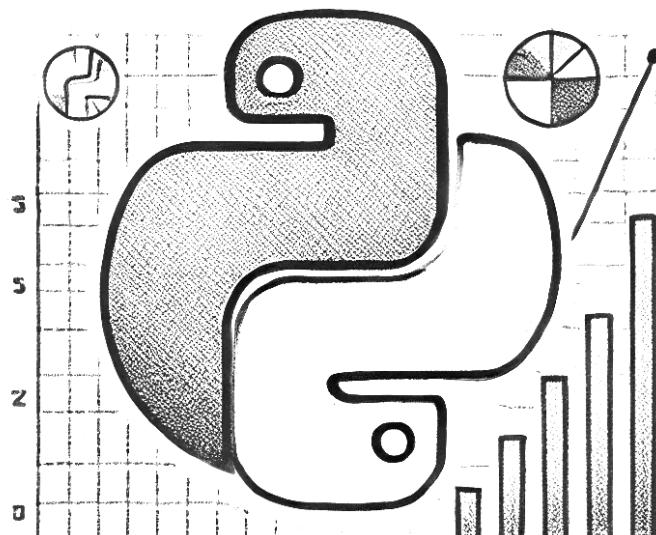


Prof. dr Predrag Katanić

Prof. dr Srđan Damjanović



PYTHON ZA EKONOMISTE

Bijeljina 2025.

PYTHON ZA EKONOMISTE

Autori:

Prof. dr Predrag Katanić
Prof. dr Srđan Damjanović

Recenzenti:

Prof. dr Branko Krsmanović
Prof. dr Slobodan Obradović
Prof. dr Danijel Mijić

Izdavač:

UNIVERZITET U ISTOČNOM SARAJEVU
FAKULTET POSLOVNE EKONOMIJE BIJELJINA

Odgovorno lice za izdavača:

Doc. dr Vesna Petrović

Odgovorni urednik:

Prof. dr Vitomir Starčević

Lektor i korektor:

Prof. dr Suzana Marković

Likovni urednik:

Vanja Đurić, dipl. ek.

Štampa:

Leader d.o.o.
Bijeljina

Za štampariju:

Mladen Dokić

Tiraž:

100 primjeraka

Godina izdanja:

2025.

ISBN: x

UDK: x

COBISS.RS-ID x

Посвета

*Бруну рада протканиу стазији успеха,
посвећујем воленом очу Радовану.*

Предраг Ђатанчић

SADRŽAJ

1. UVOD PRIJE UVODA.....	17
1.1. Python u ekonomiji.....	19
1.2. Colab okruženje za rad sa primjerima.....	24
1.3. Izmjene i prilagođenja u kodu.....	28
2. UVOD U PYTHON.....	43
2.1. Šta je Python?.....	46
2.1.1. Nastanak i razvojni put.....	46
2.1.2. Filozofija dizajna Zen Python.....	47
2.1.3. Evolucija u programiranju.....	47
2.1.4. Popularnost Pythona od škola do industrije.....	48
2.2. Šta je Jupyter Notebook.....	49
2.2.1. Koncept interaktivnih bilježnica.....	49
2.2.2. Prednosti korištenja Jupyter Notebook-a.....	50
2.2.3. Praktične primjene u učenju i istraživanju.....	50
2.3. Instalacija i podešavanje radnog okruženja.....	52
2.3.1. Verzije i subverzije Python-a.....	52
2.3.2. Instalacija Python-a u Windows okruženju.....	53
2.3.3. Instalacija Python-a u Linux okruženju.....	58
2.3.4. Instalacija Python virtualnog okruženja.....	59
2.4. Instalacija i pokretanje JupyterLab-a.....	65
2.4.1. Instaliranje JupyterLab-a.....	65
2.4.2. Pokretanje JupyterLab-a.....	66
2.4.3. Razlike između JupyterLab-a i Jupyter Notebook-a.....	67
2.5. Osnove rada u JupyterLab okruženju.....	69
2.5.1. Navigacija kroz Jupyter interfejs.....	69
2.5.2. Struktura radne sveske u JupyterLab-u.....	73
2.5.2.1. Ćelije za kod (Code cells).....	73
2.5.2.2. Markdown ćelije (Markdown cells).....	75
2.5.2.3. Raw ćelije (Raw cells).....	77
2.5.2.4. Upotreba Raw ćelije.....	77
2.5.2.5. Primjer upotrebe Raw ćelije.....	78
2.5.2.6. Prednosti Raw ćelija.....	78
2.5.3. Osnovne komande i prečice u JupyterLab-u.....	78
2.5.3.1. Izvršavanje ćelija.....	78

2.5.3.2. Dodavanje i brisanje ćelija.....	79
2.5.3.3. Prebacivanje između tipova ćelija.....	79
2.5.3.4. Prelazak između načina uređivanja i komandi.....	79
2.5.4. Savjeti za efikasan rad u JupyterLab-u.....	80
2.6. Prvi koraci sa Python-om u Jupyter-u.....	82
2.6.1. Pisanje jednostavnih izraza.....	82
2.6.2. Rad sa komentarima.....	83
2.6.3. Upoznavanje sa greškama.....	83
2.6.4. Praktične vježbe.....	84
2.7. Dodatni resursi i naredni koraci.....	86
3. OSNOVNI KONCEPTI.....	89
3.1. Tipovi podataka.....	92
3.1.1. Integer (cjelobrojne vrijednosti).....	93
3.1.2. Float (realni brojevi s decimalnim dijelom).....	94
3.1.3. String (niz znakova).....	95
3.1.4. Bool (logička vrijednost).....	95
3.1.5. Konverzija prostih tipova podataka.....	96
3.1.6. Liste (lists).....	96
3.1.7. n-Torke (tuples).....	97
3.1.8. Rječnici (dictionaries).....	98
3.1.9. Skupovi (sets).....	98
3.2. Varijable, operatori i izrazi.....	99
3.2.1. Definisanje i korištenje varijabli.....	99
3.2.2. Operatori i izrazi.....	100
3.2.2.1. Matematički operatori.....	100
3.2.2.2. Relacioni operatori.....	101
3.2.2.3. Logički operatori.....	102
3.2.2.4. Izrazi.....	103
3.3. Kontrolne strukture.....	104
3.3.1. Uslovne naredbe (if, elif, else).....	104
3.3.2. Petlje (for, while).....	108
3.3.2.1. For petlja.....	109
3.3.2.2. While petlja.....	111
3.3.3. Kombinovanje uslovnih naredbi i petlji.....	113
3.4. Funkcije i moduli.....	115
3.4.1. Korisnički definisane funkcije.....	115
3.4.2. Lambda funkcije.....	116

3.4.3. Ugrađene funkcije.....	119
3.4.4. Funkcije iz modula i biblioteka.....	122
3.5. Osnovni tipovi fajlova.....	125
3.5.1. CSV (Comma-Separated Values).....	125
3.5.1.1. Čitanje podataka iz CSV fajla.....	125
3.5.1.2. Upisivanje podataka u CSV fajl.....	126
3.5.2. Excel (XLS, XLSX).....	127
3.5.2.1. Čitanje podataka iz Excel fajla.....	128
3.5.2.2. Upisivanje podataka u Excel fajl.....	129
3.5.3. JSON (JavaScript Object Notation).....	130
3.5.3.1. Čitanje podataka iz JSON fajla.....	130
3.5.3.2. Upisivanje podataka u JSON fajl.....	131
3.5.4. YAML (Yet Another Markup Language).....	132
3.5.4.1. Čitanje podataka iz YAML fajla.....	132
3.5.4.2. Upisivanje podataka u YAML fajl.....	133
4. OSNOVNI ALATI.....	135
4.1. Math modul.....	138
4.1.1. Matematičke konstante.....	138
4.1.2. Funkcije za zaokruživanje i apsolutne vrijednosti.....	139
4.1.3. Eksponencijalne i logaritamske funkcije.....	140
4.1.4. Funkcije za faktore i kombinatoriku.....	140
4.1.5. Trigonometrijske funkcije.....	141
4.1.6. Ostale korisne funkcije.....	142
4.1.7. Upotreba modula Math u ekonomiji.....	142
4.2. NumPy Modul.....	144
4.2.1. Kreiranje nizova.....	145
4.2.2. Sumiranje elemenata niza.....	146
4.2.3. Prosječna vrijednost niza.....	149
4.2.4. Jedinična (identična) matrica.....	150
4.2.5. Inverzija matrice.....	150
4.2.6. Determinanta matrice.....	151
4.2.7. Rang matrice.....	152
4.2.8. Transponovanje matrice.....	153
4.3. SciPy modul.....	154
4.3.1. Optimizacija.....	154
4.3.2. Integracija.....	155
4.3.3. Interpolacija.....	155

4.3.4. Linearna algebra.....	156
4.3.5. Numerički izvodi.....	157
4.3.6. Diferencijalne jednačine.....	158
4.4. Matplotlib modul.....	160
4.4.1. Kreiranje osnovnog grafika.....	160
4.4.2. Prilagođavanje grafika.....	162
4.4.3. Tipovi grafika.....	166
4.4.3.1. Linijiški grafik.....	167
4.4.3.2. Grafik u obliku stubova.....	167
4.4.3.3. Grafik u obliku torte (pie).....	168
4.4.3.4. Histogram grafik.....	168
4.4.3.5. Kombinovani grafik.....	169
4.4.3.6. Scatter plot (grafik sa dvije varijable).....	170
4.5. Pandas modul.....	171
4.5.1. Osnovne strukture podataka.....	172
4.5.1.1. Series.....	173
4.5.1.2. DataFrame.....	174
4.5.2. Rad sa Series i DataFrame objektima.....	175
4.5.2.1. Kreiranje Series objekta.....	175
4.5.2.2. Kreiranje DataFrame objekata.....	176
4.5.2.3. Pristup i manipulacija podacima.....	179
Pristup redovima ili kolonama pomoću indeksa pozicije - iloc.....	179
Pristup redovima ili kolonama pomoću imena - loc.....	181
Dodavanje nove kolone.....	182
Dodavanje novog reda.....	182
Brisanje kolone i reda.....	184
Ažuriranje postojećih podataka.....	185
4.5.2.3. Učitavanje podataka.....	186
4.5.3.1. Učitavanje podataka iz CSV datoteke.....	186
4.5.3.2. Učitavanje podataka iz Excel datoteke.....	187
4.5.3.3. Učitavanje podataka iz JSON datoteke.....	188
4.5.3.4. Učitavanje podataka iz baze podataka.....	189
4.5.4. Snimanje podataka.....	190
4.5.4.1. Snimanje podataka u CSV format.....	190
4.5.4.2. Snimanje podataka u Excel format.....	191
4.5.4.3. Snimanje podataka u JSON format.....	192
4.5.4.4. Snimanje podataka u MySQL bazu.....	193

4.5.5. Analiza i manipulacija podacima.....	194
4.5.5.1. Filtriranje i selekcija.....	194
4.5.5.2. Grupisanje i agregacija.....	195
4.5.5.3. Spajanje podataka.....	196
4.5.5.4. Rad s nedostajućim podacima.....	199
4.5.5.5. Rad sa datumima i vremenom.....	201
4.5.6. Vizualizacija podataka koristeći Pandas.....	202
4.5.6.1. Osnovni grafik u Pandas-u.....	203
4.5.6.2. Integracija s Matplotlib.....	205
4.5.6.3. Tipovi grafika.....	207
Bar grafik.....	207
Histogram grafik.....	207
Boxplot grafik.....	208
Density grafik.....	209
Scatter grafik.....	210
Hexbin grafik.....	211
Pie grafik.....	212
5. PYTHON I MATEMATIKA.....	214
5.1. Matrice.....	217
5.1.1. Osnovni pojmovi o matricama.....	217
5.1.2. Tipovi matrica.....	221
5.1.3. Osnovne operacije nad matricama.....	222
5.1.3.1. Sabiranje i oduzimanje matrica.....	222
5.1.3.2. Sabiranje (oduzimanje) matrice i skalara.....	223
5.1.3.3. Množenje matrica.....	225
5.1.3.4. Množenje matrice i skalara.....	225
5.1.3.5. Transponovanje matrice.....	226
5.1.4. Napredne operacije sa matricama.....	227
5.1.4.1. Determinanta.....	228
5.1.4.2. Inverzna matrica.....	229
5.1.4.3. Rang matrice.....	230
5.1.4.4. Dijeljenje matrice matricom.....	232
5.1.4.5. Dijeljenje matrice skalarom.....	233
5.2. Sistemi linearnih jednačina.....	234
5.2.1. Kroneker-Kapelijeva teorema.....	236
5.2.2. Homogenost sistema.....	238
5.2.3. Model tržišne ravnoteže.....	240

5.2.4. Input-Output Analiza.....	245
5.3. Funkcije.....	248
5.3.1. Linearne funkcije.....	248
5.3.2. Kvadratna funkcija.....	251
5.3.3. Eksponencijalne funkcije.....	254
5.3.4. Logaritamske funkcije.....	256
5.3.5. Primjer optimizacije profita.....	259
5.4. Nizovi.....	262
5.4.1. Aritmetički niz.....	262
5.4.2. Geometrijski niz.....	263
5.4.3. Grafički prikaz aritmetičkog i geometriskog niza.....	263
5.4.4. Granična vrijednost niza.....	265
5.5. Diferencijalni račun.....	267
5.5.1. Granične vrijednosti i neprekidnost funkcije.....	267
5.5.2. Izvod funkcija.....	271
5.6. Integralni račun.....	274
5.6.1. Primjer računanja neto viška dobiti.....	276
5.6.2. Primjer računanja prosječne akumulirane dobiti.....	278
6. PYTHON I STATISTIKA.....	280
6.1. Osnove statistike.....	282
6.1.1. Skup podataka i vrste podataka.....	282
6.1.2. Distribucija frekvencija.....	284
6.1.3. Mjere distribucije podataka (Kvantili).....	286
6.1.4. Mjere centralne tendencije.....	289
6.1.4.1. Aritmetička sredina.....	289
6.1.4.2. Geometrijska sredina.....	290
6.1.4.3. Harmonijska sredina.....	292
6.1.4.4. Medijana.....	294
6.1.4.5. Modus.....	296
6.1.5. Mjere rasipanja (disperzije).....	298
6.1.5.1. Apsolutne mjere disperzije.....	298
Razmak (Range).....	299
Interkvartilni razmak.....	300
Varijansa i standardna devijacija.....	301
6.1.5.2. Empirijsko pravilo i Čebišovljeva teorema.....	303
6.1.5.3. Relativne mjere disperzije.....	308
Koeficijent varijacije (Coefficient of Variation - CV).....	308

Relativni razmak (Range Ratio).....	309
6.2. Osnovni pojmovi vjerovatnoće.....	311
6.2.1. Pojam vjerovatnoće.....	311
6.2.2. Zakon velikih brojeva.....	312
6.2.3. Teorema o ukupnoj vjerovatnoći (Law of Total Probability).....	313
6.2.4. Bayesova teorema.....	315
6.3. Statističko zaključivanje (Inferencijalna statistika).....	316
6.3.1. Osnovni pojmovi testiranja hipoteza.....	316
6.3.1.1. Nulta i alternativna hipoteza.....	316
6.3.1.2. Statistička značajnost i P-vrijednost.....	316
6.3.1.3. Greške tipa I i tipa II.....	317
6.3.2. Parametarski testovi.....	317
6.3.2.1. Studentov t-test.....	317
6.3.2.2. Varijacije Studentovog t-testa.....	319
6.3.2.3. ANOVA test.....	319
6.3.3. Neparametarski testovi.....	321
6.3.3.1. Chi-kvadrat test.....	321
6.3.3.2. Mann-Whitney u-test.....	323
6.4. Primjena statistike u ekonomiji.....	326
6.4.1. Deskriptivna statistika u ekonomiji.....	326
6.4.1.1. Analiza tržišta.....	326
Izračunavanje prosječne cijene.....	327
Raspon cijena (Range, Min, Max).....	328
Stopa inflacije.....	328
6.4.1.2. Makroekonomski indikatori.....	329
Bruto domaći proizvod (BDP / GDP).....	329
Nezaposlenost.....	331
Kamatne stope.....	335
6.4.1.3. Praćenje poslovnih ciklusa.....	338
6.4.2. Regresiona i korelaciona analiza.....	340
6.4.2.1. Korelaciona analiza.....	341
6.4.2.2. Regresiona analiza.....	343
6.4.3. Prognoziranje u ekonomiji (Forecasting).....	347
6.4.3.1. Jednostavne metode ekstrapolacije.....	347
Pokretni prosjek (Moving Average - MA).....	348
Eksponencijalno poliranje (Exponential Smoothing - ES).....	349
Linearna ekstrapolacija (Trend Projection).....	351

6.4.3.2. Vremenske serije.....	353
6.4.3.3. Složeni ekonometrijski modeli.....	358
7. FINANSIJSKA MATEMATIKA.....	360
7.1. Procentni račun.....	363
7.2. Prost interesni račun.....	366
7.3. Složeni interesni račun.....	370
7.4. Nominalna i efektivna kamatna stopa.....	376
7.5. Obračun potrošačkog kredita.....	379
7.6. Faktor akumulacije - Prve finansijske tablice.....	385
7.7. Diskontni faktor - Druge finansijske tablice.....	390
7.8. Faktor dodatnih uloga - Treće finansijske tablice.....	396
7.9. Faktor aktualizacije - Četvrte finansijske tablice.....	400
7.10. Faktor povrata - Pete finansijske tablice.....	405
8. Prilog A.....	412
A.1. Komande u Windows Command Prompt-u (CMD).....	413
A.1.1. Navigacija kroz direktorijume.....	413
A.1.2. Upravljanje direktorijumima (direktorijumima).....	413
A.1.3. Upravljanje fajlovima.....	414
A.1.4. Pregled i čitanje sadržaja fajlova.....	415
A.1.5. Dodatne korisne komande.....	415
A.2. Komande u Linux terminalu.....	416
A.2.1. Navigacija kroz direktorijume.....	416
A.2.2. Upravljanje direktorijumima.....	416
A.2.3. Upravljanje fajlovima.....	417
A.2.4. Pregled i čitanje sadržaja fajlova.....	418
A.2.5. Upravljanje pravima pristupa.....	418
A.2.6. Dodatne korisne komande.....	419
A.3. Dodatne napomene.....	419
9. LITERATURA.....	420

PREDGOVOR

Programski jezik Python izabran je kako zbog svoje jednostavnosti, tako i zbog svoje otvorenosti i opšte prisutnosti. U knjizi se ne razmatraju sve mogućnosti i specifičnosti ovog programskog jezika, već je pažnja najviše bila usmjerena na primjenu ovog programskog jezika u ekonomiji. Kao i svaka druga vještina, programiranje zahtjeva od početnika strpljenje i upornost. Olakšavajuća okolnost u procesu učenja ovog programskog jezika se ogleda u mogućnosti da se uloženi rad vrlo brzo testira na racunaru. Zbog svoje jednostavnosti u pisanju programa i lakom praćenju toka izvršenja programa, ovaj programski jezik može da posluži početnicima u programiranju za savladavanje prvih koraka u programiranju. Cilj ovog izdanja je da se prilagodi potrebama studenata u početnoj fazi učenja programiranja, kao i svima onima koji su se ranije bavili programiranjem ali u nekom drugom programskom jeziku. Izdavanjem ove knjige nadomještena je literatura, koja nedostaje za izvođenje predavanja i vježbi iz predmeta Programski jezici, Finansijska matematika, Teorija i analiza bilansa, Upravljački informacioni sistemi, Metodologija naučnoistraživačkog rada i Poslovne baze podataka na Fakultetu poslovne ekonomije u Bijeljini.

Knjiga je napisana zbog nedostatka na tržištu literature na srpskom jeziku sa praktičnim primjerima za programske jezike Python. Naime, postoje samo uputstva za rad sa raznim verzijama programskog jezika Python, koja su napisana na engleskom jeziku i dostupna su u elektronskom obliku. Na relativno skromnom broju stranica izložena je jedna konzistentna cjelina pogodna za edukaciju u obrazovnim ustanovama. Knjiga je posebno aktuelna za studente i čitaoce, koji se samostalno bave programiranjem u programskom jeziku Python. Ponuđeni primjeri su, zbog konciznosti izlaganja, djelimično uprošćeni, ali se nevelikim trudom mogu brzo i efikasno dovesti do profesionalne aplikacije. Obradene oblasti su raznovrsne i brižljivo birane, kako bi čitaoci mogli da uvide višestruke primjene obrađenog materijala u praksi. Iako je fokus ove knjige na ekonomskim disciplinama, primjeri i pristupi obrađeni u knjizi mogu se primjeniti i u drugim oblastima. Svi primjeri iz knjige dostupni su na internetu putem GitHub servisa, a u knjizi je detaljno objašnjen način njihove upotrebe, kao i korišćenje GitHub platforme za verzionisanje i preuzimanje koda.

Knjiga je u osnovi podijeljena na sedam poglavlja, koja čine dvije zasebne cjeline. Prvu cjelinu čini poglavlje jedan, dva, tri i četiri u kojima su date osnove za rad u programskom jeziku Python. Drugu cjelinu čine poglavlja pet do sedam, u kojima su date osnove primjene ovog programskog jezika u matematici, statistici i finansijskoj matematici.

Za svakog programera je bitno da prepozna oblasti u kojima se može koristiti programski jezik, koji se želi naučiti. Python je izuzetno moćan i svestran alat, koji se široko koristi u analizi podataka i automatizaciji procesa. Dati su osnovni koraci, koji čine proces pisanja programa. Na kraju prvog poglavlja kako bi knjigu učini korisnom,

a u isto vrijeme i zanimljivom za čitanje, predstavljen je praktičan primjer programa za analizu cijena kretanja akcija.

Postupak instaliranja programa Python predstavljen je u drugom poglavlju. Opisana su radna okruženja u kojima se može pisati i izvršavati kod ovog programskog jezika. Poseban akcenat stavljen je na okruženje Google Colab, koje omogućava rad sa Python-om bez potrebe za lokalnom instalacijom softvera, što ga čini idealnim za studente i korisnike koji žele brzo testirati kod u oblaku. Takođe, obrađeno je i okruženje Jupyter Lab, jedno od najmoćnijih razvojnih okruženja za analizu podataka i interaktivno programiranje, koje se može jednostavno instalirati i pokrenuti na personalnom računaru.

U trećem poglavlju opisani su osnovni tipovi podataka, koji se koriste prilikom pisanja programa u programskom jeziku Python. Uz svaki tip podatka prikazan je primjer programa, koji opisuje kako se on koristi. Detaljno su opisani razni postupci pravljenja grananja u programu kroz primjere, kako bi se što lakše mogli koristiti za pisanje programa. Predstavljen je i način višestrukog ponavljanja nekih naredbi, kako da se značajno smanji broj kodnih linija u programu.

Python raspolaže velikim brojem modula i biblioteka, koji se mogu koristiti za različite aspekte analize podataka i modeliranja. Za početno upoznavanje primjene Python-a u ekonomskoj analizi, najznačajniji alati su Math, NumPy, SciPy, Pandas i Matplotlib. Ove biblioteke pokrivaju širok opseg matematičkih, statističkih i vizuelizacionih potreba i opisane su u četvrtom poglavlju. U ovo poglavlju dati su praktični primjeri primjene ovih ključnih alata za uspješnu i efikasnu ekonomsku analizu u Python-u.

U petom poglavlju predstavljena je primjena ovog programskog jezika u matematici. Ekonomija se oslanja na brojne matematičke metode, kako bi analizirala podatke i donosila argumentovane zaključke. Korištenjem ovih metoda može se preciznije opisati i predvidjeti ponašanje tržišta, izračunati različiti finansijski pokazatelji ili izvršiti kompleksne analize troškova i prihoda. Uvođenje matematičkih i statističkih koncepta u ekonomske analize povećava tačnost modela i pomaže u boljem razumijevanju odnosa među varijablama.

Statistika je moćan alat u ekonomiji, a praktični primjeri primjene dati su u šestom poglavlju. Primjenom statističkih metoda, ekonomisti mogu donositi utemeljene zaključke zasnovane na podacima, identifikovati važne obrasce i trendove, te predvidjeti buduća kretanja u različitim segmentima ekonomije. Takođe, statistika pruža osnovu za razumijevanje i kvantifikaciju kompleksnih odnosa između ekonomskih varijabli, čime postaje neophodna za donošenje ispravnih odluka u poslovanju, upravljanju i ekonomskim politikama.

U sedmom poglavlju predstavljena je primjena ovog programskog jezika u finansijskoj matematici. Finansijska matematika predstavlja nezaobilazan stub ekonomskih analiza i donošenja racionalnih odluka u poslovanju. Ona se bavi

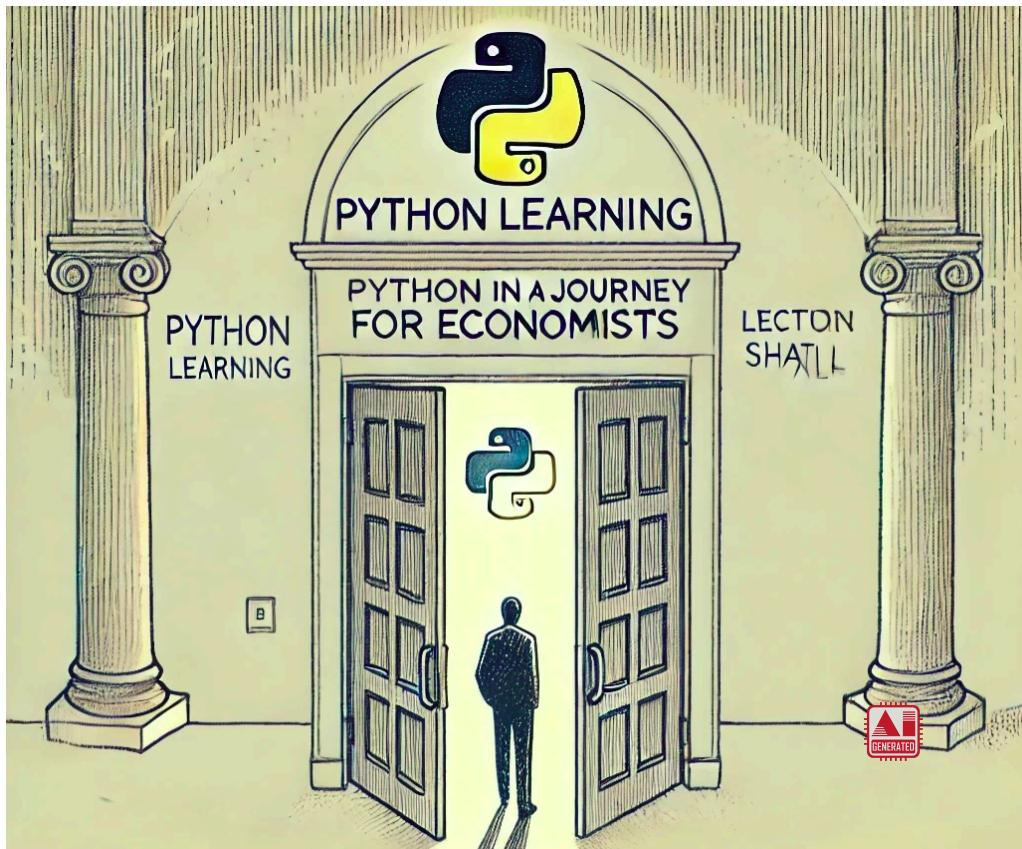
proračunima vrijednosti investicija, kredita i novčanih tokova u različitim vremenskim periodima, što čini osnovu za planiranje, budžetiranje i strategijsko odlučivanje. Upravo tu se Python, kao jednostavan i izuzetno fleksibilan programski jezik, pokazuje kao moćan saveznik. Zahvaljujući bibliotekama poput NumPy, Pandas i SciPy, ekonomski i finansijska analiza podataka može se značajno ubrzati i pojednostaviti.

Ova knjiga pokriva osnovne koncepte rada u Python-u i njegovu primjenu u ekonomiji, dok će u narednim proširenim izdanjima autori obuhvatiti i druge specifične oblasti, kao što su računovodstvo, finansijske analize i berzansko poslovanje, napredne metode u finansijskoj matematici, analiza rizika, optimizacija portfelja, kao i primjena mašinskog učenja u ekonomskoj prognozi i donošenju poslovnih odluka.

Nadamo se da će ova kniga biti od koristi svima onima, koji se bave programiranjem u programskom jeziku Python. Sugestije i pitanja mogu se slati na adresu predrag.katanic@fpe.ues.rs.ba i srdjan.damjanovic@fpe.ues.rs.ba.

Autori

1. UVOD PRIJE UVODA



“Somewhere, something incredible is waiting to be known.”
“Negdje, nešto nevjerojatno čeka da bude otkriveno.”

— Carl Sagan¹

¹ Carl Sagan (1934. – 1996.) bio je američki astronom i naučni pisac. Kao popularna i uticajna ličnost u Sjedinjenim Američkim Državama, bio je kontroverzan u naučnim, političkim i religijskim krugovima zbog svojih stavova o vanzemaljskoj inteligenciji, nuklearnom oružju i religiji. Izvor: Britannica

Dugo smo razmišljali o tome kako napraviti dobar uvod u školski udžbenik. Kako započeti knjigu za studente, a da nakon prve stranice ne počnu letimično prelistavati knjigu od korice do korice i potom je odložiti na policu. Da bismo izbjegli taj monotoni pristup odlučili smo da na samom početku, prije uvoda kroz nekoliko praktičnih, ali ne previše složenih primjera zainteresiramo čitaoca i djelimično im prikažemo snagu programskog jezika Python i njegovu primjenu u ekonomiji.

Da bi vas zaintrigirali za rad u Python okruženju, na kraju ovog poglavlja, pokazat ćemo vam kako te primjere možete generisati uz pomoć **vještačke inteligencije (AI)**². Naša ideja je da demistifikujemo kompleksnost učenja i rješavanja problema, ali istovremeno da istaknemo neophodnost poznavanja osnovnih alata koje koristimo, kako bi što efikasnije mogli iskoristiti pomoć AI tehnologije.

1.1. Python u ekonomiji

Da li ste ikada razmišljali koliko vremena i truda je potrebno da prikupite podatke o kretanju akcija neke kompanije ili da dođete do informacija o bruto domaćem proizvodu (BDP) određene zemlje? Možda ste se suočili sa izazovom pronalaženja pouzdanih izvora, ručnog preuzimanja podataka i njihovog organizovanja u format pogodan za analizu. Nakon toga, postavlja se pitanje: Kako efikasno da obraditi te podatke i kako ih prezentovati na jasan i razumljiv način, koji će omogućiti donošenje pravilnih odluka.

U savremenom okruženju, dostupnost podataka je veća nego ikada prije. Internet obiluje informacijama iz različitih izvora: od finansijskih portalova, preko statističkih baza podataka, do otvorenih podataka vladinih agencija. Međutim, ovo obilje podataka može biti mač sa dvije oštice. Dok s jedne strane imamo pristup ogromnoj količini informacija, s druge strane se suočavamo sa izazovom njihove efikasne obrade i analize. Potrebni su nam alati, koji mogu da se nose sa ovim izazovima i pretvore sirove podatke u upotrebljive informacije.

Python je upravo takav alat. Riječ je o moćnom i svestranom programskom jeziku, koji je postao standard u oblasti analize podataka, mašinskog učenja i automatizacije procesa. Njegova jednostavnost, čitljivost i širok spektar biblioteka čine ga idealnim izborom za ekonomiste, koji žele da unaprijede svoje analitičke sposobnosti.

U ovom poglavlju - uvodu prije uvoda, želimo pokazati kako se bez početnih znanja iz Pythona, može brzo i jednostavno pristupiti podacima, obraditi ih i napraviti grafike. Neka vas Python kod koji slijedi ne obeshrabri, fokusirajte se na rezultate koje možemo postići. Čak i ako nemate prethodno iskustvo u programiranju, kroz ovu knjigu ćete naučiti kako da savladate osnove i primijenite ih u praktičnim situacijama.

² AI (vještačka inteligencija) – vaš digitalni kolega koji nikada ne spava, ne piće kafu i ne pita za odmor. Radi sve što mu kažete (ako znate šta da mu kažete) i često daje odgovore za koje ste sigurni da ih je našao u nekom paralelnom univerzumu. Izvor: ovaj tekst je kreirala AI, a vi čitate rezultat zajedničkog truda čoveka i mašine. 

Šta možemo sa Python-om?

Python je izuzetno moćan i svestran alat, koji se široko koristi u analizi podataka i automatizaciji procesa. Njegova primjena omogućava sljedeće:

- Automatizovati prikupljanje podataka: Umjesto ručnog preuzimanja podataka, moguće je pisati skripte, koje automatski preuzimaju najnovije informacije sa relevantnih web stranica ili baza podataka.
- Efikasno obraditi velike skupove podataka: Sa bibliotekama poput Pandas i NumPy, manipulacija i analiza podataka postaju znatno brže i jednostavnije.
- Vizualizovati podatke na profesionalan način: Biblioteke kao što su Matplotlib i Seaborn omogućavaju kreiranje visokokvalitetnih grafika i dijagrama, koji olakšavaju interpretaciju rezultata i donošenje odluka.

Zašto baš Python?

Postoji mnogo razloga zašto je Python postao jedan od najpopularnijih programskih jezika, a neki od ključnih razloga su:

- Jednostavnost i čitljivost: Python sintaksa je intuitivna i bliska prirodnom jeziku, što olakšava učenje i razumijevanje koda.
- Zajednica i podrška: Velika i aktivna zajednica korisnika znači da je dostupno mnoštvo resursa, tutorijala i odgovora na pitanja, koja se mogu pojaviti tokom učenja.
- Svestranost: Python se koristi u raznim industrijama i naučnim oblastima, što znači da će znanje ovog jezika biti korisno bez obzira na to gdje vas karijera odvede.

Šta može Python u ekonomiji?

Iako je fokus ove knjige na ekonomskim disciplinama, primjeri i pristupi obrađeni u knjizi mogu se primjeniti i u drugim oblastima. Međutim, u ekonomiji Python ima posebne prednosti:

- Bržu analizu: Bilo da se radi o analizi finansijskih tržišta, predviđanju trendova ili istraživanju potrošačkih navika, Python pruža alate za efikasnu obradu i analizu podataka.
- Poboljšanu preciznost: Automatizacijom procesa smanjuje se mogućnost ljudske greške, što čini rezultate tačnijim.
- Uštedu vremena i resursa: Umjesto da trošite sate na ručne proračune i manipulaciju podacima, možete se fokusirati na analizu i tumačenje rezultata, a zatim i strateško donošenje odluka.

Šta možete očekivati u nastavku?

Kako bi knjigu učini korisnom, a u isto vrijeme i zanimljivom za čitanje, svako poglavlje sadrži sljedeće elemente:

- Praktične primjere: Kroz konkretnе slučajeve, pokazaćemo kako da se primjeni Python u različitim oblastima ekonomije.
- Korak-po-korak uputstva: Detaljna objašnjenja će pomoći da se razumije svaki korak procesa, od prikupljanja podataka do njihove vizualizacije.
- Vježbe za samostalni rad: Kako biste utvrdili naučeno, pripremili smo zadatke koji će vas podstići na kreativno razmišljanje i primjenu stečenih znanja.

Ne dozvolite da vas tehnološki izazovi obeshrabre. Svaki početak je težak, ali uz prave alate i podršku, možete savladati i najsloženije zadatke. Naša je želja da vam kroz ovu knjigu pružimo znanje i vještine, koje će vam biti od koristi tokom studija, ali i kasnije u profesionalnom radu.

Pridružite nam se na ovom uzbudljivom putovanju kroz svijet Pythona i otkrijte kako možete unaprijediti svoje analitičke sposobnosti, donijeti bolje odluke i postati konkurentniji na tržištu rada. Vaš trud će se svakako isplatiti, a mi smo tu da vas vodimo kroz svaki korak.

U nastavku ćemo kroz tri primjera pokušati dočarati samo jedan mali djelić mogućnosti Python jezika, kako bi vas zainteresovali za dalje čitanje knjige. Prva dva primjera su vrlo praktična i njih ćemo zajedno da realizujemo. Treći primjer ćemo raditi zajedno sa AI i kroz njega želimo da pokažemo praktičnu upotrebu AI u rješavanju sličnih zadataka.

Primjer 1. Analiza kretanja cijena akcija kompanija

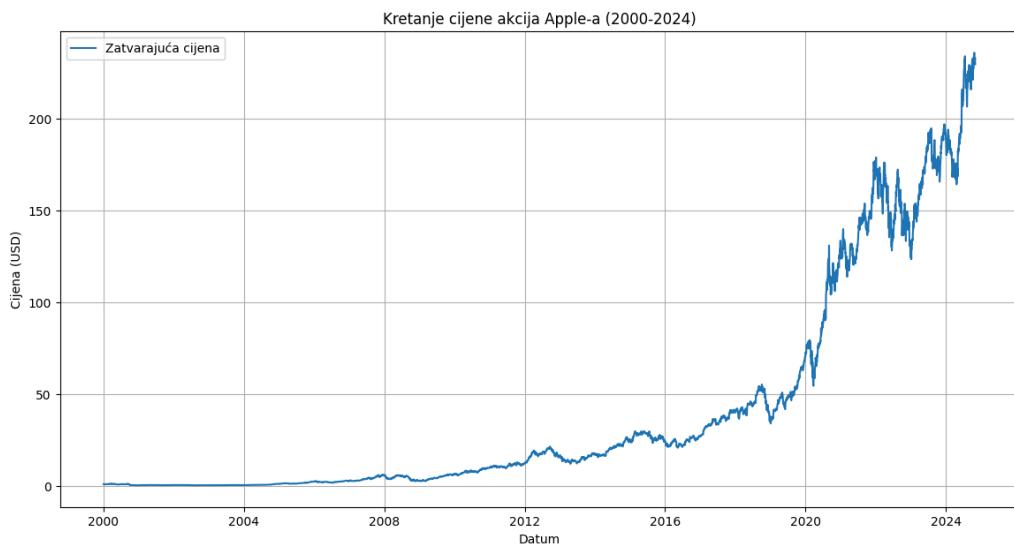
Finansijska tržišta su dinamična i kompleksna, a cijene akcija kompanija poput Apple, Google ili Tesle su često u fokusu investitora, analitičara i ekonomista. Razumijevanje kako se cijene ovih akcija mijenjaju kroz vrijeme može pružiti dragocjene uvide u performanse kompanija, tržišne trendove i šire ekonomske indikatore.

Problem koji želimo riješiti:

- Kako efikasno pristupiti istorijskim podacima o cijenama akcija?
- Kako obraditi te podatke za analizu?
- Kako vizualizirati kretanje cijena na način, koji olakšava interpretaciju?

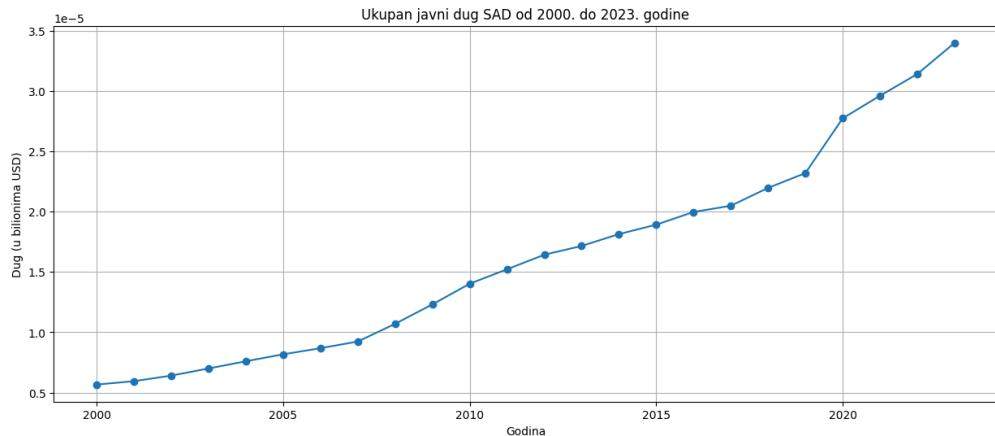
Tradicionalni pristup mogao bi uključivati ručno preuzimanje podataka sa finansijskih web stranica, unos tih podataka u Excel tabele i zatim kreiranje grafika. Ovaj proces je dugotrajan i podložan greškama. Uz pomoć Pythona, možemo automatizirati ovaj proces i fokusirati se na analizu i donošenje zaključaka.

Pogledajte grafik na slici 1.1. koji prikazuje kretanje cijene akcija kompanije Apple. Iako grafik i podaci na njemu nisu posebno stvaralačko dostignuće, vrijedno je napomenuti da svako od vas može pronaći sličan grafički prikaz na internetu. Međutim, takav prikaz bi bio samo statičan grafik, odnosno obična slika. Ono što želimo postići je upravljanje ovom slikom, kombinovanje podataka i pronalaženje zavisnosti među njima.



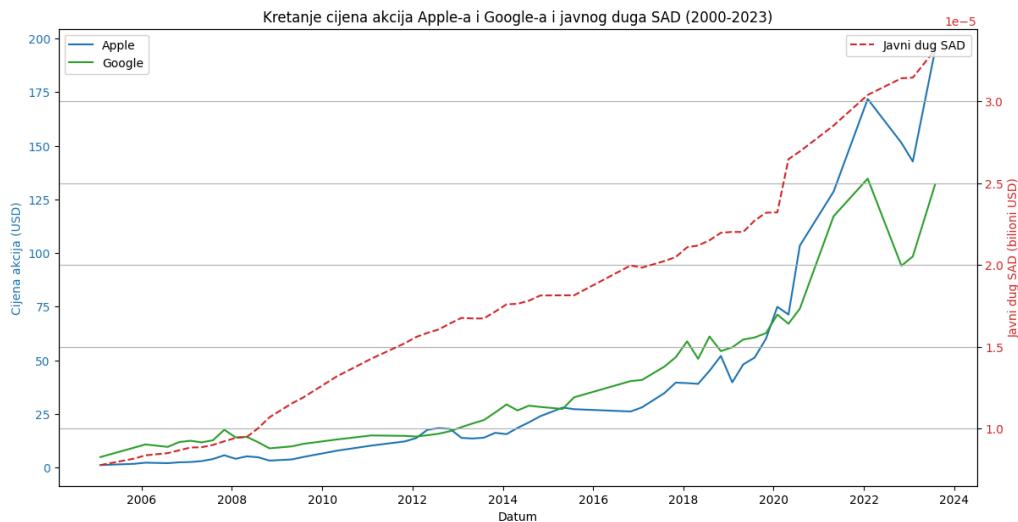
Slika 1.1 Kretanje cijene akcija Apple (2000-2024), Izvor: Autori.

Zamislite da možete uporediti cijene akcija kompanije Apple sa trendom zaduženje SAD u istom periodu, koje je prikazano na sljedećem grafiku.



Slika 1.2. Kretanje javnog duga SAD (2000-2023), Izvor: Autori.

Ovo su dva potpuno odvojena pokazatelja i ako bismo željeli prikazati korelaciju između njih, morali bismo primijeniti dodatne obrade podataka, što može predstavljati izazov. Međutim, ukoliko ovlastate osnovnim alatima programskog jezika Python, taj zadatak može postati veoma lak i inspirativan. Sljedeći grafik upravo prikazuje tu korelaciju.



Slika 1.3. Uporedni grafikon kretanja cijena akcija i javnog duga, Izvor: Autori.

U nastavku nećemo prikazivati kompletan Python kod, koji je generisao ove grafike, kako bismo izbjegli moguću konfuziju. Nije nam ideja da direktno prepisujete kod u Python fajl ili neko razvojno okruženje. Umjesto toga, uradit ćemo nešto drugačije: pokazat ćemo vam kako da pristupite izvornom kodu, koji je kreirao ove grafike, kao i način na koji možete da ga pokrenete i eventualno izmjenite.

1.2. Colab okruženje za rad sa primjerima

Prepostavimo da imate pristup Google Gmail servisu i povezanim servisom Google Colab. Ukoliko trenutno nemate ove servise, ne brinite. Date primjere ćete moći isprobati nakon prvog poglavlja, kada ćete naučiti kako da postavite svoje Python okruženje koristeći Jupyter Lab ili direktno Python na svom računaru.

Savjetujemo vam da, ukoliko nemate Gmail nalog, otvorite jedan prije nastavka čitanja ovog poglavlja. Na taj način moći ćete u potpunosti sagledati snagu i jednostavnost rada u Pythonu. Sam proces otvaranja Gmail naloga je vrlo jednostavan i besplatan, te ne zahtijeva nikakve finansijske troškove.

U nastavku ćemo vas korak po korak uputiti kako da koristite Colab okruženje. Prikazaćemo:

- Kako da pristupite izvornom kodu za svaku skriptu?
- Kako da učitate Python skriptu?
- Kako da pokrenete skriptu koristeći Google Colab?

Ove instrukcije su primjenjive na sve primjere, koje ćemo pokazati u narednim poglavljima ove knjige. Uvijek se možete vratiti na ove korake, kako biste otvorili neki primjer koda, izvršili analizu ili napravili željene izmjene.

U ovom poglavlju nismo se posebno fokusirali na detaljno objašnjenje svih funkcionalnosti Colab okruženja. Akcenat smo stavili na konkretnе primjere i objasnili samo neophodne korake za rad. U sljedećem poglavlju ćemo detaljnije obraditi rad u Jupyter Lab okruženju, koje ima slične funkcionalnosti kao i Colab, ali pruža veću fleksibilnost za lokalni rad i prilagođavanje korisničkom okruženju.

Redoslijed koraka je:

1) Otvaranje Google Colab-a

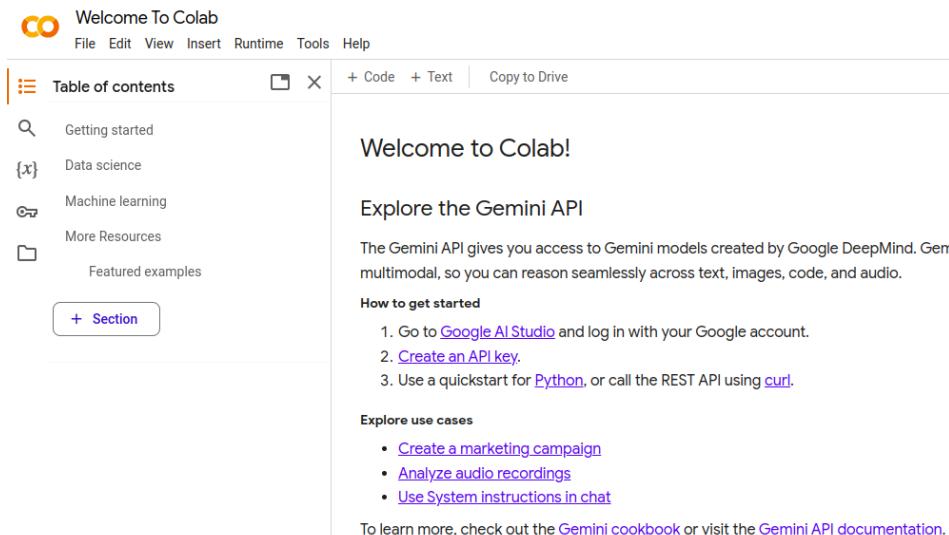
- U web pretraživaču otvorite link: <https://colab.research.google.com/>.
- Alternativno, u Google pretraživaču ukucajte "Google Colab" i kliknite na odgovarajući link.

2) Prijavljanje na Google nalog

- Ako ste povezani sa svojim Google (Gmail) nalogom:
 - Automatski ćete dobiti personalizovani pristup Google Colab-u.
 - Moći ćete da snimate fajlove i kasnije ih ponovo pokrećete.
- Ako niste prijavljeni:
 - Dobićete anonimni pristup unutar trenutne sesije.
 - Nakon unošenje promjene od vas će se tražiti da se logujete na svoj Gmail nalog.

3) Otvaranje radnog okruženja

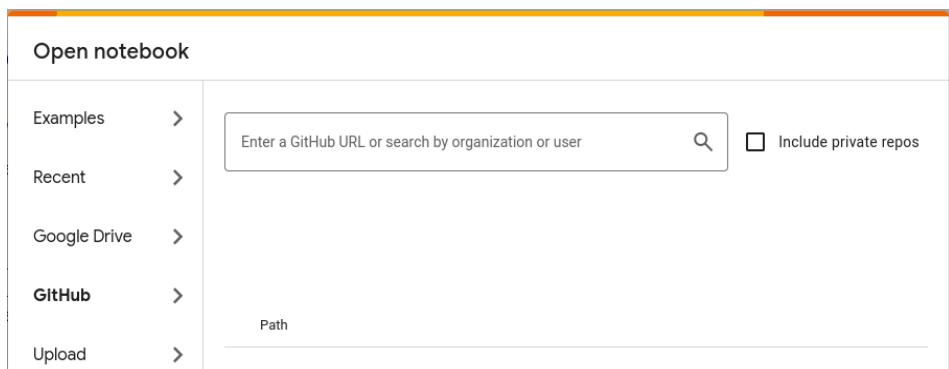
- Nakon prijavljivanja, prikazaće vam se radna površina za unos Python koda.



Slika 1.4. Početno radno okruženje Colab, Izvor: Autori.

4) Učitavanje gotovog koda iz GitHub-a

- U gornjem lijevom uglu, kliknite na "File" (Datoteka), a zatim odaberite "Open notebook" (Otvari svesku kao na slici 1.5).
- U novom prozoru, odaberite karticu "GitHub".



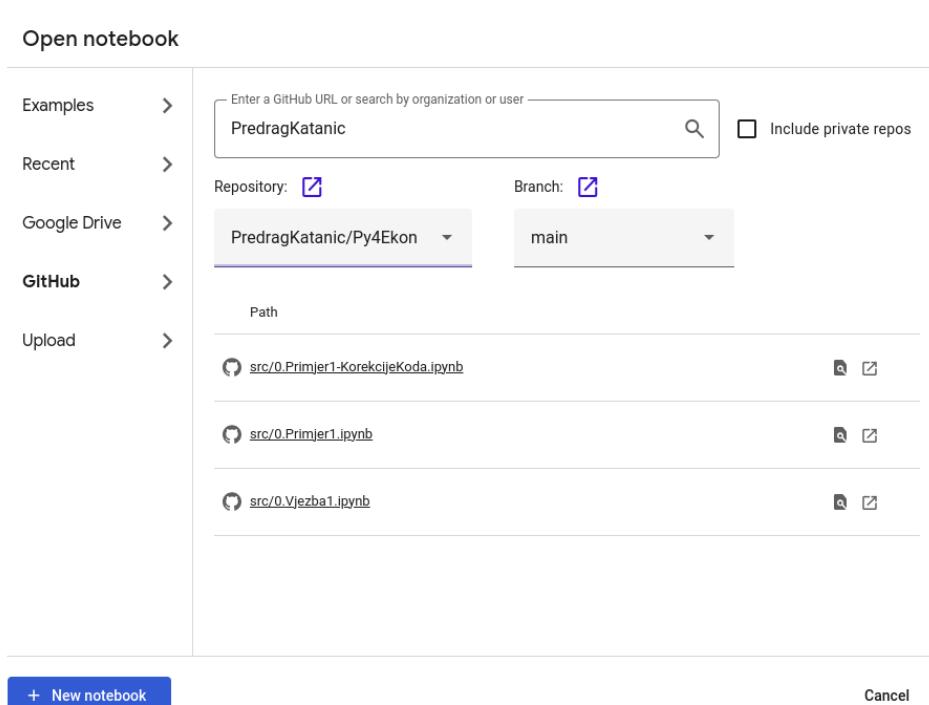
Slika 1.5. Povezivanje GitHub -a sa Colab okruženjem, Izvor: Autori.

5) Pronalaženje repozitorijuma

- U polje "Enter a GitHub URL or search by organization or user" unesite korisničko ime: [PredragKatanic](#).
- Pritisnite Enter ili kliknite na ikonu pretrage (🔍).

6) Izbor repozitorijuma i datoteke

- Iz liste repozitorijuma odaberite [PredragKatanic/Py4Ekon](#).
- U donjem dijelu prozora prikazaće se lista dostupnih notebook-ova (sveski), odnosno dokumenata sa ekstenzijom .ipynb kao na slici 1.6.



Slika 1.6. Pregled repozitorijuma, Izvor: Autori.

7) Otvaranje primjera

- Za prvi primjer, kliknite dva puta na datoteku [ch1/Primjer1.ipynb](#).
- Otvoriće vam se notebook sa kodom programa spremnim za izvršavanje.

8) Pokretanje koda

Kod unutar otvorene datoteke organizovan je u čelijama, koje su vertikalno poredane. Svaka čelija koja sadrži kod može se zasebno pokrenuti, čime se izvršava kod unutar te čelije. U ovom konkretnom primjeru, datoteka sadrži dvije čelije. Prva čelija sadrži instrukcije za instalaciju potrebnih modula, dok druga čelija sadrži kod za obradu podataka.

Ovaj način organizacije omogućava modularno izvršavanje koda i pojednostavljuje testiranje i izmjene u kodu.

Instaliranje neophodnih modula odvija se kroz sljedeće korake:

- **Prva ćelija** u notebook-u obično sadrži komande za instalaciju potrebnih biblioteka.
- Kliknuti na dugme "**Play**" [▶] koje se nalazi sa lijeve strane prve ćelije.
- **Sačekati** da se izvrši instalacija. Vidjećete poruku o uspješnoj instalaciji modula.
- **Napomena:** U okviru jedne sesije nije potrebno svaki put instalirati module. Ovo je potrebno samo ako zatvorite prozor i ponovo pokrenete Colab.

Pokretanje Python koda odvija se kroz sljedeće korake:

- Nakon instalacije modula, možete pokrenuti sljedeće ćelije sa kodom.
- Kliknite na dugme "**Play**" [▶] pored svake ćelije.
- **Rezultati** će se pojaviti ispod svake ćelije nakon izvršavanja.

```
# Import module
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt

# Preuzimanje podataka o akcijama kompanije Apple (AAPL)
symbol = 'AAPL' # Ticker za Apple
```

Slika 1.7. Pregled koda u Colab, Izvor: Autori.

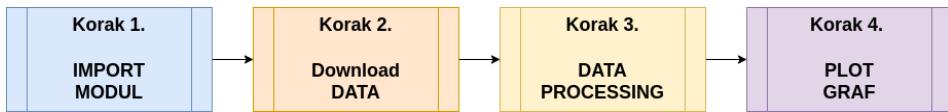
Prva ćelija je instalacija neophodnih modula, dok je druga ćelija skripta koja treba da kreira grafički prikaz kretanja cijena akcija za kompaniju Apple.

1.3. Izmjene i prilagođenja u kodu

Prije nego što započnemo sa izmjenama u kodu, pokušajmo ukratko da prođemo kroz postojeći kod i istaknemo neke od njegovih osnovnih cjelina, kao i principe koje smo koristili i koje ćemo koristiti u nastavku knjige.

Kod koji ćemo pisati u početku predstavlja skup komandi, koje se izvršavaju linijski, jedna za drugom i za njih ćemo koristiti naziv **skripta**.

Na sljedećoj slici 1.8. prikazan je blok dijagram izvršenje Python skripte, koja analizira kretanje cijene akcija kompanije Apple, a koju smo učitali u naše Colab okruženje.



Slika 1.8. Blok dijagram izvršenje Python skripte, Izvor: Autori.

Skripta je podijeljena u blokove, gdje svaki blok predstavlja jednu logičku cjelinu i hronološki korak izvršenja koda. Ovo je urađeno sa ciljem da lakše možemo da pratimo logiku izvršenja koda korak po korak, tako imamo:

Korak 1. Uvoz modula

U prvom koraku importujemo potrebne biblioteke za rad naše skripte. Ovo je standardizovano prvi korak u Python skripti, ali sam proces importovanja biblioteke moguće je izvesti u bilo kom dijelu koda. Bitno je da modul učitamo prije samog korišćenja neke od funkcionalnosti iz traženog modula.

```
# Uvoz modula
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt
```

Gdje su:

- **yfinance** modul – Omogućava preuzimanje finansijskih podataka direktno sa interneta, posebno sa Yahoo Finance API-ja.
- **pandas** modul – Biblioteka za manipulaciju i analizu podataka, posebno pogodna za rad sa tabelarnim podacima.
- **matplotlib.pyplot** modul – Biblioteka za vizualizaciju podataka, koja omogućava kreiranje različitih tipova grafika.

Nakon izvršenja ovih linija koda, naša skripta ima sve potrebne module za obradu podataka.

Korak 2. Preuzimanje podataka

U drugom koraku definijemo dvije varijable, `symbol` i `data`, koje će imati vrijednosti potrebne za našu analizu. Varijabli `symbol` smo dodijelili vrijednost '`AAPL`', što predstavlja ticker, odnosno kod kompanije Apple na berzi. U varijablu `data`, koristeći `yfinance` modul, preuzimamo podatke za kompaniju Apple sa interneta. U nastavku koda, štampaćemo nekoliko prvih redova podataka, kao provjeru da li smo uspješno preuzeeli podatke.

```
# Preuzimanje podataka o akcijama kompanije Apple (AAPL)
symbol = 'AAPL' # Ticker za Apple
data = yf.download(symbol, start='2000-01-01', end='2024-10-31')

# Provjera prvih nekoliko redova podataka
print("Prvih pet redova podataka:")
print(data.head())
```

Nakon ovog koraka, imamo spremne podatke i možemo ih obraditi.

Korak 3. Obrada podataka

U trećem koraku skripte vršimo dodatnu obradu podataka. Inicijalno, za prikaz cijene akcije nije potrebna dodatna obrada. Te vrijednosti možemo direktno prikazati na grafiku. Međutim, u ovom primjeru imamo dvije dodatne obrade podataka:

- 1) Računanje dnevног povrata
- 2) Računanje kumulativnog povrata

Ove vrijednosti inicijalno nisu prikazane na grafiku, ali ćemo u nastavku pokazati kako da ih dodamo.

```
# Izračunavanje dnevnih povrata (returns)
data['Daily Return'] = data['Close'].pct_change()

# Izračunavanje kumulativnog povrata
data['Cumulative Return'] = (1 + data['Daily Return']).cumprod()
```

- `data['Daily Return']` – Ova komanda uzima novu kolonu koja sadrži dnevne povrate, izračunate kao procentualna promjena zatvarajuće cijene u odnosu na prethodni dan.
- `data['Cumulative Return']` – Ova komanda uzima novu kolonu koja sadrži kumulativni povrat investicije, izračunatu kumulativnim množenjem (koristeći kumulativni proizvod) dnevnih povrata.

Nakon ovog koraka, imamo spremne preračunate podatke za dnevni i kumulativni povrat investicije, pa ih možemo također prikazati na grafiku.

Korak 4. Prikaz grafika

U četvrtom koraku skripte podešavamo osnovne parametre za grafik i prosljeđujemo mu podatke potrebne za generisanje.

```
# Grafik zatvarajuće cijene
plt.figure(figsize=(14, 7))
plt.plot(data.index, data['Close'], label='Zatvarajuća cijena')
plt.title('Kretanje cijene akcija Apple-a (2000-2024)')
plt.xlabel('Datum')
plt.ylabel('Cijena (USD)')
plt.legend()
plt.grid(True)
plt.show()
```

Gdje komande imaju sljedeće značenje:

- `plt.figure(figsize=(14,7))` – Postavlja veličinu grafika. Inicijalno setovanje veličine je u inčima (1in=2.54 cm)
- `plt.plot(data.index,data['Close'],label='Zatvarajuća cijena'):` Crta linijski grafik zatvarajućih cijena kroz vrijeme.
- `plt.title(...)` – Postavljaju naslov grafika.
- `plt.xlabel(...)` – Postavljaju naslov y ose grafika.
- `plt.ylabel(...)` – Postavljaju naslov x ose grafika.
- `plt.legend()` – Uključuje prikazivanje legende grafika.
- `plt.grid(True)` – Uključuje prikazivanje mreže na grafik za bolju preglednost.
- `plt.show()` – Prikazuje grafik. Ova komanda inicijalizuje crtanje grafika.

Svaka od ovih linija koda vrši određena podešavanja grafika, a posljednja komanda vrši prikaz grafika sa postavljenim parametrima.

Izmjene u kodu

U nastavku ćemo predložiti odredene promjene u kodu, koje možete jednostavno sprovesti i pratiti rezultate tih promjena. Sve ove izmjene su vam dostupne u fajlu `ch1\Primjer1-KorekcijeKoda.ipynb`. Da bi ste pogledali ove izmjene i provjerili da li ste ispravno izmjenili svoj kod, pratite instrukcije koje smo objasnili u prethodnom naslovu “Colab okruženje za rad sa primjerima”.

Zatim ćemo uraditi tri izmjene, koje smo definisali u nastavku:

1) Promjena tikera kompanije. Možemo promijeniti ticker kompanije, odnosno skup podataka, a time i samu kompaniju kao predmet analize. Uzećemo, na primjer, kompaniju Tesla. Pokazaćemo i kako da dođete do svih kompanija, koje su dostupne u bazi.

Izmjena koda: Promijenite vrijednost varijable `symbol`:

```
symbol = 'TSLA' # Ticker za Tesla
```

Postupak pronalaženja ticker-a druge kompanije:

- Posjetite web stranicu Yahoo Finance (<https://finance.yahoo.com/>).
- U polje za pretragu unesite naziv kompanije (npr. "Tesla").
- Pronađite odgovarajući ticker (npr. "TSLA" za Tesla).

Ponovo pokrenite skriptu i posmatrajte kako se mijenja grafik i izračunati podaci za novu kompaniju.

2) Prikaz kumulativnog povrata. Ukoliko želite da prikažete grafike za izračunati kumulativni povrat investicije i dnevni povrat, možete birati između dva pristupa: dodavanje serije podataka na isti grafik ili kreiranje posebnog grafika sa novom serijom podataka. Izbor zavisi od prirode podataka i mogućnosti prikazivanja na istoj x osi. Za narednu izmjenu koda, koristićemo pristup dodavanja nove serije podataka na postojeći grafik.

Izmjena koda: Dodavanje linije koda za crtanje kumulativnog povrata.

```
plt.plot(data.index, data['Cumulative Return'], label='Kumulativni povrat')
```

U principu, ovu liniju možete dodati u bloku koji crta grafik na proizvoljno mjesto, ali prije komande `plt.show()`, koja je uvijek zadnja komanda, koja se izvršava ili koja generiše grafik. Generisani grafik je statička slika, koja nema interaktivnost i ukoliko je želite modifikovati, poslije izmjene nekog od parametara, morate ponovo pozvati komandu `plt.show()`.

```
# grafik zatvarajuće cijene i kumulativnog povrata
plt.figure(figsize=(14, 7))
plt.plot(data.index,
          data['Close'], label='Zatvarajuća cijena')
plt.plot(data.index,
          data['Cumulative Return'], label='Kumulativni povrat')
plt.title('Kretanje cijene akcija i kumulativni povrat investicije')
plt.xlabel('Datum')
plt.ylabel('Vrijednost')
plt.legend()
plt.grid(True)
plt.show()
```

3) Prikaz tri grafika. U nastavku ćemo proširiti našu početnu obradu podataka. Blok obrade podataka definiše tri vrijednosti, tako da osim kretanja cijene akcije imamo i

kumulativni povrat investicije i dnevni povrat investicije. Izmjenama koda u bloku za crtanje grafika (PLOT GRAF), prikazat ćemo sve obrađene podatke u obliku grafika.

Izmjena koda: U predloženom primjeru prikazat ćemo dva pristupa i to:

- Multipliciranje koda kopiranjem bloka za crtanje grafika i promjenom podataka u `plt.plot()` i `plt.title()` omogućava generisanje tri odvojena grafika. Ovo je jednostavna, ali manje efikasna metoda.
- Nasuprot tome, korišćenje funkcije `subplots()` omogućava prikaz više različitih grafika unutar jednog okvirnog grafika, čime se svi grafici objedinjuju u jednoj slici. Ovaj pristup je bolje organizovan i praktičan za rad sa većim brojem grafika.

Multipliciranje grafika postižemo izmjenom sljedećeg koda, dok dio predstavljen sa tri tačke (...) ostaje nepromjenjen u odnosu na početni kod.

```
# grafik 1
...
plt.plot(data.index, data['Close'], label='Zatvarajuća cijena')
plt.title('Kretanje cijene akcija ')
...
# grafik 2
...
plt.plot(data.index, data['Cumulative Return'], label='Kumulativni povrat')
plt.title('Kumulativni povrat investicije (2000-2024)')
...
# grafik 3
...
plt.plot(data.index, data['Daily Return'], label='Dnevni povrat')
plt.title('Dnevni povrat investicije (2000-2024)')
...
```

Multipleksiranje više grafika u jednu sliku.

```
# Prikaz tri grafika
fig, axs = plt.subplots(3, 1, figsize=(14, 18))

# Zatvarajuća cijena
axs[0].plot(data.index, data['Close'], label='Zatvarajuća cijena', color='blue')
axs[0].set_title('Kretanje cijene akcija (2000-2024)')
axs[0].set_xlabel('Datum')
axs[0].set_ylabel('Cijena (USD)')
axs[0].legend()
axs[0].grid(True)

# Kumulativni povrat
axs[1].plot(data.index, data['Cumulative Return'], label='Kumulativni povrat',
color='green')
axs[1].set_title('Kumulativni povrat investicije (2000-2024)')
axs[1].set_xlabel('Datum')
axs[1].set_ylabel('Kumulativni povrat')
axs[1].legend()
axs[1].grid(True)
```

```

# Dnevni povrat
axs[2].plot(data.index, data['Daily Return'], label='Dnevni povrat', color='red')
axs[2].set_title('Dnevni povrat investicije (2000-2024)')
axs[2].set_xlabel('Datum')
axs[2].set_ylabel('Dnevni povrat')
axs[2].legend()
axs[2].grid(True)
plt.tight_layout()
plt.show()

```

Kroz ovaj primjer smo prošli osnovne korake izrade skripte u Pythonu za finansijsku analizu, uključujući uvoz modula, preuzimanje i obradu podataka, te vizualizaciju rezultata. Također smo pokazali kako jednostavnim izmjenama u kodu možemo prilagoditi našu analizu različitim kompanijama ili dodati nove parametre za analizu.

Ohrabrujemo vas da nastavite sa eksperimentisanjem:

- Mijenjajte vrijednosti u kodu: Isprobajte različite kompanije promjenom vrijednosti variabile symbol.
- Dodajte nove kompanije: Analizirajte više kompanija istovremeno.
- Uključite dodatne finansijske metrike: Kao što su pokretni prosjeci, tržišni indeks, itd.

Ovo će vam pomoći da bolje razumijete kako Python može biti moćan alat u ekonomskim analizama.

Napomena: Pri izmjenama koda, pazite na ispravno pisanje naziva varijabli i funkcija, kao i na sintaksu, kako bi skripta ispravno funkcionala. Naravno, tokom vaših vježbi uvijek imate pristup do skripti na GitHub koje prate ovu knjigu. Zato se u početku vraćajte na uputstvo "Instrukcije za rad sa primjerima u Colab" kako bi savladali korišćenje GitHub-a i Colab okruženja.

Vježba:

1. Prikažite za jednom ili više grafika kretanje cijena akcija za Apple, Google i Amazon
2. Prikažite na jednom ili više grafika kretanje cijena akcija za Volkswagen, Mercedes-Benz i Toyota
3. Prikažite za jednom ili više grafika kretanje cijena akcija za Coca-Cola, Procter & Gamble i Shell

Napomena: Rješenja se nalaze na GitHub u fajlu ch1/Vjezba1.ipynb

Primjer 2. Analiza kretanja bruto domaćeg proizvoda

Bruto domaći proizvod (BDP) je jedan od ključnih ekonomskih indikatora, koji mjeri ukupnu ekonomsku aktivnost zemlje. Analiza BDP-a kroz vrijeme omogućava uvid u ekonomski rast i razvoj zemlje. U ovom primjeru, izvršena je analiza kretanje BDP-a Bosne i Hercegovine od 2000. do 2021. godine, te je izvršeno poređenje sa susjednim zemljama: Hrvatskom, Srbijom i Crnom Gorom. Pored ove analize BDP za zemlje

regionala, izvršena je i analizu ukupnog javnog duga SAD, kao i neke komparacije kretanja tog parametra sa kretanjem cijena akcija.

U metodološkom smislu, koriste se podaci Svjetske banke o BDP-u izraženom u tekućim cijenama u američkim dolarima (USD). Podaci su preuzeti korišćenjem modula (biblioteke) `pandas_datareader`, a zatim obrađeni i vizualizovani pomoću modula `matplotlib`.

U prethodnom primjeru objasnili smo osnovnu strukturu Python skripte i njene ključne blokove. Takođe, na blok dijagramu (slika 1.8.) definisali smo tok obrade podataka kroz četiri koraka. Ovaj blok dijagram se koristi i za ovaj primjer, kao i za sve ostale primjere obrađene u ovoj knjizi.

U okviru bloka *IMPORT MODUL* izvršićemo uvoz potrebnih modula za analizu i crtanje dijagrama. Za razliku od prethodnog slučaja, preuzimanje podataka ovde vršimo pomoću modula `pandas_datareader`, kojem prosleđujemo parametar `wb`. Ovaj parametar referencira World Bank website i omogućava preuzimanje podataka direktno sa te platforme.

```
import pandas as pd
from pandas_datareader import wb # referenciranje na WorldBank podatke
import matplotlib.pyplot as plt
```

Blok *Download DATA* zadužen je za preuzimanje podataka sa zvanične web platforme Svjetske banke (*World Bank*). Umesto direktnog navođenja URL adrese (<https://www.worldbank.org/>), koristi se ugradena klasa `wb`, koja omogućava automatski pristup i preuzimanje podataka direktno iz baze Svjetske banke. Ova baza podataka pruža javno dostupne informacije, koje se mogu koristiti bez potrebe za registracijom ili specijalnim ovlašćenjima. Ovakav pristup omogućava bržu i jednostavniju analizu podataka, čineći ga pogodnim za akademske i istraživačke potrebe.

```
# Definisanje zemalja i indikatora
countries = ['BA'] # Bosna i Hercegovina
indicator = 'NY.GDP.MKTP.CD' # BDP (trenutne cijene u USD)

# Preuzimanje podataka sa Svjetske banke
data = wb.download(indicator=indicator, country=countries, start=2000, end=2021)
```

Blok *DATA PROCESSING* u ovom primjeru vrši reindeksiranje podataka, kako bi izdvojio kolone `country` i `year`, uz istovremeno sortiranje podataka po koloni `year`. Pored toga, unutar ovog bloka vrši se i rotiranje tabele (*pivot table*) kako bi podaci bili bolje strukturirani i pripremljeni za lakšu i efikasniju vizualizaciju. Ovakav pristup omogućava jasnije prikazivanje trendova i poređenja između različitih kategorija ili vremenskih perioda.

```

# Resetovanje indeksa da bi dobili kolone 'country' i 'year'
data = data.reset_index()
# Sortiranje podataka po godini u rastućem redoslijedu
data = data.sort_values('year')
# Pivotiranje podataka za lakšu vizualizaciju
data_pivot = data.pivot(index='year', columns='country', values=indicator)

```

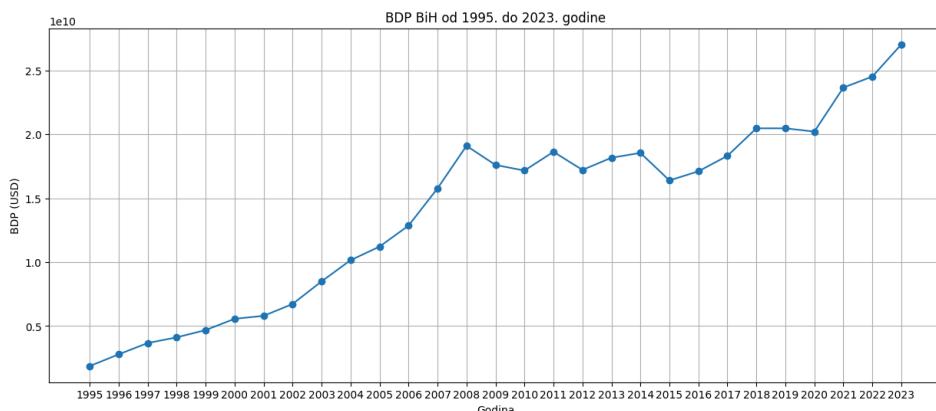
Blok *PLOT GRAF* služi za vizualizaciju podataka, koji su prethodno preuzeti i pripremljeni. Unutar bloka komandom `plt.figure()` definije se veličina grafika, a komandom `plt.title()` naslov koji će biti prikazan na vrhu grafika. Pored toga, postavljaju se nazivi za x i y osu, radi jasnijeg predstavljanja podataka, a uključuje se i prikaz grid linija, koje olakšavaju praćenje i interpretaciju vrijednosti na grafiku. Na kraju, komandom `show` grafik se prikazuje u radnom okruženju, omogućavajući korisniku da sagleda rezultate analize u grafičkom formatu.

Konačno, grafik koji prikazuje kretanja BDP u Bosni i Hercegovini prikazan je na sljedećoj slici 1.9.

```

plt.figure(figsize=(15,6))
plt.plot(data['year'], data[indicator], marker='o', linestyle='-' )
plt.title('BDP BiH od 1995. do 2023. godine')
plt.xlabel('Godina')
plt.ylabel('BDP (USD)')
plt.grid(True)
plt.show()

```



Slika 1.9. Dijagram kretanja BDP za BiH (1995-2023), Izvor: Autori.

Napomena: Ovaj primjer, kao i varijacije koje ćemo pokazati u nastavku se nalazi takođe na GitHub zajedno sa svim primjerima. Možete ih koristiti kroz okruženje Google Colab, koji smo ranije opisali. Naziv fajla koji sadrži kod za analizu BDP Bosne i Hercegovine je [ch1/Primjer2.ipynb](#).

U nastavku ćemo pokazati neka od izmjena na ovom primjeru, kroz koje možemo da proširimo našu analizu.

Proširena analiza BDP-a - Poređenje sa okruženjem

Posmatramo kretanje BDP-a Bosne i Hercegovine u periodu od 2000. do 2021. godine i poredimo trendove sa susjednim zemljama: Srbijom, Hrvatskom i Crnom Gorom. Ova analiza pruža širu perspektivu regionalnog ekonomskog razvoja i omogućava bolje razumijevanje ekonomskih trendova u okruženju.

1) Proširenje skupa podataka (*Download DATA*)

Početna skripta će biti proširena kako bi obuhvatili podatke za susjedne zemlje. U bloku *Download DATA*, lista zemalja će biti definisana na sljedeći način:

```
countries = ['BA', 'HR', 'RS', 'ME'] # BiH, Hrvatska, Srbija, Crna Gora
```

Ova lista obuhvata ISO kodove za tražene zemlje i omogućava jednostavno preuzimanje podataka za svaku od njih (<https://www.iban.com/country-codes>).

2) Obrada podataka (*DATA PROCESSING*)

U bloku *DATA PROCESSING* nema izmjena, jer će prethodno definisane operacije reindeksiranja, sortiranja i “pivotiranja” table automatski obraditi prošireni skup podataka.

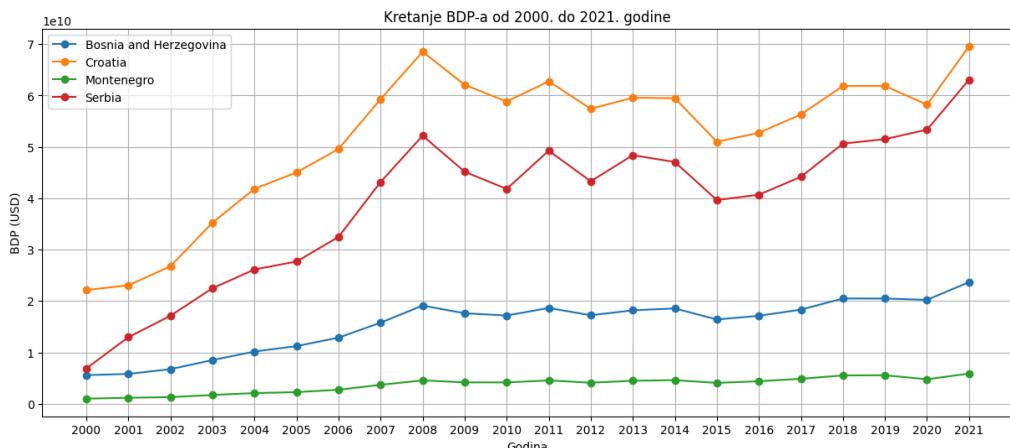
3) Vizualizacija podataka (*PLOT GRAF*)

U bloku *PLOT GRAF* modifikovaćemo dio koda za dodavanje podataka na grafik. Korišćenjem petlje, podaci za svaku zemlju će biti prikazani na grafiku:

```
for country in data_pivot.columns:
    plt.plot(data_pivot.index, data_pivot[country], marker='o', linestyle='-' ,
label=country)
plt.legend()
```

Dodavanje legende (`plt.legend()`) je korigovano, kako bi se jasno razlikovali podaci za svaku zemlju. Ovdje su izostavljeni parametri i sam modul će da rasporedi boje i sadržaje labela.

Ovaj pristup omogućava jednostavnu nadogradnju analize, jer je dovoljno proširiti listu zemalja ukoliko želimo uključiti i druge države. Vizualizacija sa više linijskih grafika na istom prikazu omogućava lako poređenje i prepoznavanje ekonomskih trendova. Konačan izgled grafika je prikazan na sljedećoj slici.



Slika 1.10. Poređenje kretanja BDP za zemlje regiona, Izvor: Autori.

Primjer je pokazao kako možemo koristiti Python za prikupljanje i analizu ekonomskih podataka iz pouzdanih izvora kao što je Svjetska banka. Upoređivanje BDP-a Bosne i Hercegovine sa susjednim zemljama pruža potpuniji uvid u regionalne ekonomske trendove, što može poslužiti za druge analize i predviđanja ekonomskih kretanja.

Vježba:

1. Uporedite kretanje BDP-a Bosne i Hercegovine sa Njemačkom
2. Uporedite kretanje BDP-a Kine i SAD

Napomena: Rješenja se nalaze na GitHub u fajlu ch1\Vjezba2.ipynb

Primjer 3. Analiza kretanja javnog duga USA-a

Kretanje javnog duga Sjedinjenih Američkih Država (SAD) u periodu od 2000. do 2024. godine predstavlja značajan globalni parametar za analizu ekonomskih tokova. Javna dugovanja SAD-a često su polazna tačka za mnoge ekonomske analize, jer pružaju uvid u makroekonomske trendove i omogućavaju procjenu fiskalne održivosti jedne od najvažnijih svjetskih ekonomija. Cilj ovog primjera je da demonstriramo metode za prečenja dugoročnih trendova javnog duga SAD-a, čime se stiče dublje razumijevanje fiskalnih izazova i prilika, što je ključno za donošenje ekonomskih odluka.

U rješavanju ovog primjera mijenjamo pristup i uključujemo vještačku inteligenciju (AI). Uzakujemo na dva ključna aspekta prilikom rješavanja problema upotrebom AI:

- 1) **Razumijevanje problema:** Precizno definisanjem problema kroz opis koji ćemo dati AI alatu, osiguravamo tačnost odgovora. Ukoliko ne razumijem sam problem i skup mogućih rješenja, odgovori koje dobijamo od AI, samo će dodatno komplikovati rješenje i vrlo često nam dati pogrešne odgovore.

- 2) **Razumijevanje odgovora:** Poznavanje strukture dobijenog koda i rezultata koje AI generiše, kao i postupka rješavanja problema i interpretaciju podataka, pomoći će nam da dođemo do konačnog rješenja.

Upotreba AI omogućava brže i efikasnije rješavanje zadatka, ali takođe pomaže u razvijanju kritičkog mišljenja prilikom korišćenja modernih tehnologija. Ova knjiga ima upravo taj cilj, da vam pomogne u razumijevanju odgovora, olakša postavljanje problema i formira kritičko mišljenje o korišćenju ili nekorišćenju AI i novih tehnologija u radu.

Napomena: Ovaj primjer se nalazi na GitHub zajedno sa svim primjerima. Kod koji je generisao AI nalazi se u fajlu [Primjer3.ipynb](#)

Da bi testirali postavku problema i odgovore generisane od strane AI, potrebno je imati nalog za interakciju sa AI platformom. Konkretno, ovaj primjer je testiran na ChatGPT proizvodu kompanije OpenAI. Ukoliko nemate pristup ovoj platformi, možete isprobati slične primjere na alternativnim platformama kao što su Google Gemini ili druge slične usluge. Većina ovih platformi nudi besplatne modove uz određena ograničenja.

Odgovori u ovoj knjizi generisani su korišćenjem modela ChatGPT-4o, koji je komercijalni servis. Ukoliko koristite druge modele može doći do manjih odstupanja u strukturi i izgledu koda, ali ne i u funkcionalnosti.

U nastavku se nalazi tekst instrukcija, koje su korišćene za AI kako bi generisao Python skriptu za rješavanje ovog problema.

ChatGPT

Potrebna mi je Python skriptu koja analizira kretanje javnog duga SAD-a od 2000. do 2023. godine. Podatke treba preuzeti sa FRED baze podataka koristeći modul pandas_datareader, i ekonomski indikator koji opisuje te podatke. Skripta treba da:

1. Importuje potrebne module za rad skripte
2. Definiše vremenski opseg analize (od 2000. do 2023. godine).
3. Preuzme podatke o javnom dugu SAD-a.
4. Prilagodi vremenske format radi lakše obrade.
5. Agregira podatke na godišnjem nivou, uzimajući posljednju vrijednost za svaku godinu.
6. Kreira linijski grafik koji prikazuje kretanje javnog duga kroz posmatrani period, sa:
 - Osa X koja prikazuje godine.
 - Osa Y koja prikazuje vrijednost javnog duga u hiljada milijardi USD.
7. Doda naslov i mrežu na grafik za bolju preglednost.

Odgovor koji smo dobili na postavljeno pitanje od ChatGPT AI modela 4o je:

ChatGPT: Evo Python skripte koja analizira kretanje javnog duga SAD-a u periodu od 2000. do 2023. godine, koristeći ekonomski indikator [GFDEBTN](#) sa FRED baze podataka.

```

import pandas as pd
import matplotlib.pyplot as plt
from pandas_datareader import data as web

# Definisanje vremenskog opsega analize
start_date = '2000-01-01'
end_date = '2023-12-31'

# Preuzimanje podataka o javnom dugu SAD-a sa FRED baze
debt = web.DataReader('GFDEBTN', 'fred', start_date, end_date)

# Konvertovanje vremenskih oznaka u datetime format radi lakše obrade
debt.index = pd.to_datetime(debt.index)

# Agregacija podataka na godišnjem nivou - posljednja vrijednost svake godine
annual_debt = debt.resample('YE').last()

# Kreiranje grafika
plt.figure(figsize=(12, 6))
plt.plot(annual_debt.index.year, annual_debt['GFDEBTN'] / 1e12, marker='o',
linestyle='-', color='b', label='Javni dug SAD')

# Dodavanje naslova, osi i mreže
plt.title('Kretanje javnog duga SAD-a (2000-2023)', fontsize=14)
plt.xlabel('Godina', fontsize=12)
plt.ylabel('Dug (u bilionima USD)', fontsize=12)
plt.grid(True, linestyle='--', alpha=0.6)
plt.legend()
plt.tight_layout()
# Prikaz grafika
plt.show()

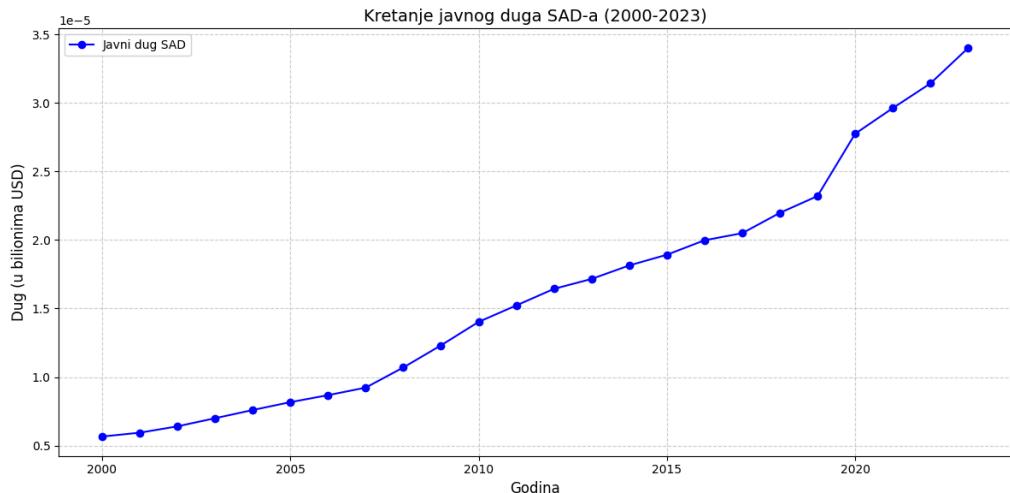
```

Objašnjenje skripte:

- Importovanje modula: Skripta koristi `pandas` za obradu podataka, `matplotlib` za vizualizaciju, i `pandas_datareader` za preuzimanje podataka sa FRED baze.
- Preuzimanje podataka: Podaci o javnom dugu SAD-a preuzimaju se koristeći indikator `GFDEBTN` (Total Public Debt Outstanding).
- Obrada podataka: Vremenske oznake se konvertuje u datetime format radi lakše obrade. Podaci se agregiraju na godišnjem nivou, uzimajući posljednju vrijednost svake godine.
- Vizualizacija: Linijski grafik prikazuje kretanje javnog duga od 2000. do 2023. godine. Osa X prikazuje godine, dok osa Y prikazuje dug u bilionima USD. Mreža i legenda poboljšavaju preglednost grafika.

Ovaj kod pruža jasnu vizualizaciju kretanja javnog duga i može se lako proširiti za dodatne analize.

Nakon pokretanja skripte u našem Google Colab okruženju, generiše se sljedeći grafik, koji prikazuje kretanje javnog duga SAD u traženom periodu.



Slika 1.11. Kretanje javnog duga SAD. Izvor: Autori.

Ukoliko se želi dodatno analizirati kretanje javnog duga SAD-a i uporediti ga sa cijenama akcija kompanija Apple i Google, moraju se kombinovati podaci iz različitih izvora i prilagoditi skale na y-osi za odgovarajući prikaz. Iako je ovo tehnički izvodljivo uz adekvatne izmjene u kodu, trenutno nemamo dovoljno znanja za implementaciju takvih modifikacija. Ono što trenutno znamo je kako budući grafik treba da izgleda, pa ćemo na osnovu tih znanja modifikovati upita za AI kako bi dobili izmjene u kodu.

ChatGPT

Potrebna mi je Python skripta koja analizira kretanje cijena akcija kompanija Apple i Google, zajedno sa kretanjem javnog duga Sjedinjenih Američkih Država (SAD) u periodu od 2000. do 2023. godine. Skripta treba da:

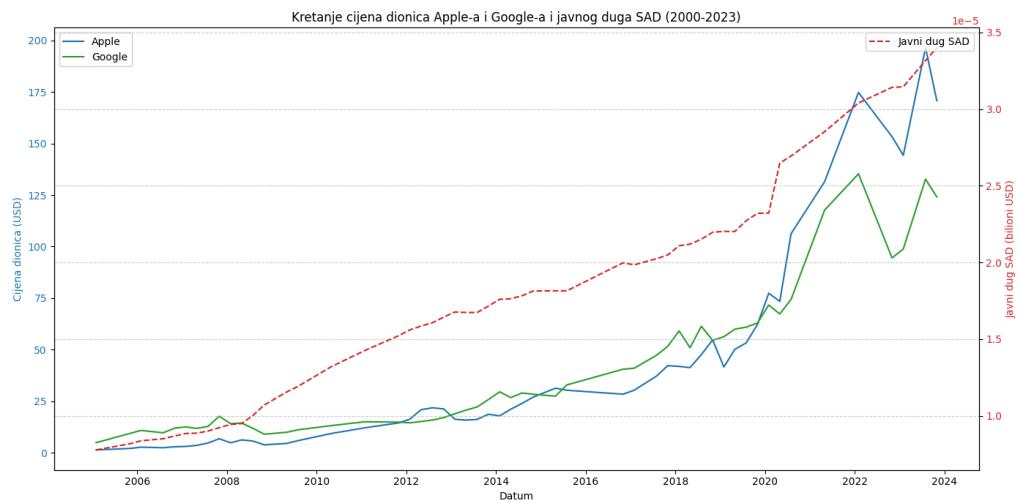
1. Importuje potrebne module za preuzimanje, obradu i prikaz podataka.
2. Definiše vremenski opseg analize od 2000. do 2023. godine.
3. Preuzme podatke o cijenama akcija Apple-a i Google-a koristeći Yahoo Finance API yfinance.
4. Preuzme podatke o javnom dugu SAD-a sa FRED baze podataka koristeći odgovarajući indikator i pandas_datareader.
5. Prilagodi podatke:
 - Preuzeti podatke sa download i WebReader
 - Konvertuje vremenske oznake za javni dug u odgovarajući format.
 - Agregira podatke o javnom dugu na mjesecni nivo, uzimajući posljednju dostupnu vrijednost svakog mjeseca.
6. Spoji podatke o akcijama i javnom dugu u jedan skup podataka.
7. Kreira uporedni grafik sa sljedećim karakteristikama:
 - Na lijevoj y-osi prikazuje kretanje cijena akcija Apple-a i Google-a u USD.
 - Na desnoj y-osi prikazuje kretanje javnog duga SAD-a u bilionima USD.
 - Dodaje legendu za svaku os.
8. Doda naslov i mrežu na grafik za jasniji pregled parametara.

ChatGPT: Evo Python skripte koja ispunjava sve navedene zahtjeve:

```
import pandas as pd
import matplotlib.pyplot as plt
import yfinance as yf
from pandas_datareader import data as web

# Definisanje vremenskog opsega analize
start_date = '2000-01-01'
end_date = '2023-12-31'
# Definisanje simbola kompanija
symbols = ['AAPL', 'GOOGL']
# Preuzimanje podataka o cijenama akcija koristeći Yahoo Finance API
stock_data = yf.download(symbols, start=start_date, end=end_date)[['Close']]
# Uklanjanje vremenske zone iz podataka
stock_data.index = stock_data.index.tz_localize(None)
# Preuzimanje podataka o javnom dugu SAD-a sa FRED baze podataka
debt_data = web.DataReader('GFDEBTN', 'fred', start_date, end_date)
# Konvertovanje vremenskih oznaka za javni dug u datetime format
debt_data.index = pd.to_datetime(debt_data.index)
# Agregacija podataka o javnom dugu na mjesecni nivo (uzimajući posljednju dostupnu vrijednost u mjesecu)
debt_monthly = debt_data.resample('ME').last()
# Spajanje podataka o cijenama akcija i javnom dugu u jedan DataFrame
data_combined = pd.concat([stock_data, debt_monthly], axis=1).dropna()
# Kreiranje uporednog grafika
fig, ax1 = plt.subplots(figsize=(14, 7))
# Prva y-osa (lijeva): cijene akcija
ax1.set_xlabel('Datum')
ax1.set_ylabel('Cijena akcija (USD)', color='tab:blue')
ax1.plot(data_combined.index, data_combined['AAPL'], label='Apple',
          color='tab:blue')
ax1.plot(data_combined.index, data_combined['GOOGL'], label='Google',
          color='tab:green')
ax1.tick_params(axis='y', labelcolor='tab:blue')
ax1.legend(loc='upper left')
# Druga y-osa (desna): javni dug
ax2 = ax1.twiny() # Dvostruka y-osa
ax2.set_ylabel('Javni dug SAD (bilioni USD)', color='tab:red')
ax2.plot(data_combined.index, data_combined['GFDEBTN'] / 1e12, label='Javni dug SAD',
          color='tab:red', linestyle='--')
ax2.tick_params(axis='y', labelcolor='tab:red')
ax2.legend(loc='upper right')
# Dodavanje naslova i mreže
plt.title('Kretanje cijena akcija Apple-a i Google-a i javnog duga SAD (2000-2023)')
plt.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()
# Prikaz grafika
plt.show()
```

Nakon pokretanja skripte u Google Colab okruženju dobijamo grafik, koji prikazuje tražene parametre u posmatranom vremenskom intervalu.



Slika 1.12. Poređenje kretanja cijena akcija i javnog duga SAD, Izvor: Autori.

2. UVOD U PYTHON



Our vision was that every student on campus should have access to a computer, and any faculty member should be able to use a computer in the classroom whenever appropriate. It was as simple as that. (Authors of the BASIC programming language)³

Naša vizija je bila da svaki student na kampusu ima pristup računaru i da svaki profesor može koristiti računar u učionici kad god je to pogodno. Bilo je tako jednostavno. (Autori BASIC programskog jezika)

— John G. Kemeny, Thomas E. Kurtz

³ TIME, By Harry McCracken April 29, 2014 Izvor: <https://time.com/69316/basic/>

Ciljevi ovog poglavlja

U ovom poglavlju cilj nam je da:

- Predstavimo osnovne karakteristike programskog jezika Python, uključujući njegov nastanak, filozofiju dizajna i razloge zašto je idealan alat za studente ekonomije.
- Upoznavanje sa Jupyter Notebook-om kao interaktivnim okruženjem, koje omogućava jednostavno pisanje, izvršavanje i dokumentovanje Python koda, što je posebno korisno za analizu podataka i ekomska istraživanja.
- Opis procesa instalacije i podešavanja radnog okruženja, uključujući instalaciju Pythona i Jupyter Notebook-a na Windows i Linux operativnim sistemima, kreiranje virtualnih okruženja i instaliranje potrebnih paketa.
- Objasnjenje osnove rada u Jupyter Notebook-u, uključujući kako se koriste celije za kod i Markdown, osnovne komande i prečice, te savjete za efikasan rad i organizaciju vaših projekata.
- Opis kako da se naprave prvi koraci sa Python-om u Jupyter-u, kroz pisanje jednostavnih izraza, rad sa komentarima, upoznavanje sa uobičajenim greškama i njihovo ispravljanje, kao i praktične vježbe relevantne za ekonomiju.
- Predstavljanje i pružanje dodatnih resursa i smjernica za dalje učenje, uključujući online materijale, zajednice za podršku i pripremu za naredna poglavlja, koja će produbiti vaše znanje i vještine u programiranju i analizi podataka.

Ovim ciljevima želimo da vas ospesobimo za samostalno korištenje Python-a i Jupyter Notebook-a, što će vam biti dragocjeno u daljem studiranju, istraživanju i profesionalnom radu u oblasti ekonomije.



2.1. Šta je Python?

Python je moderan, višenamjenski programski jezik visokog nivoa, poznat po svojoj jednostavnosti i čitljivosti. Dizajniran je tako da bude lako razumljiv kako početnicima, tako i iskusnim programerima, čime postaje idealan alat za učenje programiranja i primjenu u širokom spektru oblasti, uključujući ekonomiju. Njegova najveća prednost leži u ogromnoj zajednici korisnika i bogatoj kolekciji modula, što ga čini primjenjivim u gotovo svim poslovnim sektorima. Python se koristi za sve, od programiranja IoT (Internet of Things) uređaja do treniranja modela za vještačku inteligenciju (AI).

2.1.1. Nastanak i razvojni put

Python je razvio Guido van Rossum početkom 1990-ih godina u Holandiji. Prvobitno je nastao kao nasljednik programskog jezika ABC, s ciljem da bude jednostavan za korištenje, a istovremeno dovoljno moćan za rješavanje složenih zadataka. Naziv "Python" inspirisan je britanskom komičnom grupom "Monty Python", čime je autor želio da jezik učini zabavnim i prijatnjim za korištenje.

Prva verzija Pythona (Python 0.9.0) objavljena je u februaru 1991. godine i uključivala je osnovne funkcionalnosti poput rukovanja izuzecima, definisanja funkcija i osnovnih tipova podataka, po kojima je Python danas prepoznatljiv.

Python 2.0, objavljen 16. oktobra 2000. godine, uveo je mnoge značajne funkcije, uključujući bolje upravljanje memorijom i podršku za Unicode. Ove inovacije omogućile su efikasniji rad i prilagodile Python potrebama savremenih aplikacija. Još jedna ključna promjena bila je uvođenje transparentnijeg procesa razvoja, podržanog od strane zajednice, čime se obezbijedila njegova dugoročna relevantnost.

Python 3.0, objavljen 3. decembra 2008. godine, predstavljao je veliko unapređenje jezika, sa dodatnim funkcijama i poboljšanjima. Iako nije bio unazad kompatibilan sa prethodnim verzijama, mnoge nove funkcije bile su prilagođene i uključene u verzije 2.6 i 2.7, koje su služile kao prelazna rješenja.

Python se koristi u razvoju aplikacija u organizacijama poput **Facebooka**, **Twittera**, **CERN-a**, i **NASA-e**, kao i u finansijskom sektoru, za analizu podataka i modeliranje. Njegova fleksibilnost omogućava programerima i analitičarima da brzo razvijaju rješenja za složene probleme, što ga čini ključnim alatom za istraživanja i inovacije.

Tokom godina, Python je doživio značajan razvoj i postao jedan od najpopularnijih programskih jezika na svijetu. Njegova otvorena i aktivna zajednica kontinuirano doprinosi unapređenju jezika, uvodeći nove funkcionalnosti i biblioteke, koje dodatno proširuju njegove mogućnosti i oblasti primjene.

2.1.2. Filozofija dizajna Zen Python

Python-ova filozofija dizajna naglašava čitljivost koda i jednostavnost, što je najbolje izraženo kroz „Zen Python⁴“. Ova zbirka aforizama, koju je napisao Tim Peters, jedan od vodećih saradnika Pythona, sadrži principe koji usmeravaju dizajn jezika. Neka od najpoznatijih načela uključuju: „Lijepo je bolje nego ružno“, „Jednostavno je bolje od složenog“ i „Čitljivost je važna“. Ovi principi su bili ključni u razvoju Pythona, obezbjeđujući da jezik ostane intuitivan i jednostavan za održavanje.

Jedna od najizraženijih karakteristika Pythona je njegova upotreba značajnog razmaka. Za razliku od mnogih drugih jezika, Python koristi uvlačenje za definisanje blokova koda, čime se eliminiše potreba za vitičastim zagradama. Ova karakteristika, iako u početku kontroverzna, pripisuje se podsticanju programera da pišu čistiji i čitljiviji kod.

Ovako definisana filozofija čini Python u isto vrijeme pristupačnim za početnike i efikasnim za iskusne programere.

2.1.3. Evolucija u programiranju

Python je prešao dug put od skromnog skriptnog jezika do jednog od najvažnijih alata u savremenom programiranju. Njegova evolucija obilježena je kontinuiranim razvojem biblioteka i okvira, koji zadovoljavaju širok spektar potreba. Od razvoja web aplikacija koristeći okvire poput Django i Flask, do naučnih proračuna sa bibliotekama kao što su NumPy i SciPy, pa sve do mašinskog učenja uz alate poput TensorFlow i PyTorch, Python se pokazao kao univerzalni alat za različite primjene.

Jedan od ključnih razloga za rast njegove popularnosti leži u zajednici, koja upravlja njegovim razvojem. Proces *Python Enhancement Proposal* (PEP)⁵ omogućava članovima zajednice da predlažu, diskutuju i glasaju o promjenama jezika. Ovaj demokratski pristup ne samo da osigurava njegovu tehnološku relevantnost, već podstiče inkluzivnost i život Python zajednice, što doprinosi njegovom globalnom usvajanju.

Pythonov uticaj u programiranju je ogroman, posebno zbog svoje jednostavnosti i pristupačnosti. Kao prvi programske jezik, snizio je barijeru za ulazak u svijet programiranja, čineći ga idealnim za početnike. U akademskim krugovima, Python je zamjenio jezike poput Java i C++ zbog svoje čitljivosti i lakoće učenja, što ga čini omiljenim izborom za nastavu programiranja.

U profesionalnom okruženju, njegova fleksibilnost i široka paleta biblioteka učinile su ga nezaobilaznim alatom u oblastima poput nauke o podacima, vještacke inteligencije i

⁴ PEP 20 – The Zen of Python, Izvor: <https://peps.python.org/pep-0020/>

⁵ PEP 0 – Index of Python Enhancement Proposals (PEPs), Izvor: <https://peps.python.org/>

mašinskog učenja, podstičući inovacije i istraživanja. Njegova primjena se proteže kroz različite domene, od razvoja web aplikacija do najsloženijih proračuna, što ga čini ključnim alatom za programere i istraživače širom svijeta. Python-ov rast dodatno potvrđuju njegova konstantna visoko rangirana pozicija među najpopularnijim programskim jezicima.

Za ekonomiste, Python predstavlja moćan alat za analizu podataka, modeliranje i automatizaciju zadataka. Njegova jednostavnost i bogata zajednica pružaju mogućnosti za brzo usvajanje i primenu, čineći ga nezaobilaznim u savremenim ekonomskim istraživanjima i poslovanju.

2.1.4. Popularnost Pythona od škola do industrije

O popularnosti Python jezika govori i istraživanje iz 2015. godine u osnovnim školama u Velikoj Britaniji.⁶ Statistike pokazuju da 60% roditelja više bi voljelo da njihova djeca uče Python nego francuski. Ovo govori o globalnoj svijesti o značaju programiranja i činjenici da se Python pozicionira kao ključan alat za budućnost.

Pored toga, 75% djece u osnovnim školama izjavilo je da bi radije naučilo kako da upravlja robotom nego da pohađa časove francuskog jezika. Ovaj trend ukazuje na povećano interesovanje za STEM oblasti (nauka, tehnologija, inženjering i matematika), u kojima Python igra centralnu ulogu zbog svoje pristupačnosti i široke primjene.

Python je stekao značajnu popularnost u različitim industrijama zahvaljujući svojoj jednostavnosti, fleksibilnosti i širokom spektru biblioteka. U oblasti vještacke inteligencije, Python je postao glavni alat za razvoj projekata mašinskog učenja i dubokog učenja. Njegova fleksibilnost omogućava programerima da biraju između objektno orijentisanog ili skriptovanog pristupa, smanjujući mogućnost grešaka i omogućavajući kombinovanje sa drugim programskim jezicima.

U avio industriji, Python se koristi za modeliranje i simulaciju različitih komponenti letjelica. Na primjer, biblioteka "pyaviation" pruža skup formula i principa za avio industriju, omogućavajući korisnicima da kreiraju modele aviona i vrše osnovne proračune brzine i drugih parametara. Takođe, Python se koristi u mehaničkom i aeronautečkom inženjeringu za analizu podataka, simulacije i razvoj softverskih alata. Knjiga "Python for Mechanical and Aerospace Engineering"⁷ pruža primjere korišćenja Pythona u ovim oblastima, uključujući grafički prikaz podataka, modeliranje satelitskih orbita i kreiranje korisničkih interfejsa.

⁶ <https://www.ocadogroup.com/media/news/python-overtakes-french-most-popular-language-taught-primary-schools>

⁷ <https://github.com/alexkenan/pymae>

2.2. Šta je Jupyter Notebook

Jupyter Notebook je interaktivno radno okruženje, koje omogućava korisnicima da kombinuju kôd, tekst, vizualizacije i druge medijske sadržaje u jednom dokumentu koji se naziva **sveska** (*notebook*). Ovo ga čini moćnim alatom za učenje, istraživanje i prezentaciju, posebno u oblastima koje zahtijevaju analizu podataka i matematičke proračune, poput ekonomije.

2.2.1. Koncept interaktivnih bilježnica

Jupyter Notebook predstavlja evoluciju koncepta interaktivnog programiranja, gdje korisnici mogu pisati i izvršavati kod u malim dijelovima nazvanim **ćelije**. Svaka ćelija može sadržavati kod, tekst ili multimedijalni sadržaj, omogućavajući tako kombinaciju programskog koda sa objašnjenjima, komentarima i rezultatima izvršavanja.

Pristup kroz interaktivne bilježnice omogućava korisniku:

- Pisanje i izvršavanje koda u realnom vremenu. Promjene u kodu mogu se odmah testirati i rezultati se odmah vide.
- Promjenu redoslijeda izvršenja koda. Ćelije su poredane hronološki, ali izvršenje koda u ćelijama može da se odvija na proizvoljan način.
- Dokumentovanje procesa programiranja. Uz kod se mogu dodati opisi, matematičke formule i slike, koje objašnjavaju šta kôd radi.
- Vizualizacija prikaza podataka. Grafici i druge vizualizacije mogu se direktno prikazati unutar bilježnice.

Kako Jupyter povezuje kod, rezultate i dokumentaciju?

Odgovor na ovo pitanje se pronalazi u strukturi Jupyter Notebook, koji čuvanje i interpretiranje koda bazira na ćelijama. U osnovi ima dva tipa ćelija koje koristi i to:

- Ćelije za kod. U njih se unosi programski kod (npr. u Pythonu), koji se može izvršiti pojedinačno. Rezultati izvršavanja, poput vrijednosti ili grafika, prikazuju se odmah ispod ćelije.
- Ćelije za tekst. Koristeći **Markdown**⁸ sintaksu, omogućavaju formatiranje teksta, dodavanje naslova, listi, slika i matematičkih izraza koristeći LaTeX sintaksu.

Ova kombinacija omogućava da se kod, njegovi rezultati i objašnjenja nalaze zajedno, što olakšava razumijevanje i dijeljenje rada sa drugima.

⁸ Markdown je jednostavan jezik za formatiranje teksta koji omogućava jednostavno pisanje i formatiranje dokumenta koristeći običan tekst. Spada u grupu "markup language" jezika kao i XML, HTML i sl. Razvijen od strane Johna Grubera i Aarona Swartza 2004. godine

2.2.2. Prednosti korištenja Jupyter Notebook-a

Jedna od ključnih prednosti korištenja Jupyter Notebook-a je njegova interaktivnost i mogućnost vizualizacije podataka. Ovaj alat omogućava korisnicima da eksperimentišu u realnom vremenu, što znači da mogu brzo mijenjati i izvršavati kôd, isprobavajući različite pristupe i odmah vidjeti rezultate. Ova interaktivnost podstiče dublje razumijevanje problema i omogućava efikasnije pronalaženje rješenja. Studenti i istraživači mogu testirati hipoteze, prilagođavati parametre i neposredno posmatrati kako te promjene utiću na ishod, što je posebno korisno u dinamičnim oblastima kao što je ekonomija.

Pored toga, Jupyter podržava vizualizaciju podataka direktno unutar radne sveske, odnosno notebook fajla. Korisnici mogu generisati grafike, tabele i interaktivne vizualizacije, koje olakšavaju interpretaciju kompleksnih setova podataka (datasets). Ovo je ključno za analizu ekonomskih podataka, gdje vizuelni prikazi mogu pomoći u otkrivanju trendova, obrazaca i anomalija, koje nisu odmah vidljive iz sirovih podataka.

Druga značajna prednost je lakoća dijeljenja i tumačenja analiza. Jupyter radna sveska i podaci u njoj se mogu jednostavno dijeliti sa kolegama ili profesorima, bilo kao datoteke ili putem servisa kao što su na primjer GitHub. Ovo olakšava saradnju na projektima, razmjenu ideja i zajedničko rješavanje problema. Budući da bilježnice sadrže sav kôd i podatke potrebne za analizu, drugi korisnici mogu lako reprodukovati rezultate i razumjeti proces, koji je doveo do njih. Ovo je princip koji koristimo i u ovoj knjizi, gdje su svi primjerni dostupni čitaocu da ih preuzme i testira.

Kombinacija interaktivnosti, vizualizacije i mogućnosti dijeljenja čini Jupyter Notebook moćnim alatom, koji podržava efikasno učenje, istraživanje i komunikaciju u oblasti ekonomije. Korištenjem Jupyter-a, korisnici mogu poboljšati svoju produktivnost, bolje razumjeti složene podatke i efikasnije prenositi svoje ideje i rezultate drugima.

2.2.3. Praktične primjene u učenju i istraživanju

Dokumentovanje istraživačkih procesa

Jupyter Notebook omogućava studentima i istraživačima da efikasno dokumentuju svoje istraživačke procese. Kroz ovu interaktivnu platformu, korisnici mogu sistematski pratiti tok svog rada i istraživanja. Evidentiranjem svakog koraka, korištene metoda i postupka, lakše je razumjeti tok istraživanja i kasnije se vratiti na prethodne korake, ako je potrebno. Ova transparentnost pomaže ne samo u vlastitom razumijevanju procesa, već i u omogućavanju drugima da prate i reprodukuju istraživanje.

Dodavanje komentara putem tekstualnih celija ili formatiranje formula u LaTEX jeziku, pruža prostor za detaljno objašnjenje korištenih metoda, teorijske osnove i interpretaciju rezultata. Na taj način radnu svesku pretvaramo u dokument, koji kombinuje kôd sa narativom, čineći ga lakšim za čitanje i razumijevanje. Ovo je posebno korisno kada se rade složene analize, koje zahtijevaju dodatna pojašnjenja ili kada se rezultati prezentuju publici, koja možda nije upoznata sa svim tehničkim detaljima.

Integracijom referenci i dodatnih materijala, moguće je direktno u radnu svesku uključiti linkove do relevantnih izvora, literature i dodatnih resursa koji su korišteni tokom istraživanja. Ovo obogaćuje sadržaj i pruža čitaocima mogućnost da se dublje upoznaju sa temom ili da provjere izvore informacija. Takva integracija čini radnu svesku ne samo alatom za analizu, već i centralnim skladištem svih relevantnih informacija vezanih za projekat. Na bazi ovoga razvijen je i projekat Jupyter{Book} koji predstavlja novi pogled na interpretaciju sadžaja knjiga u elektronskom formatu.⁹

Prezentacija rezultata kolegama i profesorima

Jupyter Notebook je izuzetno koristan alat za prezentaciju rezultata istraživanja kolegama i profesorima. Kombinacija koda, rezultata i objašnjenja u jedinstvenom dokumentu omogućava jasnu i razumljivu prezentaciju rezultata. Umjesto da se kôd i rezultati prezentuju odvojeno, sve je objedinjeno, što olakšava praćenje logičkog slijeda i razumijevanje kako su dobijeni određeni zaključci. Ovo je posebno korisno u akademskim okruženjima gdje je transparentnost i detaljno objašnjenje metodologije od ključnog značaja.

Pored toga, Jupyter Notebook omogućava interaktivne prezentacije. Moguće je koristiti radnu svesku za izvođenje kôda uživo tokom seminara ili predavanja, što omogućava dinamično demonstriranje, kako promjene u kôdu ili podacima utiču na rezultate.

Ovim pristupom, Jupyter Notebook se ističe kao svestran alat, koji ne samo da olakšava analizu i obradu podataka, već i poboljšava komunikaciju i saradnju među studentima i istraživačima. Korištenjem ovog alata, proces učenja i istraživanja postaje interaktivniji, transparentniji i efikasniji, što doprinosi kvalitetu obrazovanja i naučnog rada.

⁹ Jupyter{Book}. Build beautiful, publication-quality books and documents from computational content.

Izvor: <https://jupyterbook.org/>

2.3. Instalacija i podešavanje radnog okruženja

Za efikasno korištenje Python-a i Jupyter Notebook-a, potrebno je pravilno instalirati i podešiti radno okruženje. U ovom dijelu, prolazimo kroz korake instalacije Python-a i Jupyter Notebook-a na Windows i Linux operativnim sistemima, kreiranje virtualnih okruženja, instalaciju paketa pomoću komande [pip](#) i upoznavanje sa razlikama između Jupyter Lab-a i Jupyter Notebook-a. Takođe, upoznaćemo se s korištenjem Pythona u terminalnom modu (CLI).

U nastavku će detaljno biti prikazane instrukcije za instalaciju radnog okruženja, koje će omogućiti da svako samostalno prilagodi radno okruženje na svom računaru. Na ovaj način stvaraju se preduslovi za praćenje sadržaja ove knjige, kao i za samostalno testiranje koda ili istraživanje dodatnih funkcionalnosti.

Alternativno, može se koristiti **Google Colab**, kako je opisano u poglavlju „Uvod prije uvida“. Ova platforma omogućava rad u Python okruženju bez potrebe za instalacijom softverskih paketa na vašem računaru, uz jednostavnu razmjenu podataka između radnih stanica.

Ipak, u praksi preferiramo korišćenje lokalno instaliranog okruženja, jer ono omogućava veću kontrolu nad instaliranim paketima, veću fleksibilnost u radu i bolji pregled funkcionalnosti, koje istražujete. Rad u lokalnom okruženju je posebno koristan za složenije analize i veće projekte, koji zahtjevaju specifične resurse ili prilagođena podešavanja.

2.3.1. Verzije i subverzije Python-a

Prije nego što pređemo na postupak instalacije Python-a, važno je razumjeti razlike između glavnih verzija Python-a, v2 i v3, kao i ulogu podverzija unutar glavnih verzija.

Python 2 više nije podržan, što znači da se ne ažurira i ne koristi u modernim aplikacijama. Nasljednik je verziji Python 3, koju ćemo koristiti u ovoj knjizi. Python 3 donosi brojne inovacije, promjene i poboljšanja, uključujući uklanjanje funkcionalnosti, koje su bile nejasne ili zbunjujuće za nove programere.

Iako se Python 2 još uvijek koristi u nekim starijim projektima, većina modernog razvoja koristi Python 3. Dobra vijest je da je oko 90% sintakse i koda između Python-a 2 i Python-a 3 isto, što znači da, ako naučite Python 3, s minimalnim trudom možete razumjeti kod napisan u Python-u 2.

Podverzije unutar Python-a 3

Python 3 se kontinuirano razvija i unapređuje kroz podverzije. Podverzije su manje verzije koje unose ispravke grešaka, optimizacije i nove funkcionalnosti, a pri tome ne narušavaju kompatibilnost s postojećim kodom. Trenutno su aktuelne podverzije:

- **Python 3.13** – Eksperimentalni Just-In-Time (JIT) kompjajler, poboljšan interpreter i druge promjene.
- **Python 3.12** – Uvodi fleksibilnije f-stringove, omogućavajući složenije izraze unutar f-stringova i druge promjene.
- **Python 3.11** – Fokusiran na poboljšanje performansi i uvođenje dodatnih funkcionalnosti za tipove podataka i druge promjene.

Zašto koristiti najnoviju verziju?

Najnovije podverzije uvijek donose poboljšanja u performansama, sigurnosti i funkcionalnostima, čineći rad efikasnijim i sigurnijim. Ukoliko na svom računaru imate stariju verziju Python-a, preporučuje se njeno ažuriranje.

2.3.2. Instalacija Python-a u Windows okruženju

Preduslov: Potrebno je da imate računar sa Windows 10 operativnim sistemom ili višom verzijom. Niže verzije operativnog sistema mogu imati neusaglašenosti sa novijim verzijama Python-a, pa će trebati dodatna podešavanja ili ćete morati instalirati starije verzije Python-a. Arhitektura CPU (Central Processing Unit) ne utiče na proces instalacije, samo je bitno izabrati odgovarajući instalacioni fajl prilagođen arhitekturi vašeg računara.

U nastavku su navedeni koraci za instalaciju sa instrukcijama:

Korak 1. Preuzimanje instalacionog fajla

- 1) Posjetite zvaničnu web stranicu Python organizacije kako bi preuzeли fajl za Windows. Link: <https://www.python.org/downloads/windows/>
- 2) U sekciji “Stable Releases” pronađite najnoviju verziju Python-a, koja odgovara arhitekturi vašeg procesora. Većina modernih računara koristi 64 bitnu arhitekturu. Ako niste sigurni koja je arhitektura vašeg CPU, detaljna uputstva možete da pronađete na zvaničnoj Microsoft web stranici. Link: <http://support.microsoft.com/kb/827218>
- 3) Preuzmite instalacioni fajl koji odgovara vašem sistemu.

Napomena: U ovoj knjizi su prikazani primjeri programa napravljeni sa verzijom Python-a **3.13.0**, ali većina primjera biće kompatibilna i sa novijim verzijama Python-a.

Korak 2. Pokretanje instalacije

- 1) Nakon što ste preuzeeli instalacioni fajl, dvaput kliknite na izvršnu **.exe** datoteku (na primjer: **python-3.13.2-amd64.exe**) da biste pokrenuli Python instalater.

- 2) Označite opciju „*Install launcher for all users*“. Ova opcija, ukoliko se pojavi na vašem operativnom sistemu, omogućava da svi korisnici računara imaju pristup Python aplikaciji.
- 3) Označite opciju „*Add Python to PATH*“. Ova opcija je veoma važna jer omogućava pokretanje Python-a direktno iz komandne linije bez dodatnih podešavanja.



Slika 2.1. Početni prozor instalacionog programa, Izvor: Autori.

- 4) Ako tek počinjete da koristite Python i želite da ga instalirate sa podrazumjevanim funkcijama, jednostavno kliknite na ***Install Now*** i predite direktno na **Korak 4 — Provjera Python instalacije**.

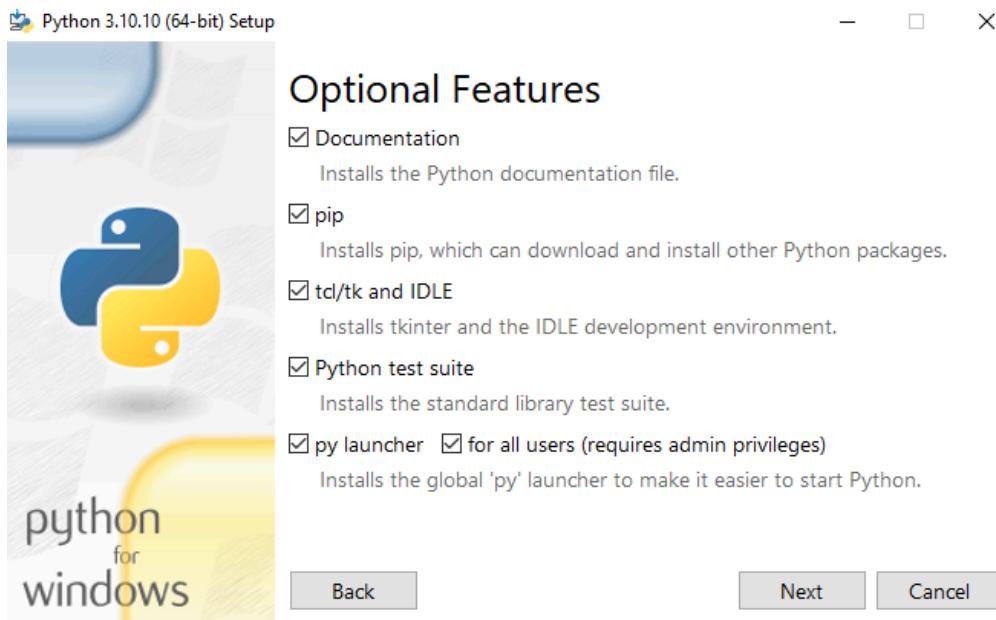
Korak 3. Prilagođavanje instalacije - opciono

Ukoliko želite da prilagodite svoju instalaciju i promijeniti neke predefinisane postavke u vašem Python okruženju, izaberite opciju ***Customize installation*** i pratite sljedeće korake:

- 1) Opcione funkcije (*Optional Features*): Ovo obuhvata alate i resurse koji su korisni za napredne korisnike Python-a, kao što su pip, IDLE (integrisano okruženje za razvoj), dokumentacija i alati za razvoj. Preporučuje se da ih sve instalirate, čak i ako trenutno ne planirate da ih koristite.

Odaberite neke ili sve sljedeće opcije:

- *Documentation*: Preporučuje se instalacija dokumentacije.
- *pip*: Preporučuje se ako želite da instalirate druge Python pakete, poput NumPy ili pandas.
- *tcl/tk i IDLE*¹⁰: Preporučuje se ako planirate da koristite IDLE ili pratite tutorijale koji ga koriste.
- *Python test suite*: Preporučuje se za testiranje i učenje.
- *py launcher*: Preporučuje se kako bi se omogućilo korisnicima da pokreću Python iz komandne linije.



Slika 2.2. Prilagođavanje instalacije, Izvor: Autori.

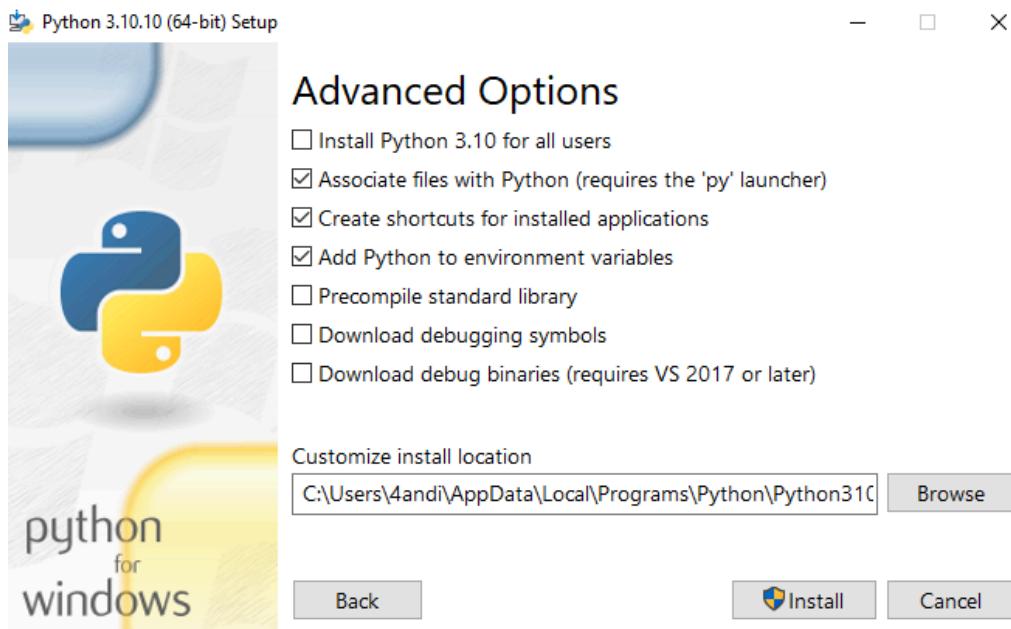
- 2) Nakon podešavanja opcionih funkcionalnosti, kliknuti na **Next** za nastavak instalacije.
- 3) Napredne opcije (*Advanced Options*): Opcije koje se podešavaju u sljedećem dialog prozoru vezane su za integraciju Python u Windows operativni sistem.

Odaberite opcije koje odgovaraju vašim potrebama:

- *Install for all users*: Preporučuje se ako niste jedini korisnik ovog računara.

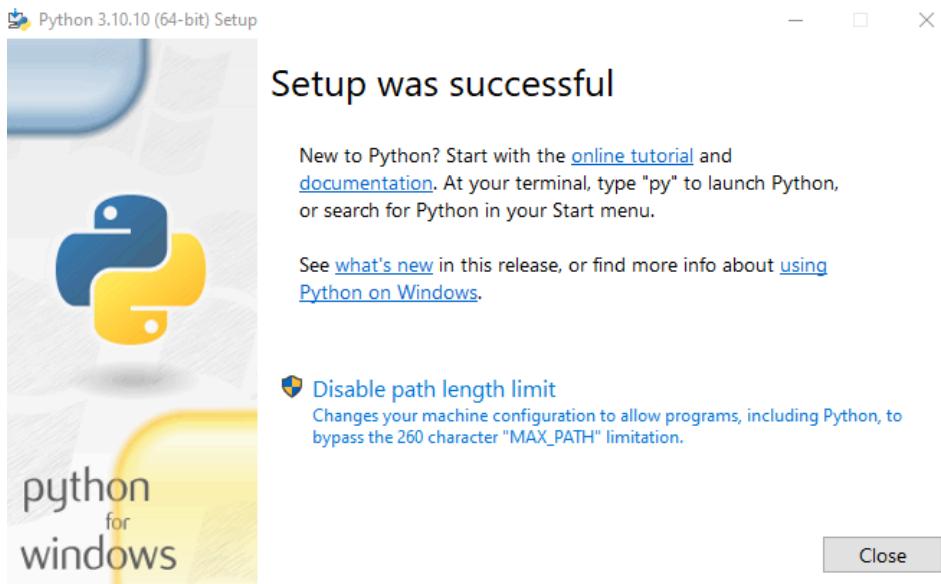
¹⁰ IDLE -Integrated Development and Learning Environment. Link: <https://docs.python.org/3/library/idle.html>

- *Associate files with Python*: Preporučuje se, jer ova opcija povezuje sve tipove Python datoteka sa Python lančerom ili editorom.
- *Create shortcuts for installed applications*: Postavlja simboličke linkove (prečice) za Python aplikacije na desktop.
- *Add Python to environment variables*: Omogućava pokretanje Python-a sa bilo koje pozicije, bez navođena putanje.
- *Precompile standard library*: Nije neophodno, jer može usporiti instalaciju. Pokreće ponovno kompajliranje svih biblioteka.
- *Download debugging symbols i Download debug binaries*: Preporučuje se samo ako planirate da kreirate ekstenzije u C ili C++ jeziku.



Slika 2.3. Podešavanje naprednih opcija, Izvor: Autori.

- 4) Nakon završenog izbora opcija kliknite na **Install** da bi ste započeli instalaciju.
- 5) Nakon što se instalacija završi, prikazaće se poruka **Setup was successful**, što ukazuje na uspješno završenu instalaciju. Ukoliko niste dobili ovu poruku, ili ste dobili neko upozorenje potrebno je potražiti pomoć. Python Forum je odlično mjesto gdje možete pronaći odgovore na vaša pitanja. Link je na web adresi: <https://python-forum.io/>



Slika 2.4. Poruka o uspješno završenoj instalaciji , Izvor: Autori.

Korak4. Provjera Python instalacije

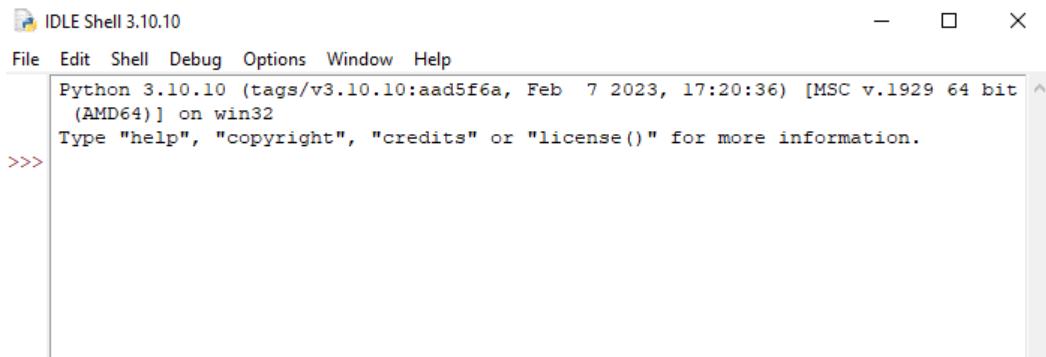
Nakon što ste završili instalaciju, možete provjeriti da li je Python uspješno instaliran i koja je verzija instalirana. To možemo da uradimo na dva načina. Prvi način je putem komandne linije (CMD), a drugi je korišćenjem aplikacije IDLE (Integrated Development and Learning Environment).

- 1) Koristeći **CMD**. Idite na **Start** i u traku za pretragu unesite **cmd**, zatim kliknite na **Command Prompt**. U komandnoj liniji unesite sljedeću komandu:

```
python --version
```

Ukoliko je instalacija uspješna, u konzoli će biti ispisani odgovor u kome se navodi verzija instaliranog Python-a (**Python 3.13.0**). Ako dobijete neki drugi odgovor, morate provjeriti postupak instalacije i da li je uključena opcija „*Add Python to PATH*“.

- 2) Koristeći **IDLE** aplikaciju: Idite na **Start** i u traku za pretragu unesite **python**. Kliknite na aplikaciju **IDLE**, na primjer: **IDLE (Python 3.13 64-bit)**. Kada se otvori IDLE, automatski će prikazati verziju Python-a u naslovnoj traci ili uvodnoj poruci, kao na sljedećoj slici.



Slika 2.5. Radna konyola u IDLE okruženju, Izvor: Autori.

Ako je instalacija uspješno završena i dobili ste poruku da je Python instaliran, vaš računar je spremjan za rad. Provjerili ste instalaciju pomoću komandne linije ili IDLE aplikacije, što potvrđuje da je Python pravilno postavljen i funkcionalan.

Korisnici Windows operativnog sistema sada mogu preći na sljedeći dio – “Podešavanje virtuelnog okruženja”, jer su svi koraci za instalaciju već objašnjeni.

Ako nađete na bilo kakve probleme tokom instalacije ili provjere, preporučuje se da se obratite zvaničnoj dokumentaciji Python-a ili da potražite pomoć na Python zajedničkim forumima.

2.3.3. Instalacija Python-a u Linux okruženju

Linux korisnici, s druge strane, treba da identifikuju svoju distribuciju kako bi prilagodili komande za instalaciju Python-a. S obzirom na veliki broj dostupnih distribucija, fokusiraćemo se na dvije najpopularnije: Ubuntu/Debian i Fedora. Za ostale distribucije, koje nisu derivati ovih sistema, preporučujemo da odgovarajuće uputstvo potražite na Python Forumu.

Većina Linux distribucija dolazi sa unaprijed instaliranim Python-om, jer je on često ključni dio funkcionalnosti operativnog sistema. U nastavku ćemo objasniti kako da provjerite da li je Python instaliran, koja je verzija prisutna i kako da ga instalirate ukoliko nije dostupan na vašem sistemu.

Korak 1. Provjera verzije instaliranog Python-a

Da bi ste provjerili da li imate instaliran Python i koja je verzija instalirana, otvorite Terminal i u njemu unesite komandu:

```
python --version
```

Ako je Python već instaliran na vašem Linux računaru, komanda za provjeru verzije, vratiće odgovor s prikazom verzije instaliranog Python-a (npr. Python 3.13.0). U tom slučaju, nema potrebe da nastavite s instalacijom, već možete odmah preći na dio "Podešavanje virtuelnog okruženja".

Međutim, ako dobijete poruku poput „*command not found*“, to znači da Python nije prepoznat na vašem sistemu i treba nastaviti sa sljedećim koracima za instalaciju.

Napomena: Ako koristite računar s instalacijom Linux operativnog sistema starijom od 2020. godine, postoji mogućnost da imate instaliran Python 2.7 i neku verziju Python-a 3.x. U tom slučaju, komanda `python` može pozivati Python 2, dok za testiranje Python-a 3 treba koristiti komandu `python3`. Preporučuje se da, u slučaju starijih distribucija, obavezno izvršiti ažuriranje Python-a na posljednju dostupnu verziju, kako biste imali pristup novim funkcionalnostima i unapređenjima.

Korak 2. Instalacija Pythona 3 (ako nije instaliran)

Otvorite Terminal i pokrenite sljedeće komande u zavisnosti od vaše linux distribucije:

```
# Ubuntu/Debian
sudo apt update
sudo apt install python3 python3-pip
# Fedora:
sudo dnf install python3 python3-pip
```

Ako su komande za ažuriranje i instalaciju uspješno završene i ako tokom procesa nisu prijavljene greške, preporučuje se da još jednom provjerite verziju instaliranog Python-a.

Ukoliko je prikazana odgovarajuća verzija Python-a, vaš računar je uspješno pripremljen za dalji rad u Python programskom jeziku. Sada možete nastaviti s podešavanjem virtuelnog okruženja.

2.3.4. Instalacija Python virtualnog okruženja

Virtuelno okruženje u Python-u predstavlja izolovani prostor, koji omogućava instalaciju specifičnih verzija Python paketa za određeni projekat, bez uticaja na globalnu instalaciju Python-a ili druge projekte. Ovo je posebno korisno kada radite na više projekata, koji zahtijevaju različite verzije istih paketa ili kada želite da izbjegnete konflikte između zavisnosti.

Da bismo bolje objasnili koncept virtuelnog okruženja, zamislite ga kao zaseban prostor u kojem svaki projekat ima sopstvenu instalaciju potrebnih modula i biblioteka. Kada završite instalaciju Python-a, stvorili ste osnovno okruženje za rad. Međutim, različiti projekti mogu zahtijevati različite module i verzije. Na primjer,

jedan projekat može zahtjevati noviju verziju određene biblioteke, dok drugi koristi stariju verziju. Bez virtuelnog okruženja, ovi konflikti mogu otežati rad i destabilizovati vaše okruženje. Virtuelno okruženje omogućava vam da za svaki projekat kreirate sopstvenu kopiju Python okruženja, koje možete prilagoditi bez uticaja na globalne postavke ili druge projekte.

Korišćenjem virtualnih okruženja možete:

- Izolovati projektne zavisnosti: Svaki projekat može imati svoje pakete i verzije, što osigurava stabilnost i predvidljivost.
- Testirati različite verzije paketa: Lako možete eksperimentisati sa različitim verzijama biblioteka bez rizika po druge projekte.
- Pojednostaviti razvoj i distribuciju: Omogućava lakšu reprodukciju okruženja na drugim računarima ili pri djeljenju koda sa kolegama.

U nastavku su korak po korak prikazane instrukcije za instalaciju i pripremu virtualnog okruženja za Python, koje će biti često korištene u ovoj knjizi. Ove instrukcije mogu poslužiti kao referenca na koju se možete vraćati kad god vam zatreba podsjetnik za instalaciju.

Za pokretanje novog Python virualnog okruženje potrebno je uraditi sljedeće korake:

- 1) Pokretanja komandne linije
- 2) Kreiranje direktorijuma za projekat
- 3) Kreiranje virtuelnog okruženja
- 4) Aktiviranje virtuelnog okruženja
- 5) Testiranje virtuelnog okruženja
- 6) Deaktiviranje virtuelnog okruženja

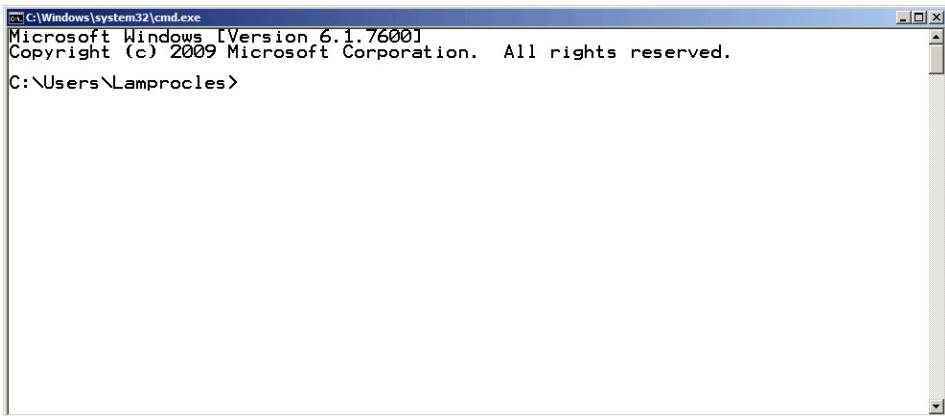
Na svom računaru možete kreirati neograničen broj virtualnih okruženja, odnosno projekata, a jedini limit je prostor za skladištenje podataka. Prosječna veličina virtuelnog okruženje koje sadrži osnovne Python module *pandas*, *numpy* i *scipy*, iznosi oko 300 MB, a sa dodatnim modulima se povećava. Praćenjem ovih instrukcija lako ćete organizovati svoje projekte i održavati stabilnost razvojnih okruženja. U nastavku slijede instrukcije korak po korak.

Korak 1. Pokretanje komandne linije

Postoji više načina za kreiranjem virtuelnog okruženja, a u ovoj knjizi će biti obrađen standardizovan način kroz komandnu liniju (Terminal ili Konzolu). Ovakav pristup se koristi jer je upotrebljiv kako u Windows, tako i u Linux i MacOS okruženju. Takođe, kroz komandnu liniju se kreiraju potrebni direktorijumi (folderi) u kojima se smješta željeni projekat.

- 1) Windows okruženje, *Command Prompt* (CMD):
 - a) Kliknuti na **Start** meni ili pritisniti **Windows** dugme na tastaturi.

- b) U polje za pretragu upisati "cmd".
- c) Kliknuti na aplikaciju "**Command Prompt**" koja se pojavi u rezultatima pretrage.
- d) Windows konzola je spremna za rad, kao na slici 2.6.. Pozadina po želji može da bude crna ili bijela u zavisnosti od podešavanja.



Slika 2.6. Windows konzola, Izvor: Autori.

- 2) Linux okruženje, Linux terminal
 - a) Korištenjem prečice na tastaturi: Pritisnuti **Ctrl + Alt + T**.
 - b) Ili kroz meni aplikacija i pokretanje programa Terminal.
 - c) Nakon toga Linux terminal je spreman za rad, kao na slici 2.7. Pozadina može da bude crna ili bijela u zavisnosti od podešavanja.



Slika 2.7. Linux konzola, Izvor: Autori.

Korak 2. Kreiranje direktorijuma za projekat

Kreiranje posebnog direktorijuma za svaki novi projekat pomaže u organizaciji i olakšava upravljanje datotekama i okruženjima. U ovom koraku su potrebne osnovne komande za rad sa direktorijumima. Na kraju knjige u Prilogu 1. može se pronaći pregled osnovnih komandi za upravljanje fajlovima i direktorijumima u Windows i Linux okruženju.

- 1) Odaberite lokaciju gdje želite smjestiti projekat. Može se koristiti postojeći direktorijum ili kreirati novi unutar vaše radne površine. Koristiti komande za kretanje kroz direktorijume.
- 2) Kreiranje direktorijuma za projekat sljedećom komandom:

```
mkdir moj_projekat      # Windows/Linux
```

- 3) Nakon kreiranja, potrebno se pozicionirati u kreirani direktorijuma koristeći komandu `cd` (*change directory*):

```
cd moj_projekat # Windows/Linux
```

- 4) Sada ste unutar direktorijuma `moj_projekat` i spremni ste za kreiranje virtuelnog okruženja.

Korak 3. Kreiranje virtuelnog okruženja

Za kreiranje Python virtualnog okruženja koristimo modul `venv`, koji je ugrađen u Python. U okviru direktorijuma koji je kreiran za naš projekat pokrećemo sljedeću komandu:

```
python -m venv naziv_okruzenja # Windows/Linux
```

- `python`: Poziva Python interpreter¹¹.
- `-m venv`: Pokreće modul za kreiranje virtuelnog okruženja.
- `naziv_okruzenja`: Zamjenite ovo željenim imenom vašeg okruženja. Izbor imena za okruženje je proizvoljno, ali je preporuka da bude tako da vas asocira da je to sistemski direktorijum npr. (`venv`, `env`, `.venv`, `.env`). U nastavku knjige, koristićemo `.venv` za naziv našeg virtualnog okruženja (napomena: postoji tačka ispred `venv`, time postižemo da je direktorijum skriven.)

Korak 4. Aktiviranje virtuelnog okruženja

¹¹ Python interpreter je osnovna komponenta Python programskog jezika koja omogućava direktno izvršavanje koda. On čita i izvršava Python skripte liniju po liniju, omogućavajući brzu provjeru rezultata i lakše otklanjanje grešaka.

Nakon kreiranja virtuelnog okruženja, potrebno je aktivirati okruženje kako bi mogli da nastavimo da radimo na našem projektu. Za aktiviranje virtualnog okruženje potrebno je otkucati sljedeću komandu:

```
.venv\Scripts\activate      # Windows  
source .venv/bin/activate  # Linux
```

Kada je virtuelno okruženje aktivirano, prompt će se promijeniti i ispred će se pojaviti naziv okruženja:

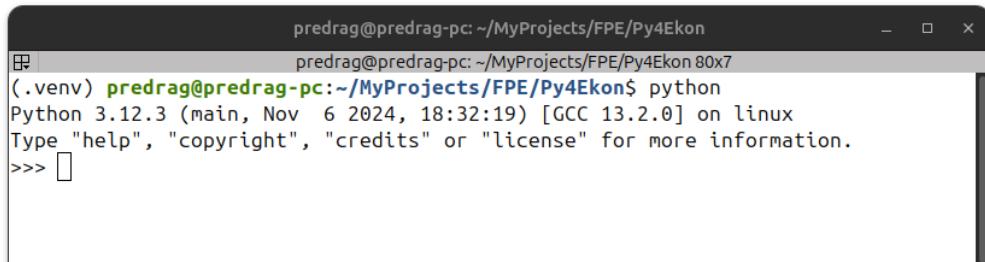
```
(.venv) C:\Users\predrag\Py4Ekon>      # Windows  
(.venv) predrag@predrag-pc:~/Py4Ekon$  # Linux
```

Sada je naše virtualno okružene spremno za rad.

Korak 5. Testiranje Python virtualnog okruženja

Nakon kreiranja i aktiviranja Python virtuelnog okruženja, potrebno je provjeriti da li sve funkcioniše kako treba. U ovom koraku će biti pokazano kako testirati virtuelno okruženje iz komandne linije, instalirati modul `numpy`, i potvrditi njegovu instalaciju pomoću komande `freeze`.

- 1) Pokretanje Python-a iz komandne linije. Ukoliko je aktivno virtualno okruženje, promijenjen je izgled prompta. Python interpreter pokrećemo komandom `python` i nakon toga prelazimo u interaktivni mod. Ako Python ispravno radi, trebalo bi da se prikaže interaktivni Python CLI (*Command Line Interface*) sa verzijom Python-a i promptom `>>>`, kao na sljedećoj slici.



```
predrag@predrag-pc: ~/MyProjects/FPE/Py4Ekon  
predrag@predrag-pc: ~/MyProjects/FPE/Py4Ekon 80x7  
(.venv) predrag@predrag-pc:~/MyProjects/FPE/Py4Ekon$ python  
Python 3.12.3 (main, Nov  6 2024, 18:32:19) [GCC 13.2.0] on linux  
Type "help", "copyright", "credits" or "license" for more information.  
>>> 
```

Slika 2.8. Python CLI, Izvor: Autori.

- 2) Instalacija modula `numpy`. Izaći iz Python CLI-ja unosom komande `exit()` ili kombinacijom tastera `Ctrl+Z` + `Enter`. Zatim instalirati `numpy` unutar virtuelnog okruženja pomoću `pip` komande.

```
pip install numpy
```

Ako je instalacija uspješna, pojaviće se poruka koja je slična sljedećoj poruci:
Successfully installed numpy-2.1.1

- 3) Provjera instalacije iz Python CLI-ja. Ponovo pokrenuti Python CLI komandom `python`. U okviru interaktivnog Python okruženja, provjerite da li je modul numpy ispravno instaliran tako što unesete sljedeći kod:

```
import numpy  
print(numpy.__version__)
```

Ako je instalacija uspješna, Python će ispisati verziju numpy modula.

- 4) Provjera instaliranih paketa pomoću komande `freeze`.

```
pip freeze
```

Korak 6. Deaktiviranje virtuelnog okruženja

Kada se završi rad u virtuelnom okruženju, preporučljivo ga je deaktivirati, kako biste se vratili na globalno okruženje. Deaktiviranje se može postići komandom za deaktiviranje ili zatvaranje terminala odnosno konzole. Komanda za deaktiviranje je:

```
deactivate      # Windows/Linux
```

Nakon što izvršite ovu komandu, prompt će se vratiti u prvočitno stanje, bez naziva virtuelnog okruženja ispred.

2.4. Instalacija i pokretanje JupyterLab-a

JupyterLab predstavlja novu generaciju interaktivnog okruženja za rad s radnim sveskama (*notebooks*), kodom i podacima. Kao moćan alat za interaktivno programiranje i analizu podataka, JupyterLab pruža napredne mogućnosti u odnosu na tradicionalni Jupyter Notebook. U ovom poglavlju fokus je na procesu instalacije JupyterLab-a, načinu njegovog pokretanja, te objašnjenju ključne razlike između JupyterLab-a i Jupyter Notebook-a. Ovo će omogućiti korisnicima da odaberu radno okruženje, koje najbolje odgovara potrebama u učenju i istraživanju.

2.4.1. Instaliranje JupyterLab-a

Instaliranje JupyterLab-a je jednostavan proces, koji se može obaviti putem menadžera paketa [pip](#), koji dolazi uz Python. Preporuka je da se instalacija izvrši unutar virtuelnog okruženja, kako bi izolovali zavisnosti novog projekta.

Preduslovi:

- Python 3.5 ili noviji: Provjerite da li imate instaliranu odgovarajuću verziju Pythona.
- Virtuelno okruženje: Iako nije obavezno, preporučljivo je koristiti virtuelno okruženje za instalaciju JupyterLab-a, kako bi izbjegli konflikte s drugim Python paketima.

Koraci za instalaciju:

- 1) Kreiranje i aktiviranje virtuelnog okruženja

```
python -m venv .venv  
.venv\Scripts\activate
```

- 2) Ažuriranje [pip](#) menadžer paketa

```
pip install --upgrade pip
```

- 3) Instalacija JupyterLab-a

```
pip install jupyterlab
```

Ova komanda će preuzeti i instalirati sve potrebne zavisnosti za JupyterLab.

4) Provjera uspješnosti instalacije

```
jupyter-lab --version
```

Ako instalacija nije uspjela ili se pojave greške, provjerite internet konekciju i da li je [pip](#) ispravno ažuriran.

2.4.2. Pokretanje JupyterLab-a

Nakon uspješne instalacije, JupyterLab možete pokrenuti iz komandne linije. Prije pokretanja, obavezno provjerite:

- Da se nalazite se u željenom radnom direktorijumu ili projektu.
- Da je aktivirano virtuelno okruženje u kojem je JupyterLab instaliran, kako je objašnjeno u prethodnim koracima.

Ovi koraci su potrebni svaki put kada želite pokrenuti JupyterLab, posebno nakon restartovanja računara ili zatvaranja konzole u kojoj je aplikacija prethodno bila aktivirana.

U terminalu ili komandnoj liniji (bez obzira na operativni sistem – Windows, Linux ili macOS), JupyterLab se pokreće pomoću sljedeće komande:

```
jupyter-lab
```

Ova komanda pokreće JupyterLab server i automatski otvara interaktivno okruženje u vašem web pretraživaču, kao što su Microsoft Edge, Google Chrome, Mozilla Firefox, ili bilo koji drugi pretraživač, koji je podešen kao podrazumijevani na vašem računaru.

Šta se dešava prilikom pokretanja?

Prilikom pokretanja JupiterLab aplikacije na vašem računaru će se desiti sljedeće promjene:

- Pokretanje servera: JupyterLab pokreće lokalni server na vašem računaru, koji omogućava interakciju sa radnom sveskom i drugim datotekama.
- Otvaranje u web pretraživaču: Automatski se otvara nova kartica ili prozor u vašem web pretraživaču s interfejsom JupyterLab-a.
- URL adresa: Standardna web adresa na kojoj je pokrenut web servis i na kojoj možete pristupiti aplikaciji je <http://localhost:8888/lab>, port je 8888. Ako je ovaj port zauzet, koristite se naredni slobodan port.

Ako se želi zaustaviti JupyterLab aplikacija, važno je znati da web pretraživač služi samo kao interfejs za jednu aktivnu sesiju rada. Zatvaranjem prozora web pretraživača ne prekida se rad JupyterLab servera, koji će i dalje ostati aktivan u pozadini.

Da bi se u potpunosti zaustavio JupyterLab server, potrebno se vratiti u terminal iz kojeg je i pokrenut. Pritisnuti kombinaciju tastera **Ctrl + C**. Terminal će zatim dati poruku sa pitanjem da li želite da prekinuti rad servera. Da bi završili proces potrebno je unijeti **y** i pritisnuti **Enter**.

2.4.3. Razlike između JupyterLab-a i Jupyter Notebook-a

Iako su JupyterLab i Jupyter Notebook dio istog Jupyter ekosistema i dijele mnoge funkcionalnosti, postoje značajne razlike u njihovom dizajnu i mogućnostima.

Jupyter Notebook nudi jednostavan i linearan interfejs, fokusiran na rad sa pojedinačnim radnim sveskama. Njegovo okruženje se sastoji od liste fajlova i mogućnosti otvaranja pojedinačnih radnih svesaka u posebnim tabovima pretraživača. Ovaj pristup omogućava osnovno uređivanje i izvršavanje ćelija sa kodom i tekstom, što ga čini idealnim za brze prototipove, jednostavne analize i edukativne svrhe. Međutim, njegova funkcionalnost je ograničena kada su u pitanju složeniji projekti, jer pruža minimalnu podršku za proširenja i rad sa različitim vrstama datoteka.

Nasuprot tome, **JupyterLab** donosi moderan i fleksibilan interfejs, koji se ponaša kao višepanelna radna površina. Omogućava otvaranje više radnih svesaka, terminala, tekstualnih editora i konzola unutar istog prozora, pružajući korisnicima veću kontrolu nad radnim prostorom. Osim osnovnih funkcionalnosti, JupyterLab podržava integraciju različitih tipova datoteka, uključujući CSV, slike i PDF, kao i napredne alate za vizualizaciju, debagovanje i organizaciju projekta. Njegova podrška za proširenja, poput Git integracije ili dodatnih vizualnih alata, čini ga pogodnim za složene projekte, koji zahtijevaju rad sa više vrsta datoteka i različitim alatima.

Za jednostavne analize i edukaciju, Jupyter Notebook može biti dovoljan. Međutim, za složenije projekte i potrebe organizacije, JupyterLab pruža sveobuhvatniji pristup, koji omogućava efikasniji rad i bolju prilagodljivost.

U sljedećoj tabeli su navedene ključne razlike između JupyterLab i Jupyter Notebook okruženja.

Tabela 2.1. Razlike između JupyterLab i Jupyter Notebook okruženja

Karakteristika	JupyterNotebook	JupyterLab
Interfejs	Jednostavan, linearan	Višepanelni, prilagodljiv
Upravljanje datotekama	Osnovni pregled fajlova	Napredni preglednik s podrškom za različite tipove datoteka
Proširenja i dodaci	Ograničena podrška	Bogata podrška za proširenja
Prilagodljivost	Minimalna	Visoka
Integracija alata	Osnovna	Integrисани terminal, editori, konzole
Učenje krivulje	Jednostavnije za početnike	Zahtijeva malo više vremena za navikavanje

Preporuka koje okruženje odabrati?

- **Jupyter Notebook:**
 - Ako ste početnik i želite jednostavno okruženje za učenje Pythona i Jupyter-a.
 - Za brze i jednostavne zadatke ili demonstracije.
- **JupyterLab:**
 - Ako radite na kompleksnijim projektima, koji zahtijevaju rad s više tipova datoteka i alata.
 - Ako želite naprednije funkcionalnosti i veću produktivnost.
 - Ako planirate raditi na dugoročnim projektima gdje je organizacija radnog prostora važna.

Napomena za studente: Iako JupyterLab na prvi pogled može djelovati složenije, njegova fleksibilnost i dodatne mogućnosti značajno će vam uštediti vrijeme i poboljšati produktivnost, posebno kada budete radili na složenijim projektima. Ova knjiga koristi isključivo JupyterLab okruženje za sve primjere i funkcionalnosti koje obrađujemo. Kada savladate rad u JupyterLab-u, prelazak na Jupyter Notebook biće jednostavan, jer predstavlja pojednostavljenu verziju JupyterLab-a.

2.5. Osnove rada u JupyterLab okruženju

JupyterLab predstavlja napredno interaktivno okruženje, koje omogućava korisnicima da kombinuju kod, tekst, vizualizacije i druge resurse u jedinstvenom radnom prostoru. Kao nasljednik tradicionalnog Jupyter Notebook-a, JupyterLab pruža veću fleksibilnost i bolje alate za organizaciju i upravljanje projektima. U ovom poglavlju biće detaljno istražene osnovne komponente JupyterLab-a, uključujući navigaciju kroz interfejs, strukturu radne sveske (*notebook*), rad sa različitim tipovima celija i korisne komande koje pomažu da se efikasno koristiti ovo moćno okruženje.

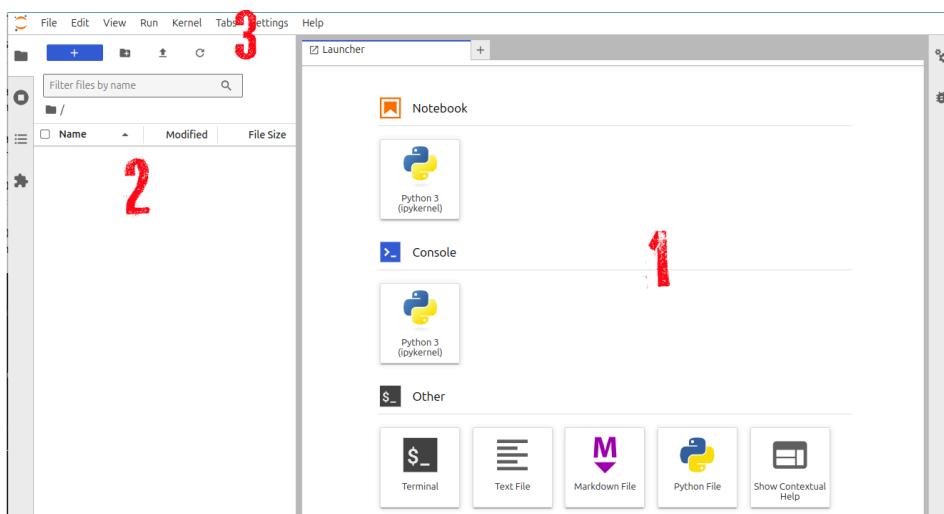
2.5.1. Navigacija kroz Jupyter interfejs

Kada se pokrene JupyterLab, pojavljuje se moderno i prilagodljivo radno okruženje, koje se otvara u podrazumijevanom web pretraživaču. Interfejs JupyterLab-a sastoji se od nekoliko ključnih elemenata, koji omogućavaju jednostavnu navigaciju i organizaciju vaših projekata.

Glavni elementi interfejsa:

1) *Launcher* (Pokretač):

Pri prvom pokretanju, u centralnom dijelu ekrana prikazuje se *Launcher*, koji omogućava brzo kreiranje novih radnih sveski, tekstualnih fajlova, terminala i drugih resursa. Ovdje se može odabratи željeni kernel (npr., Python 3) za vašu radnu svesku. Sljedeća slika 2.9. pozicija 1, prikazuje standardni JupyterLab *Launcher*.



Slika 2.9 JupyterLab Launcher, Izvor: Autori.

2) *File Browser*:

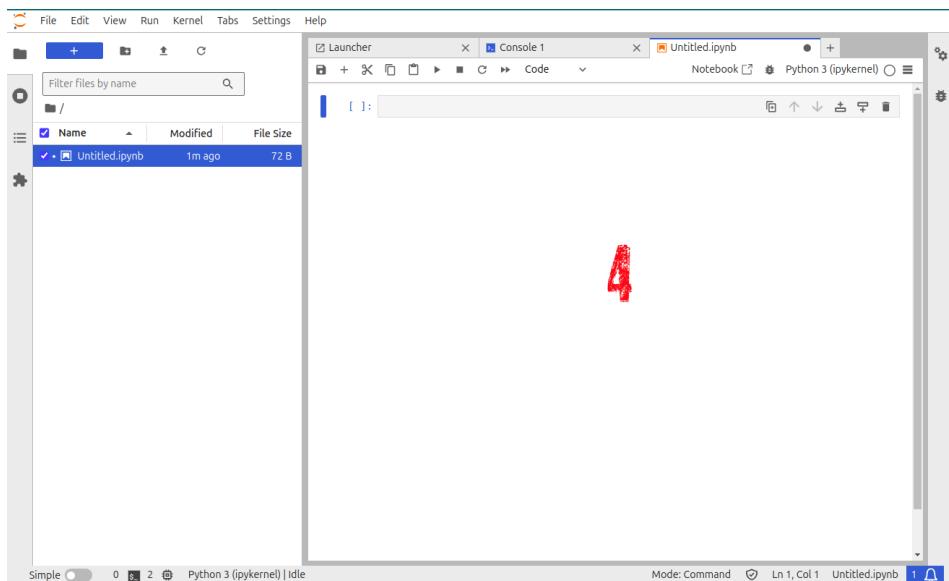
Sa lijeve strane nalazi se *File Browser*, koji prikazuje strukturu direktorijuma vašeg radnog okruženja. Ovdje se mogu pregledati, otvoriti, kreirati i organizovati fajlovi i direktorijumi koji se koriste u projektu. Ukoliko je pokrenut novi projekat, on nema početnih struktura direktorijuma i fajlova. Kako budemo dodavali naše radeve u ovom dijelu (slika 2.9. pozicija 2) će se pojavljivati fajlovi i direktorijumi.

3) *Menu Bar (Meni)*:

Na vrhu ekrana nalazi se *Menu Bar* sa standardnim opcijama kao što su *File*, *Edit*, *View*, *Run*, *Kernel*, *Tabs*, *Settings* i *Help*. Ove opcije pružaju pristup različitim funkcionalnostima JupyterLab-a. (slika 2.9. pozicija 3)

4) *Main Work Area (Glavni radni prostor)*:

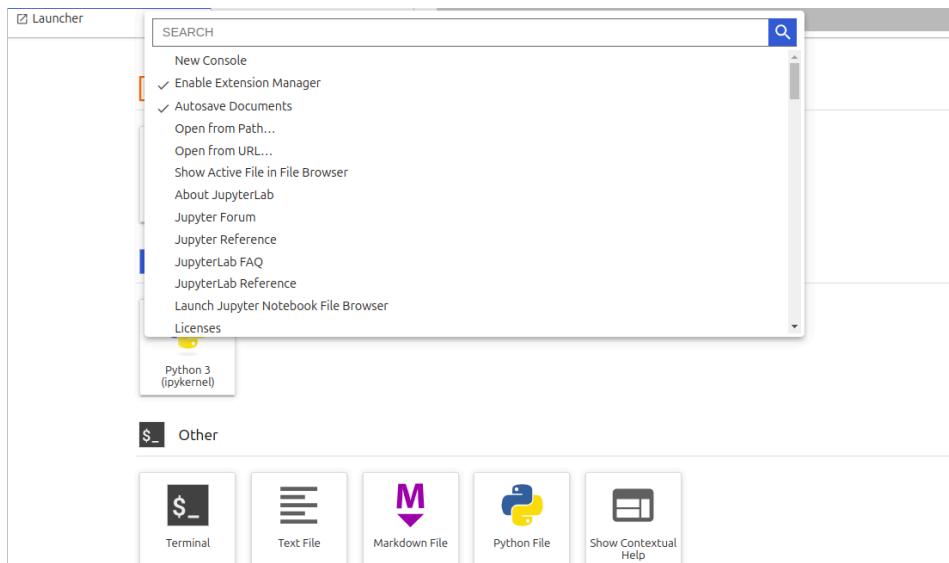
Centralni dio ekrana je Glavni radni prostor, gdje se otvaraju radne sveske, tekstualni fajlovi, konzole i drugi elementi. JupyterLab podržava višestruke tabove i prozore, koji se mogu rasporediti po želji, omogućavajući da se istovremeno radi na više resursa. Novu radnu svesku otvaramo iz Launcher-a klikom na ikonu ispod Notebook, ili kroz meni na opciju File->New->Notebook. U novom tabu se otvara prazna radna sveska kao što je prikazano na slici 2.10. pozicija 4.



Slika 2.10. *JupyterLab Notebook*, Izvor: Autori.

5) *Command Palette* (Paleta komandi):

Command Palette možete otvoriti pritiskom na **Ctrl + Shift + C** (ili **Cmd + Shift + C** na macOS-u). Ona omogućava brzo pokretanje komandi i akcija u JupyterLab-u, slično kao u modernim editorima koda. Na sljedećoj slici 2.11. prikazana je paleta komandi za brzi izbor.



Slika 2.11 *Command Palette-a sa pretragom komandi*, Izvor: Autori.

6) Console (Konzola):

Konzola je alat u JupyterLab-u, koji omogućava direktno izvršavanje koda u realnom vremenu. Nalazi se u centralnom radnom prostoru i otvara se kroz Launcher klikom na opciju Console, ili putem menija odabirom File -> New -> Console. Konzola je povezana s kernelom i omogućava testiranje koda liniju po liniju, uz trenutne rezultate koji se prikazuju direktno ispod unijetog koda. Idealna je za brzo ispitivanje funkcija ili podataka dok radite na projektu. Konzola je identično Python CLI, koji smo ranije opisali u testiranju Python instalacije.

7) Terminal

Terminal u JupyterLab-u omogućava pristup komandnoj liniji vašeg sistema direktno iz interfejsa, pružajući funkcionalnosti poput bash-a na Linux-u ili Command Prompt-a na Windows-u. Može se otvoriti kroz Launcher odabirom opcije Terminal, ili putem menija izborom File > New > Terminal. Koristi se

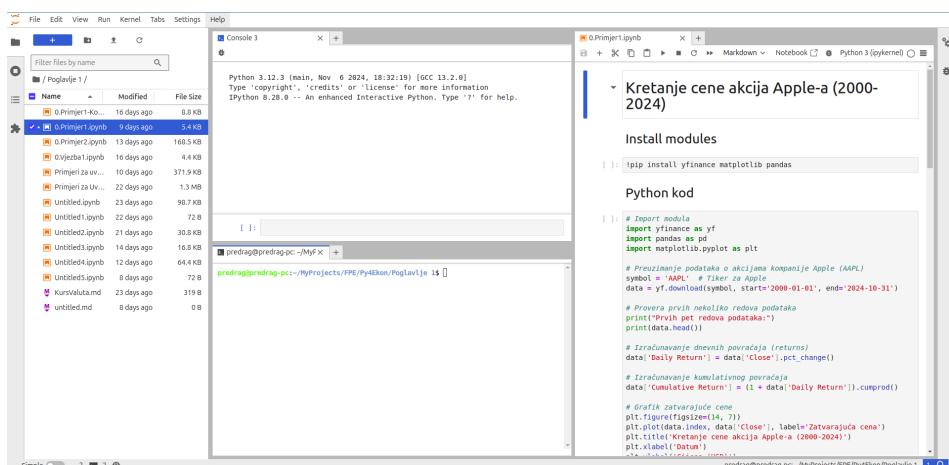
za izvršavanje sistemskih komandi, upravljanje datotekama, pokretanje skripti ili instalaciju Python paketa, a sve to bez izlaska iz JupyterLab okruženja.

Terminal se otvara kao kartica unutar glavnog radnog prostora, omogućavajući paralelan rad sa više radnih sveski i konzolama. Može se otvoriti više terminala istovremeno za obavljanje različitih zadataka, dok JupyterLab postavke omogućavaju osnovno prilagođenje izgleda terminala, poput promjene veličine fonta i boje pozadine.

Navigacija i organizacija

Otvaranje fajlova u JupyterLab-u je intuitivno i brzo. Dvostrukim klikom na bilo koji fajl u File Browser-u, taj fajl se otvara u glavnom radnom prostoru. Ako otvarate radne sveske, one će se automatski prikazati u novom tabu, omogućavajući istovremeni rad na više njih. Ovo je posebno korisno kada se istovremeno radi na različitim analizama ili projektima.

Raspoređivanje prozora pruža fleksibilnost u organizaciji radnog prostora. Tabovi se mogu prevlačiti i raspoređivati horizontalno ili vertikalno, čime se raspored prilagođava svojim potrebama. Na primjer, moguće je imati radnu svesku otvorenu pored terminala za izvršavanje komandi ili konzole za interaktivni rad. Ovo omogućava efikasno korištenje ekrana i lakši pristup svim resursima. Preraspodjela tabova se radi tehnikom drag-and-drop, tako što se mišem uzmima jezičak taba i prenese na željenu lokaciju. Sistem vam simulira izgled i poziciju prije nego otpustite tab. (Slika 2.12.)



Slika 2.12. Prikaz radne površine sa više tabova, Izvor: Autori.

Pretraga fajlova olakšava pronalaženje potrebnih dokumenata. U gornjem dijelu File Browser-a nalazi se polje za pretragu, koje omogućava brzo lociranje fajlova prema njihovom nazivu. Ova funkcionalnost je posebno korisna kada se radi sa velikim brojem fajlova i želi smanjiti vrijeme traženja.

Ove funkcionalnosti čine JupyterLab moćnim alatom za navigaciju i organizaciju, omogućavajući jednostavno upravljanje radnim prostorom i resursima, bez obzira na složenost projekta.

2.5.2. Struktura radne sveske u JupyterLab-u

Jupyter radne sveske su strukturane na način, koji omogućava intuitivno i efikasno organizovanje rada. Osnovna jedinica rada u Jupyter radnim sveskama su **ćelije**, koje pružaju fleksibilnost i modularnost u pisanju, uređivanju i izvršavanju sadržaja. Svaka ćelija ima specifičnu funkciju, a tip ćelije se može lako odabrati ili promijeniti prema potrebi u toku rada.

Postoje tri glavne vrste ćelija koje se mogu koristiti u radnim sveskama:

1) Ćelije za kod (Code cells):

Ove ćelije predstavljaju osnovu interaktivnog rada u Jupyter radnim sveskama. Koriste se za unos i izvršavanje Python koda (ili drugog jezika podržanog kernelom). Nakon izvršavanja, rezultati se prikazuju neposredno ispod ćelije, omogućavajući direktni uvid u rezultate analize ili eventualne greške.

2) Markdown ćelije (Markdown cells):

Markdown ćelije omogućavaju unos formatiranog teksta koristeći Markdown sintaksu. Koriste se za dodavanje opisa, naslova, uputstava ili komentara unutar radne sveske, čineći vaš rad preglednjim i razumljivim. Ove ćelije su idealne za dokumentovanje analiza i olakšavaju dijeljenje radnih svesaka s drugima.

3) Raw ćelije (Raw cells):

Raw ćelije su neobrađene ćelije koje se ne izvršavaju niti prikazuju u izlaznom formatu. Koriste se za specifične svrhe, poput uključivanja sadržaja, koji je potreban za prilagođeno procesiranje ili konverziju u druge formate, kao što su PDF ili LaTeX.

Struktura Jupyter radnih svesaka omogućava korisnicima da efikasno kombinuju kod, tekst i dodatne informacije na jednom mjestu. Ova organizacija čini Jupyter radne sveske izuzetno korisnim alatom za interaktivno programiranje, analizu podataka i dokumentovanje procesa rada, posebno u kontekstu ekonomskih analiza.

2.5.2.1. Ćelije za kod (Code cells)

Ćelije za kod predstavljaju osnovni element Jupyter radnih svesaka, gdje se unosi i izvršava Python kod. Svaka ćelija može sadržavati jedan ili više redova koda, koji se zajedno izvršavaju kao cjelina. Ove ćelije omogućavaju iterativni pristup programiranju, što znači da se mogu testirati, prilagođavati i ponovno pokretati dijelove koda tokom rada, bez potrebe za ponovnim izvršavanjem cjelokupnog programa.

Redoslijed izvršenja ćelija određuje redoslijed pokretanja dijelova koda. Nije potrebno pratiti redoslijed kojim su ćelije unešene. Mogu se pokretati u bilo kojem redoslijedu. Međutim, ovaj pristup može uticati na rezultate, jer neke promjene ili varijable mogu zavisiti od prethodno izvršenih ćelija. Da bi izbjegli zabunu i greške, preporučuje se da redoslijed izvršenja ćelija prati logički tok od vrha ka dnu sveske. Na taj način osiguravate da su svi dijelovi koda izvedeni pravilnim redoslijedom i da rezultati odgovaraju očekivanjima.

Unesite sljedeći jednostavan kod u prethodno otvorenu radnu svesku u Jupyter-u.

```
# Izračunavanje jednostavnog izraza
a = 5
b = 3
suma = a + b
print("Suma je:", suma)
```

Na slici 2.13. označene su ključne pozicije za rad s ćelijama u Jupyter radnim sveskama, uključujući izbor vrste ćelije i dugme za pokretanje koda u aktivnoj ćeliji. Aktivna ćelija je ona sa kojom se trenutno radi, a njen izbor vrši se jednostavnim pozicioniranjem kurzora u ćeliju i klikom na lijevi taster miša.

Aktivna ćelije se prepoznaju po plavom vertikalnom markeru koji se pojavljuje s lijeve strane aktivne ćelije. Ovaj vizuelni marker olakšava rad, jer jasno pokazuje koja je ćelija spremna za uređivanje ili izvršavanje, omogućavajući preciznu kontrolu nad tokom rada.



Slika 2.13. Izbor vrste ćelije i dugme za izvršenje koda, Izvor: Autori.

Kada se u ćeliju unese ovaj kod i pokrene program (na ikonicu Run >), ispod ćelije će se ispisati sljedeći rezultat: Suma je: 8

Načini izvršavanja koda:

- 1) Klik na dugme "Run": U paleti alata iznad radne sveske nalazite dugme "Run". Klikom na njega izvršava se trenutna ćeliju.

- 2) Pritisom na **Shift + Enter**: Ova kombinacija izvršava ćeliju i automatski prelazi na sljedeću, olakšavajući kontinuirani rad.
- 3) Pritisom na **Ctrl + Enter**: Ova opcija izvršava ćeliju, ali zadržava fokus na istoj ćeliji, što je korisno kada se želi više puta testirati isti kod.

Ćelije za kod su idealne za interaktivno programiranje, jer omogućavaju korisnicima da brzo testiraju fragmente koda i odmah vide rezultate. Ovaj pristup je posebno koristan za analizu podataka, testiranje funkcija i debagovanje koda. Uz fleksibilnost koju nude, ćelije za kod čine rad u Jupyter radnim sveskama jednostavnim i efikasnim.

Rezultati izvršavanja ćelije u Jupyter radnim sveskama prikazuju se neposredno ispod te ćelije. Ovi rezultati mogu biti različiti, uključujući tekstualne izlaze, grafike ili čak interaktivne elemente. Upravljanje izlazima omogućava da se organizuje i prilagodi način prikaza rezultata rada.

Primjeri upravljanja izlazom:

- Sakrivanje izlaza: Ako želite ukloniti prikazane rezultate, desnim klikom na izlaz odaberite opciju "*Clear Output*".
- Ponovno prikazivanje izlaza: Rezultati će se ponovo generisati jednostavnim ponovnim izvršavanjem ćelije.
- Zadržavanje izlaza pri ponovnom pokretanju: Izlazi se mogu sačuvati zajedno s radnom sveskom, omogućavajući njihov pregled i nakon što je sveska ponovo otvorena.

Ove opcije omogućavaju veću fleksibilnost u radu, pomažući da se jasno organizuju i analiziraju rezultati bez pretpavanja radnog prostora.

2.5.2.2. Markdown ćelije (Markdown cells)

Markdown ćelije koriste se za pisanje formatiranog teksta, objašnjenja, naslova i uključivanje multimedijalnih sadržaja. Ove ćelije podržavaju Markdown sintaksu, koja omogućava jednostavno i efikasno formatiranje teksta. U ovom dijelu knjige biće objašnjene osnovne funkcionalnosti i sintaksa Markdown jezika, dok detaljniji vodič se može pronaći na zvaničnoj web stranici (<https://www.markdownguide.org/>).

Da biste promijenila aktivnu ćeliju u Markdown ćeliju, potrebno je promijeniti njen tip, kako je prikazano na slici 2.14. Nakon unosa teksta sa sintaksom za formatiranje, Markdown ćeliju treba izvršiti na isti način kao i ćeliju za kod – korištenjem komande **Run** ili kombinacijom tastera **Shift + Enter**.

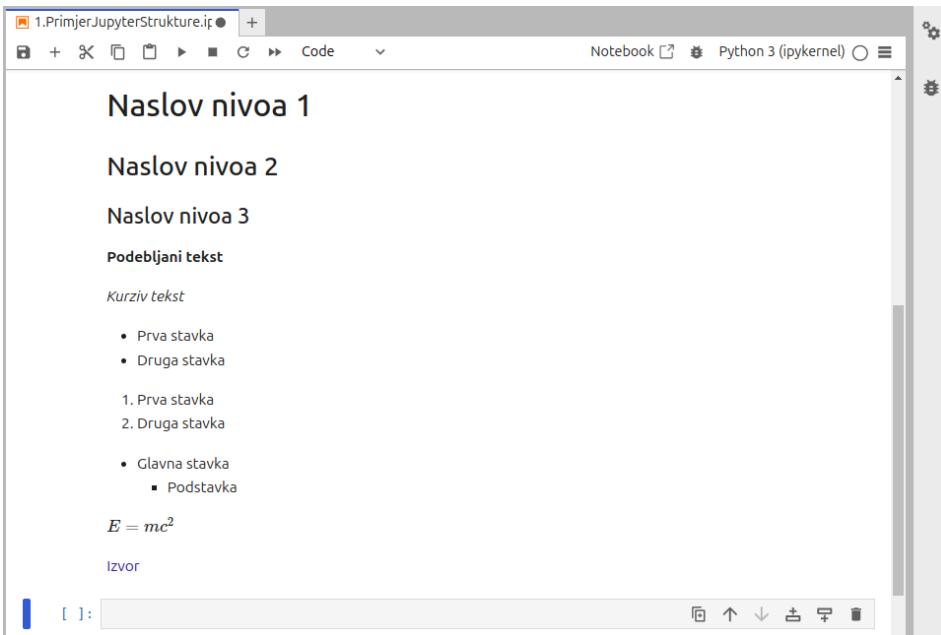
Nakon izvršenja, tekst se u ćeliji prikazuje u formatiranom obliku, a ćelija prelazi u *read-only* mod, što znači da više nije moguće unositi promjene direktno. Ako se želi izmijeniti formatirani tekst, potrebno je dvaput kliknute na ćeliju, kako bi je ponovo otvorili za uređivanje. Nakon što se unese izmjena, ćeliju je potrebno ponovo izvršiti, kako bi se prikazale nove promjene u formatiranom obliku.

Ova funkcionalnost omogućava lako prebacivanje između uređivanja i prikaza formatiranog teksta, što čini Markdown celije idealnim za dokumentovanje i strukturisanje vašeg rada u radnoj svesci.

U sljedećoj tabeli su prikazana osnovna uređenja teksta, a na slici 2.14. je prikazan rezultat ovog načina formatiranja.

Tabela 2.2. Osnove uređenja teksta

Sintaksa	Izgled	Opis
# Naslov nivoa 1	Naslov nivoa 1	Formatiranje naslova ekvivalentno MS Word Heading 1.
## Naslov nivoa 2	Naslov nivoa 2	Formatiranje naslova ekvivalentno MS Word Heading 2.
### Naslov nivoa 3	Naslov nivoa 3	Formatiranje naslova ekvivalentno MS Word Heading 3.
Podebljani tekst	Podebljani tekst	Koristi se za podebljavanje teksta.
Kurziv tekst	<i>Kurziv tekst</i>	Koristi se za isticanje teksta kurzivom.
- Prva stavka - Druga stavka	<ul style="list-style-type: none"> ● Prva stavka ● Druga stavka 	Stavke u neuređenoj listi.
1. Prva stavka 2. Druga stavka	<ol style="list-style-type: none"> 1. Prva stavka 2. Druga stavka 	Stavke u uređenoj listi.
- Glavna stavka - Podstavka	<ul style="list-style-type: none"> ● Glavna stavka <ul style="list-style-type: none"> ○ Podstavka 	Stavka i podstavka u neuređenoj listi.
\$E = mc^2\$	Formula: $E = mc^2$	Pisanje matematičkih izraza koristeći LaTeX sintaksu unutar znaka \$.
[Izvor](https://www.primjer.com)	izvor	Dodavanje hiperveza koje vode do određenih web stranica.



Slika 2.14. Prikaz formatiranog teksta u Jupyter-u, Izvor: Autori.

2.5.2.3. Raw ćelije (Raw cells)

Raw ćelije su specifične u Jupyter radnim sveskama, koje omogućavaju unos sirovog, neobrađenog sadržaja. Za razliku od ćelija za kod ili Markdown ćelija, Raw ćelije se ne izvršavaju niti formatiraju unutar Jupyter okruženja. Umjesto toga, njihov sadržaj ostaje nepromijenjen i koristi se u posebnim kontekstima, kao što su prilagođeni procesi pretvaranja radne sveske u druge formate (npr. LaTeX, PDF, ili HTML).

Raw ćelije su korisne kada se radi na projektima, koji zahtijevaju očuvanje originalnog sadržaja bez obrade. To može uključivati dodatne komentare, kod koji treba koristiti van Jupyter-a, ili specifične formate koji nisu podržani direktno u radnoj svesci.

2.5.2.4. Upotreba Raw ćelije

Markdown ćelije omogućavaju formatiranje teksta u JupyterLab-u, ali ponekad je potrebno sačuvati sadržaj bez formatiranja – u tom slučaju koristi se Raw ćelija.

- Dodavanje Raw ćelije. Da bi dodali Raw ćeliju, promijenite tip ćelije u "Raw" pomoću padajućeg menija u paleti alata (kao što je prikazano na slici 2.13.).
- Unos sadržaja. Upisati sirovi tekst ili kod koji se želi sačuvati u neobradenom obliku.

2.5.2.5. Primjer upotrebe Raw ćelije

Sljedeći primjer uključuje LaTeX kod, koji će se koristiti prilikom konverzije radne sveske u PDF format.

```
\section{Naslov poglavlja} Ovo je sirovi LaTeX sadržaj koji će biti procesiran prilikom konverzije u PDF.
```

Prilikom izvoza radne sveske u PDF koristeći LaTeX alatke, sadržaj ove ćelije će biti prepoznat i procesiran prema LaTeX sintaksi, dok bi u Jupyter-u ostao nepromijenjen.

2.5.2.6. Prednosti Raw ćelije

Raw ćelije su idealne za napredne korisnike, koji žele kombinovati interaktivnost Jupyter radnih svesaka s prilagođenim procesima formatiranja ili konverzije. One nam omogućavaju sljedeće prednosti:

- Fleksibilnost. Mogućnost uključivanja sadržaja koji nije prilagođen Jupyter formatu.
- Prilagođeni procesi. Korisno za specifične alate za konverziju, koji zahtijevaju sirove podatke.
- Čuvanje sadržaja. Omogućava unos sadržaja, koji se ne smije mijenjati ili obrađivati tokom rada u Jupyter-u.

2.5.3. Osnovne komande i prečice u JupyterLab-u

Efikasan rad u JupyterLab-u zahtijeva poznавanje osnovnih komandi i prečica, koje značajno ubrzavaju radni proces i olakšavaju navigaciju unutar radnih svesaka. U ovom poglavlju biće detaljno obrađene ključne komande i prečice koje se koriste tokom rada.

2.5.3.1. Izvršavanje ćelija

Izvršavanje ćelija je osnovna operacija u JupyterLab-u, a sljedeće prečice omogućavaju različite pristupe:

- **Shift + Enter** – Izvršava trenutnu ćeliju i prelazi na sljedeću. Ako sljedeća ćelija ne postoji, automatski se kreira nova.
- **Ctrl + Enter** – Izvršava trenutnu ćeliju, ali zadržava fokus na njoj bez prelaska dalje.
- **Alt + Enter** – Izvršava trenutnu ćeliju i umeće novu ćeliju ispod nje.

Ove prečice su idealne za iterativan rad, omogućavajući korisnicima da brzo testiraju kod ili modifikuju sadržaj ćelija. Vizualni prikaz tastature sa označenim tipkama može biti koristan za bolje pamćenje ovih prečica.

2.5.3.2. Dodavanje i brisanje ćelija

Rad sa ćelijama često zahtijeva dodavanje novih ili brisanje postojećih ćelija:

- **Dodavanje nove ćelije:**
 - Tipka **B** – U komandnom načinu umetanje ćelije ispod trenutne.
 - Tipka **A** – Umetanje ćelije iznad trenutne.
- **Brisanje ćelije:**
 - Tipka **D, D** – Pritisnite tipku **D** dva puta zaredom u komandnom načinu da biste obrisali trenutnu ćeliju.

Ove prečice omogućavaju brzu manipulaciju ćelijama, čineći proces uređivanja radne sveske jednostavnijim i efikasnijim.

2.5.3.3. Prebacivanje između tipova ćelija

Tip ćelije može se lako promijeniti pomoću sljedećih prečica:

- Tipka **Y** – Postavlja trenutnu ćeliju kao ćeliju za kod.
- Tipka **M** – Postavlja trenutnu ćeliju kao Markdown ćeliju.

Ove opcije omogućavaju prilagodđavanje radne sveske, čineći je preglednijom i prilagođenom različitim potrebama, poput dokumentovanja i izvršavanja koda.

2.5.3.4. Prelazak između načina uređivanja i komandi

Poznavanje osnovnih komandi i prečica u JupyterLab-u ne samo da ubrzava rad već i omogućava bolju organizaciju i produktivnost. Tokom rada na složenijim projektima, ove prečice postaju neophodan alat za efikasno upravljanje radnim sveskama.

JupyterLab ima dva načina rada:

- **Edit Mode (Način uređivanja):**
 - Omogućava uređivanje sadržaja ćelije.
 - Aktivira se klikom unutar ćelije ili pritiskom na **Enter**.
 - Indikator je zeleni okvir oko ćelije.
- **Command Mode (Način komandi):**
 - Omogućava izvršavanje komandi na nivou ćelije (kopiranje, lijepljenje, brisanje).
 - Aktivira se pritiskom na **Esc** ili klikom na sivu marginu lijevo od ćelije.
 - Indikator je plavi okvir oko ćelije.

Vizualni prikaz razlike između Edit i Command moda može dodatno pomoći u razumijevanju ove funkcionalnosti, jer jasno ilustruje kako se prelazi iz jednog načina rada u drugi.

2.5.4. Savjeti za efikasan rad u JupyterLab-u

Efikasan rad s JupyterLab-om podrazumijeva dobru organizaciju sadržaja, korištenje prečica, pravilno upravljanje projektima i implementaciju dodatnih alata. U nastavku su savjeti koji će vam pomoći da maksimalno iskoristite mogućnosti JupyterLab-a.

a) Korištenje naslova i podnaslova:

- Organizovati rad pomoću naslova u Markdown ćelijama, što olakšava navigaciju i razumijevanje strukture dokumenta.
- Dodati **Table of Contents** ekstenziju koja automatski generiše sadržaj na osnovu naslova i podnaslova, što je posebno korisno kod većih projekata.

b) Grupisanje ćelija:

- Grupisanje povezanih ćelija pomaže u boljoj organizaciji, posebno kod obimnijih radnih svesaka. Na taj način se može logički podijeliti analiza na cjeline, koje se lako prepoznaju i prate. Grupisanje povezanih ćelija podrazumjeva umetanje markdown ćelija između ćelija sa kodom, odvajajući ih u logičke cjeline.

c) Čuvanje promjena:

- Iako JupyterLab automatski čuva promjene, preporučljivo je ručno sačuvati rad pritiskom na **Ctrl + S**.
- Indikator sačuvanih promjena nalazi se u naslovnoj traci kao ikonica diska.

d) Eksportovanje radne sveske:

- Bilježnicu možete eksportovati u različite formate putem opcije **File > Export Notebook As....**
- Dostupni formati uključuju **HTML**, **PDF**, **Markdown**, pa čak i slajdove za prezentacije.
- Vizualni prikaz menija za eksportovanje može dodatno olakšati razumijevanje procesa.

e) Korištenje verzionog sistema:

- Za praćenje promjena i saradnju, preporučuje se korištenje sistema za kontrolu verzija, poput **Git-a**. Na taj način se osigurava povratak na prethodne verzije i efikasno dijeljenje rada s timom.

f) Korištenje prečica:

- U **Command Palette-u** možete pronaći listu svih dostupnih prečica, a takođe ih možete prilagoditi prema svojim potrebama, što značajno ubrzava radni proces.

g) Dodavanje ekstenzija:

- JupyterLab podržava veliki broj ekstenzija koje proširuju funkcionalnosti, uključujući dodatne teme, integraciju s Git-om, napredne alate za vizualizaciju i mnoge druge korisne dodatke.

h) Čišćenje izlaza prije dijeljenja:

- Prije dijeljenja radne sveske, uklonite sve izlaze putem opcije **Edit > Clear All Outputs**. Ovo smanjuje veličinu fajla i izbjegava moguće konflikte prilikom otvaranja na drugim uređajima.

Prateći ove savjete, značajno se poboljšava efikasnost rada u JupyterLab-u, omogućava bolju organizaciju svojih projekata i olakšava dijeljenje radnih svesaka s kolegama. Korištenjem dodatnih alata i ekstenzija, JupyterLab postaje još moćniji alat za analizu podataka, dokumentovanje i vizualizaciju.

2.6. Prvi koraci sa Python-om u Jupyter-u

U prethodnim poglavljima upoznali smo se sa osnovama Jupyter okruženja, kreiranjem i organizacijom bilježnica, te smo stekli uvid u različite tipove ćelija i načine efikasnog rada. Sada je vrijeme da započнемo konkretni rad sa Python-om u Jupyter-u. U ovom poglavlju fokusiraćemo se na osnovne koncepte programiranja u Python-u, što uključuje pisanje jednostavnih izraza, izvođenje matematičkih operacija, prikaz rezultata korištenjem funkcije `print()`, rad sa komentarima i razumijevanje poruka o greškama. Uz to, prikazaćemo i praktične vježbe, koje će vam pomoći da primijenite naučeno u kontekstu ekonomskih zadataka.

2.6.1. Pisanje jednostavnih izraza

Jedna od ključnih prednosti Python-a je njegova jednostavna i intuitivna sintaksa. Čak i ako prethodno niste programirali, osnovne operacije u Python-u možete savladati veoma brzo.

Matematičke operacije

Python podržava osnovne matematičke operacije bez potrebe za dodatnim bibliotekama:

- Sabiranje (+), oduzimanje (-), množenje (*) i dijeljenje (/)
- Cjelobrojno dijeljenje (//), stepenovanje (**) i ostatak pri dijeljenju (%)

```
# Sabiranje i množenje
x = 5 + 3
y = x * 2
print("Vrijednost y je:", y)

# Stepenovanje i cjelobrojno dijeljenje
a = 10
b = a ** 2    # a na kvadrat
c = a // 3    # cjelobrojno dijeljenje a sa 3
print("b =", b, "c =", c)
```

```
Vrijednost y je: 16
b = 100 c = 3
```

Ovi primjeri prikazuju kako se jednostavni izrazi mogu pisati i izvršavati direktno u ćeliji za kod u Jupyter-u. Nakon izvršavanja, rezultati se odmah prikazuju ispod ćelije, što vam omogućava brzu provjeru ispravnosti koda.

Korištenje funkcije `print()`

Funkcija `print()` u Python-u služi za ispis vrijednosti u izlaz. Ovo je osnovni alat za provjeru rezultata i praćenje toka programa.

```
rezultat = 25 * 4
print("Rezultat množenja je:", rezultat)

Rezultat množenja je: 100
```

Ovo će prikazati tekst i vrijednost varijable `rezultat` odmah ispod ćelije. Pomoću `print()` možete ispisivati kombinacije teksta i varijabli, što olakšava interpretaciju rezultata, kao što je to prikazano u prethodnom primjeru (`print("b=",b,"c=",c)`).

2.6.2. Rad sa komentarima

Komentari su dijelovi koda koje Python interpreter preskače prilikom izvršavanja i služe isključivo programerima i onima koji čitaju kod. Dobro komentarisanje olakšava razumijevanje logike, objašnjava svrhu određenih dijelova koda i pomaže u vršenju izmjena i održavanju projekata.

U Python-u se komentari označavaju znakom `#`. Sve što se nađe iza ovog znaka u istoj liniji smatra se komentarom.

U primjeru, komentari objašnjavaju značenje varijabli i korake koji se izvode. Ovo je naročito važno u složenijim programima i ekonomskim analizama, gdje je potrebno jasno dokumentovati razloge i način računanja određenih veličina.

```
# Ova linija je komentar i Python je neće izvršiti
x = 10 # x predstavlja količinu robe
y = 20 # y predstavlja cijenu jedinice
ukupno = x * y # izračunavamo ukupni iznos
print(ukupno)
```

2.6.3. Upoznavanje sa greškama

Prilikom programiranja neminovno dolazi do grešaka. Greške nisu nešto čega se treba plašiti. Naprotiv, one pružaju priliku da bolje razumijete kod i naučite nove koncepte.

Python generiše detaljne poruke o greškama, koje ukazuju na mjesto gdje je došlo do problema i često sugeriraju o kakvoj je grešci riječ. Najčešće kategorije grešaka su:

- **SyntaxError** – Greška u sintaksi koda (izostavljeni ili pogrešno slovo pri pisanju neke naredbe ili izostavljeni zagradni znak).
- **NameError** – Pozivanje varijable ili funkcije koja nije definisana.
- **TypeError** – Operacija izvedena na nekompatibilnim tipovima podataka (npr. sabiranje stringa i broja).
- **ValueError** – Neodgovarajući tip vrijednosti za određenu operaciju (npr. konverzija stringa "abc" u broj).

- ***ZeroDivisionError*** – Pokušaj dijeljenja s nulom.

```
x = 10
y = 0
rez = x / y # Ovdje će doći do ZeroDivisionError-a

-----
ZeroDivisionError          Traceback (most recent call last)
Cell In[6], line 3
  1 x = 10
  2 y = 0
----> 3 rez = x / y # Ovdje će doći do ZeroDivisionError-a
ZeroDivisionError: division by zero
```

Jupyter će ispisati poruku o grešci ispod ćelije, ukazujući na problematičnu liniju (ispisujući broj linije – line 3) i tip greške. Čitanjem i razumijevanjem ovih poruka možete brzo locirati i ispraviti grešku.

Osnovne tehnike za ispravljanje grešaka

- **Provjera sintakse** – Provjeriti da su komande ispravno napisane da su zgrade, dvotačke i navodnici pravilno zatvoreni.
- **Provjera imena varijabli** – Uvijek provjerite da li varijabla postoji prije nego što je koristite i da li ste pravilno upisali njeno ime (Python razlikuje velika i mala slova).
- **Tipovi podataka** – Razmislite o tipovima podataka uključenim u operaciju. Ako dobijete TypeError, možda sabirate broj i string umjesto dva broja.
- **Logička provjera** – Ako nema sintaksnih i tipnih grešaka, razmislite o logici koda i da li su logički uslovi dobro napisani. Da li su variable inicijalizovane? Da li je redoslijed operacija ispravan?

2.6.4. Praktične vježbe

Kako biste što bolje savladali osnove Pythona, predstavljamo nekoliko vježbi koje su relevantne za ekonomski koncepte i proračune.

Vježba 1. Izračunavanje jednostavnih finansijskih formula

Prepostavimo da imate iznos novca **P** i godišnju kamatnu stopu **r** u procentima. Želite izračunati iznos novca nakon jedne godine, ako se kamata obračunava jednom godišnje.

Formula za to je:

$$[A = P \times \left(1 + \frac{r}{100}\right)]$$

Primjer koda koji rješava postavljeni problem je prikazan u nastavku.

```
P = 1000 # početni iznos
r = 5    # godišnja kamatna stopa u %

A = P * (1 + r/100)
print("Iznos nakon jedne godine je:", A)

Iznos nakon jedne godine je: 1050.0
```

Isprobajte različite vrijednosti **P** i **r** da vidite kako se iznos mijenja.

Vježba 2. Pisanje kratkih programa koji rješavaju ekonomski probleme

Možete pisati kratke programe koji izračunavaju, na primjer, mjesecnu ratu kredita ili vrijednost investicije nakon određenog broja godina.

Mjesecna rata kredita po formuli za anuitetno otplaćivanje:

$$[R = P \frac{r(1+r)^n}{(1+r)^n - 1}]$$

Gdje je:

- P – iznos kredita
- r – mjesecna kamatna stopa (npr. godišnja stopa podijeljena sa 12)
- n – broj mjeseci otplate

```
P = 20000 # Iznos kredita
god_r = 6 # Godišnja kamatna stopa u %
r = (god_r / 100) / 12 # Mjesecna kamatna stopa
n = 60 # Broj mjeseci otplate (5 godina)

R = P * (r * (1 + r)**n) / ((1 + r)**n - 1)
print("Mjesecna rata iznosi:", R)
```

```
Mjesecna rata iznosi: 386.6560305885654
```

Ovakve praktične vježbe omogućavaju vam da odmah primijenite naučeno u kontekstu ekonomskih problema. Isprobajte različite vrijednosti i scenarije (različite kamatne stope, trajanja otplate, iznose kredita) kako bi stekli osjećaj za dinamiku ovih formula.

Ovo su temeljne vještine koje će biti dragocjene u daljim koracima, bilo da se bavite naprednjim matematičkim modeliranjem, statističkom analizom ili ekonomskim simulacijama. Kako nastavite s učenjem, ove osnove će vam omogućiti da se brže i efikasnije snalazite u kompleksnijim izazovima.

2.7. Dodatni resursi i naredni koraci

U prethodnim poglavljima prikazane su osnove za rad s Python-om i Jupyter okruženjem. Opisano je kako instalirati i pokrenuti Jupyter Notebook i JupyterLab, organizovati radno okruženje, koristiti osnovne komande i raditi sa različitim tipovima celija. Upoznali smo se i sa osnovama programiranja u Python-u, uključujući pisanje jednostavnih izraza, korištenje funkcije `print()`, rad s komentarima i razumijevanje uobičajenih poruka o greškama.

Sada je vrijeme da se prošire znanje i upoznaju dodatni resursi, koji mogu pomoći u daljem razvoju programskih i analitičkih vještina. Takođe ćemo se ukratko osvrnuti na naredna poglavja i dati smjernice o tome kako se pripremiti za naprednije koncepte.

Korištenjem zvanične dokumentacije, online kurseva, forumskih zajednica i drugih online dokumenata, moguće je značajno unaprijediti svoje znanje i vještine. U nastavku ćemo navesti samo neke osnovne izvore gdje možete tražiti odgovore na vaša pitanja i nedoumice.

Zvanična Python dokumentacija

Jedan od najpouzdanijih izvora informacija je zvanična Python dokumentacija, dostupna na <https://docs.python.org/3/>. Ovdje možete pronaći:

- Detaljan opis svih ugrađenih funkcija, tipova podataka i modula.
- Primjere upotrebe i kratke tutorijale.
- Vodiče za napredne teme poput asinhronog programiranja, upravljanja memorijom i sigurnosnih aspekata koda.

Redovno korišćenje zvanične dokumentacije pomoći će vam da razumijete specifične aspekte jezika i riješite nedoumice bez oslanjanja isključivo na sekundarne izvore.

Tutorijali i kursevi prilagođeni ekonomistima

Budući da je cilj ovog udžbenika da vas upozna s Python-om u kontekstu ekonomije, korisno je potražiti tutorijale i kurseve posebno usmjereni na ekonomske i finansijske analize. Neki od preporučenih izvora uključuju:

- **MOOC platforme** (npr. Coursera, edX, Udemy): Kursevi koji se fokusiraju na analizu podataka, ekonometriju ili mašinsko učenje u ekonomiji često koriste Python i nude praktične primjere.
- **Specijalizirani blogovi i web-stranice:** Mnogi ekonomski analitičari i istraživači dijele svoje projekte i code snippet-e, objašnjavajući kako analizirati vremenske serije, modelirati tržišta ili raditi sa velikim setovima podataka.

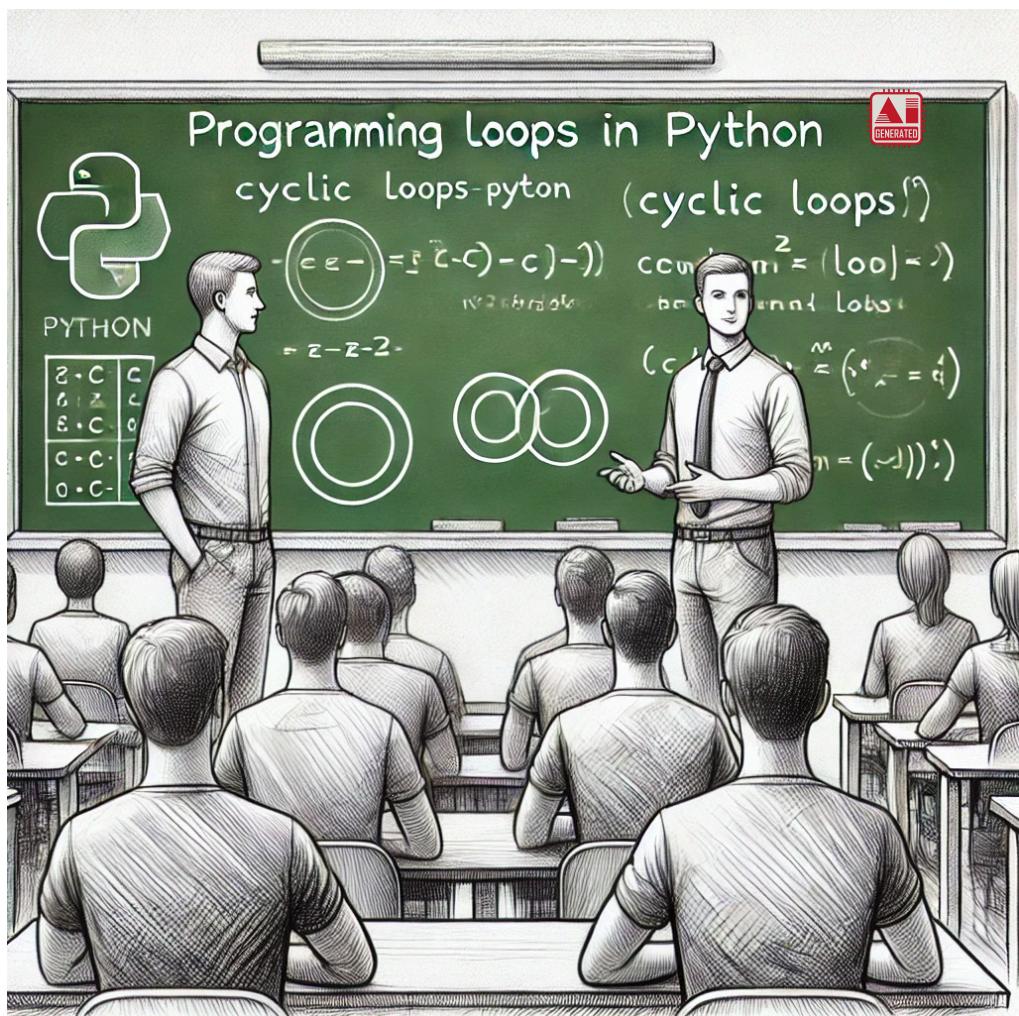
Posjetite i akademske resurse, koji nude radne bilježnice (notebooke) sa primjerima ekonomskih analiza, često dostupne uz istraživačke radove ili prateće materijale za udžbenike u ekonometriji i finansijama.

Forumi poput Stack Overflow-a

Ukoliko nađete na problem tokom programiranja ili analize, *Stack Overflow* (<https://stackoverflow.com/>) je jedna od najkorisnijih zajednica za dobijanje pomoći. Ovo je platforma gdje programeri svih nivoa postavljaju pitanja i dijele rješenja. Postoji velika vjerovatnoća da je neko već postavio pitanje slično vašem, a ako nije, možete kreirati novo pitanje uz jasan opis problema i djelić koda.

Pretraživanje ključnih pojmoveva i poruka o greškama koje dobijete u Python-u ili Jupyter-u često će vas dovesti do Stack Overflow-a, gdje ćete pronaći detaljna objašnjenja i rješenja.

3. OSNOVNI KONCEPTI



"Everything should be made as simple as possible, but not simpler."
"Sve treba pojednostaviti koliko je moguće, ali ne više od toga."

— Albert Einstein¹²

¹² Albert Ajnštajn (1879–1955) jedan od najznačajnijih naučnika 20. vijeka. Ajnštajn je uvijek isticao važnost jednostavnosti i jasnog razumevanja, bez nepotrebnog pojednostavljivanja složenih pojava.

Ciljevi ovog poglavlja

U ovom poglavlju želimo da ekonomski koncepte i analize koje su posmatrane putem tabelarnih i matematičkih prikaza, povežemo sa osnovnim elementima programiranja u Python-u. Osnovni ciljevi ovog poglavlja su:

- Upoznavanje s tipovima podataka: Razumijevanje ključnih tipova podataka (brojevi, tekst, logičke vrijednosti), koji su neophodni za predstavljanje ekonomskih veličina (npr. cijene, količine, datumi, kategorije) i njihovo pravilno tumačenje u Python-u.
- Varijable i organizacija podataka: Shvatiti kako se podaci mogu čuvati u varijablama radi daljeg računanja i manipulacije, te kako takav pristup olakšava rad s ekonomskim pokazateljima, indeksima i finansijskim indikatorima.
- Operatori i izrazi: Savladati osnovne matematičke (+, -, * i /) i logičke (=, >, <, >=, <= i < >) operatore, kako bi mogli efikasno izvoditi računanja nad ekonomskim podacima, od jednostavnih formula do složenijih finansijskih modela.
- Kontrolne strukture (naredbe grananja i petlje): Naučiti kako donijeti odluku u programu (kako pokrenuti određene dijelove koda, kada se prede određena zadata vrijednost na finansijskom tržištu) i kako automatizovati ponavljanje određenih radnji (ponavljanje kroz redove podataka ili godine posmatranja).
- Funkcije i moduli: Razumjeti koncept grupisanja koda u funkcije za ponovnu upotrebu, što omogućava lakše održavanje i proširivanje analiza. Upoznavanje s modulima i bibliotekama (tj. gotovim funkcijama i paketima) omogućava da se brzo iskoristite postojeći alat za složenije ekonomski proračune, bez potrebe da se logika piše od nule.
- Osnovni tipovi fajlova: Savladati rad sa ključnim formatima za skladištenje i razmjenu podataka, kao što su CSV, Excel, JSON i YAML, kako bi se efikasno upravljalo obradama podataka. Cilj je naučiti kako čitati i pisati podatke u ove formate koristeći odgovarajuće alate, te razumjeti njihove prednosti i nedostatke u kontekstu analize podataka i izrade finansijskih izvještaja.

Ovi ciljevi zajedno treba da omoguće da s lakoćom koristite programske strukture i time unaprijedite analitičke sposobnosti prilikom rada sa podacima i ekonomskim modelima.

3.1. Tipovi podataka

Kada započinjemo rad sa Python-om, jedan od prvih koraka je razumjeti osnovne tipove podataka s kojima ćemo raditi. Tipovi podataka određuju na koji način računar čuva i manipuliše informacijama, a izbor pravilnog tipa podataka uveliko olakšava analizu i obradu ekonomskih veličina. U ekonomiji se često susrećemo s numeričkim vrijednostima kao što su cijene, kamatne stope, količine, stopa inflacije, ali i sa tekstualnim podacima, kao i s logičkim uslovima (npr. da li je neka promjena veća ili manja od očekivane vrijednosti).

Razumijevanje tipova podataka i njihovih karakteristika ključan je preduslov za kasnije analize, bilo da se bavimo jednostavnim proračunima ili kompleksnim statističkim i ekonometrijskim modelima.

Kada govorimo o tipovima podataka u Python-u, možemo ih razvrstati u dvije glavne kategorije: **prosti** i **složeni** tipovi podataka. Ova podjela nam pomaže da bolje razumijemo kako Python obraduje i organizuje informacije, što je ključno za efikasnu analizu podataka.

Prosti tipovi podataka

Prosti tipovi podataka predstavljaju osnovne strukture u programiranju. Oni su direktno povezani s načinom na koji računar čuva i obrađuje osnovne vrijednosti. Prosti tipovi podataka su osnovni i ne sastoje se od drugih tipova. Oni su ključni za izvođenje osnovnih matematičkih i logičkih operacija.

U Python-u, prosti tipovi podataka uključuju:

- **Cijele brojeve (integers)** – Koriste se za predstavljanje cijelih vrijednosti, poput količina ili godina.
 - Primjer: `5, 2023`
- **Decimalni brojevi (floating-point)** – Koriste se za predstavljanje vrijednosti s decimalama, kao što su kamatne stope ili inflacija.
 - Primjer: `3.14, 0.05`
- **Tekstualne vrijednosti (strings)** – Koriste se za čuvanje tekstualnih podataka poput imena kompanija ili proizvoda.
 - Primjer: `"Ekonomija", "USD"`
- **Logičke vrijednosti (booleans)** – Koriste se za prikaz rezultata logičkih uslova (`True` ili `False`), kao što je provjera da li je neka vrijednost veća od određene granice.
 - Primjer: `True, False`

Složeni tipovi podataka

Složeni tipovi podataka omogućavaju čuvanje i organizaciju više vrijednosti u jednoj strukturi. U Python-u, najčešće korišćeni složeni tipovi uključuju:

- **Liste (lists)** – Lista vrijednosti koja može da sadrži različite tipove podataka u kojoj je moguće mijenjanje podataka.
 - Primjer: `[1, 2, 3], ["USD", "EUR", "JPY"]`
- **n-Torke¹³ (tuples)** – Kolekcije vrijednosti koje su nepromjenjive nakon kreiranja, a mogu se sastojati od različitih tipova podataka.
 - Primjer: `(1, "USD", 3.14)`
- **Rječnici (dictionaries)** – Strukture podataka koje povezuju ključeve sa konkretnim vrijednostima, korisne za organizovanje podataka.
 - Primjer: `{"valuta": "USD", "vrijednost": 100}`
- **Skupovi (sets)** – Skupovi jedinstvenih vrijednosti, korisni za rad s podacima bez duplikata, gdje je redoslijed podataka proizvoljan.
 - Primjer: `{"USD", "EUR", "JPY"}`

Složeni tipovi omogućavaju rad s većim i kompleksnijim strukturama podataka, poput analiza velikih setova podataka ili rada sa strukturiranim informacijama.

Razlike između prostih i složenih tipova podataka

Struktura: Prosti tipovi podataka čuvaju pojedinačne vrijednosti, poput brojeva ili teksta i koriste se za jednostavne operacije i proračune. Nasuprot tome, složeni tipovi podataka omogućavaju čuvanje više vrijednosti, koje mogu biti iste ili različite vrste, čineći ih pogodnim za organizaciju i obradu složenijih podataka.

Fleksibilnost: Prosti tipovi podataka su brži i jednostavniji za obradu, što ih čini idealnim za osnovne operacije i proračune. S druge strane, složeni tipovi omogućavaju rad s grupama podataka i nalaze primjenu u složenijim analizama i strukturiranim zadacima.

Primjena: Prosti tipovi podataka koriste se za osnovne proračune i logičke uslove, gdje je jednostavnost ključna za efikasno rješavanje zadataka. Nasuprot tome, složeni tipovi služe za preuzimanje podataka iz tabela ili baza podataka, omogućavajući rad s većim i kompleksnijim setovima podataka.

Razumijevanje razlike između prostih i složenih tipova podataka omogućava vam da odaberete odgovarajući tip za svaku situaciju.

3.1.1. Integer (cjelobrojne vrijednosti)

Integer ili skraćeno **int**, predstavljaju cjelobrojne vrijednosti, bez decimalnog dijela. U ekonomskim primjenama cjelobrojne vrijednosti se često koriste za predstavljanje diskretnih veličina, npr.:

¹³ U matematici, n-torka (engl. tuple) je konačan niz (također poznat kao "uređena lista") od n objekata, svaki od kojih je specificiranog tipa. Izvor: <https://realpython.com/python-tuple/>

- **Količina proizvoda** – Koliko jedinica robe je proizvedeno ili prodato (npr. 100 jedinica automobila).
- **Godine ili periodi** – Predstavljanje godine (npr. 2023) ili broja perioda u nekom modelu (npr. 12 mjeseci).
- **Brojač iteracija** – Korištenje cijelobrojne vrijednosti kao brojača prilikom ponavljanja operacija (npr. iteracija kroz 10 različitih kompanija).

Primjer:

```
godina = 2023
kolicina = 350
print("Godina:", godina, "Količina proizvedenih jedinica:", kolicina)
```

```
Godina: 2023 Količina proizvedenih jedinica: 350
```

Integeri se koriste kada nam decimalni dio nije potreban, ili kada bi on bio nelogičan (nije moguće imati 3.7 prodatih jedinica proizvoda).

3.1.2. Float (realni brojevi s decimalnim dijelom)

Float tipovi predstavljaju realne brojeve, koji se sastoje od cijelog i decimalnog dijela, odvojenih decimalnim simbolom, koji je u Pythonu tačka (.). Ovi podaci su od suštinskog značaja u ekonomskoj analizi, jer većina pokazatelja nije cijelobrojna.

- **Cijene i kamatne stope** – Cijena jednog proizvoda može biti 15.99, kamatna stopa 5.25%.
- **Vrijednosti indeksa i pokazatelja** – Berzanski indeks može imati vrijednost 3500.75, dok inflacija može biti 2.3%.
- **Rezultati statističkih proračuna** – Prosječna vrijednost određene veličine ili standardna devijacija rijetko će biti cijelobrojna.

Primjer:

```
cijena = 15.99
kamatna_stopa = 5.25
print("Cijena proizvoda:", cijena, "Kamatna stopa:", kamatna_stopa, "%")
```

```
Cijena proizvoda: 15.99 Kamatna stopa: 5.25 %
```

Korištenje float tipa podataka omogućavamo preciznije računanje i analizu. Ipak, treba imati u vidu da je ovaj tip podataka podložan zaokruživanjima. To može da dovede do greške, kada se sabiraju već zaokružene numeričke vrijednosti. Ovo je posebno važno u finansijskim izvještajima, gdje se traži poklapanje svih rezulata.

3.1.3. String (niz znakova)

String je tip podatka koji predstavlja niz znakova, obično korišten za tekstualne podatke. U ekonomiji, imena kompanija, šifre proizvoda, nazivi zemalja ili valuta, sve se to može čuvati u stringovima.

Primjeri:

- **Identifikatori kompanija:** "AAPL" za Apple, "GOOGL" za Google.
- **Nazivi zemalja i valuta:** "Bosna i Hercegovina", "USD", "EUR".
- **Kategorije proizvoda ili usluga:** "Elektronika", "Prevoz".

```
ime_kompanije = "Apple Inc."
valuta = "USD"
print("Kompanija:", ime_kompanije, "Valuta:", valuta)
```

```
Kompanija: Apple Inc. Valuta: USD
```

3.1.4. Bool (logička vrijednost)

Bool ili logički tip podatka može imati samo dvije vrijednosti: **True** (tačno) ili **False** (netačno). Ovaj tip se obično koristi u uslovnim strukturama koda i kontrolnim mehanizmima.

Primjer:

- **Provjera uslova:** Da li je prodaja u ovom kvartalu bila veća od one u prethodnom kvartalu?
- **Signalizacija odobrenja ili neuspjeha transakcije:** Da li je transakcija validna (True) ili nije (False)?

```
prodaja_prethodni = 1000
prodaja_tekuci = 1200
rast_prodaje = prodaja_tekuci > prodaja_prethodni # Rezultat poređenja može biti
True ili False
print("Da li je došlo do rasta prodaje?", rast_prodaje)
```

```
Da li je došlo do rasta prodaje? True
```

Logičke vrijednosti omogućavaju programima da "donose odluke" i usmjeravaju tok izvršavanja programa.

3.1.5. Konverzija prostih tipova podataka

U ekonomskoj analizi, podaci mogu dolaziti iz različitih izvora i u različitim formatima. Neki brojevi mogu biti predstavljeni kao stringovi (npr. "1000.50"), ali se moraju pretvoriti u numerički format (float) da bi se mogle izvršiti aritmetičke operacije, odnosno uvrstiti u neku matematičku formulu.

Metode konverzije:

- `int(x)` konvertovanje promjenjive `x` u cijelobrojnu vrijednost.
- `float(x)` konvertovanje promjenjive `x` u realnu vrijednost s decimalnim dijelom.
- `str(x)` konvertovanje promjenjive `x` u string.

Primjer:

```
vrijednost_string = "1000.50"
vrijednost_float = float(vrijednost_string) # Pretvaranje stringa u float
vrijednost_int = int(float(vrijednost_string)) # Pretvaranje float u int
vrijednost_string2 = str(vrijednost_float + 1.234) # Dodavanje vrijednosti i
# pretvaranje u string

print("Cijela vrijednost:", vrijednost_int)
print("Vrijednost kao float:", vrijednost_float)
print("Ispis teksta:", vrijednost_string2)
```

```
Cijela vrijednost: 1000
Vrijednost kao float: 1000.5
Ispis teksta: 1001.734
```

Konverzija je česta potreba u analizi podataka. Na primjer, možete učitati podatke iz CSV¹⁴ (comma-separated values) fajla gdje su sva polja stringovi, pa ćete ih morati konvertovati u float ili int prije statističkih proračuna ili izgradnje modela.

3.1.6. Liste (lists)

Liste predstavljaju uredene nizove, koji mogu sadržavati različite tipove podataka i čija se dužina može mijenjati. Moguće je dodavati nove elemente, mijenjati postojeće ili brisati one koji više nisu potrebni.

Primjer iz ekonomije: Zamislite da imate listu valuta u kojima poslujete ili niz vrijednosti berzanskog indeksa tokom određenog perioda.

¹⁴ CSV (Comma-Separated Values) format predstavlja način čuvanja tabelarnih podataka u tekstualnom obliku, gdje su polja (kolone) medusobno odvojena zapetom, a pojedinačni zapisi (redovi) razdvojeni prelaskom u novi red. Izvor: <https://www.ietf.org/rfc/rfc4180.txt>

```

valute = ["USD", "EUR", "JPY"]
cijene_akcija = [100.5, 101.2, 99.8, 102.0]

print("Valute:", valute)
print("Cijene akcija:", cijene_akcija)

```

```

Valute: ['USD', 'EUR', 'JPY']
Cijene akcija: [100.5, 101.2, 99.8, 102.0]

```

Liste su fleksibilne i omogućavaju pretraživanje, sortiranje i filtriranje vrijednosti. Na primjer, možete dodati novu valutu, ukloniti staru ili izračunati prosječnu cijenu akcije iz liste cijena.

```

valute = ["USD", "EUR", "JPY"]
cijene_akcija = [100.5, 101.2, 99.8, 102.0]
# Dodavanje nove valute
valute.append("BAM")
print("Valute nakon dodavanja BAM:", valute)
# Brisanje valute JPY
valute.remove("JPY")
print("Valute nakon brisanja JPY:", valute)
# Računanje srednje vrijednosti cijena akcija
projek_cijena = sum(cijene_akcija) / len(cijene_akcija)
print("Prosječna cijena akcija:", projek_cijena)

```

```

Valute nakon dodavanja BAM: ['USD', 'EUR', 'JPY', 'BAM']
Valute nakon brisanja JPY: ['USD', 'EUR', 'BAM']
Prosječna cijena akcija: 100.875

```

3.1.7. n-Torke (tuples)

n-Torke (eng. tuples) je složen tip sličan listama jer predstavlja niz elemenata, ali nakon kreiranja ovog tipa podatka nije moguće mijenjati njegove vrijednosti. Ovo je korisno kada želite biti sigurni da se neki podaci neće slučajno promijeniti tokom izvršavanja programa.

Primjer iz ekonomije: Možete koristiti n-torku za čuvanje jedne mjerne veličine sa više karakteristika. Recimo, možete čuvati podatak o transakciji kao n-torku ([iznos](#), [valuta](#), [datum](#)). Pošto je n-torka nepromjenjiva, ovi podaci se neće izmijeniti kroz program.

Primjer:

```

transakcija = (1000, "USD", "2023-05-10")
print("Transakcija:", transakcija)

```

```

Transakcija: (1000, 'USD', '2023-05-10')

```

3.1.8. Rječnici (dictionaries)

Rječnici predstavljaju strukture, koje povezuju ključeve sa vrijednostima. Ovo je izuzetno korisno kada imate podatke kojima želite pristupiti po nekom nazivu ili oznaci, umjesto po numeričkom indeksu.

Primjer iz ekonomije: Možete čuvati informacije o određenoj valuti, kao što su njeni ime, simbol, trenutna vrijednost ili referentni kurs. Umjesto da pamtite niz nejasnih indeksa, možete koristite ključeve za pristup vrijednostima.

```
info_o_valutu = {  
    "naziv": "Američki dolar",  
    "simbol": "USD",  
    "kurs_u_KM": 1.85  
}  
  
print("Naziv valute:", info_o_valutu["naziv"])  
print("Kurs u KM:", info_o_valutu["kurs_u_KM"])
```

```
Naziv valute: Američki dolar  
Kurs u KM: 1.85
```

Rječnici olakšavaju rad kada želite brzo pristupiti podacima po nazivu, a ne želite pamtitи poziciju vrijednosti u listi. Korisni su za kreiranje mapa valuta i njihovih vrijednosti, mapiranje šifri proizvoda i cijena, ili bilježenje rezultata ekonomskih indikatora po nazivu.

3.1.9. Skupovi (sets)

Skupovi (eng. sets) su nizovi jedinstvenih vrijednosti. Oni vam omogućavaju da brzo provjerite da li neka vrijednost postoji u skupu ili da eliminišete duplike.

Primjer iz ekonomije: Ako imate listu valuta u kojoj se neka valuta ponavlja više puta, skup vam može pomoći da dobijete skup jedinstvenih valuta s kojima radite.

```
valute_lista = ["USD", "EUR", "EUR", "JPY", "USD"]  
valute_skup = set(valute_lista)  
  
print("Skup jedinstvenih valuta:", valute_skup)  
  
Skup jedinstvenih valuta: {'USD', 'EUR', 'JPY'}
```

Skupovi su posebno korisni kada treba da identifikujete jedinstvene kategorije proizvoda, jedinstvene kompanije u datasetu ili bilo koju situaciju gdje ponavljajuće vrijednosti treba svesti na jedinstveni niz podataka.

3.2. Varijable, operatori i izrazi

Nakon što su definisani i objašnjeni osnovni tipovi podataka, sljedeći logički korak je razumijevanje načina njihove praktične primjene i manipulacije unutar programa. Ključni elementi ove manipulacije uključuju **variabile**, **operatori** i **izraze**. Varijable predstavljaju imenovane memoriske lokacije, koje omogućavaju čuvanje podataka radi kasnijeg pristupa i obrade. Operatori omogućavaju izvođenje različitih računskih, logičkih i drugih operacija nad podacima, dok izrazi predstavljaju kombinaciju varijabli, operatora i konkretnih vrijednosti, čime se generišu nove vrijednosti ili rezultati.

Razumijevanje ovih koncepata je od suštinskog značaja za primjenu u ekonomskoj analizi. Bez obzira na to da li je cilj izračunavanje profita na osnovu prihoda i troškova, određivanje stope rasta između dva vremenska perioda, ili računanje finansijskih pokazatelja poput marže, sposobnost manipulacije varijablama, poređenja vrijednosti i izvođenja operacija neophodna je za postizanje tačnih i korisnih rezultata.

3.2.1. Definisanje i korištenje varijabli

Varijable predstavljaju imenovane memoriske lokacije za čuvanje vrijednosti u programskom okruženju. Možemo ih zamisliti kao naljepnice s imenom, koje su povezane s određenom lokacijom u memoriji, gdje se nalazi vrijednost (npr. broj, string, lista ili drugi tip podataka). Kada se u programu pozovete na ime variabile, pristupite vrijednosti koja je u tom trenutku pohranjena u povezanoj memoriskoj lokaciji, omogućavajući manipulaciju i analizu podataka na jednostavan i organizovan način. Vrijednost smještena u varijabli ima dinamički karakter i može se tokom programa mijenjati.

Dodjela vrijednosti varijablama

U Python-u, varijable se definišu jednostavno dodjeljivanjem vrijednosti pomoću operatora **“=”**, kao u sljedećem primjeru:

```
profit = 10000
stopa_rasta = 0.05
naziv_comp = "SED TECH"
```

U ovom primjeru, varijable **profit**, **stopa_rasta** i **naziv_comp** trenutno čuvaju vrijednosti različitih tipova (int, float i string). Važno je naglasiti da Python omogućava dinamičku dodjelu tipa podataka, odnosno ne mora se unaprijed deklarisati tip varijable, a ista varijabla može kasnije dobiti i vrijednost drugog tipa.

Promjena vrijednosti varijabli

Varijable su promjenjive (mutable) u smislu da možemo mijenjati njihove vrijednosti u toku izvršavanja programa. Na primjer:

```
profit = 1000
profit = profit + 2000 # sada profit iznosi 12000
print("Novi profit je:", profit)
```

```
Novi profit je: 12000
```

Ovim fleksibilnim pristupom, možemo početi s osnovnim vrijednostima i postepeno ih prilagođavati na osnovu novih informacija (npr. dodavanje novih transakcija, ažuriranje kursa valute ili promjena kamatne stope).

3.2.2. Operatori i izrazi

Operatori su osnovni alati u programiranju koji omogućavaju izvođenje operacija nad varijablama i vrijednostima. Pomoću njih, moguće je kreirati izraze, koji predstavljaju kombinacije varijabli, operatora i literala. Takvi izrazi se evaluiraju u novu vrijednost, pružajući osnovu za izvođenje računskih, logičkih, ili drugih operacija neophodnih za obradu podataka.

3.2.2.1. Matematički operatori

U ekonomskim proračunima matematički operatori predstavljaju osnovne alate za izvođenje računskih operacija. Oni omogućavaju definisanje izraza, koji se koriste za proračun poput izračunavanja profita, prosjeka ili vrijednosti investicije. Sljedeća tabela prikazuje najčešće korišćene matematičke operatore i njihove primjene.

Tabela 3.1. Matematički operatori

Operator	Opis	Primjer
+	Sabiranje	ukupni_prihod = prihod_od_prodaje + prihod_od_usluga
-	Oduzimanje	profit = prihod - troškovi
*	Množenje	vrijednost_portfolija = broj_akcija * cijena_po_akciji
/	Dijeljenje	prosjek = suma_vrijednosti / broj_elemenata
//	Cjelobrojno dijeljenje	rezultat = 10 // 3 # Rezultat je 3
**	Stepenovanje	krajnja_vrijednost = pocetna_vrijednost * (1 + stopa_rasta)**broj_godina
%	Ostatak pri dijeljenju	ostatak = 10 % 3 # Rezultat je 1

Primjer:

```
# Definisanje varijabli
prihod = 50000
troskovi = 30000
broj_proizvoda = 1000
cijena_po_proizvodu = 50
stopa_rasta = 0.05 # 5%
broj_godina = 10

# Oduzimanje (računanje profita)
profit = prihod - troskovi
print("Profit je:", profit)

# Sabiranje (ukupni prihod iz različitih izvora)
prihod_od_usluga = 20000
ukupni_prihod = prihod + prihod_od_usluga
print("Ukupni prihod je:", ukupni_prihod)

# Množenje (vrijednost ukupne prodaje)
vrijednost_prodaje = broj_proizvoda * cijena_po_proizvodu
print("Vrijednost prodaje je:", vrijednost_prodaje)

# Dijeljenje (prosječni prihod po proizvodu)
prosjek = prihod / broj_proizvoda
print("Prosječni prihod po proizvodu je:", prosjek)

# Cjelobrojno dijeljenje (grupisanje proizvoda po pakovanjima od 12 komada)
broj_pakovanja = broj_proizvoda // 12
print("Broj pakovanja je:", broj_pakovanja)

# Stepenovanje (krajnja vrijednost investicije)
krajnja_vrijednost = prihod * (1 + stopa_rasta) ** broj_godina
print("Krajnja vrijednost investicije je:", krajnja_vrijednost)

# Ostatak pri dijeljenju (preostali proizvodi nakon grupisanja)
ostatak_proizvoda = broj_proizvoda % 12
print("Preostali proizvodi su:", ostatak_proizvoda)
```

```
Profit je: 20000
Ukupni prihod je: 70000
Vrijednost prodaje je: 50000
Prosječni prihod po proizvodu je: 50.0
Broj pakovanja je: 83
Krajnja vrijednost investicije je: 81444.7313388721
Preostali proizvodi su: 4
```

3.2.2.2. Relacioni operatori

Relacioni operatori omogućavaju poređenje vrijednosti i predstavljaju osnovni alat za donošenje odluka u kodu. Oni se često koriste za analizu podataka, kontrolu toka programa i kreiranje uslovnih izraza u ekonomskim analizama. Sljedeća tabela prikazuje operatori poređenja u Python-u.

Tabela 2.2. Relacijski operatori

Operator	Opis	Primjer
<code>==</code>	Jednako	<code>prihod == troskovi</code>
<code>!=</code>	Nije jednako	<code>prihod != troskovi</code>
<code>></code>	Veće	<code>prihod > troskovi</code>
<code><</code>	Manje	<code>prihod < troskovi</code>
<code>>=</code>	Veće ili jednako	<code>prihod >= troskovi</code>
<code><=</code>	Manje ili jednako	<code>prihod <= troskovi</code>

Relacioni operatori vraćaju logičke vrijednosti (`True` ili `False`), što ih čini korisnim za uslovne naredbe i filtriranje podataka.

```
# Definisanje vrijednosti
cijena_akcije = 120
granicna_cijena = 100

# Poređenje
print("Da li je cijena viša od granične?", cijena_akcije > granicna_cijena)
print("Da li je cijena jednaka graničnoj?", cijena_akcije == granicna_cijena)
print("Da li je cijena niža od granične?", cijena_akcije < granicna_cijena)
```

Da li je cijena viša od granične? True
 Da li je cijena jednaka graničnoj? False
 Da li je cijena niža od granične? False

Objašnjenje rezultata:

- Operator `>` vraća `True` jer je `cijena_akcije` (120) veća od `granicna_cijena` (100).
- Operator `==` vraća `False` jer `cijena_akcije` nije jednaka `granicna_cijena`.
- Operator `<` vraća `False` jer `cijena_akcije` nije manja od `granicna_cijena`.

3.2.2.3. Logički operatori

Logički operatori omogućavaju kombinovanje logičkih vrijednosti (`True` ili `False`) radi donošenja složenijih odluka u programiranju. Oni su ključni u ekonomskim analizama za provjeru ispunjenosti više uslova istovremeno, postavljanje kompleksnih pravila i donošenje informisanih odluka. Sljedeća tabela predstavlja osnovne logičke operatore u Python-u.

Tabela 3.3. Logički operatori

Operator	Opis	Primjer
and	Logičko "i": Vraća <code>True</code> ako su svi uslovi ispunjeni	<code>prihod > troškovi and marža > 0.1</code>
or	Logičko "ili": Vraća <code>True</code> ako je bar jedan uslov ispunjen	<code>prihod > troškovi or bilans_aktiva > 50000</code>
not	Logička negacija: Inverzuje vrijednost (<code>True</code> postaje <code>False</code> i obrnuto)	<code>not (prihod > troškovi)</code>

Primjer:

```
# Definisanje varijabli
cijena_akcije = 120
povoljna_cijena = cijena_akcije < 110
dobra_fundamenta = True # Pretpostavimo da smo zaključili da su dobra
# Donošenje investicijske odluke
investicijska_odluka = povoljna_cijena and dobra_fundamenta
print("Da li je preporučljivo investirati?", investicijska_odluka)
# Provjera alternativnog scenarija
alternativna_odluka = povoljna_cijena or dobra_fundamenta
print("Da li postoji alternativa za investiranje?", alternativna_odluka)
# Inverzija uslova
bez_investicije = not investicijska_odluka
print("Da li je preporučljivo ne investirati?", bez_investicije)

Da li je preporučljivo investirati? False
Da li postoji alternativa za investiranje? True
Da li je preporučljivo ne investirati? True
```

3.2.2.4. Izrazi

Izrazi su kombinacije varijabli, operatara i vrijednosti, koje Python evaluira u novu vrijednost. Unutar jednog složenog izraza grupisanje manjih izraza se vrši pomoću malih zagrada. Izvršenje izraza u zagrada ima prioritet u odnosu na izvršavanje ostalih matematičkih operatora. Na primjer izraz:

```
povrat_investicije = (krajna_vrijednost - pocetna_vrijednost) / pocetna_vrijednost * 100
```

računa procenat povrata investicije koristeći varijable `krajna_vrijednost` i `pocetna_vrijednost`, uz primjenu matematičkih operatora.

U ekonomskim i finansijskim analizama izrazi su ključni alati za generisanje novih vrijednosti iz postojećih podataka. Na primjer, mogu se koristiti za izračunavanje profitnih marži, stopa rasta, indeksa elastičnosti i drugih pokazatelja, pružajući dublji uvid u ekonomske tokove i finansijske performanse.

3.3. Kontrolne strukture

U prethodnim poglavljima predstavljeni su koncepti tipova podataka, varijabli, operatora i izraza. Međutim, bez mogućnosti da program donosi odluke i izvršava zadatke više puta, mogućnosti programskog jezika bi bile ograničene. Upravo iz tog razloga, kontrolne strukture čine ključan element u programiranju, omogućavajući fleksibilno i efikasno upravljanje tokom izvršavanja koda.

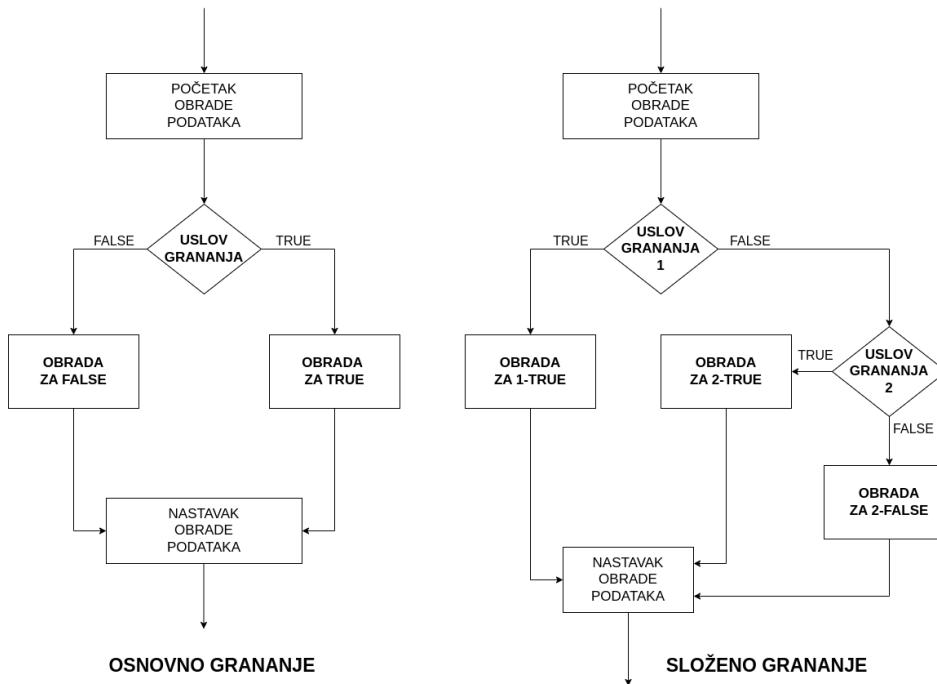
Ovo poglavlje bavi se osnovnim kontrolnim strukturama u Python-u: uslovnim naredbama i programskim petljama. Uslovne naredbe pružaju mogućnost da se na osnovu određene logičke provjere (npr. uslova vezanih za cijenu, datum, količinu ili neki indikator) odabere jedan od više puteva izvršavanja. Petlje, s druge strane, omogućavaju ponavljanje istih operacija više puta, što je posebno važno pri radu s većim skupovima podataka, pri pretraživanju informacija, statističkim analizama i simulacijama ekonomskih modela.

3.3.1. Uslovne naredbe (if, elif, else)

Uslovne naredbe, poznate i kao naredbe grananja, omogućavaju donošenje odluka u programu i usmjeravanje daljih koraka obrade podataka. Na osnovu postavljenog uslova, program može odlučiti da izvrši jedan blok koda, drugi alternativni blok, ili jednostavno preskočiti određeni dio koda i preći na sljedeću naredbu. Ove naredbe su ključne za prilagođavanje ponašanja programa promjenjivim okolnostima, poput varijacija u cijenama, promjena kamatnih stopa, dostupnosti podataka ili ispunjavanja određenih ekonomskih kriterijuma.

Uslovi mogu biti osnovni ili složeni, u zavisnosti od problema koji se rješava. Osnovno odlučivanje koristi if-else naredbe za jednostavne uslove, dok složeniji slučajevi koriste if-elif-else strukturu za donošenje odluka između više mogućnosti. Na slici 3.1. su prikazane dvije blok šeme, koje ilustruju osnovne i složene strukture grananja, prikazujući kako program odlučuje na osnovu uslova.

Primjena ovih naredbi omogućava fleksibilno i dinamično upravljanje tokom izvršenja programa. Ovo je posebno korisno u situacijama, kada je potrebno reagovati na promjene u ulaznim podacima, kao što su varijacije u prihodima i troškovima, ili kada je potrebno analizirati različite scenarije i donijeti odgovarajuće odluke u analizi podataka i simulacijama.



Slika 3.1. Blok dijagram uslovnih naredbi, Izvor: Autori.

Posmatrajmo sljedeći jednostavan primjer grananja.

```

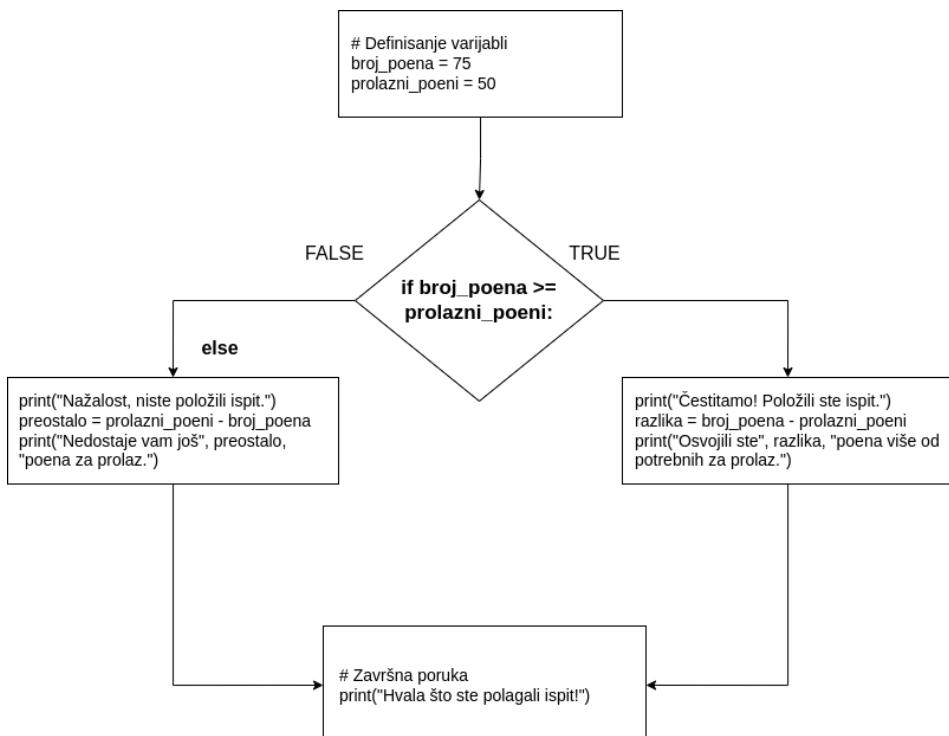
# Definisanje varijabli
broj_poena = 75
prolazni_poeni = 50

# Provjera da li je student položio
if broj_poena >= prolazni_poeni:
    print("Čestitamo! Položili ste ispit.")
    razlika = broj_poena - prolazni_poeni
    print("Osvojili ste", razlika, "poena više od potrebnih za prolaz.")
else:
    print("Nažalost, niste položili ispit.")
    preostalo = prolazni_poeni - broj_poena
    print("Nedostaje vam još", preostalo, "poena za prolaz.")

# Završna poruka
print("Hvala što ste polagali ispit!")

```

U ovom primjeru postavljen je logički test `if broj_poena >= prolazni_poeni:` i na osnovu rezultata tog testa ako je on True izvršavamo prvi blok komandi, koji je odmah ispod if uslova, odnosno ako je False izvršabamo blok, koji je iza komande else. Da bi lakše razumjeli blok dijagram, koji je prikazana na slici 3.1. na sljedećoj slici 3.2. biće prikazan prethodni primjer u obliku blok dijagrama sa komandama unutar blokova.



Slika 3.2. Blok dijagram uslovne naredbe if-else sa primjerom, Izvor: Autori.

U ekonomskim analizama često postoji potreba za donošenjem odluka na osnovu više uslova. Na primjer, prilikom analize tržišta akcija, može biti korisno klasifikovati nivo cijene akcije u različite kategorije, poput "niska", "umjerena" ili "visoka," u zavisnosti od definisanih pragova. Ova klasifikacija omogućava lakše donošenje odluka, identifikaciju prilika za ulaganje ili procjenu rizika. Za rješavanje ovakvih logičkih problema potrebne su nam složene strukture grananja. U nasatvku je primjer jedne takve složene strukture.

```
cijena_akcije = 85

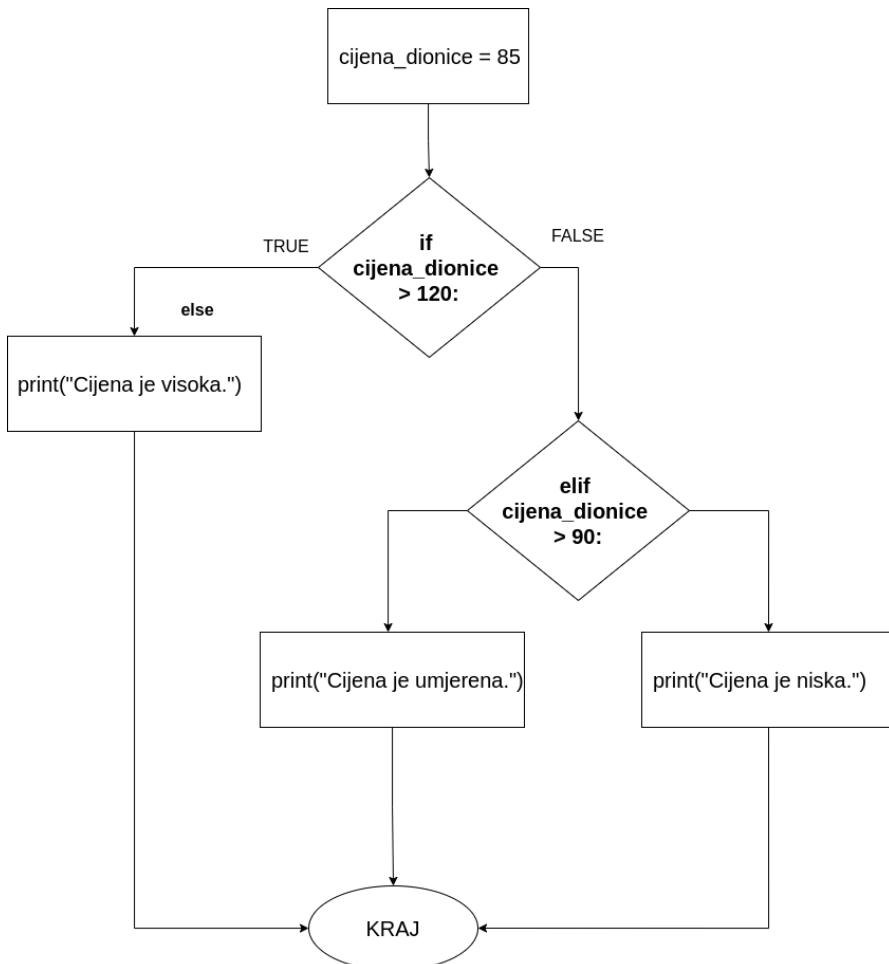
if cijena_akcije > 120:
    print("Cijena je visoka.")
elif cijena_akcije > 90:
    print("Cijena je umjerena.")
else:
    print("Cijena je niska.")
```

Objašnjenje:

- Prvi uslov (**if**) provjerava da li je cijena akcije veća od 120. Ako je taj uslov ispunjen, program ispisuje "Cijena je visoka" i završava dalju provjeru.

- Drugi uslov (**elif**) provjerava da li je cijena veća od 90, ali manja ili jednaka 120. Ako je taj uslov ispunjen, program ispisuje "Cijena je umjerena."
- Zadnji uslov (**else**) pokriva sve preostale slučajeve, tj. cijene koje su 90 ili manje i ispisuje "Cijena je niska."

U nastavku je slika koja prikazuje blok dijagram složene strukture granaњa, koja prikazuje prethodni primjer.



Slika 3.3. Blok dijagram uslovne naredbe if-elif-else sa primjerom, Izvor: Autori.

Osim klasičnih struktura **if**, **elif**, i **else**, Python nudi i druge načine za donošenje odluka i uslovno granaњe u programu. Na primjer, **ternarni operator** omogućava donošenje odluka u jednoj liniji koda, dok **match-case** struktura (dostupna od verzije 3.10) pruža moćan alat za rad sa više specifičnih slučajeva, slično **switch** naredbi u drugim jezicima. Takođe, kombinacija logičkih operatorka **and**, **or** i **not** unutar uslova može dodatno pojednostaviti izraze i omogućiti elegantniji način kombinovanja više kriterijuma.

Naprednije tehnike, poput korištenja riječnika (`dict`) za uslovno grnanje ili upravljanja greškama pomoću `try-except` blokova, otvaraju vrata za efikasniji i modularniji kod. Ove metode su korisne u složenim projektima, gdje se traži visok nivo fleksibilnosti i preciznosti u upravljanju tokom programa.

Međutim, ove tehnike zahtijevaju dublje razumijevanje programskih koncepata i često su namijenjene iskusnjim korisnicima ili specijalistima u programiranju. S obzirom na to da je cilj ove knjige uvođenje ekonomista u osnove Pythona, fokus će ostati na `if-elif-else` strukturama, koje su intuitivne i dovoljne za većinu početnih ekonomskih analiza.

Za one koji žele istražiti ove napredne tehnike, preporučujemo zvaničnu Python dokumentaciju kao vrijedan resurs. Kako budete napredovali, ove metode će vam postati korisne za sofisticirane analize i automatizaciju.

3.3.2. Petlje (for, while)

Petlje u Python-u predstavljaju kontrolne strukture, koje omogućavaju ponavljanje određenih operacija bez potrebe za ponovnim pisanjem istog koda. Ovo je naročito korisno u ekonomskim i finansijskim analizama, gdje se često treba prolaziti kroz velike skupove podataka, kao što su liste cijena kroz vremenski period, nizovi transakcija ili skup zemalja za koje se izračunavaju ekonomski indikatori. Petlje pojednostavljaju rad sa takvim podacima i omogućavaju preciznu automatizaciju ponavljačkih zadataka.

Svako pojedinačno izvršenje naredbi unutar petlje naziva se iteracija. Na primjer, ako analiziramo niz cijena akcija za svaku sedmicu u godini, svaka sedmica bi predstavljala jednu ponavljanje. Blok koda unutar petlje može se izvršiti od nula do beskonačnog broja puta, u zavisnosti od postavljenih uslova. Ovo omogućava rad sa kompleksnim skupovima podataka i pruža fleksibilnost u prilagođavanju analitičkog procesa.

Petlje su često poznate i kao strukture ponavljanja jer omogućavaju programima da ponavljaju isti proces dok ne budu ispunjeni uslovi za prekid. U Python-u postoje dvije osnovne vrste petlji koje ćemo koristiti:

- `for` petlja
- `while` petlja

U nastaktu ćemo detaljnije objasniti svaku od njih i dati odgovarajuće primjere u ekonomskoj praksi.

3.3.2.1. For petlja

For petlja je jedna od najčešće korištenih struktura u Python-u, jer omogućava ponavljanje kroz niz podataka ili sekvencu, što je izuzetno korisno u radu s velikim skupovima podataka. Ključna riječ `for` se koristi za prolazak kroz elemente određene sekvence (npr. liste, stringa, tupla ili generisanog niza pomoću `range()` funkcije). Ova petlja je idealna kada unaprijed znamo koliko puta želimo da ponovimo određeni blok koda.

Šta je sekvencia u Python-u?

Sekvenca predstavlja uređeni skup elemenata, kroz koje se možemo kretati sa pokazivačem na svaki pojedinačni element sukupa. Python nudi različite vrste sekvenci, koje se mogu koristiti u `for` petlji:

- **Liste** – Kolekcije elemenata koje mogu biti različitih tipova.
- **Stringovi** – Sekvence karaktera.
- **Tuple** – Slične listama, ali sadrže nepromjenljive podatke.
- **Range objekti** – Generišu sekvencu cijelih brojeva.

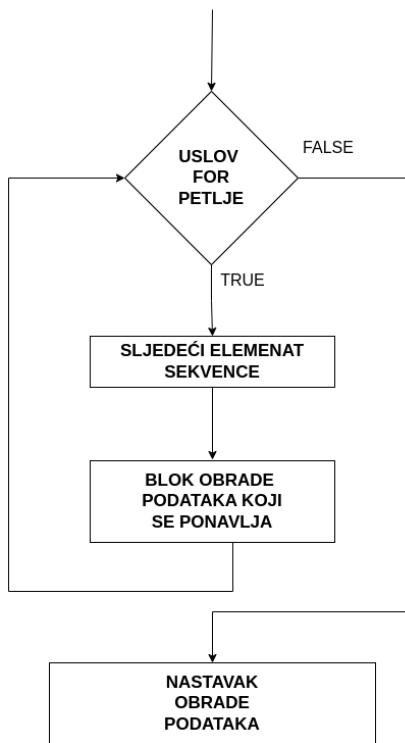
Kako for petlja funkcioniše?

For petlja prolazi kroz svaki element sekvence i izvršava kod u njenom bloku za svaki pojedinačni element. Ovo omogućava ekonomično rješavanje problema bez potrebe za manuelnim pisanjem istog koda za svaki element.

Na sljedećoj slici je prikazana blok dijagram for petlje. Dijagram prikazuje strukturu i tok izvršavanja kroz nekoliko koraka. Proces počinje inicijalizacijom petlje, gdje se prvi element sekvence uzima za obradu. Sekvenca može biti lista, string, ili niz brojeva generisan funkcijom `range`. Uslov petlje provjerava da li postoje još elemenata za obradu. Ukoliko više nema elemenata, petlja se prekida, a program nastavlja sa narednim kodom izvan petlje.

Nakon potvrde uslova, sljedeći element sekvence se obrađuje u bloku obrade podataka. U ovom dijelu se izvršavaju definisane operacije nad trenutnim elementom, kao što su proračuni, ispis podataka ili logičke projvjere. Nakon obrade, kontrola se vraća na početak petlje da provjeri postojanje narednog elementa. Proces se ponavlja dok se ne obradi posljednji element sekvence.

Kada petlja završi, program prelazi na naredni korak obrade podataka izvan petlje. Ova struktura omogućava automatsko i efikasno ponavljanje operacija nad cijelim skupom podataka, što je posebno korisno u ekonomskim analizama, kao što je računanje prosječnih vrijednosti, analiza troškova ili poređenje finansijskih pokazatelja.



Slika 3.4. Blok dijagram for petlje, Izvor: Autori.

Kako bi potpuno razumjeli kako funkcioniše for petlja pogledajmo sljedeći primjer. U njemu je potrebno proći kroz listu cijena akcija i ispisati svaku cijenu.

```

cijene_akcija = [120, 135, 145, 130, 125]

for cijena in cijene_akcija:
    print("Cijena akcije:", cijena)

```

Cijena akcije: 120
Cijena akcije: 135
Cijena akcije: 145
Cijena akcije: 130
Cijena akcije: 125

Petlja se izvršava nad listom `cijene_akcija` i ona se kreće od prvog elementa (120) do posljednjeg elements u lists (125) izvršavajući blok komandi `print("Cijena akcije:", cijena)`.

For petlja se koristi kada je potrebno ponoviti određeni blok operacija tačno definisan broj puta. Međutim, u situacijama kada nemamo unaprijed definisanu listu ili brojevi tip podataka, koji bi odredili broj ponavljanja, koristimo funkciju `range()`. Ova

funkcija generiše opseg brojeva i omogućava for petlji da vrši ponavljanje kroz sekvencu cijelih brojeva.

Pomoću funkcije `range()` možemo generisati sekvencu cijelih brojeva i koristiti je u for petlji. U sljedećem primjeru programa prolazimo kroz petlju 5 puta.

```
for broj in range(1, 6): # Generiše brojeve od 1 do 5
    print("Broj je:", broj)
```

```
Broj je: 1
Broj je: 2
Broj je: 3
Broj je: 4
Broj je: 5
```

Ova funkcija je veoma korisna kada želimo prolazak kroz unaprijed definisane intervale. Funkcija `range()` se može koristiti sa jednim, dva ili tri parametra:

- Jedan parametar: `range(5)` generiše brojeve od 0 do 4.
- Dva parametra: `range(1, 6)` generiše brojeve od 1 do 5.
- Tri parametra: `range(1, 10, 2)` generiše brojeve od 1 do 9 sa korakom 2.

3.3.2.2. While petlja

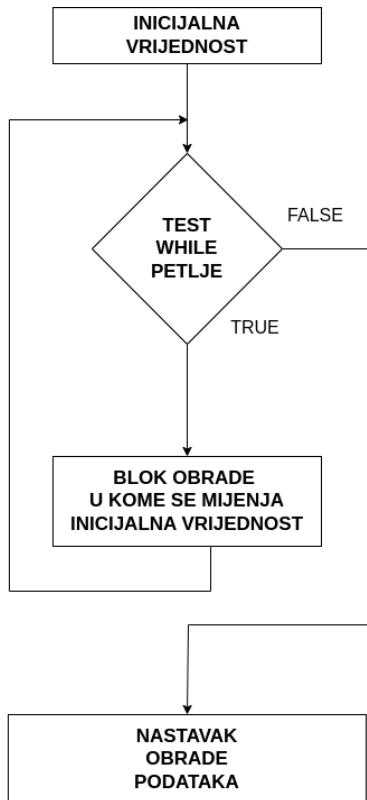
While petlja predstavlja jednu od ključnih struktura ponavljanja u Python-u, koja se koristi kada je potrebno ponavljati određeni blok koda, sve dok neki uslov ostaje tačan. Za razliku od for petlje, gdje je broj ponavljanja unaprijed poznat ili definisan, while petlja je idealna kada ne znamo koliko će iteracija biti potrebno, već se petlja izvršava sve dok ne bude ispunjen neki specifičan kriterijum.

Osnova rada while petlje leži u logičkom uslovu. Sve dok je uslov ispunjen (tačan, tj. `True`), blok koda unutar petlje će se iznova izvršavati. Ova fleksibilnost omogućava rješavanje problema, gdje se uslovi dinamički mijenjaju tokom izvršavanja, kao što su analize tržišnih podataka, simulacije ili procesi optimizacije.

Blok dijagram while petlje prikazan na slici 3.5. definiše tok izvršavanja ponavljajuće operacije zasnovane na uslovu. Prvi korak je postavljanje inicijalne vrijednosti, koja predstavlja početno stanje variable, koja će se provjeravati. Nakon toga slijedi test while petlje, gdje se provjerava da li je postavljeni uslov tačan. Ako je uslov ispunjen (tačan), izvršava se blok obrade u kojem se mijenja inicijalna vrijednost. Nakon svake iteracije, program se vraća na provjeru uslova.

U trenutku kada uslov više nije tačan, obrada izlazi iz petlje i prelazi na nastavak obrade podataka, koji slijedi izvan petlje. Ova struktura omogućava ponavljanje koda

dok se ne postigne željeni kriterijum, što je ključno za iteracije čiji broj unaprijed nije poznat.



Slika 3.5. Blok dijagram while petlje, Izvor: Autori.

Prethodni blok dijaglam će se lakše shvatiti, kroz objašnjenje rada sljedećeg primjera praćenje kamatne stope, koja se dinamički mijenja dok ne dostigne željeni nivo.

```

kamatna_stopa = 0.07 # Početna vrijednost: 7%
threshold = 0.05 # Prag ispod kojeg zaustavljamo petlju

print("Početna kamatna stopa:", kamatna_stopa)
while kamatna_stopa > threshold:
    # Simulacija smanjenja kamatne stope
    kamatna_stopa = kamatna_stopa - 0.005
    print("Trenutna kamatna stopa:", f"{kamatna_stopa:.3f}")
print("Krajnja kamatna stopa:", f"{kamatna_stopa:.3f}")
  
```

```

Početna kamatna stopa: 0.07
Trenutna kamatna stopa: 0.065
Trenutna kamatna stopa: 0.060
Trenutna kamatna stopa: 0.055
Trenutna kamatna stopa: 0.050
Trenutna kamatna stopa: 0.045
Krajnja kamatna stopa: 0.045
  
```

U ovom primjeru, na početku inicijalizujemo vrijednosti za kamatnu stopu i *threshold*, odnosno zadatu donju granicu. Petlja se ponavlja sve dok je kamatna_stopa veća od zadate donje granice. Svaka iteracija simulira pad kamatne stope za 0.005 i ispisuje trenutnu vrijednost. Čim uslov **kamatna_stopa > threshold** postane netačan (`False`), izvršavanje petlje se zaustavlja. Na kraju se ispisuje krajnja vrijednost dostignute kamete. Formatiranje izlaznih podataka `f"\{kamatna_stopa:.3f}`, koje je korišteno u primjeru definiše decimalnu vrijednost za promjenjivu kamatna_stopa sa tri decimalne cifre.

Napomena o beskonačnoj petlji

Važno je osigurati da uslov u while petlji postane netačan u nekom trenutku. Ako uslov uvijek ostaje tačan (`True`), petlja postaje beskonačna, što može uzrokovati blokiranje programa i nije moguće završiti program. Program se tada mora nasilno prekinuti. Slijedi primjer koda sa beskonačnom petljom.

```
while True:  
    print("Ovo je beskonačna petlja!")
```

Ovo je beskonačna petlja!
Ovo je beskonačna petlja!
Ovo je beskonačna petlja!
...

Situacije poput beskonačnih petlji mogu se izbjegći pravilnim definisanjem uslova ili uvođenjem mehanizama za prekid, kao što je `break` naredba, koja omogućava trenutni izlazak iz petlje i prekid njenog izvođenja. Ipak, postoje slučajevi kada su beskonačne petlje korisne i namjerno implementirane. Tipični primjeri uključuju skripte, koje kontinuirano čekaju vanjski unos podataka ili programe koji neprekidno prikupljaju, obrađuju i smještaju podatke u bazu ili fajl bez intervencije korisnika.

While petlja, zahvaljujući svojoj fleksibilnosti, predstavlja izuzetno moćan alat za ponavljanja u programiranju. Posebno je korisna kada je potrebno dinamički reagovati na promjenjive uslove tokom izvršavanja programa, što je čest zahtjev u ekonomskoj analizi i praćenju podataka.

3.3.3. Kombinovanje uslovnih naredbi i petlji

U programiranju, moć i fleksibilnost dolaze do izražaja kada se kombinuju uslovne naredbe (*if*, *elif*, *else*) sa petljama (*for*, *while*). Ova kombinacija omogućava složene procese obrade podataka, gdje se analiziraju skupovi podataka, vrše logičke provjere i selektuju relevantne informacije. U ekonomskim analizama, ovo je naročito korisno za filtriranje transakcija, klasifikaciju vrijednosti, izračunavanje statističkih pokazatelja i druge operacije, koje zahtijevaju iterativni pristup.

Posmatrajmo situaciju u kojoj imamo listu transakcija i potrebno je izdvojiti samo one, koje zadovoljavaju određeni kriterijum. Kombinacijom **for petlje** i **if naredbe**, možemo pregledati sve transakcije jednu po jednu, provjeriti postavljeni uslov i izdvojiti samo relevantne podatke.

Slijedi primjer filtriranja transakcija, čija je vrijednost veća od 1000.

```
transakcije = [500, 1200, 750, 2200, 900]
filtrirane = [] # Prazna lista za smještanje rezultat

for t in transakcije: # Iteracija kroz listu transakcija
    if t > 1000: # Provjera da li transakcija zadovoljava uslov
        filtrirane.append(t) # Dodavanje u listu ako je uslov ispunjen

print("Transakcije iznad 1000:", filtrirane)

Transakcije iznad 1000: [1200, 2200]
```

Objašnjenje koda:

- For petlja prolazi kroz sve elemente u listi **transakcije**.
- If naredba provjerava da li je vrijednost transakcije veća od 1000.
- Ako je uslov ispunjen, vrijednost se dodaje u listu **filtrirane** pomoću metode **append()**. Ova metoda jednostavno dodaje novi element u listu, koji ima vrijednost **t**.
- Na kraju, lista sa filtriranim transakcijama se ispisuje kao rezultat.

Kombinacija **uslovnih naredbi** i **petlji** omogućava efikasno pretraživanje, filtriranje i obradu velikih skupova podataka. Ovakvi alati čine programski kod fleksibilnim i prilagodljivim za rješavanje različitih problema, što je od izuzetnog značaja u ekonomskoj analizi. Kroz iteraciju podataka i primjenu logičkih uslova, možemo identifikovati trendove, filtrirati informacije i donositi zaključke zasnovane na preciznim kriterijumima.

3.4. Funkcije i moduli

Prilikom pisanja složenijih projekata i finansijskih analiza, kod koji obrađuje podatke postaje sve obimniji i složeniji. To dovodi do potrebe za organizacijom koda, koja omogućava veću efikasnost i preglednost. Funkcije i moduli predstavljaju ključan alat za postizanje tog cilja. Umjesto da se isti proračuni ili operacije pišu iznova na različitim mjestima, funkcije omogućavaju da se određeni dio koda definiše jednom, a potom koristi kad god je potrebno. Na taj način se smanjuje ponavljanje koda, pojednostavljuje njegovo održavanje i omogućava brža prilagodljivost promjenama.

U nastavku biće objašnjeni osnovni koncepti vezani za pisanje funkcija i korišćenje modula sa ugradenim funkcijama, uz fokus na njihovu praktičnu primjenu u ekonomiji. Biće predstavljene prednosti funkcija, razlike između različitih vrsta funkcija, kao i osnove korišćenja modula i biblioteka za složene analize.

3.4.1. Korisnički definisane funkcije

Funkcije su samostalni dijelovi koda, koji obavljaju određeni zadatak i mogu se koristiti više puta, bez potrebe za ponovnim pisanjem iste logike. Funkcije omogućavaju programerima da definišu složene operacije jednom, a zatim ih pozivaju u različitim dijelovima programa sa različitim ulaznim podacima. Ovo je naročito korisno u ekonomiji, gdje su proračuni često ponovljivi, kao što su izračunavanje rata kredita, diskontovane vrijednosti novčanih tokova ili analize troškova i prihoda.

Praktično, funkcije se definišu pomoću ključne riječi `def`, a njihova osnovna struktura uključuje ime funkcije, parametre i logiku obrade. Na kraju, funkcija može vratiti rezultat korišćenjem ključne riječi `return`.

Funkcija u Pythonu, slično kao u matematici, ima strukturu koja se može uporediti sa generičkim izrazom funkcije $f(x)$. Svaka funkcija ima svoje ime, koje se koristi za identifikaciju i može imati jedan ili više parametara. Parametri funkcije predstavljaju ulazne podatke, koje funkcija koristi za izvršavanje svog zadatka.

Važno je napomenuti da su parametri opcioni. Funkcija može biti definisana bez parametara, sa jednim parametrom ili sa više njih, zavisno od specifičnih potreba analize ili operacija koje se izvršavaju. Ova fleksibilnost omogućava da funkcije budu jednostavnije za korišćenje i prilagodljive različitim scenarijima.

Slijedi primjer funkcije za izračunavanje mjesecne rate kredita.

```
def rata_kredita(kredit, godisnja_kamata, broj_godina):
    r = (godisnja_kamata / 100) / 12 # Mjesečna kamata
    n = broj_godina * 12 # Ukupan broj rata
    R = kredit * r * (1+r)**n / ((1+r)**n - 1)
    return R
```

```

# Primjena funkcije
mj_rata = rata_kredita(20000, 6, 5) # Kredit od 20.000 sa 6% godišnje kamate na 5
godina
print("Mjesečna rata kredita iznosi:", mj_rata)

Mjesečna rata kredita iznosi: 386.6560305885654

```

U ovom primjeru, funkcija `rata_kredita` računa mjesecnu ratu kredita na osnovu iznosa kredita, godišnje kamatne stope i perioda otplate izraženog u godinama. Promjenom parametara funkcije, korisnik može simulirati različite kreditne uslove, što omogućava detaljnu analizu bez potrebe za ponovnim pisanjem formule.

Važno je naglasiti da redoslijed unosa parametara mora biti tačno praćen ukoliko se vrijednosti unose bez eksplicitnog imenovanja. Međutim, ako korisnik želi da promijeni redoslijed ili izostavi određeni parametar, potrebno je jasno imenovati parametre i dodijeliti im vrijednosti. Ovaj pristup povećava fleksibilnost i čitljivost koda. U nastavku je prikazan primjer unosa parametara za funkciju `rata_kredita`, gdje je redoslijed promjenjen i vrijednosti se prosljeđuju sa imenom parametra.

```

mj_rata = rata_kredita( godisnja_kamata=6, kredit=20000, broj_godina=5)
print("Mjesečna rata kredita iznosi:", mj_rata)

Mjesečna rata kredita iznosi: 386.6560305885654

```

Prednosti korisničkih funkcija:

- **Smanjenje ponavljanja koda** – Umjesto da se ista formula piše više puta, funkcija omogućava definiranje jednog rješenja, koje se može višestruko koristiti.
- **Povećana preglednost** – Grupisanjem logike računanja na jednom mjestu, kod postaje čitljiviji i lakši za razumijevanje.
- **Lakše održavanje** – Kada je potrebno izvršiti izmjenu u načinu računanja, promjena se vrši samo u funkciji, a sve primjene funkcije automatski koriste novi rezultat.
- **Ponovna upotreba** – Funkcije su univerzalne, pa kada se jednom naprave mogu se onda koristiti u različitim projektima ili analizama.

3.4.2. Lambda funkcije

Lambda funkcije u Python-u predstavljaju jednostavan način definiranja anonimnih funkcija, koje se koriste za kratke, jednolinjske operacije. Njihova glavna prednost leži u činjenici da omogućavaju brzu i efikasnu manipulaciju podacima bez potrebe za dodatnim definiranjem standardnih funkcija. Posebno su korisne u situacijama kada se funkcija koristi samo jednom ili je njen zadatok dovoljno jednostavan da ne zahtijeva posebnu strukturu.

U kontekstu ekonomije, gdje su podaci često obimni, a analize složene, lambda funkcije igraju ključnu ulogu u ubrzavanju procesa obrade. Kroz ovaj pristup, moguće je smanjiti kompleksnost koda, povećati efikasnost i postići jasniji pregled rada, što je od posebne važnosti za ekonomski analitičare i istraživače.

Ovo poglavlje će pokazati kako i zašto koristiti lambda funkcije, s posebnim fokusom na njihove primjene u ekonomiji. U narednim poglavljima često ćemo koristiti primjere sa lambda funkcijama, pa je veoma važno da razumijete samu strukturu lambda funkcije.

Šta su lambda funkcije?

Lambda funkcije su anonimne funkcije, što znači da nemaju ime i definišu se direktno na mjestu gdje se koriste. One predstavljaju brz i efikasan način za kreiranje funkcionalnosti bez potrebe za formalnim definisanjem pomoću `def`. U Python-u se pišu koristeći ključnu riječ `lambda`. Ovo ih čini pogodnim za jednostavne zadatke, koji ne zahtijevaju dodatnu strukturu ili višelinjsku logiku.

Sintaksa: `lambda argumenti: izraz`

- **Argumenti** – Lista ulaznih vrijednosti, slično parametrima u funkciji.
- **Izraz** – Jedna linija koda koja se izvršava i vraća rezultat.

Poređenje sa korisnički definisanim funkcijama

Korisnički definisane funkcije koriste ključnu riječ `def` i mogu sadržavati više linija koda, dok su lambda funkcije ograničene na jedan izraz. Za ilustraciju, razmotrite sljedeći primjer.

Korisnički definisana funkcija:

```
def kvadrat(x):
    return x ** 2
print(kvadrat(5)) # Ispisuje: 25
```

Lambda funkcija:

```
kvadrat = lambda x: x ** 2
print(kvadrat(5)) # Ispisuje: 25
```

Lambda funkcije su kompaktnije i koriste se za jednostavne zadatke, dok korisnički definisane funkcije omogućavaju veću fleksibilnost i složenost. Pored toga, lambda funkcije nemaju ime osim ako im se ne dodijeli pomoću varijable.

Korištenje lambda funkcija donosi niz prednosti, posebno u situacijama kada je potrebno implementirati jednostavne i jednolinjske logike, ali takođe imaju i određena

ogrničenja. U nastavku je pregled ključnih prednosti i nedostataka upotrebe lambda funkcije.

Prednosti:

- **Jednostavnost** – Lambda funkcije su idealne za kratke operacije, koje se koriste samo na jednom mjestu. Ovo ih čini pogodnim za brzo definisanje funkcionalnosti bez potrebe za pisanjem dodatnog koda.
- **Kompaktnost** – Njihova sintaksa omogućava smanjenje obima koda u poređenju sa korisnički definisanim funkcijama. Ovo doprinosi većoj preglednosti koda, posebno kod jednostavnih zadataka.
- **Fleksibilnost** – Lambda funkcije se često koriste u kombinaciji sa funkcijama višeg reda, poput `map`, `filter` i `reduce`, gdje omogućavaju primjenu logike direktno unutar ovih funkcija. Ovo olakšava obradu i analizu podataka u jednom koraku.

Ograničenja:

- **Ograničena složenost** – Lambda funkcije mogu sadržavati samo jedan izraz, što ih čini neadekvatnim za implementaciju složenijih logika.
- **Manja čitljivost** – Kod pisan uz korištenje lambda funkcija može postati teže razumljiv, posebno u slučaju kompleksnijih izraza ili njihove učestale upotrebe.
- **Nedostatak strukturisanosti** – Budući da lambda funkcije nemaju ime i ne podržavaju višelinjske izraze, teže ih je dokumentovati i ponovo koristiti.

Primjeri lambda funkcija u ekonomiji

Slijedi primjer filtriranja podataka iz liste pod određenim uslovom.

```
cijene = [100, 200, 300]
pdv = list(map(lambda x: x * 0.2, cijene))
print(pdv) # Ispisuje: [20.0, 40.0, 60.0]
```

Slijedi primjer računanja PDV-a za listu cijena.

```
cijene = [100, 200, 300]
skupe_cijene = list(filter(lambda x: x > 150, cijene))
print(skupe_cijene) # Ispisuje: [200, 300]
```

Primjer izračunavanja diskontirane cijene.

```
cijene = [100, 200, 300]
diskont = 0.1 # 10% popusta
diskontirane_cijene = list(map(lambda x: x * (1 - diskont), cijene))
print(diskontirane_cijene) # Ispisuje: [90.0, 180.0, 270.0]
```

Lambda funkcije su moćan alat u Python-u koji omogućava ekonomistima da brzo i efikasno rješavaju probleme. Iako imaju svoja ograničenja, njihova jednostavnost i fleksibilnost čine ih izuzetno korisnim, posebno u analizama podataka i simulacijama.

3.4.3. Ugrađene funkcije

Pored korisnički definisanih funkcija koje, omogućavaju kreiranje prilagođenih rješenja, Python nudi i širok spektar **ugrađenih funkcija**, koje su odmah dostupne, bez potrebe za dodatnim uvozom ili instalacijom. Ove funkcije pokrivaju osnovne i česte zadatke, što ih čini izuzetno korisnim u svakodnevnom programiranju, a naročito u ekonomskim analizama, koje često zahtijevaju jednostavnu obradu podataka, konverziju vrijednosti i matematičke proračune.

Korišćenjem ugrađenih funkcija, moguće je brzo i efikasno rješiti mnoge probleme, kao što su zaokruživanje brojeva, konverzija tipova podataka, mjerjenje dužine listi ili nizova i provjera tipa podataka.

U nastavku su predstavljene osnovne ugrađene funkcije u Pythonu, uz opis njihove funkcionalnosti i primjere njihove upotrebe. Radi preglednosti, funkcije su grupisane prema oblastima primjene. Ovo nije konačan pregled, jer će se dodatne funkcije uvoditi i obrađivati u kontekstu praktičnih primjera u narednim poglavljima.

Za detaljniji pregled i potpunu listu ugrađenih funkcija, sa objašnjenjima i primjerima, preporučuje se konsultovanje zvanične Python dokumentacije na sljedećim web adresama:

- <https://docs.python.org/3/library/functions.html>
- <https://realpython.com/python-built-in-functions/#built-in-functions-in-python>

Ove funkcije predstavljaju osnovni alat za rad sa podacima i omogućavaju efikasnije i intuitivnije programiranje u okviru ekonomskih i finansijskih analiza.

a) Matematičke funkcije

Matematičke funkcije u Pythonu olakšavaju obavljanje jednostavnih proračuna i manipulaciju numeričkim podacima. Za razliku od aritmetičko-logičkih operacija, koje smo ranije obradili, ove funkcije nude unaprijed definisane metode za specifične matematičke zadatke, omogućavajući preciznije i brže programiranje.

Nekoliko osnovnih matematičkih funkcija koje se često koriste:

- **abs()** – Ova funkcija računa apsolutnu vrijednost broja i rezultat je uvijek pozitivan. Primjer: `abs(-5)` vraća 5, a `abs(5)` vraća isto 5.

- **round()** – Funkcija zaokružuje broj na određeni broj decimala, što je korisno u slučajevima gdje se rezultati trebaju prikazati sa ograničenom preciznošću. Primjer: `round(3.14159, 2)` vraća `3.14`, a `round(3.146, 2)` vraća `3.15`.
- **pow()** – Ova funkcija omogućava stepenovanje brojeva. Primjer: `pow(2, 3)` vraća `8`, jer računa `2` na treći stepen.

Ove funkcije su ključne u brojnim ekonomskim i finansijskim analizama, omogućavajući jednostavnu obradu podataka bez potrebe za ručnim implementiranjem složenih algoritama.

Primjer zaokruživanje kamatne stope i računanje stepena.

```
kamatna_stopa = 5.2345
print("Zaokružena kamatna stopa:", round(kamatna_stopa, 2)) # Zaokruživanje na 2
decimale

glavnica = 1000
stopa_rasta = 0.05 # Stopa rasta izražena kao decimalni broj (5%)
periodi = 5

krajnja_vrijednost = glavnica * pow((1 + stopa_rasta), periodi)
print("Krajnja vrijednost investicije:", round(krajnja_vrijednost,2))

Zaokružena kamatna stopa: 5.23
Krajnja vrijednost investicije: 1276.28
```

U ovom primjeru, funkcija `round()` služi za prikaz kamatne stope zaokružene na dvije decimale, što je od ključne važnosti za tačnost i preglednost u izvještajima ili tabelarnim prikazima podataka. S druge strane, funkcija `pow()` omogućava jednostavan i efikasan proračun rasta vrijednosti kroz višegodišnje periode, čime se olakšava analiza dugoročnih finansijskih trendova.

b) Funkcije za konverziju tipova

U ekonomskim analizama često je potrebno konvertovati podatke iz jednog tipa u drugi, npr. kada se vrijednosti učitavaju iz fajlova ili baza podataka. Python nudi funkcije za jednostavnu konverziju:

- **int()** – Konvertuje broj u cijeli broj.
- **float()** – Konvertuje vrijednost u realan broj sa decimalama.
- **str()** – Pretvara vrijednost u string (tekstualni format).

Primjer konverzija tipova podataka.

```
prihod = "15000" # Vrijednost je string
prihod = int(prihod) # Pretvaramo je u cijeli broj
porez = prihod * 0.2 # Računanje poreza
print("Iznos poreza iznosi:", porez)
```

```
Iznos poreza iznosi: 3000.0
```

U ovom primjeru, funkcija `int()` konvertuje tekstualni unos u cijeli broj, što omogućava izvođenje matematičkih operacija poput računanja poreza.

c) Funkcije za rad sa kolekcijskim tipovima podataka

Jedna od osnovnih karakteristika Python-a je bogat skup kolekcijskih tipova podataka, koji su ugrađeni u jezik. Ovi tipovi omogućavaju organizaciju i obradu više podataka zajedno, što je ključno za analizu velikih setova podataka u ekonomiji. Kolekcije uključuju liste, torke (tuples), rječnike (dictionaries), skupove (sets) i zamrznute skupove (frozensets). Python nudi ugrađene funkcije za lako kreiranje i manipulaciju ovim strukturama.

Osnovne ugrađene funkcije za rad s kolekcijama:

- `list()` – Kreira listu iz iterabilnog objekta.
- `tuple()` – Kreira torku iz iterabilnog objekta.
- `dict()` – Kreira rječnik iz parova ključeva i vrijednosti ili kroz ključne argumente.
- `set()` – Kreira skup iz iterabilnog objekta.
- `frozenset()` – Kreira zamrznuti (nepromjenjivi) skup iz iterabilnog objekta.

Primjer: Kreiranje kolekcijskih tipova podataka

```
# Kreiranje liste iz raspona brojeva
brojevi = list(range(1, 6))
print(brojevi) # Ispisuje: [1, 2, 3, 4, 5]

# Kreiranje torke iz liste
kategorije = tuple(["Hrana", "Odjeća", "Tehnologija"])
print(kategorije) # Ispisuje: ('Hrana', 'Odjeća', 'Tehnologija')

# Kreiranje rječnika s ključevima i vrijednostima
prodaja = dict(Region1=1000, Region2=1500, Region3=1200)
print(prodaja) # Ispisuje: {'Region1': 1000, 'Region2': 1500, 'Region3': 1200}

# Kreiranje skupa za uklanjanje duplikata
artikli = set(["Jabuka", "Banana", "Jabuka", "Kruška"])
print(artikli) # Ispisuje: {'Banana', 'Kruška', 'Jabuka'}

# Kreiranje zamrznutog skupa
nepromjenjivi_skup = frozenset(["USD", "EUR", "JPY"])
print(nepromjenjivi_skup) # Ispisuje: frozenset({'EUR', 'JPY', 'USD'})

[1, 2, 3, 4, 5]
('Hrana', 'Odjeća', 'Tehnologija')
{'Region1': 1000, 'Region2': 1500, 'Region3': 1200}
{'Jabuka', 'Kruška', 'Banana'}
frozenset({'JPY', 'EUR', 'USD'})
```

Prednosti korišćenja ugrađenih funkcija:

- Brza obradu podataka bez potrebe za dodatnim pisanjem logike.
- Jednostavno rukovanje tipovima podataka, čime se izbjegavaju greške pri proračunima.
- Lako čitljiv kod koji je razumljiv i drugima.

3.4.4. Funkcije iz modula i biblioteka

Pored ugrađenih i korisnički definisanih funkcija, Python omogućava korišćenje funkcija iz modula i biblioteka, što je ključna prednost ovog programskog jezika. Moduli i biblioteke predstavljaju kolekcije unaprijed definisanih funkcija, klase i promjenljivih, organizovanih u zasebne fajlove ili pakete. Njihova primjena omogućava pristup specijalizovanim funkcijama, koje značajno olakšavaju složene zadatke kao što su složeni matematički proračuni, statistička i finansijska analiza, te vizualizacija podataka.

Moduli su dio Python-ove standardne biblioteke, dok su biblioteke često eksterni paketi, koje je potrebno dodatno instalirati putem menadžera paketa poput [pip](#). Korišćenje modula i biblioteka doprinosi modularnosti i ponovnoj upotrebi koda, čime se izbjegava ponavljanje i povećava efikasnost u pisanju programa.

U ekonomskoj analizi često se koriste sljedeće biblioteke:

- **NumPy** – Ova biblioteka pruža moćne alate za rad s višedimenzionim nizovima i matricama, omogućavajući brze matematičke i statističke operacije. Ova biblioteka omogućava računanje prosjeka, standardne devijacije, kovarijanse i korelaciju nad velikim setovima podataka.
- **Pandas** – Fokusirana je na rad s tabelarnim podacima i vremenskim serijama. Njene funkcije omogućavaju jednostavno filtriranje, grupisanje, agregiranje i transformaciju podataka, što je posebno korisno za analizu ekonomskih indikatora ili finansijskih izvještaja.
- **Matplotlib** – Omogućava kreiranje širokog spektra grafika i vizuelnih prikaza podataka. Ovo je neophodno za analizu trendova, poređenje rezultata ili prezentaciju ekonomskih zaključaka, npr. prikaz promjena u BDP-u ili kretanje inflacionih stopa.
- **SciPy** – Nudi napredne funkcionalnosti za matematičke proračune, optimizaciju i statističku analizu. Koristi se za modeliranje ekonomskih sistema, simulacije i provođenje složenih statističkih testova.

Slijedi jednostavan primjer upotrebe modula iz Python-ove standardne biblioteke.

```
import math

x = 2.5
y = math.sqrt(x) # sqrt je funkcija iz math modula
```

```
print("Kvadratni korijen od", x, "je:", y)
```

```
Kvadratni korijen od 2.5 je: 1.5811388300841898
```

U ovom primjeru, modul `math` se koristi za pristup funkciji `sqrt()`, koja računa kvadratni korijen broja. Ovakav pristup omogućava ekonomistima da se fokusiraju na analizu podataka, umjesto na implementaciju osnovnih matematičkih funkcija.

Python nudi i širok spektar eksternih biblioteka, koje proširuju mogućnosti ovog jezika. Dok standardni moduli, kao što je `math`, omogućavaju osnovne matematičke operacije poput računanja kvadratnog korijena ili trigonometrijskih funkcija, eksterne biblioteke, poput `NumPy` i `Matplotlib`, pružaju dodatne funkcionalnosti za naprednu analizu podataka, vizualizaciju i složene matematičke proračune.

Za razliku od modula iz standardne biblioteke, eksterni moduli obično nisu unaprijed instalirani u Python okruženju. Da bi se koristile ove biblioteke, potrebno ih je prethodno instalirati pomoću alata za upravljanje paketima, poput `pip`. Instalacija se obavlja jednostavnom naredbom u terminalu ili komandnoj liniji.

```
pip install numpy matplotlib
```

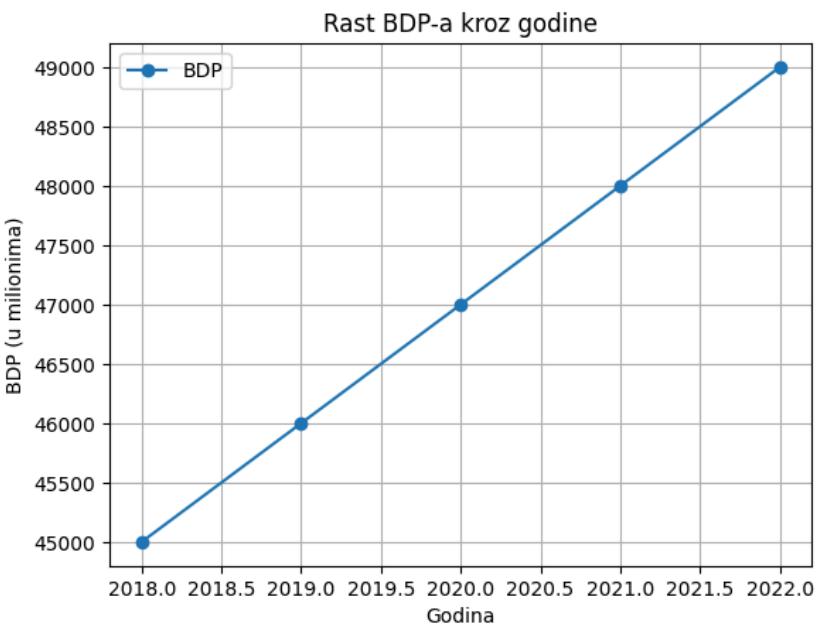
Nakon instalacije, ovi moduli postaju dostupni za korištenje u Python programima. Na primjer, `NumPy` omogućava efikasan rad s velikim nizovima i matricama, dok `Matplotlib` pruža alate za kreiranje različitih vrsta grafika i vizualizacija.

Eksterni moduli se često koriste u kombinaciji za rješavanje složenijih problema. Na primjer, možemo koristiti `NumPy` za generisanje podataka i proračune, a `Matplotlib` za vizualizaciju rezultata.

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

# Generisanje podataka
godine = np.array([2018, 2019, 2020, 2021, 2022])
bpd = np.array([45000, 46000, 47000, 48000, 49000])

# Vizualizacija podataka
plt.plot(godine, bpd, marker='o', label="BDP")
plt.title("Rast BDP-a kroz godine")
plt.xlabel("Godina")
plt.ylabel("BDP (u milionima)")
plt.grid(True)
plt.legend()
plt.show()
```



U ovom primjeru, [NumPy](#) se koristi za upravljanje podacima, dok [Matplotlib](#) omogućava vizualizaciju rasta bruto domaćeg proizvoda (BDP) kroz godine. Ovaj način predstavljanja rezultata olakšava donošenje zaključaka i predstavljanje podataka drugim stručnjacima.

Prednosti upotrebe eksternih biblioteka:

- **Specijalizovane funkcije** – Eksterne biblioteke nude funkcionalnosti, koje nisu dostupne u standardnim modulima, kao što su napredne statističke analize, regresioni modeli i optimizacione metode.
- **Jednostavnost upotrebe** – Iako su ove biblioteke napredne, njihov API (Application Programming Interface) je osmišljen tako da olakša rad i ubrza razvoj aplikacija.
- **Kompatibilnost i interoperabilnost** – Mnoge biblioteke se mogu međusobno kombinovati, omogućavajući efikasniju obradu i analizu podataka.

3.5. Osnovni tipovi fajlova

U današnjem poslovnom okruženju, obrada podataka je nezamisliva bez rada sa različitim tipovima fajlova. Možda ste već imali priliku da radite sa ovim formatima, ali u ovom poglavlju ćemo prikazati kako da koristite podatke, koji su zapisani u ovim fajlovima i kako da rezultate obrade podataka ponovo snimite u fajlove za daljne analize ili prikaze. Među najčešće korišćenim formatima u ekonomskim analizama su CSV, Excel, JSON i YAML fajlovi. Ovi formati su postali standard zahvaljujući svojoj jednostavnosti, kompatibilnosti i efikasnosti u čuvanju i razmjeni podataka. U ovom poglavlju obrađujemo osnovne karakteristike ovih fajlova i njihovu primjenu u Python-u.

3.5.1. CSV (Comma-Separated Values)

Ovaj tip fajlova je jedan od najjednostavnijih i najrasprostranjenijih formata za skladištenje tabelarnih podataka. Njihova glavna prednost leži u širokoj kompatibilnosti sa raznim alatima, uključujući Excel, baze podataka i analitičke softvere. Omogućavaju lako čuvanje i razmjenu velikih setova podataka u ljudski čitljivom formatu. Koriste se u raznim domenima, uključujući ekonomiju, gdje se često primjenjuju za skladištenje i razmjenu finansijskih podataka, prodajnih izvještaja i sličnih analiza.

Jedan od glavnih nedostataka CSV fajlova je njihova ograničena mogućnost prezentacije složenih podataka. CSV format ne podržava hijerarhijsku strukturu ili više tipova podataka unutar iste kolone, što može biti problem za složenije analize. Takođe, prilikom dijeljenja ili uvoza podataka često dolazi do problema sa različitim delimitterima ili kodnim rasporedima.

Primjer strukture CSV fajla.

Datum, Prihod, Trošak, Dobit
2023-01-01, 1000, 500, 500
2023-01-02, 1200, 700, 500

Karakteristična ekstenzija (završetak) ovog tipa fajla je '[.csv](#)', pa bi fajl u prethonom primjeru mogli imanovati kao '['podaci.csv'](#)'.

3.5.1.1. Čitanje podataka iz CSV fajla

Kod čitanja sadržaja CSV fajla, važno je razumjeti nekoliko ključnih stvari. Prva svar je delimiter podataka, odnosno kako su podaci unutar reda odvojeni. U CSV formatu delimiter je zapeta (comma-separated), ali se podaci mogu odvojiti i drugim simbolima kao što su tačka-zapeta i slično.

Putanja do fajla mora biti ispravno specificirana. Ako se fajl nalazi u istom direktorijumu kao Python skripta, dovoljno je navesti njegov naziv, na primjer, '['podaci.csv'](#)'. Međutim, za fajlove u drugim direktorijumima potrebno je koristiti apsolutnu putanju, kao što je '['C:/Users/Korisnik/podaci.csv'](#)', ili relativnu putanju (do fajla koji se nalazi na poddirektorijumu direktorijuma na kome se nalazi Python skripta) poput '['..../direktorijum/podaci.csv'](#)'.

Na kraju, ključno je osigurati da fajl bude zatvoren nakon upotrebe. Ovo sprečava greške i osigurava integritet podataka. U Python-u, ovo se lako postiže korištenjem `with` naredbe, koja automatski zatvara fajl, kada se završi rad sa njim.

Primjer čitanja podataka iz CSV fajla.

```
import csv

# čitanje CSV fajla
with open('podaci.csv', 'r') as file:
    reader = csv.reader(file)
    for row in reader:
        print(row)

['Datum', ' Prihod', ' Trošak', ' Dobit']
['2023-01-01', ' 1000', ' 500', ' 500']
['2023-01-02', ' 1200', ' 700', ' 500']
```

Opis koraka:

- 1) **Uvoz modula** – Koristi se `csv` modul koji je ugrađen u Python, specijalizovan za rad sa CSV fajlovima. Nema potrebe za dodatnim instaliranjem ovog modula.
- 2) **Otvaranje fajla** – Fajl `podaci.csv` otvaramo pomoću funkcije `open()`. Argument `'r'` označava da se otvoreni fajl može samo čitati. U ovom režimu nije moguće upisivati vrijednosti u fajl.
- 3) **Kreiranje čitača** – Funkcija `csv.reader()` pretvara sadržaj fajla u objekat, koji možemo ponavljati u petlji. Svaka linija u CSV fajlu se konvertuje u listu vrijednosti. Predefinisan delimited simbol je zapeta `,`. Ako u fajlu korišten drugi simbol, onda je potrebno u funkciji `csv.reader()` definisati parametar `delimiter=';'` koji opisuje novi simbol.
- 4) **Iteracija kroz redove** – Koristeći `for` petlju, prolazimo kroz svaki red u fajlu. Svaki red se ispisuje na konzolu pomoću funkcije `print()`.
- 5) **Zatvaranje fajla** – Koristi se `with` komanda, kako bi se automatski zatvorio fajl nakon izvršavanja bloka koda, osiguravajući sigurnost i efikasnost.

3.5.1.2. Upisivanje podataka u CSV fajl

Kod upisivanja podataka u CSV fajl, primjenjuju se iste logike, koje su ranije objašnjene u dijelu o čitanju podataka, kao što su izbor delimitera i specifikacija

putanje do fajla. Značajno je da delimiter bude dosljedan i jasno definisan u funkciji za upisivanje podataka, dok putanja do fajla mora biti pravilno određena, bilo da je apsolutna ili relativna.

Osim toga, ključno je osigurati da se fajl zatvori nakon što se podaci upišu, kako bi se spriječile greške i oštećenje fajla. Python-ov `with` komanda, kao i kod čitanja, automatski zatvara fajl po završetku rada, čineći proces sigurnim i efikasnim.

Primjer upisivanja podataka u CSV fajl.

```
import csv

# pisanje u CSV fajl
with open('izlaz.csv', 'w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(['Datum', 'Prihod', 'Trošak', 'Dobit'])
    writer.writerow(['2023-01-01', 1000, 500, 500])
    writer.writerow(['2023-01-02', 1200, 700, 500])
```

Rezultat ovog koda je fajl `izlaz.csv` koji sadrži sljedeće podatke:
Datum,Prihod,Trošak,Dobit
2023-01-01,1000,500,500
2023-01-02,1200,700,500

Opis koraka:

- 1) **Uvoz modula** – Koristi se `csv` modul, koji je ugrađen u Python, specijalizovan za rad sa CSV fajlovima. Nema potrebe za dodatnom instalacijom ovog modula.
- 2) **Otvaranje fajla** – Fajl `izlaz.csv` se otvara pomoću funkcije `open()`. Argument `'w'` označava režim pisanja, pri čemu se postojeći sadržaj fajla prepisuje. Ako fajl ne postoji, automatski će biti kreiran.
- 3) **Kreiranje pisača** – Funkcija `csv.writer()` omogućava pisanje podataka red po red u CSV fajl. Predefinisani delimiter je zapeta `,`. Ako se koristi drugi simbol, treba definisati parametar `delimiter=';'`.
- 4) **Upisivanje redova** – Metoda `writer.writerow()` upisuje jedan red podataka u CSV fajl. Svaka vrijednost u redu mora biti predstavljena kao stavka u listi.
- 5) **Zatvaranje fajla** – Upotrebom `with` komande, fajl se automatski zatvara nakon završetka pisanja, čime se osigurava da su svi podaci pravilno zapisani i fajl ostaje konzistentan.

3.5.2. Excel (XLS, XLSX)

Excel je jedan od najpoznatijih i najčešće korišćenih formata za skladištenje tabelarnih podataka. Nastao je kao format Microsoft Excel aplikacije. Excel fajlovi se mogu koristiti i u drugim programima, koji podržavaju rad sa tabelama, poput Google Sheets

ili LibreOffice Calc. Njegova popularnost leži u fleksibilnosti i bogatim funkcijama, koje omogućavaju organizaciju, analizu i vizualizaciju podataka.

Za učitavanje Excel fajlova u Python-u postoji više načina, a najčešće korišćene biblioteke su [pandas](#) i [openpyxl](#). Obje su eksterni moduli koji se moraju prethodno instalirati kako bi mogli da se koriste. Dok je [pandas](#) idealan za naprednu obradu i analizu podataka, [openpyxl](#) pruža jednostavan i direktni način za rad sa Excel fajlovima, posebno za osnovne operacije čitanja i pisanja.

U ovom poglavlju ćemo koristiti [openpyxl](#) za demonstraciju primjera rada sa Excel fajlovima, dok će [pandas](#) biti obrađen u posebnom poglavlju, gdje ćemo detaljnije objasniti njegove napredne mogućnosti za analizu podataka. Ovaj pristup omogućava postupno uvođenje u rad sa Excel fajlovima, počevši od osnovnih funkcionalnosti ka složenijim analizama.

Karakteristične ekstenzije za ovaj tip fajla su '[.xls](#)' i '[.xlsx](#)'. Format '[.xls](#)' predstavlja stariju verziju Excel fajlova i kompatibilan je sa starijim verzijama softvera, dok je '[.xlsx](#)' noviji format koji nudi bolje performanse i veću podršku u modernim alatima. Na primjer, prethodni fajl sa podacima mogao bi se imenovati kao '['podaci.xlsx'](#)'.

3.5.2.1. Čitanje podataka iz Excel fajla

Čitanje podataka iz Excel fajla uz pomoć biblioteke [openpyxl](#) omogućava direktni pristup podacima unutar ćelija radnog lista. Ova biblioteka je jednostavna za korištenje i posebno pogodna za osnovne operacije sa Excel fajlovima. U nastavku je primjer kako učitati i prikazati sadržaj Excel fajla.

```
from openpyxl import load_workbook

# Učitavanje Excel fajla
wb = load_workbook('podaci.xlsx') # Učitava fajl sa nazivom 'podaci.xlsx'
sheet = wb.active # Pristupa aktivnom radnom listu
for row in sheet.iter_rows(values_only=True): # Prolazi kroz sve redove, čitajući
    samo vrijednosti
    print(row) # Ispisuje red po red

('Datum', 'Prihod', 'Trošak', 'Dobit')
(datetime.datetime(2023, 1, 1, 0, 0), 1000, 500, 500)
(datetime.datetime(2023, 1, 2, 0, 0), 1200, 700, 500)
```

Opis koraka:

- 1) **Uvoz modula** – Biblioteka [openpyxl](#) se koristi za rad sa Excel fajlovima. Ovo je eksterni modul, koji je potrebno instalirati prije korištenja, npr. komandnom [pip install openpyxl](#).

- 2) **Učitavanje fajla** – Funkcija `load_workbook()` otvara Excel fajl. U primjeru se koristi fajl sa nazivom `'podaci.xlsx'`. Ako se fajl nalazi u drugom direktorijumu, potrebno je navesti absolutnu ili relativnu putanju do fajla.
- 3) **Pristup radnom listu** – Metoda `wb.active` automatski bira aktivni (glavni) radni list u fajlu. Ovo je radni list koji je bio otvoren posljednji put prilikom čuvanja fajla.
- 4) **Ponavljanje kroz redove** – Funkcija `sheet.iter_rows()` omogućava ponavljanje kroz sve redove radnog lista. Parametar `values_only=True` osigurava da se čitaju samo vrijednosti iz ćelija, bez dodatnih informacija o formatiranju.
- 5) **Ispis redova** – Svaki red se ispisuje u obliku tuple-a, gdje svaka stavka odgovara vrijednosti iz ćelije u tom redu.

Ukoliko se želite pristupiti specifičnim ćelijama, poput `B1` i `B2`, koristeći biblioteku `openpyxl`, potrebno je direktno adresirati ćeliju, kojoj se želite pristupiti. Ovo se postiže unosom adrese ćelije u objekat koji referencira radni list, na primjer `sheet['B1']`. Pristup vrijednosti upisanoj u toj ćeliji omogućava metoda `.value`. Sljedeći primjer pokazuje kako direktno pročitati podatke iz ćelija na radnom listu.

```
from openpyxl import load_workbook

# Učitavanje Excel fajla
wb = load_workbook('podaci.xlsx') # Učitava fajl sa nazivom 'podaci.xlsx'
sheet = wb.active # Pristupa aktivnom radnom listu

# Čitanje specifičnih ćelija
value_b1 = sheet['B1'].value # Čita vrijednost iz ćelije B1
value_b2 = sheet['B2'].value # Čita vrijednost iz ćelije B2

# Ispis vrijednosti
print("Vrijednost u ćeliji B1:", value_b1)
print("Vrijednost u ćeliji B2:", value_b2)

Vrijednost u ćeliji B1: Prihod
Vrijednost u ćeliji B2: 1000
```

3.5.2.2. Upisivanje podataka u Excel fajl

Upisivanje podataka u Excel fajl uz pomoć biblioteke `openpyxl` omogućava jednostavno dodavanje ili izmjenu sadržaja ćelija na radnom listu. U nastavku je primjer kako upisati podatke u specifične ćelije.

```
from openpyxl import Workbook

# Kreiranje novog Excel fajla
wb = Workbook() # Kreira novi Workbook objekat
sheet = wb.active # Pristupa aktivnom radnom listu

# Upisivanje podataka u specifične ćelije
sheet['B1'] = "Prihod" # Upisuje tekst u ćeliju B1
```

```
sheet['B2'] = 1000 # Upisuje broj u ćeliju B2  
# Čuvanje fajla  
wb.save('izlaz.xlsx') # Čuva fajl pod nazivom 'izlaz.xlsx'
```

Opis koraka:

- 1) **Kreiranje Workbook objekta** – Koristimo `Workbook()` iz biblioteke `openpyxl` za kreiranje novog Excel fajla. Ako se želi raditi sa postojećim fajlom, koristiti `load_workbook()` umjesto `Workbook()`.
- 2) **Pristup radnom listu** – Metoda `wb.active` omogućava rad sa aktivnim radnim listom u Workbook objektu.
- 3) **Upisivanje podataka** – Da biste upisali podatke u specifične ćelije, koristite sintaksu `sheet['B1'] = vrijednost`. U ovom primjeru, tekst "Prihod" upisuje se u ćeliju B1, a broj 1000 u ćeliju B2.
- 4) **Čuvanje fajla** – Nakon što su svi podaci upisani, fajl se čuva pomoću metode `wb.save()`, sa nazivom i ekstenzijom, npr. `'izlaz.xlsx'`.

3.5.3. JSON (JavaScript Object Notation)

JSON (JavaScript Object Notation) je lagan i čitljiv format za razmjenu podataka, koji se koristi za strukturisano predstavljanje podataka u obliku ključeva i vrijednosti. U Python-u, JSON se prirodno mapira na strukturu ugrađenog tipa `dict` (rječnika). Ključevi u JSON-u odgovaraju ključevima u Python rječnicima, dok vrijednosti mogu biti jednostavnii tipovi podataka (stringovi, brojevi, booleani) ili složenije strukture poput listi i ugnježdenih objekata.

Za razliku od CSV-a i Excel-a, JSON nije primarno tabelarni format, već je dizajniran za hijerarhijsko skladištenje podataka, što ga čini pogodnim za kompleksnije strukture. Njegova jednostavnost i široka primjena u web aplikacijama i API komunikaciji čine ga ključnim formatom za razmjenu podataka u savremenim softverskim rješenjima.

Fajlovi koje sadrže podatke u json formatu standarno imaju ekstenziju `.json`. U suštini to su tekstualni fajlovi u kojima su podaci organizovani u json strukturi.

3.5.3.1. Čitanje podataka iz JSON fajla

Čitanje JSON fajlova u Python-u je jednostavno zahvaljujući ugrađenoj biblioteci `json`. Ova biblioteka omogućava da sadržaj JSON fajla direktno mapiramo u Python-ove strukture podataka, kao što su rječnici i liste. Biblioteka je dio standarde instalacije Python-a i nije potrebno dodatno instaliranje.

Ukoliko bi prethodni primjer sadržaja fajlova CSV i Excel, zapisali u json strukturi, tada bi fajl podaci.json izgledao kao u sljedećem primjeru.

```
{  
    "datum": "2023-01-01",  
    "prihod": 1000,  
    "trošak": 500,  
    "dubit": 500  
}
```

Posmatrajmo u nastavku primjer učitavanja sadržaja fajla podaci.json u promjenjivu tipa riječnik i štampanje promjenjive podaci.

```
import json  
  
# Čitanje JSON fajla  
with open('podaci.json', 'r') as file:  
    podaci = json.load(file) # Učitava sadržaj JSON fajla kao Python rječnik  
    print(podaci) # Ispisuje učitane podatke  
  
{'datum': '2023-01-01', 'prihod': 1000, 'trošak': 500, 'dubit': 500}
```

Opis koraka:

- 1) **Uvoz modula** – Koristi se `json` modul, koji je ugrađen u Python, pa nije potrebna dodatna instalacija.
- 2) **Otvaranje fajla** – Fajl `podaci.json` otvara se u režimu čitanja (`'r'`) pomoću funkcije `open()`.
- 3) **Učitavanje podataka** – Funkcija `json.load()` konvertuje sadržaj JSON fajla u Python rječnik ili listu, zavisno od strukture podataka.
- 4) **Rad sa podacima** – Nakon učitavanja, podaci se mogu koristiti na isti način kao Python rječnik.

3.5.3.2. Upisivanje podataka u JSON fajl

Pisanje podataka u JSON fajl omogućava lako čuvanje i dijeljenje novi podataka u standardizovanom formatu. Biblioteka `json` nudi metodu `json.dump()` za konvertovanje Python rječnika u JSON format i zapisivanje u fajl.

```
import json  
  
# Podaci koje želimo zapisati  
podaci = {  
    "datum": "2023-01-01",  
    "prihod": 1000,  
    "trošak": 500,  
    "dubit": 500  
}  
  
# Pisanje u JSON fajl  
with open('izlaz.json', 'w') as file:  
    json.dump(podaci, file, indent=4) # Zapisuje podatke sa formatiranjem
```

Opis koraka:

- 1) **Definisanje podataka** – Podaci se predstavljaju u obliku Python rječnika, koji sadrži ključeve i odgovarajuće vrijednosti.
- 2) **Otvaranje fajla** – Fajl `izlaz.json` otvara se u režimu pisanja ('`w`') pomoću funkcije `open()`.
- 3) **Zapisivanje podataka** – Funkcija `json.dump()` konvertuje Python rječnik u JSON format i zapisuje ga u fajl. Parametar `indent=4` dodaje formatiranje za bolju čitljivost.
- 4) **Zatvaranje fajla** – Korišćenje `with` osigurava automatsko zatvaranje fajla nakon zapisivanja podataka.

3.5.4. YAML (Yet Another Markup Language)

YAML je format za razmjenu podataka, koji je srođan JSON-u, ali je dizajniran da bude lakši za čitanje i pisanje ljudima. Ključna razlika između YAML-a i JSON-a je u njegovoj sintaksi. YAML koristi uvlačenje za definisanje struktura, dok JSON koristi zagrade i zapete. Ova karakteristika čini YAML preglednijim i čitljivijim, posebno za kompleksnije hijerarhije podataka.

YAML je često korišćen za konfiguracione datoteke u softverskim aplikacijama. Njegova fleksibilnost i podrška za napredne tipove podataka čine ga odličnim izborom za aplikacije, koje zahtijevaju strukturisane podatke, dok zadržava jednostavnost.

```
- datum: "2023-01-01"
  prihod: 1000
  trošak: 500
  dobit: 500

- datum: "2023-01-02"
  prihod: 1200
  trošak: 700
  dobit: 500
```

3.5.4.1. Čitanje podataka iz YAML fajla

YAML fajlovi se lako učitavaju i konvertuju u Python-ove strukture podataka poput rječnika i listi, čime se omogućava jednostavna manipulacija sadržajem. Funkcija `yaml.safe_load()` osigurava sigurno i efikasno parsiranje sadržaja, konvertujući YAML strukturu u Python-ove tipove podataka.

```
import yaml
# Čitanje YAML fajla
with open('podaci.yaml', 'r') as file:
    podaci = yaml.safe_load(file) # Učitava sadržaj YAML fajla kao Python rječnik
    print(podaci) # Ispisuje učitane podatke
```

```
[{'datum': '2023-01-01', 'prihod': 1000, 'trošak': 500, 'dubit': 500}, {'datum': '2023-01-02', 'prihod': 1200, 'trošak': 700, 'dubit': 500}]
```

Opis koraka:

- 1) **Uvoz modula** – Koristi se `yaml` modul, koji se mora instalirati pomoću komande `pip install pyyaml`.
- 2) **Otvaranje fajla** – Fajl `podaci.yaml` otvara se u režimu čitanja (`'r'`) pomoću funkcije `open()`.
- 3) **Učitavanje podataka** – Funkcija `yaml.safe_load()` konvertuje sadržaj YAML fajla u Python rječnik.
- 4) **Rad sa podacima** – Učitani podaci mogu se koristiti kao i svaki drugi Python rječnik.

3.5.4.2. Upisivanje podataka u YAML fajl

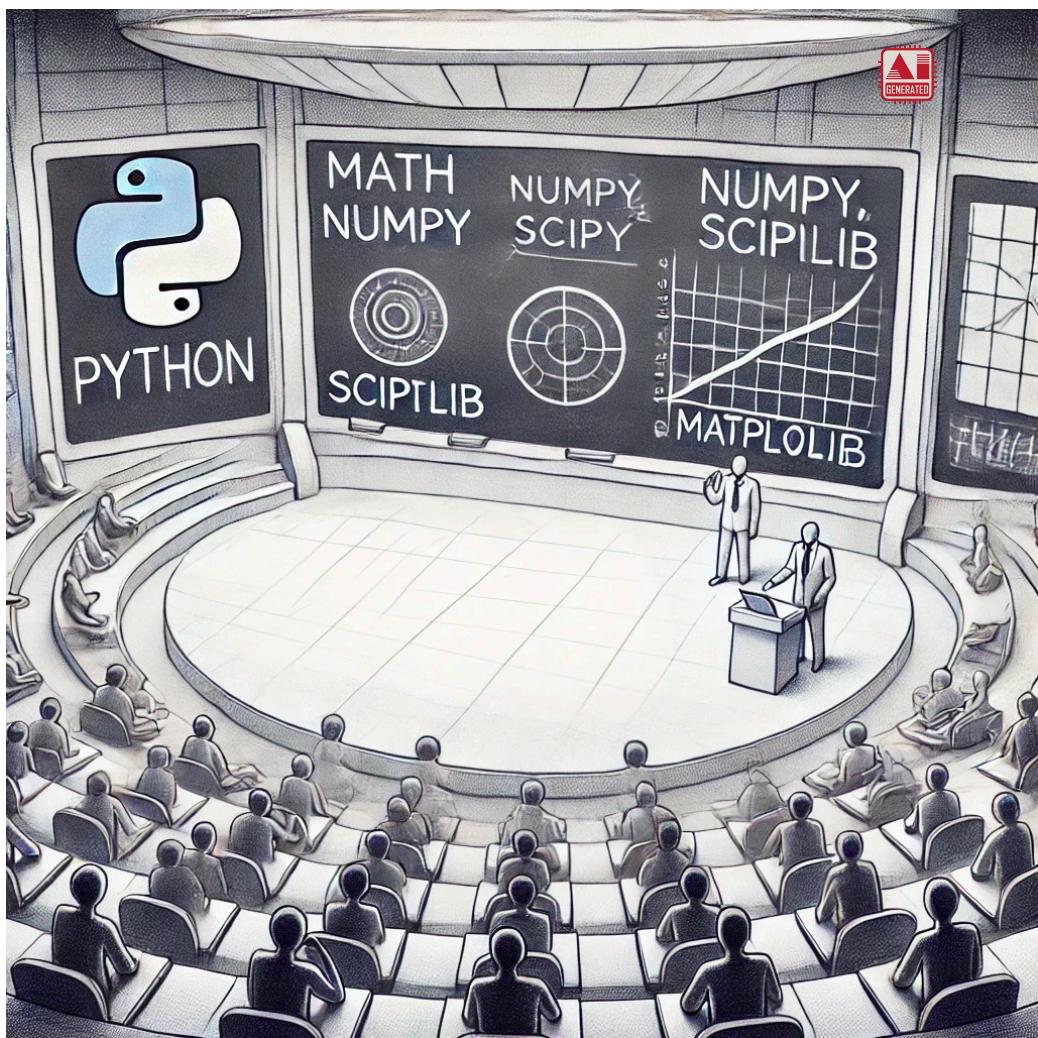
Pisanje podataka u YAML fajl omogućava generisanje čitljivih i strukturisanih fajlova za različite svrhe, kao što su konfiguracije ili izvještaji. Korišćenjem funkcije `yaml.dump()`, Python rječnici ili liste se konvertuju u YAML format sa mogućnošću dodavanja prilagođenog formatiranja za bolju preglednost.

```
import yaml
# Podaci koje želimo zapisati
podaci = {
    "datum": "2023-01-01",
    "prihod": 1000,
    "trošak": 500,
    "dubit": 500
}
# Pisanje u YAML fajl
with open('izlaz.yaml', 'w') as file:
    yaml.dump(podaci, file, default_flow_style=False) # Zapisu u YAML formatu
```

Opis koraka:

- 1) **Definisanje podataka** – Podaci se predstavljaju u obliku Python rječnika.
- 2) **Otvaranje fajla** – Fajl `izlaz.yaml` otvara se u režimu pisanja (`'w'`).
- 3) **Zapisivanje podataka** – Funkcija `yaml.dump()` konvertuje rječnik u YAML format i zapisuje ga u fajl. Parametar `default_flow_style=False` osigurava da se podaci formatiraju na čitljiv način

4. OSNOVNI ALATI



"In God we trust. All others must bring data."
"U Boga vjerujemo. Svi ostali moraju donijeti podatke."
— W. Edwards Deming¹⁵

¹⁵ W. Edwards Deming (1900–1993) bio je američki inženjer, statističar i savjetnik za upravljanje kvalitetom. Njegov rad imao je veliki uticaj na razvoj statističkih metoda u poslovanju, ekonomiji i industriji. Izvor: Oxford Essential Quotations (6 ed.)

Python raspolaže velikim brojem modula i biblioteka, koji se mogu koristiti za različite aspekte analize podataka i modeliranja. Za početno upoznavanje primjene Python-a u ekonomskoj analizi, najznačajniji alati su **Math**, **NumPy**, **SciPy**, **Pandas** i **Matplotlib**. Ove biblioteke pokrivaju širok opseg matematičkih, statističkih i vizuelizacionih potreba. Ovladavanje ovim osnovnim bibliotekama na samom početku omogućava postepeno nadograđivanje znanja i korištenje drugih specijalizovanih rješenja u budućnosti. Upravo zato, u ovo poglavlju ćemo napraviti pregled i praktičnu primjenu ovih ključnih alata za uspješnu i efikasnu ekonomsku analizu u Python-u.

Ključne teme koje će biti obrađene u ovom poglavlju su:

- **Math** – standardna modul za osnovne matematičke operacije i funkcije.
- **NumPy** – je modula sa skupovima alatima za rad sa nizovima i matricama, te osnovne linearne algebarske proračune.
- **SciPy** – je modul koji predstavlja skup naprednih matematičkih i naučnih modula, koji proširuju mogućnosti NumPy-ja, uključujući optimizaciju, integraciju i statistiku.
- **Pandas** - jedna od najpopularnijih Python modula za analizu i manipulaciju podataka.
- **Matplotlib** – najčešće korišten modul za vizualizaciju podataka i rezultata analiza putem raznovrsnih grafika i dijagrama.

4.1. Math modul

U ekonomiji, matematičke funkcije igraju ključnu ulogu u analizi podataka i donošenju odluka. Od osnovnih proračuna kao što su kamatni račun i diskontovanje novčanih tokova, do složenih modela optimizacije i simulacija, razumijevanje matematičkih principa je neophodno za interpretaciju ekonomskih fenomena. Modul Math u Pythonu pruža osnovne alate za izvođenje numerički preciznih matematičkih operacija, što ga čini nezamjenjivim u svakodnevnim analizama. Pod numerički precizne matematičke operacije, podrazumjevaju se operacije sa brojevima, koji imaju visok nivo preciznosti decimalnog dijela broja.

Iako modul Math pruža širok spektar osnovnih matematičkih funkcija, uključujući logaritme, trigonometriju i rad s konstantama, njegova funkcionalnost često služi kao osnova za naprednije biblioteke poput NumPy-a i SciPy-a. NumPy omogućava efikasnu obradu velikih nizova podataka i matričnih operacija, dok SciPy proširuje mogućnosti na numeričku optimizaciju, statističku analizu i rješavanje diferencijalnih jednačina. Kombinacija ovih alata daje ekonomistima moćne alate za analizu složenih podataka i kreiranje modela, koji olakšavaju strateško donošenje odluka.

Modul Math je integriran u osnovnom setu biblioteka, koje se instaliraju prilikom standardne instalacije Python okruženje i za razliku od NumPy, SciPy i Matplotlib-a nije potrebno naknadno je instalirati. Jednostavno prilikom pisanja skripte samo importuje modul, kao što će biti pokazano u primjerima.

U nastavku je pregled osnovnih funkcija i konstanti, koje se koriste u različitim ekonomskim disciplinama, uz objašnjenja i praktične primjene. Kompletan pregled svih funkcija možete naći na zvaničnoj web adresi Python dokumentacije: <https://docs.python.org/3/library/math.html>.

4.1.1. Matematičke konstante

Matematičke konstante, poput π i e , ključne su u mnogim matematičkim modelima i proračunima, koji se koriste u ekonomiji. Konstanta π se često pojavljuje u analizi kružnih fenomena, kao što su sezonski trendovi, dok je e osnova eksponencijalnog rasta, neophodna u modelima kamatnih stopa i populacijskih projekcija.

Sintaksa:

`math.pi` – konstanta $\pi = 3.141592\dots$
`math.e` – osnova prirodnog logaritma $e = 2.718281\dots$

Primjer:

```
import math  
radius = 5
```

```

obim = 2 * math.pi * radius
povrsina = math.pi * radius**2
print("Obim:", obim, "Površina:", povrsina)

```

Obim: 31.41592653589793 Površina: 78.53981633974483

4.1.2. Funkcije za zaokruživanje i absolutne vrijednosti

Veoma često se susrećemo sa situacijama gdje je potrebno zaokružiti broj na najbliži cijeli broj prema gore ili dolje, ili pronaći absolutnu vrijednost broja. Python ima ugrađene funkcije `round()` i `abs()`, koje omogućavaju jednostavno zaokruživanje realnih brojeva i računanje absolutne vrijednosti.

Sintaksa:

`round(x, ndig)` – zaokruživanje broja `x` na ukupno `ndig` decimala
`abs(x)` – absolutna vrijednost broja `x`

Modul Math sadrži dodatne funkcije za zaokruživanje koje su specifične za određene situacije, poput finansijskih proračuna ili optimizacija resursa. Za razliku od `round()` funkcije, `math.ceil()` i `math.floor()` pružaju precizniju kontrolu nad načinom zaokruživanja.

Sintaksa:

`math.ceil(x)` – Zaokružuje broj `x` na najbliži cijeli broj na više.
`math.floor(x)` – Zaokružuje broj `x` na najbliži cijeli broj na niže.
`math.fabs(x)` – Vraća absolutnu vrijednost broja `x`, uvijek kao decimalan broj

Primjer:

```

# Ugrađene funkcije round() i abs()
print(round(3.256,2))      # Rezultat 3.26
print(round(3.256,1))      # Rezultat 3.3
print(round(3.256))        # Rezultat 3
print(abs(-3.256))         # Rezultat 3.256
print(abs(3.256))          # Rezultat 3.256
# Zaokruživanje prema gore
import math
print(math.ceil(4.2))       # Rezultat: 5
print(math.ceil(-3.7))      # Rezultat: -3
# Zaokruživanje prema dole
print(math.floor(4.8))      # Rezultat: 4
print(math.floor(-3.2))     # Rezultat: -4
# Apsolutna vrijednost
print(math.fabs(-5))        # Rezultat: 5.0
print(math.fabs(3.2))        # Rezultat: 3.2

```

4.1.3. Eksponencijalne i logaritamske funkcije

Eksponencijalne i logaritamske funkcije se koriste u mnogim ekonomskim analizama. Funkcija `math.exp` omogućava modeliranje eksponencijalnog rasta, dok funkcije `math.log` i `math.log10` služe za analizu podataka u logaritamskoj skali, što je korisno pri modeliranju trendova i predviđanju ekonomskih pojava. Ove funkcije su osnova za izračunavanje vremenskih perioda potrebnih za rast investicija, analizu kamatnih stopa i procjenu populacijskih trendova.

Sintaksa:

`math.exp(x)` – Eksponencijalna funkcija (e^x).
`math.log(x, base)` – Logaritam broja x sa osnovom `base`.
`math.log10(x)` – Logaritam broja x sa osnovom 10.

Primjer:

```
import math
print(math.log(7.389)) # ln(7.389) ≈ 2
print(math.log(100, 10)) # log10(100) ≈ 2
print(math.log10(1000)) # log10(1000) ≈ 3
print(math.log10(0.1)) # log10(0.1) ≈ -1
```

```
1.9999924078065106
2.0
3.0
-1.0
```

4.1.4. Funkcije za faktore i kombinatoriku

Funkcija `math.factorial` često se koristi za proračune u teoriji vjerovatnoće, dok `math.comb` omogućava jednostavno računanje broja kombinacija, što je ključno za analize izbora u portfolijima investicija, scenarijima odlučivanja i optimizaciji resursa.

Sintaksa:

`math.factorial(x)` – Računa faktorijal broja x .
`math.comb(n, k)` – Predstavlja broj načina da se iz skupa od n elemenata izabere k elemenata bez obzira na redoslijed.

Primjer:

```
# Faktorijal
import math
broj = 5
faktorijal = math.factorial(broj)
print(f"Faktorijal broja {broj} je: {broj}!={faktorijal}") # Rezultat: 120
# Binomni broj
ukupno_elemenata = 10
```

```

izbor = 3
kombinacije = math.comb(ukupno_elemenata, izbor)
print(f"Broj kombinacija za odabir {izbor} elemenata iz skupa od {ukupno_elemenata}
je: {kombinacije}") # Rezultat: 120

Faktorijal broja 5 je: 5!=120
Broj kombinacija za odabir 3 elemenata iz skupa od 10 je: 120

```

4.1.5. Trigonometrijske funkcije

Trigonometrijske funkcije su koriste za analizu cikličnih pojava i modeliranje sezonskih trendova u ekonomiji. Funkcije poput `math.sin`, `math.cos` i `math.tan` omogućavaju proračun sinusnih, kosinusnih i tangensnih vrijednosti uglova izraženih u radijanima, dok `math.degrees` i `math.radians` olakšavaju pretvaranje između stepeni i radijana, što je neophodno u vizualizaciji podataka i proračunima, koji zahtijevaju različite mjere uglova.

Sintaksa:

`math.sin(x)`, `math.cos(x)`, `math.tan(x)` – Sinus, kosinus i tangens ugla u radijanima.
`math.degrees(x)` – Pretvara radijane u stepene.
`math.radians(x)` – Pretvara stepene u radijane.

Primjer:

```

import math

# Sinus, kosinus i tangens ugla u radijanima
ugao_radijani = math.pi / 4 # 45 stepeni u radijanima
sinus = math.sin(ugao_radijani)
kosinus = math.cos(ugao_radijani)
tangens = math.tan(ugao_radijani)

print(f"Sinus od 45° (u radijanima): {sinus}")
print(f"Kosinus od 45° (u radijanima): {kosinus}")
print(f"Tangens od 45° (u radijanima): {tangens}")

# Pretvaranje između stepeni i radijana
ugao_stepeni = 180 # 180 stepeni
ugao_radijani = math.radians(ugao_stepeni)
ugao_stepeni_iz_radijana = math.degrees(math.pi)

print(f"{ugao_stepeni} stepeni u radijanima: {ugao_radijani}")
print(f"{math.pi} radijana u stepenima: {ugao_stepeni_iz_radijana}")

```

```

Sinus od 45° (u radijanima): 0.7071067811865475
Kosinus od 45° (u radijanima): 0.7071067811865476
Tangens od 45° (u radijanima): 0.9999999999999999
180 stepeni u radijanima: 3.141592653589793
3.141592653589793 radijana u stepenima: 180.0

```

4.1.6. Ostale korisne funkcije

Modul Math sadrži i mnoge druge funkcije koje su korisne u različitim matematičkim i ekonomskim analizama. Neke od najčešće korištenih su prikazana u nastavku. Za sve ostale funkcije koje budemo koristili u nastavku knjige pogledajte detaljnije opise u zvanično dokumentaciji na web stranici, koju smo dali na početku poglavlja.

Sintaksa:

- `math.sqrt(x)` – Kvadratni korijen broja `x`.
- `math.pow(x, y)` – Stepenovanje broja `x` na stepen `y` `x^y`.
- `math.gcd(a, b)` – Najveći zajednički djelilac brojeva `a` i `b`.

Primjer:

```
# Kvadratni korijen
import math
broj = 16
korijen = math.sqrt(broj)
print(f"Kvadratni korijen broja {broj} je: {korijen}") # Rezultat: 4.0
# Potenciranje
baza = 2
eksponent = 3
rezultat = math.pow(baza, eksponent)
print(f"{baza} na {eksponent} je: {rezultat}") # Rezultat: 8.0
# Najveći zajednički djelilac
broj1 = 28
broj2 = 35
nzs = math.gcd(broj1, broj2)
print(f"Najveći zajednički djelilac brojeva {broj1} i {broj2} je: {nzs}") #
Rezultat: 7
```

```
Kvadratni korijen broja 16 je: 4.0
2 na 3 je: 8.0
Najveći zajednički djelilac brojeva 28 i 35 je: 7
```

4.1.7. Upotreba modula Math u ekonomiji

U nastavku su prikazana dva jednostavna primjera upotrebe **Math** modula u ekonomskim analizama. U ovim primjerima je korišten samo Math modula i osnovne operacije u Python-u.

Računanje standardne devijacije. Funkcija `math.sqrt` može se koristiti za računanje standardne devijacije kod analize varijanse finansijskih performansi. Modul Math nema ugrađenu funkciju za standardnu devijaciju, pa ćemo u sljedećem primjeru primjeniti opštu formulu za računanje standardne devijacije.

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2}$$

Gdje je:

- σ – standardna devijacija,
- N – ukupan broj elemenata u skupu podataka,
- x_i – pojedinačna vrijednost u skupu podataka,
- μ – srednja vrijednost skupa podataka, koja se računa kao:

$$\mu = \frac{1}{N} \sum_{i=1}^N x_i$$

Primjer računanja standardne devijacije.

```
import math
vrijednosti = [10, 12, 23, 23, 16]
prosjek = sum(vrijednosti) / len(vrijednosti)
varijansa = sum((x - prosjek) ** 2 for x in vrijednosti) / len(vrijednosti)
standardna_devijacija = math.sqrt(varijansa)
print(f"Standardna devijacija: {standardna_devijacija}")

Standardna devijacija: 5.418486873657627
```

Modeli rasta kamate omogućavaju analizu vremena potrebnog da investicija dostigne određeni cilj, uzimajući u obzir početni iznos, kamatu stopu i broj obračunskih perioda. Funkcija `math.log` koristi se za računanje vremenskog parametra u modelima sa diskretnim obračunom kamate.

Za diskretni obračun kamate, formula za rast investicije je:

$$V(t) = V_0 \cdot (1 + r)^t$$

gdje je:

- $V(t)$ – vrijednost investicije nakon t perioda,
- V_0 – početni iznos investicije,
- r – kamatna stopa po periodu,
- t – broj perioda.

Za računanje vremena t , koristi se transformacija logaritmom:

$$t = \frac{\ln\left(\frac{V(t)}{V_0}\right)}{\ln(1+r)}$$

Primjer računanja vremenskog perioda otplate investicije.

```
import math
pocetni_iznos = 1000
konaci_iznos = 2000
kamatna_stopa = 0.05
vrijeme = math.log(konaci_iznos / pocetni_iznos) / math.log(1 + kamatna_stopa)
print(f"Vrijeme potrebno za udvostručenje investicije: {vrijeme} godina")

Vrijeme potrebno za udvostručenje investicije: 14.206699082890461 godina
```

4.2. NumPy Modul

NumPy je modul za rad sa numeričkim podacima u Pythonu, koja omogućava efikasno upravljanje nizovima, vektorima i matricama. Njegove moćne funkcionalnosti čine ga neophodnim alatom za ekonomsku analizu, posebno kada se radi o velikim setovima podataka i kompleksnim proračunima.

NumPy se koristi za:

- Efikasnu obradu podataka: Računanje statistika poput prosjeka, varijanse i standardne devijacije.
- Linearne algebarske operacije: Rješavanje sistema linearnih jednačina, inverzija matrica, determinanti.
- Analizu i modeliranje: Implementacija ekonomskih modela poput input-output analiza.

NumPy pruža osnovnu strukturu pod nazivom `array`, koja je optimizovana za brzu manipulaciju podacima. Dok obične Python liste mogu da čuvaju različite tipove podataka, NumPy nizovi su homogeni, što omogućava brže računanje i manju potrošnju memorije. To znači da svi elementi u nizu moraju biti istog tipa, što omogućava da se podaci efikasnije obrađuju. NumPy koristi optimizirane algoritme za simultanu obradu čitavih nizova.

NumPy je eksterni modul, koji se mora instalirati prije korišćenja u Python skriptama. Instalacija se vrši pomoću menadžera modula `pip`, kao i za ostale eksterne module. Komanda za instalaciju je:

```
pip install numpy
```

Da bi se modul koristio unutar Python skripte, potrebno ga je, nakon instalacije, importovati. Pri importovanju modula NumPy, uobičajeno je dodijeliti mu skraćeno ime, kako bi se pojednostavio pristup njegovim funkcijama. Standardna praksa u Python skriptama je korišćenje generičkih skraćenica, pa se za NumPy koristi `np`, za Pandas `pd`, a za Matplotlib `plt`. NumPy se importuje na sljedeći način.

```
import numpy as np
```

NumPy pruža niz funkcionalnosti, koje omogućavaju efikasnu obradu numeričkih podataka. Ove funkcionalnosti se oslanjaju na rad s vektorima (jednodimenzionalni nizovi), matricama (dvodimenzionalni nizovi) i višedimenzionalnim nizovima. Njihova primjena se može lako povezati s matematičkim modelima, čime se omogućava lakše razumijevanje i praktična primjena u ekonomiji.

U nastavku su predstavljene osnovne funkcionalnosti, koje NumPy pruža. Sve funkcije su ugrađene u modul i koriste se u formatu `np.naziv_funkcije()`, gdje je `np` uobičajena oznaka za importovani NumPy modul. Na ovaj način omogućava se standardizovan i čitljiv način korištenja funkcija u Python skriptama. U ovom poglavlju su objašnjene osnovne funkcije iz NumPy modula, koje će biti korištene u poglavljima koja slijede. To nije kompletan skup funkcija, koje pruža modul. Ostatak funkcionalnosti možete potražiti na web adresi gdje se nalazi zvanična dokumentacija za NumPy modul. (<https://numpy.org/doc/stable/index.html>)

4.2.1. Kreiranje nizova

Funkcija `np.array()` koristi se za kreiranje nizova i predstavlja osnovu za rad u NumPy-u. Ovom funkcijom se definišu vektori, matrice i višedimenzioni nizovi, čime se omogućava organizacija podataka na intuitivan način, koji je prilagođen matematičkim modelima i operacijama.

Vektori (jednodimenzioni nizovi)

U matematici, vektor predstavlja uređeni niz vrijednosti. U Python-u, vektor se kreira pomoću `np.array()` funkcije, pri čemu svi elementi moraju biti istog tipa podataka. Na primjer, prodaja proizvoda u četiri kvartala može se predstaviti kao vektor.

```
import numpy as np
vektor = np.array([10, 20, 30, 40])
print("Vektor:", vektor)
```

Vektor: [10 20 30 40]

Matrice (dvodimenzioni nizovi)

Matrica je tabelarna struktura, koja se sastoji od brojeva raspoređenih u redove i kolone. U ekonomiji, matrice se često koriste za analizu podataka poput prihoda i troškova različitih sektora ili za modeliranje input-output odnosa u proizvodnim procesima. U NumPy-u, matrice se jednostavno definišu pomoću funkcije `np.array()`, koja omogućava organizaciju podataka u ovom formatu na intuitivan i efikasan način.

```
import numpy as np
matrica = np.array([[1, 2], [3, 4]])
print("Matrica:\n", matrica)
```

Matrica:
[[1 2]
 [3 4]]

Višedimenzioni nizovi

Višedimenzioni nizovi predstavljaju strukture koje omogućavaju organizaciju podataka kroz više slojeva ili dimenzija. U ekonomiji se često koriste za analizu kompleksnih podataka, poput prihoda različitih proizvoda, prikazanih po gradovima i godinama. Ova struktura omogućava jasno odvajanje i paralelnu obradu podataka po različitim kriterijumima. U NumPy-u, višedimenzionalni nizovi se takođe kreiraju pomoću funkcije `np.array()`.

```
import numpy as np

# Prihodi proizvoda kroz različite gradove i godine
visedimenzionalni = np.array([
    [[100, 150], [200, 250]], # Prihodi za godinu 1
    [[120, 170], [210, 260]] # Prihodi za godinu 2
])
print("Višedimenzionalni niz:\n", visedimenzionalni)

Višedimenzionalni niz:
[[[100 150]
  [200 250]

  [[120 170]
  [210 260]]]
```

4.2.2. Sumiranje elemenata niza

Funkcija `np.sum()` omogućava efikasno sumiranje elemenata niza u NumPy-u, bilo da se radi o vektorima (jednodimenzionim nizovima), matricama (dvodimenzionim nizovima) ili višedimenzionim nizovima. Sumiranje se može primijeniti na sve elemente niza ili selektivno po određenoj dimenziji, što omogućava prilagodljivost u analizi podataka.

Sintaksa:

```
np.sum(niz, axis=None)
```

Gdje je:

- `niz` – Niz (vektor, matrica ili višedimenzioni niz) čiji zbir treba izračunati.
- `axis` (opciono) – Dimenzija po kojoj se računa suma:
 - `None` (podrazumijevano) – Sabira sve elemente.
 - `axis=0` – Sabira elemente po kolonama.
 - `axis=1` – Sabira elemente po redovima.
- Funkcija vraća:– Skalar ili niz sa sumama po zadatoj dimenziji.

U nasatvku je primjer, koji pokazuje upotrebu funkcije `np.sum` na vektoru i dvodimenzionalnoj matrići.

```

import numpy as np

# Definisanje vektora od 10 članova
vektor = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

# Sumiranje elemenata vektora
suma_vektora = np.sum(vektor)
print("Suma elemenata vektora:", suma_vektora)

# Definisanje matrice dimenzija 6x4
matrica = np.array([
    [1, 2, 3, 4],
    [5, 6, 7, 8],
    [9, 10, 11, 12],
    [13, 14, 15, 16],
    [17, 18, 19, 20],
    [21, 22, 23, 24]
])

# Sumiranje elemenata matrice po kolonama (axis=0)
suma_po_kolonama = np.sum(matrica, axis=0)
print("Suma elemenata po kolonama:", suma_po_kolonama)

# Sumiranje elemenata matrice po redovima (axis=1)
suma_po_redovima = np.sum(matrica, axis=1)
print("Suma elemenata po redovima:", suma_po_redovima)

# Sumiranje svih elemenata matrice
suma_matrice = np.sum(matrica)
print("Suma svih elemenata matrice:", suma_matrice)

```

```

Suma elemenata vektora: 55
Suma elemenata po kolonama: [66 72 78 84]
Suma elemenata po redovima: [10 26 42 58 74 90]
Suma svih elemenata matrice: 300

```

Sljedeći primjer predstavlja višedimenzionu matricu i operacije sumiranja na njoj. Prikazano je kako se funkcija koristi za različite strukture podataka, poput prihoda proizvoda kroz kvartale, gradove i godine. Zamislimo set podataka koji predstavlja prihode ostvarene u dva grada tokom dva kvartala (Kvartal 1 i Kvartal 2) za dvije godine (2023. i 2024.). Ovi podaci mogu se organizovati kao višedimenzionalni niz u NumPy-u, gdje je:

- Prva dimenzija (osa 0) predstavlja godine (2023. i 2024.),
- Druga dimenzija (osa 1) predstavlja gradove (Grad 1 i Grad 2),
- Treća dimenzija (osa 2) predstavlja kvartale (Kvartal 1 i Kvartal 2).

```

import numpy as np

# Definisanje podataka: prihodi kroz kvartale (vektor), gradove (matrica) i
# godine (višedimenzionalni niz)
podaci = np.array([
    [[100, 200], [300, 400]], # Prihodi za godinu 1
    [[150, 250], [350, 450]] # Prihodi za godinu 2
])

```

```

# Sumiranje svih elemenata
ukupni_prihod = np.sum(podaci)
print("Ukupan prihod:", ukupni_prihod)

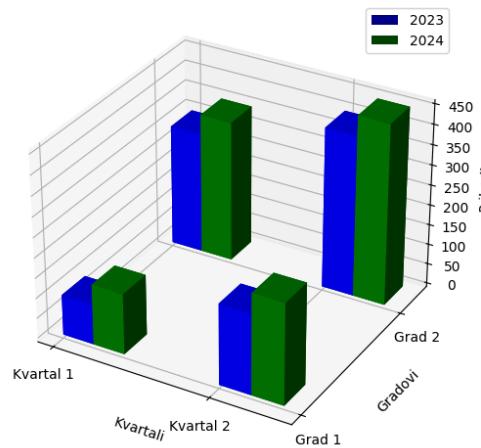
# Sumiranje po gradovima (dimenzija 1)
prihodi_po_gradovima = np.sum(podaci, axis=1)
print("Prihodi po gradovima:\n", prihodi_po_gradovima)

# Sumiranje po kvartalima (dimenzija 2)
prihodi_po_kvartalima = np.sum(podaci, axis=2)
print("Prihodi po kvartalima:\n", prihodi_po_kvartalima)

```

Ukupan prihod: 2200
 Prihodi po gradovima:
 [[400 600]
 [500 700]]
 Prihodi po kvartalima:
 [[300 700]
 [400 800]]

Prihodi po kvartalima, gradovima i godinama



Objašnjenje:

Ukupno sumiranje: Funkcija `np.sum()` bez dodatnih parametara računa zbir svih elemenata u nizu. U primjeru, to je ukupni prihod iz svih godina, gradova i kvartala.

Sumiranje po osi: Dodavanjem parametra `axis` funkciji, možemo specifično sumirati elemente:

- `axis=1` – Sumira elemente po gradovima, omogućavajući analizu prihoda za svaki kvartal u svim godinama.
- `axis=2` – Sumira elemente po kvartalima, pružajući pregled ukupnih prihoda za svaki grad tokom svih godina.

4.2.3. Prosječna vrijednost niza

Funkcija `np.mean()` omogućava računanje aritmetičke srednje vrijednosti elemenata u nizovima. Ova funkcija može se primijeniti na sve elemente niza ili selektivno po određenim dimenzijama (osama), što je korisno za analizu prosječnih vrijednosti u različitim kontekstima.

Sintaksa:

```
np.mean(niz, axis=None)
```

Gdje je:

- `niz` – Niz (vektor, matrica ili višedimenzioni niz) čiji prosjek treba izračunati.
- `axis` (opciono) – Dimenzija po kojoj se računa prosjek:
 - `None` (podrazumijevano) – Računa prosjek svih elemenata.
 - `axis=0` – Računa prosjek po kolonama.
 - `axis=1` – Računa prosjek po redovima.
- Funkcija vraća – Skalar ili niz sa prosjecima po zadatoj dimenziji.

Primjenimo funkciju na prethodni primjer.

```
import numpy as np

podaci = np.array([
    [[100, 200], [300, 400]], # Prihodi za 2023.
    [[150, 250], [350, 450]] # Prihodi za 2024.
])

# Prosjek svih elemenata
ukupni_prosjek = np.mean(podaci) print("Ukupan prosjek prihoda:", ukupni_prosjek)

# Prosjek po kvartalima za svaki grad
prosjek_po_gradovima = np.mean(podaci, axis=2) print("Prosjek prihoda po gradovima:\n", prosjek_po_gradovima)

# Prosjek po gradovima za svaki kvartal
prosjek_po_kvartalima = np.mean(podaci, axis=1) print("Prosjek prihoda po kvartalima:\n", prosjek_po_kvartalima)

# Prosjek po gradovima i kvartalima
prosjek_po_godinama = np.mean(podaci, axis=(1, 2)) print("Prosjek prihoda po godinama:", prosjek_po_godinama)

Ukupan prosjek prihoda: 275.0
Prosjek prihoda po gradovima:
[[150. 350.]
 [200. 400.]]
Prosjek prihoda po kvartalima:
[[200. 300.]
 [250. 350.]]
Prosjek prihoda po godinama: [250. 300.]
```

4.2.4. Jedinična (identična) matrica

NumPy nudi moćne alate za rad s linearom algebrrom, između ostalog i praktičnu funkciju `np.eye()`, pomoću koje jednostavno možemo kreirati jediničnu (identičnu) matricu. Podsećamo da je riječ o posebnoj vrsti dijagonalne matrice, kod koje su svi elementi na glavnoj dijagonali jednaki 1, dok su svi ostali elementi 0. Jedinična matrica je od presudnog značaja u mnogim matematičkim operacijama, budući da prilikom množenja ne mijenja izgled i vrijednost drugih matrica (djeluje kao “broj 1” u množenju realnih brojeva).

Sintaksa:

```
np.eye(n)
```

Gdje je:

- `n` – veličina (broj redova i kolona) kvadratne jedinične matrice.
- Funkcija vraća jediničnu matricu dimenzija `n×n`.

U primjeru ispod pokazaćemo kako kreiramo i ispisujemo jediničnu matricu dimenzija 4×4 .

```
import numpy as np

# Definisanje jedinične matrice 4x4
jedinicna_matrica = np.eye(4)

# Ispis matrice
print("Jedinična matrica 4x4:\n", jedinicna_matrica)
```



```
Jedinična matrica 4x4:
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
```

Kao što vidimo, na glavnoj dijagonali (od gornjeg lijevog do donjeg desnog ugla) svi elementi su `1`, dok su izvan dijagonale svi elementi `0`, što u potpunosti odgovara definiciji jedinične (identične) matrice.

4.2.5. Inverzija matrice

NumPy pruža moćne alate za linearu algebru kroz modul `np.linalg`, uključujući funkciju `np.linalg.inv()`, koja se koristi za računanje inverzne matrice. Inverzna matrica je ključna za mnoge matematičke operacije, uključujući rješavanje sistema

linearnih jednačina. Da bi matrica imala inverziju, ona mora biti kvadratna (isti broj redova i kolona) i njena determinanta mora biti različita od nule.

Sintaksa:

```
np.linalg.inv(matrica)
```

Gdje je:

- **matrica** – Kvadratna matrica čiji se inverzija računa.
- Funkcija vraća inverznu matricu, ako postoji.

Posmatrajmo matricu dimenzija 2×2 i izračunajmo njenu inverznu matricu.

```
import numpy as np

# Definisanje matrice
matrica = np.array([[1, 2], [3, 4]])

# Računanje inverzne matrice
inverzna_matrica = np.linalg.inv(matrica)

# Ispis originalne matrice i njene inverzne
print("Originalna matrica:\n", matrica)
print("Inverzna matrica:\n", inverzna_matrica)
```

Originalna matrica:
[[1 2]
 [3 4]]
Inverzna matrica:
[[-2. 1.]
 [1.5 -0.5]]

4.2.6. Determinanta matrice

Funkcija `np.linalg.det()` koristi se za računanje determinante matrice. Determinanta je broj povezan s matricom, koji ima značajnu ulogu u linearnoj algebri. Ako je determinanta matrice različita od nule, matrica je invertibilna (ima inverznu matricu). U suprotnom, matrica nije invertibilna i ne može se koristiti za rješavanje određenih sistema linearnih jednačina.

Sintaksa:

```
np.linalg.det(matrica)
```

Gdje je:

- **matrica** – Kvadratna matrica čiji se inverzija računa.
- Funkcija vraća skalar (determinanta matrice).

Posmatrajmo matricu dimenzija 2×2 i izračunajmo njenu determinantu.

```
import numpy as np

# Definisanje matrice
matrica = np.array([[1, 2], [3, 4]])

# Računanje determinante matrice
determinanta = np.linalg.det(matrica)

# Ispis originalne matrice i njene determinante
print("Originalna matrica:\n", matrica)
print("Determinanta matrice:", determinanta)
```

Originalna matrica:
[[1 2]
[3 4]]
Determinanta matrice: -2.000000000000004

4.2.7. Rang matrice

Rang matrice je broj nezavisnih redova ili kolona matrice. Rang igra ključnu ulogu u analizi linearnih sistema, jer pokazuje da li je sistem jednačina konzistentan i da li ima jedinstveno rješenje. U NumPy-u, rang matrice se računa pomoću funkcije `np.linalg.matrix_rank()`.

Sintaksa:

```
np.linalg.matrix_rank(matrica)
```

Gdje je:

- `matrica` – Matrica čiji se rang računa.
- Funkcija vraća broj linearno nezavisnih redova ili kolona matrice.

Definišimo jednostavnu matricu i izračunajmo njen rang.

```
import numpy as np

# Definisanje matrice
matrica = np.array([[1, 2, 3],
                    [4, 5, 6],
                    [7, 8, 9]])

# Računanje ranga matrice
rang = np.linalg.matrix_rank(matrica)

# Ispis originalne matrice i njenog ranga
print("Originalna matrica:\n", matrica)
print("Rang matrice:", rang)
```

```
Originalna matrica:  
[[1 2 3]  
 [4 5 6]  
 [7 8 9]]  
Rang matrice: 2
```

4.2.8. Transponovanje matrice

Transponovanje matrice podrazumijeva zamjenu redova i kolona. Drugim riječima, element koji se nalazi u i-tom redu i j-toj koloni u originalnoj matrici prelazi u j-ti red i i-tu kolonu u transponovanoj matrici. U NumPy-u, za transponovanje matrice koristi se funkcija `np.transpose()`. Ova funkcija je korisna za operacije, koje zahtijevaju promjenu strukture matrice, poput množenja matrica ili preračunavanja podataka.

Sintaksa:

```
np.transpose(matrica)
```

Gdje je:

- `matrica` – Matrica koja se transponuje.
- Funkcija vraća transponovanu matricu.

Primjer računaja transponovane matrice.

```
import numpy as np  
  
# Definisanje matrice  
matrica = np.array([[1, 2, 3],  
                    [4, 5, 6]])  
  
# Računanje transponovane matrice  
transponovana_matrica = np.transpose(matrica)  
  
# Ispis originalne i transponovane matrice  
print("Originalna matrica:\n", matrica)  
print("Transponovana matrica:\n", transponovana_matrica)
```

```
Originalna matrica:  
[[1 2 3]  
 [4 5 6]]  
Transponovana matrica:  
[[1 4]  
 [2 5]  
 [3 6]]
```

4.3. SciPy modul

SciPy je jedan od najvažnijih Python modula za naučne i inženjerske proračune. Nadovezuje se na **NumPy**, nudeći napredne algoritme i matematičke metode za optimizaciju, integraciju, interpolaciju, analizu signala i rješavanje diferencijalnih jednačina. U ekonomiji se SciPy može koristiti za modeliranje, analizu podataka i rješavanje složenih problema.

SciPy je složen modul, koji sadrži mnoge različite podmodule namenjene primjeni u raznim naučnim disciplinama. U nastavku su prikazani osnovni podmoduli i njihove funkcionalnosti, koje se mogu koristiti u različitim oblastima ekonomije. Pregled ostalih funkcija koje pruža modul možete potražiti na web adresi <https://docs.scipy.org/doc/scipy/index.html> gdje se nalazi zvanična dokumentacija za SciPy modul.

4.3.1. Optimizacija

Modul za minimizaciju funkcija, rješavanje nelinearnih jednačina i optimizaciju. Ove metode su ključne za analizu i optimizaciju ekonomskih modela, poput minimizacije troškova, maksimizacije profita ili pronalaženja ravnotežne tačke u tržišnim modelima.

Sintaksa:

```
scipy.optimize.minimize(fun, x0, method, options)
```

Gdje je:

- **fun** – Funkcija koja se minimizuje.
- **x0** – Početna tačka.
- **method** – Metod optimizacije (npr. 'BFGS').
- **options** – Dodatne opcije, kao što su tolerancija i maksimalni broj iteracija.

Primjer optimizacija troškova proizvodnje. Funkcija troška opisuje ukupne troškove proizvodnje određene količine proizvoda. Cilj je pronaći količinu proizvodnje, koja minimizira trošak.

```
from scipy.optimize import minimize

# Funkcija troška
trosak = lambda x: (x - 2)**2 + 100

rezultat = minimize(trosak, x0=0)
print("Optimalna količina proizvodnje:", rezultat.x)
print("Minimalni trošak:", rezultat.fun)
```

```
Optimalna količina proizvodnje: [1.99999955]
Minimalni trošak: 100.0000000000002
```

U ovom primjeru, rezultat vraća optimalnu količinu proizvodnje (x) i minimalnu vrijednost troškova (fun).

4.3.2. Integracija

Numeričko rješavanje definisanih integrala i diferencijalnih jednačina. Koristi se za računanje kumulativnih vrijednosti, kao što su ukupni prihod ili trošak tokom određenog perioda.

Sintaksa:

```
scipy.integrate.quad(func, a, b)
```

Gdje je:

- `func` – Funkcija koja se integriše.
- `a, b` – Donja i gornja granica integracije.

Računanje ukupnih prihoda kroz vremenski period. Funkcija prihoda definisana je kao kvadrat vremena (t), a cilj je izračunati ukupni prihod na intervalu od 0 do 5 godina.

```
from scipy.integrate import quad

# Funkcija prihoda
prihod = lambda t: 100 * t**2

ukupni_prihod, greska = quad(prihod, 0, 5)
print("Ukupni prihod:", ukupni_prihod)
```

Ukupni prihod: 4166.666666666668

Rezultat pokazuje ukupni prihod kao integral funkcije prihoda, dok se "greska" odnosi na procijenjenu numeričku grešku.

4.3.3. Interpolacija

Procjena vrijednosti između poznatih tačaka koristeći različite metode interpolacije. Ovo je korisno u ekonomiji za procjenu podataka, koji nedostaju ili za predviđanje trendova.

Sintaksa:

```
scipy.interpolate.interp1d(x, y, kind)
```

Gdje je:

- `x, y` – Poznate tačke.
- `kind` – Tip interpolacije ('linear', 'cubic', itd.).

Primjer procjene BDP-a za godine bez dostupnih podataka. Na osnovu poznatih vrijednosti BDP-a za nekoliko godina, interpolacija omogućava procjenu vrijednosti za 2008. godinu.

```
from scipy.interpolate import interp1d
godine = [2000, 2005, 2010, 2015]
BDP = [1000, 1500, 2000, 2500]
interpolacija = interp1d(godine, BDP, kind='linear')
print("Procijenjeni BDP za 2008. godinu:", interpolacija(2008))
Procijenjeni BDP za 2008. godinu: 1800.0
```

Ova metoda procjenjuje vrijednost BDP-a linearno između 2005. i 2010. godine.

4.3.4. Linearna algebra

Napredne funkcije za rad sa matricama omogućavaju efikasno rješavanje sistema linearnih jednačina, pronalaženje sopstvenih vrijednosti i primjenu dekompozicija. Ove metode igraju ključnu ulogu u ekonomiji, posebno u analizi i optimizaciji resursa, jer pomažu u strukturiranju i pojednostavljinju složenih problema, koji uključuju višestruke zavisnosti i ograničenja.

Sintaksa:

```
scipy.linalg.solve(a, b)
```

Gdje je:

- `a` – Koeficijenti sistema jednačina.
- `b` – Vektori rezultata.

Primjer alokacija resursa. Sistem linearnih jednačina modelira optimalnu alokaciju resursa u dva sektora proizvodnje.

```
from scipy.linalg import solve
A = [[2, 1], [1, 3]] # Koeficijenti
B = [100, 200] # Resursi
```

```
rjesenje = solve(A, B)
print("Optimalna alokacija resursa:", rjesenje)

Optimalna alokacija resursa: [20. 60.]
```

Rezultat prikazuje količinu resursa, koja treba biti dodijeljena svakom sektoru.

4.3.5. Numerički izvodi

Izračunavanje prvog, drugog i viših redova izvoda funkcija. Prvi izvod predstavlja brzinu promjene funkcije (npr. marginalni trošak), dok drugi izvod daje informacije o ubrzanju promjene (konkavnosti ili konveksnosti). Viši redovi izvoda koriste se za analizu kompleksnijih obrazaca promjena.

Sintaksa:

```
scipy.misc.derivative(func, x0, dx, n)
```

Gdje je:

- **func** – Funkcija za koju se računa derivacija (izvod).
- **x0** – Tačka u kojoj se računa derivacija (izvod).
- **dx** – Korak za numeričku derivaciju.
- **n** – Red izvoda.

Primjer računanja marginalnog troška. Funkcija troška definiše ukupne troškove proizvodnje, a derivacija izračunava marginalni trošak.

```
from scipy.misc import derivative

# Funkcija ukupnog troška
trosak = lambda x: x**3 + 2*x**2 + 10

marginalni_trosak = derivative(trosak, 5.0, dx=1e-6)
print("Marginalni trošak za proizvodnju 5 jedinica:", marginalni_trosak)

Marginalni trošak za proizvodnju 5 jedinica: 95.0000001594344
```

Napomena: Funkcija `scipy.misc.derivative` je zastarjela u verziji SciPy 1.10.0 i biće potpuno uklonjena u verziji 1.12.0. Preporučuje se prelazak na korištenje biblioteka poput:

- **findiff** (<https://github.com/maroba/findiff>) ili
- **numdifftools** (<https://github.com/pbrod/numdifftools>) za računanje numeričkih izvoda.

4.3.6. Diferencijalne jednačine

Numeričko rješavanje diferencijalnih jednačina omogućava simulaciju dinamičkih sistema u ekonomiji, kao što su rast investicija, promjena potrošnje ili predviđanje tržišnih trendova. Modul `scipy.integrate.solve_ivp` pruža fleksibilan način za rješavanje ovakvih problema.

Sintaksa:

```
scipy.integrate.solve_ivp(fun, t_span, y0, t_eval)
```

Gdje je:

- `fun` – Funkcija diferencijalne jednačine.
- `t_span` – Interval vremena (početak i kraj).
- `y0` – Početni uslovi.
- `t_eval` – Vrijednosti vremena za evaluaciju rješenja.

Primjer modela eksponencijalnog rasta investicija. Ovaj primjer ilustruje kako investicije rastu eksponencijalno tokom vremena uz konstantnu stopu rasta.

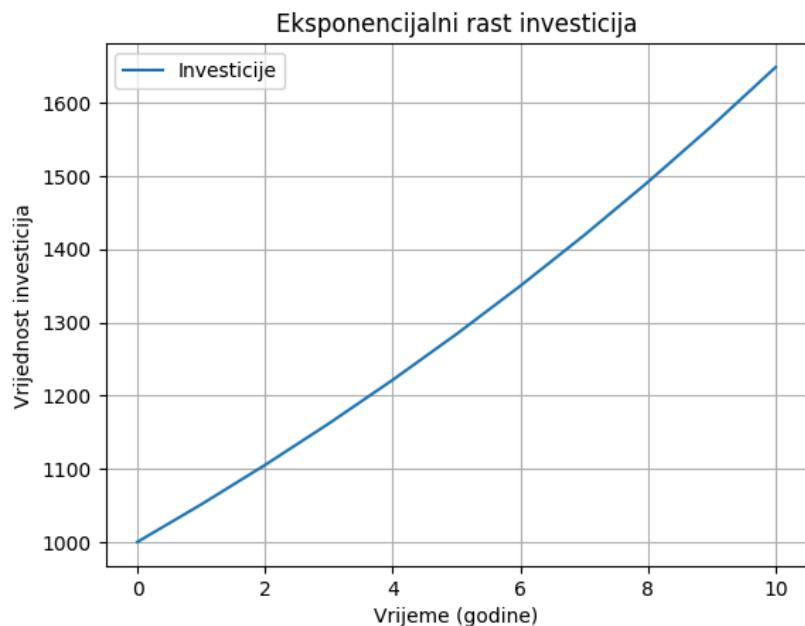
```
from scipy.integrate import solve_ivp

# Diferencijalna jednačina: dI/dt = r * I
rast = lambda t, I: 0.05 * I # Stopa rasta 0.05

rezultat = solve_ivp(rast, [0, 10], [1000], t_eval=range(0, 11))
print("Vrijednosti investicija kroz vrijeme:", rezultat.y[0])

# Vizualizacija rasta
import matplotlib.pyplot as plt
plt.plot(rezultat.t, rezultat.y[0], label="Investicije")
plt.title("Eksponencijalni rast investicija")
plt.xlabel("Vrijeme (godine)")
plt.ylabel("Vrijednost investicija")
plt.legend()
plt.grid()
plt.show()

Vrijednosti investicija kroz vrijeme: [1000. 1051.27109807 1105.17091822
1161.83498632 1221.40448744 1284.02740756 1349.86029476 1419.06825919 1491.8249731
1568.31267083 1648.72214882]
```



Ovaj primjer jasno pokazuje kako investicije rastu eksponencijalno kroz posmatrano vrijeme.

4.4. Matplotlib modul

Matplotlib je moćan Python modul, koji omogućava kreiranje grafika i vizualizaciju podataka. Njegova primjena u ekonomiji je posebno značajna, jer olakšava analizu složenih podataka i omogućava njihovo jasno i pregledno predstavljanje u vidu grafika. Ekonomisti i istraživači koriste Matplotlib za praćenje ekonomskih trendova, vizualizaciju finansijskih podataka i donošenje odluka na osnovu grafičkih prikaza.

Matplotlib je sveobuhvatan alat sa brojnim podmodulima i funkcionalnostima, koji omogućava generisanje raznovrsnih grafičkih prikaza, od osnovnih grafika do naprednih vizualizacija. U ovom poglavlju fokusiraćemo se na funkcionalnosti relevantne za crtanje grafika, koji će se koristiti u narednim poglavljima posvećenim specifičnim ekonomskim disciplinama. Za dodatne informacije o mogućnostima Matplotlib-a, čitaoci mogu posjetiti zvaničnu stranicu na adresi: www.matplotlib.org.

Osnovne karakteristike Matplotlib-a:

- Generiše statičke dijagrame, koji se mijenjaju promjenom ulaznih podataka i ponovnim pokretanjem skripte.
- Širok spektar tipova grafika: linijski grafici, stubići, pie dijagrami, histogrami i mnogi drugi.
- Jednostavan za upotrebu: Omogućava brzo kreiranje osnovnih grafika sa samo nekoliko linija koda.
- Visoka prilagodljivost: Može se detaljno podešavati izgled grafika prema svojim potrebama.

Primjena Matplotlib-a u ekonomiji se ogleda u njegovoj sposobnosti da prikaže ključne informacije, poput kretanja prihoda, troškova, BDP-a, javnog duga i drugih važnih pokazatelja. Grafici generisani ovom bibliotekom omogućavaju bolju interpretaciju podataka i identifikaciju trendova, koji bi inače ostali neprimjećeni.

4.4.1. Kreiranje osnovnog grafika

Matplotlib nudi poseban modul pod nazivom `pyplot`, koji predstavlja intuitivni interfejs za kreiranje grafika u Pythonu. Koristeći funkcije ovog modula, moguće je jednostavno kreirati i prilagođavati grafike sa samo nekoliko linija koda. Modul omogućava direktnu manipulaciju grafičkim elementima, poput linija i tačaka, uz minimalan napor. Ključne funkcije koje su obavezne u pravljenju grafika su `plot` i `show`.

Sintaksa:

`plt.plot(x, y)` – Koristi se za crtanje linijskog grafika
`plt.show()` – Prikazuje grafik nakon svih podešavanja.

Gdje su:

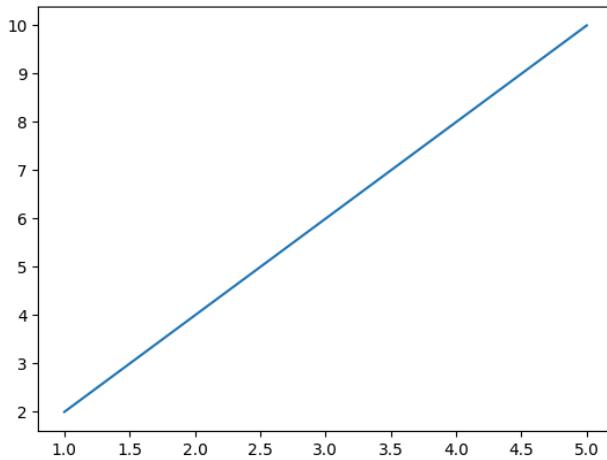
- `x`, `y` liste podataka koje predstavljaju tačke na grafiku.

Kao ilustraciju, pogledajmo jednostavan primjer u kojem ćemo kreirati grafik sa 5 zadatih tačka.

```
import matplotlib.pyplot as plt

# Zadate vrijednosti tačaka
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

# Kreiranje grafika
plt.plot(x, y)
plt.show()
```



Gdje je:

1. Liste `x` i `y`:
 - Lista `x` sadrži vrijednosti, koje se koriste za horizontalnu osu grafika. U ovom primjeru, `x` je niz brojeva od 1 do 5.
 - Lista `y` sadrži vrijednosti za vertikalnu osu, pri čemu su one dvostruko veće od odgovarajućih elemenata liste `x`.
2. Funkcija `plt.plot(x, y)`:
 - Ova funkcija koristi podatke iz lista `x` i `y` kako bi nacrtala linijski grafik. Vrijednosti iz `x` liste postaju tačke na horizontalnoj osi, dok vrijednosti iz `y` liste odgovaraju tačkama na vertikalnoj osi.
3. Funkcija `plt.show()`:
 - Ova funkcija služi za prikazivanje grafika nakon što su svi elementi i podešavanja dodati.

Rezultat ovog koda je linijski grafik, koji prikazuje linearnu zavisnost između `x` i `y` vrijednosti. Ovo je osnova za dalje istraživanje naprednijih tipova grafika.

4.4.2. Prilagođavanje grafika

Dodavanje naslova i oznaka osa grafika

Jedan od osnovnih načina prilagođavanja grafika uključuje dodavanje naslova i oznaka za horizontalnu i vertikalnu osu. Ovo omogućava korisnicima da jasno razumiju šta grafik predstavlja.

Sintaksa:

- `plt.title(label)` – Postavlja naslov grafika.
- `plt.xlabel(label)` – Dodaje naziv za x-osu.
- `plt.ylabel(label)` – Dodaje naziv za y-osu.

Gdje je:

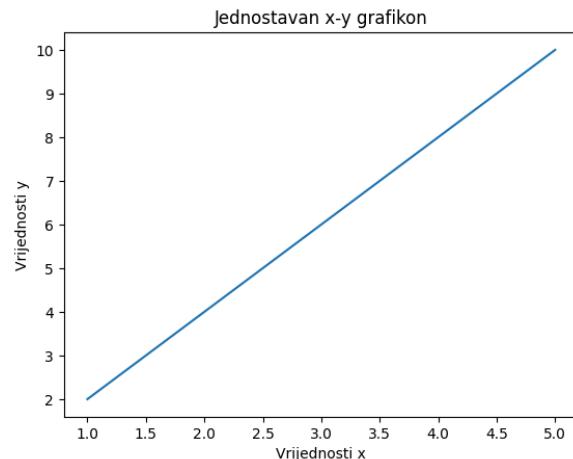
- `label` tekst koji se prikazuje na grafiku.

Uzećemo naš osnovni primjer i dodati naslov i oznake osa.

```
import matplotlib.pyplot as plt

# Zadate vrijednosti tačaka
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

# Kreiranje grafika sa dodatim naslovom i oznakama osa
plt.plot(x, y)
plt.title("Jednostavan x-y grafik")
plt.xlabel("Vrijednosti x")
plt.ylabel("Vrijednosti y")
plt.show()
```



Rezultat je grafik sa jasno označenim naslovom i osama, što olakšava interpretaciju podataka.

Promjena boje linije i stila linije grafika

Matplotlib omogućava jednostavno prilagođavanje boje i stila linije na graficima, čime se povećava preglednost i vizualni utisak. Ovo se postiže korištenjem argumenata `color` i `linestyle` unutar funkcije `plot`.

Gdje je:

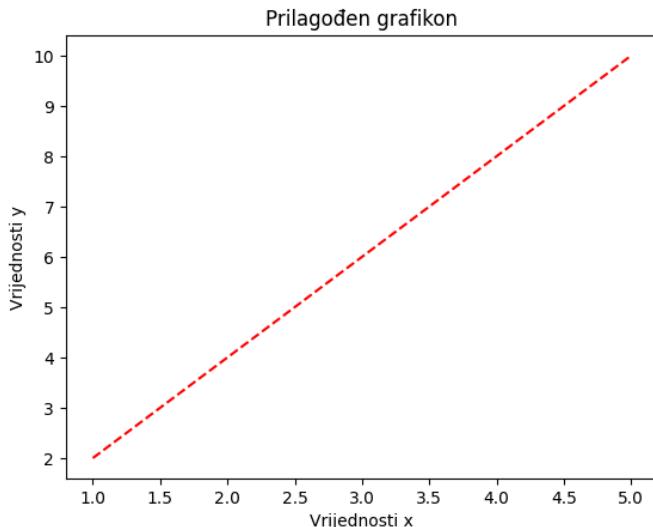
- `color` – Definiše boju linije, npr. `'red'`, `'blue'`, ili heksadecimalni kod boje.
- `linestyle` – Definiše stil linije, npr. `'--'` za isprekidanu liniju, `'-. '` za liniju sa tačkama, ili `'-'` za punu liniju.

Uzećemo naš osnovni primjer i dodati prilagođenu boju i stil linije.

```
import matplotlib.pyplot as plt

# Zadate vrijednosti tačaka
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

# Kreiranje grafika sa prilagođenom bojom i stilom linije
plt.plot(x, y, color='red', linestyle='--')
plt.title("Prilagođen grafik")
plt.xlabel("Vrijednosti x")
plt.ylabel("Vrijednosti y")
plt.show()
```



Ovim podešavanjima grafik postaje vizuelno atraktivniji i jasniji.

Dodavanje mreže i podešavanje fonta grafika

Dodavanje mreže pomaže u boljem praćenju vrijednosti na grafiku, dok podešavanje fontova naslova i oznaka doprinosi profesionalnijem izgledu.

Sintaksa za mrežu grafika:

- `plt.grid(color=, linestyle=, linewidth=)` – Aktivira prikaz mreže na grafiku.
 - `color` – Definiše boju linija mreže, npr. `'gray'`.
 - `linestyle` – Stil linija mreže, npr. `'--'` za isprekidane linije.
 - `linewidth` – Debljina linija mreže, npr. `0.5`.

Sintaksa za fontove:

- `fontsize` – Podešava veličinu fonta naslova, oznaka i legende.
- `fontweight` – Podešava stila fonta, npr. `'bold'` za podebljani tekst.

```
import matplotlib.pyplot as plt

# Zadate vrijednosti tačaka
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

# Kreiranje grafika sa mrežom i prilagođenim fontovima
plt.plot(x, y, color='red', linestyle='--')
plt.title("grafik sa svim podešavanjima", fontsize=14, fontweight='bold')
plt.xlabel("Vrijednosti x", fontsize=12)
plt.ylabel("Vrijednosti y", fontsize=12)
plt.grid(color='gray', linestyle=':', linewidth=0.5)
plt.show()
```



Dodavanje legende i granica za x i y osu grafika

Dodavanje legende i postavljanje granica za x i y osu omogućava dodatnu fleksibilnost i informativnost grafika. Legenda pomaže da se razlikuju linije na grafiku, dok granice definišu opseg podataka koji će biti prikazan. Prikaz legende, odnosno dodatnog opisa podataka ima smisla kada se prikazuje više setova podataka na istom grafiku.

Sintaksa za legendu:

- `plt.legend(loc)` – Prikazuje legendu na grafiku. `loc`: predstavlja poziciju legende, npr. `'upper left'`, `'lower right'`.

Sintaksa za granice osa:

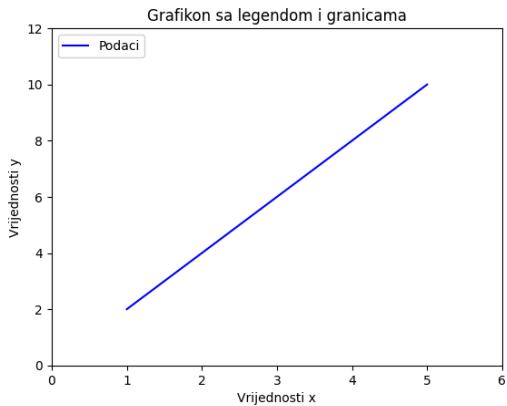
- `plt.xlim(min, max)` – Postavlja minimalnu i maksimalnu vrijednost za x-osu.
- `plt.ylim(min, max)` – Postavlja minimalnu i maksimalnu vrijednost za y-osu.

Primjer upotrebe granica za x i y osu i prikaza legende, odnosno dodatnog opisa podataka na grafiku.

```
import matplotlib.pyplot as plt

# Zadate vrijednosti tačaka
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

# Kreiranje grafika sa legendom i granicama osa
plt.plot(x, y, label="Podaci", color='blue')
plt.title("grafik sa legendom i granicama")
plt.xlabel("Vrijednosti x")
plt.ylabel("Vrijednosti y")
plt.legend(loc='upper left')
plt.xlim(0, 6)
plt.ylim(0, 12)
plt.show()
```



Rezultat je grafik sa legendom, koja označava liniju podataka i postavljenim granicama osa koje definišu prikazani opseg podataka.

4.4.3. Tipovi grafika

Vizualizacija podataka je ključna za analizu i prezentaciju složenih informacija. Matplotlib podržava širok spektar grafika, koji omogućavaju različite načine interpretacije podataka. U nastavku ćemo objasniti najvažnije tipove grafika i njihove primjene, kao i kako se oni mogu kombinovati za detaljniju analizu.

Prije nego što detaljno obradimo svaki tip grafika, biće prikazane ključne funkcije u Matplotlib-u koje omogućavaju njihovo crtanje.

- `plt.plot(x, y)` – Kreira linijski grafik spajanjem tačaka definisanih listama `x` i `y`.
- `plt.bar(x, height)` – Kreira stubičasti grafik, gdje `x` predstavlja kategorije, a `height` njihove vrijednosti.
- `plt.pie(x, labels)` – Kreira tortni dijagram koristeći `x` za vrijednosti i `labels` za oznake kategorija.
- `plt.hist(x, bins)` – Kreira histogram, gdje `x` predstavlja podatke, a `bins` definiše intervale.
- `plt.scatter(x, y)` – Kreira grafik koji prikazuje tačke definisane sa `x` i `y` koordinatama.

Ove funkcije će biti osnova za primjere i analize u nastavku.

4.4.3.1. Linijski grafik

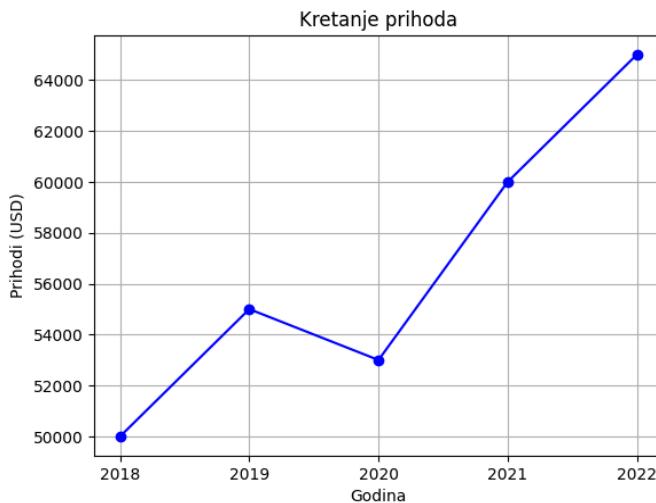
Linijski grafik je idealan za praćenje promjena kroz vrijeme ili za prikaz trendova u podacima. Ovaj tip grafika prezentuje podatke pomoću linija, čineći promjene lako vidljivim.

Primjer:

```
import matplotlib.pyplot as plt

godine = ["2018", "2019", "2020", "2021", "2022"]
prihodi = [50000, 55000, 53000, 60000, 65000]

plt.plot(godine, prihodi, marker='o', linestyle='-', color='blue')
plt.title("Kretanje prihoda")
plt.xlabel("Godina")
plt.ylabel("Prihodi (USD)")
plt.grid(True)
plt.show()
```



4.4.3.2. Grafik u obliku stubova

Ovaj tip grafika se koristi za poređenje veličina različitih kategorija. Svaka kategorija je predstavljena stubom, čija visina odgovara vrijednosti podataka. Mogu da budu vertikalni sa upotrebom funkcije `bar()` ili horizontalni ako se koristi funkciju `barh()`.

Primjer:

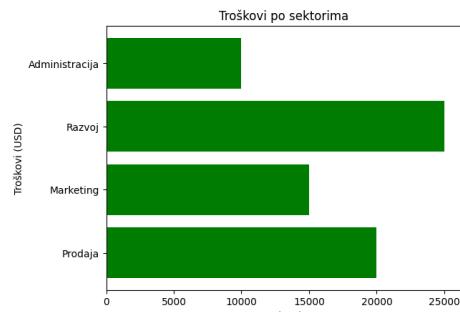
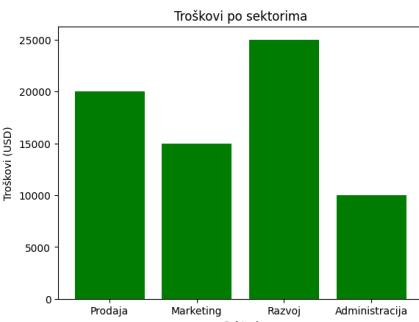
```
import matplotlib.pyplot as plt
kategorije = ["Prodaja", "Marketing", "Razvoj", "Administracija"]
troskovi = [20000, 15000, 25000, 10000]

plt.bar(kategorije, troskovi, color='green')
```

```

plt.title("Troškovi po sektorima")
plt.xlabel("Sektori")
plt.ylabel("Troškovi (USD)")
plt.show()

```



4.4.3.3. Grafik u obliku torte (pie)

Grafik u obliku torte prikazuje relativne udjele različitih kategorija u ukupnoj veličini. Koristi se za analizu procentualnih učešća.

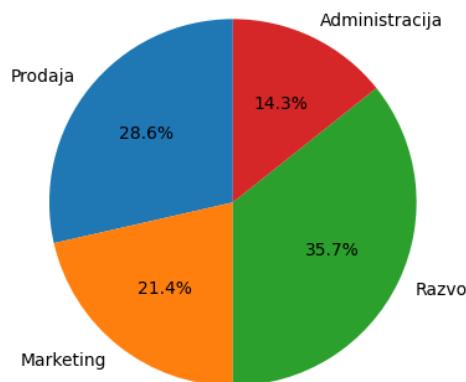
Primjer:

```

import matplotlib.pyplot as plt
kategorije = ["Prodaja", "Marketing", "Razvoj", "Administracija"]
troskovi = [20000, 15000, 25000, 10000]
plt.pie(troskovi, labels=kategorije, autopct='%1.1f%%', startangle=90)
plt.title("Udio troškova po sektorima")
plt.show()

```

Udio troškova po sektorima



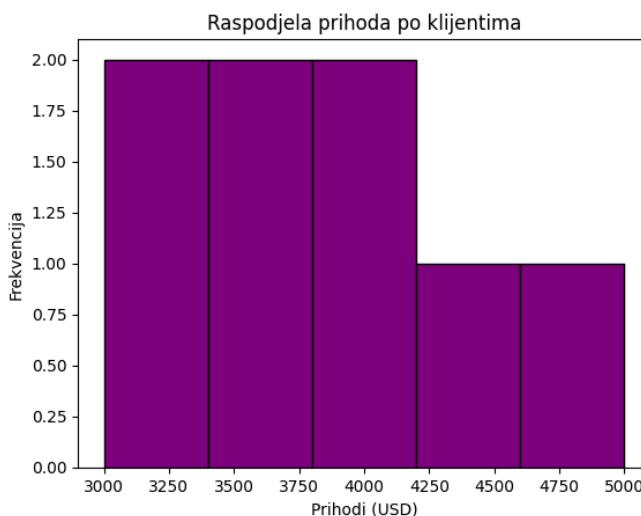
4.4.3.4. Histogram grafik

Histogram grafik prikazuju distribuciju podataka grupisanih u intervale. Koristi se za analizu frekvencija unutar opsega podataka.

Primjer:

```
import matplotlib.pyplot as plt
prihodi_po_klijentima = [3000, 4000, 3500, 3000, 5000, 4500, 4000, 3500]

plt.hist(prihodi_po_klijentima, bins=5, color='purple', edgecolor='black')
plt.title("Raspodjela prihoda po klijentima")
plt.xlabel("Prihodi (USD)")
plt.ylabel("Frekvencija")
plt.show()
```

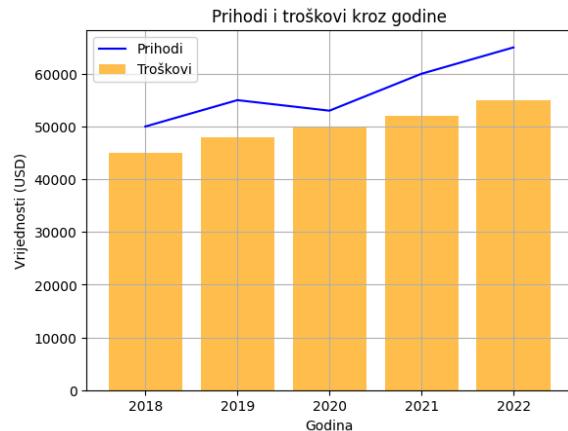


4.4.3.5. Kombinovani grafik

Matplotlib omogućava kombinaciju više grafika u jednoj slici, čime se podaci iz različitih izvora mogu analizirati zajedno.

Primjer:

```
import matplotlib.pyplot as plt
plt.plot(godine, prihodi, label="Prihodi", color='blue')
plt.bar(godine, [45000, 48000, 50000, 52000, 55000], label="Troškovi",
        color='orange', alpha=0.7)
plt.title("Prihodi i troškovi kroz godine")
plt.xlabel("Godina")
plt.ylabel("Vrijednosti (USD)")
plt.legend()
plt.grid(True)
plt.show()
```

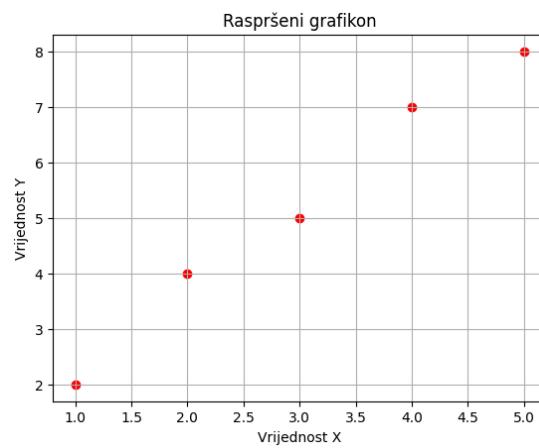


4.4.3.6. Scatter plot (grafik sa dvije varijable)

Scatter plot koristi tačke za prikaz odnosa između dvije varijable. Idealan je za analizu korelacija.

Primjer:

```
import matplotlib.pyplot as plt
x = [1, 2, 3, 4, 5]
y = [2, 4, 5, 7, 8]
plt.scatter(x, y, color='red')
plt.title("grafik sa dvije varijable")
plt.xlabel("Vrijednost X")
plt.ylabel("Vrijednost Y")
plt.grid(True)
plt.show()
```



4.5. Pandas modul

Pandas je Python modul otvorenog koda, koja omogućava rad s tabelarnim i vremenskim podacima na intuitivan način. Modul se oslanja na moće numeričke kapacitete NumPy modula i pruža dodatne funkcionalnosti za strukturiranje, analizu i obradu podataka.

U ovom poglavlju ćemo proći kroz sve segmente rada sa Pandas modulom. Ovaj modul je osnova za sve buduće ekonomske analize, koje ćemo obrađivati u sljedećim poglavljima i primjerima koji prate ta poglavlja.

Zašto koristiti Pandas?

- **Efikasna obrada podataka:** Pandas podržava manipulaciju velikim setovima podataka, koji se ne mogu lako obrađivati u alatima poput Excel-a.
- **Jednostavna integracija:** Kompatibilan je s brojnim formatima datoteka, uključujući CSV, Excel, SQL baze podataka i JSON.
- **Analitička fleksibilnost:** Nudi bogat set funkcionalnosti za filtriranje, grupisanje, agregaciju i vizualizaciju podataka.
- **Skalabilnost:** Pogodan je za jednostavne analize, kao i za kompleksne ekonometrijske modele.

Pandas nudi nekoliko ključnih prednosti, koje ga izdvajaju kao idealan alat za analizu podataka:

- 1) **Intuitivan rad s podacima:** Manipulacija podataka u Pandas-u je slična radu s tabelama u Excel-u, ali uz dodatnu fleksibilnost i brzinu.
- 2) **Automatsko rukovanje nedostajućim podacima:** Pandas olakšava identifikaciju i obradu nedostajućih vrijednosti (“NaN”), bilo da ih treba popuniti ili ukloniti.
- 3) **Napredna analiza:** Podržava grupisanje i agregaciju podataka, što je korisno za analize poput prosječnih zarada po sektoru ili izrade statističkih izvještaja.
- 4) **Jednostavna vizualizacija podataka:** Pandas se lako integrše s bibliotekama kao što su Matplotlib i Seaborn za kreiranje grafika.
- 5) **Automatizacijom obrade podataka:** Ovim se eliminšu postupci, koji se ponavljaju tokom uređivanja i obrade podataka, što omogućava fokusiranje na dovođenje odluka.

Iako je Excel popularan alat za analizu podataka, Pandas nudi značajne prednosti u odnosu na njega. U tabeli 4.1. prikazane su uporedne karakteristike, koje ukazuju na ključne prednosti upotrebe biblioteke Pandas.

Tabela 4.1. Poređenje karakteristika Pandas biblioteke i Excel-a

Karakteristika	Pandas	Excel
Skalabilnost	Podržava velike datasetove, milione redova.	Ograničen broj redova i kolona (1.048.576 redova i 16.384 kolona).
Automatizacija	Skriptabilnost olakšava ponovnu upotrebu koda.	Ručno unošenje ili ograničena automatizacija.
Preciznost analiza	Napredne funkcije za analizu i statistiku.	Ograničene analitičke funkcije. Proširenje se postiže tek sa upotrebom VBA ili integracijom Python-a.
Integracija	Jednostavno povezivanje s bazama podataka.	Složeno povezivanje s vanjskim izvorima podataka. Načće preko ODBC drivera.
Vizualizacija	Moćne opcije vizualizacije uz dodatne biblioteke.	Ograničene opcije vizualizacije, bez mogućnosti proširenja.

U ovom poglavlju će biti obrađeni osnovni koncepti upotrebe Pandas biblioteka, sa primjerima obrade i vizuelizacije podataka. Napredne funkcionalnost možete potražiti na zvaničnoj web stranici: <https://pandas.pydata.org/docs/index.html>

Prije nego što počnemo s opisom osnovnih struktura podataka, važno je napomenuti da je Pandas eksterni modul, koji treba instalirati prije korišćenja. Instalacija se može obaviti korišćenjem sljedeće komande u terminalu ili komandnoj liniji:

```
pip install pandas
```

Nakon instalacije, modul je potrebno uvesti u Python skriptu, kako bi bio dostupan za korišćenje. Uobičajena praksa je da se Pandas uvozi pod simboličkim imenom `pd` radi kraćeg i preglednijeg koda:

```
import pandas as pd
```

Kada je instalacija i uvoz završen, možemo koristiti bogat set funkcionalnosti, koje Pandas nudi za rad s podacima.

4.5.1. Osnovne strukture podataka

Pandas nudi dvije osnovne strukture podataka (Series i DataFrame) koje su intuitivne i moćne za analizu i manipulaciju. Ove strukture predstavljaju osnovu za rad s podacima, omogućavajući efikasan i organizovan pristup složenim setovima podataka. Njihova fleksibilnost i jednostavnost korišćenja čini ih idealnim za analizu ekonomskih i poslovnih podataka.

4.5.1.1. Series

Series predstavlja jednodimenzionalni niz podataka, sličan koloni u tabeli. Svaki element ima indeks, što omogućava brz pristup i manipulaciju podacima. Series objekti su korisni za rad s podacima, koji su povezani s vremenskim ili kategorijskim indeksima, poput cijena akcija ili dnevnih stopa inflacije.

Sintaksa:

```
pd.Series(data=None, index=None, dtype=None, name=None,  
          copy=False)
```

Gdje je:

- **data** – Podaci za kreiranje serije. Može biti lista, niz (array), scalar ili rječnik.
- **index** – Lista ili niz indeksa koji se dodjeljuju elementima serije. Ako nije naveden, kreira se podrazumijevani indeks.
- **dtype** – Tip podataka u seriji (npr. `int`, `float`, `str`). Ako nije naveden, Pandas će ga automatski odrediti.
- **name** – Naziv serije, što olakšava identifikaciju kada je dio DataFrame-a.
- **copy** – Ako je `True`, podaci se kopiraju, a u suprotnom, koristi se referenca na originalne podatke.

Primjer:

```
import pandas as pd  
  
cijene_akcija = pd.Series([150, 152, 153, 155], index=["2023-01-01", "2023-01-02",  
                           "2023-01-03", "2023-01-04"])  
print(cijene_akcija)  
  
2023-01-01    150  
2023-01-02    152  
2023-01-03    153  
2023-01-04    155  
dtype: int64
```

Series je konceptualno sličan Python-ovom tipu podataka Riječnik (dict). Ključevi iz Riječnika odgovaraju indeksima Series objekta, dok vrijednosti u Riječniku postaju “data” Series objekta. Ova strukturalna sličnost omogućava jednostavnu konverziju između ova dva tipa podataka i olakšava rad sa strukturiranim podacima.

U sljedećem primjeru je pokazano kako se pretvara Riječnik u Series.

```
import pandas as pd  
data = {"2023-01-01": 150, "2023-01-02": 152, "2023-01-03": 153, "2023-01-04": 155}  
cijene_akcija = pd.Series(data)  
print(cijene_akcija)
```

```
2023-01-01    150
2023-01-02    152
2023-01-03    153
2023-01-04    155
dtype: int64
```

Ovaj način pretvaranja je posebno koristan za organizovanje i manipulaciju podataka, koji su inicijalno spremljeni kao rječnici, pružajući jednostavan prelaz ka naprednjim funkcionalnostima koje nudi Pandas. Ova konverzija se često koristi kod preuzimanja podataka sa interneta ili kod importovanja iz fajlova.

4.5.1.2. DataFrame

DataFrame je dvodimenzionalna struktura podataka, koja je suštinski tabela s redovima i kolonama. Može sadržavati podatke različitih tipova (npr. brojeve, tekst). DataFrame objekti omogućavaju rad s podacima, koji su organizovani u obliku tabela, što ih čini posebno korisnim za ekonomski analize i finansijske izvještaje.

Sintaksa:

```
pd.DataFrame(data=None, index=None, columns=None, dtype=None, copy=False)
```

Gdje je:

- **data** – Izvor podataka za DataFrame. Može biti lista, rječnik, niz (array), ili drugi DataFrame.
- **index** – Indeksi za redove. Ako nije naveden, automatski se generišu kao cjelobrojni indeksi.
- **columns** – Nazivi kolona. Ako nisu navedeni, koristi se podrazumijevana vrijednost.
- **dtype** – Tip podataka u kolonama. Ako nije naveden, automatski se određuje.
- **copy** – Ako je **True**, kopira se uneseni izvor podataka, inače se koristi referenca na postojeći izvor.

Primjer:

```
import pandas as pd
podaci = {
    "Država": ["BiH", "Hrvatska", "Srbija"],
    "BDP": [20000, 55000, 48000],
    "Populacija": [3.3, 4.0, 7.0]
}
dataframe = pd.DataFrame(podaci)
print(dataframe)
```

	Država	BDP	Populacija
0	BiH	20000	3.3
1	Hrvatska	55000	4.0
2	Srbija	48000	7.0

DataFrame je jedan od najčešće korištenih objekata u Pandas biblioteci. Slično kao i Series, DataFrame može prihvati različite vrste ulaznih podataka, uključujući:

- Rječnik (dict) jednodimenzionalih nizova (ndarray), lista, rječnika ili Series.
- Dvodimenzioni numpy.ndarray.
- Excel tabela: Tabela se prethodno može učitati pomoću Pandas metode `read_excel`, a zatim konvertovati u rječnik za dalje prilagođavanje podataka.
- Drugi DataFrame.

Pored podataka, moguće je opcionalno proslijediti parametre index (oznake redova) i columns (oznake kolona). Ako se definišu index i/ili columns, oni garantuju odgovarajuću strukturu rezultirajućeg DataFrame-a. Na primjer, kada se koristi rječnik Series objekata uz definisan indeks, podaci koji ne odgovaraju proslijđenim indeksima automatski se odbacuju.

Primjer učitavanja podataka iz Excel fajla i pretvaranje u rječnik.

```
import pandas as pd

# Učitavanje podataka iz Excel fajla
excel_data = pd.read_excel("podaci.xlsx")

# Konverzija Excel podataka u rječnik
data_dict = excel_data.to_dict(orient="list")

# Kreiranje DataFrame-a iz rječnika
dataframe = pd.DataFrame(data_dict)
print(dataframe)
```

Ova fleksibilnost čini Pandas moćnim alatom za rad s različitim vrstama ulaznih podataka i omogućava jednostavnu integraciju s postojećim izvorima informacija.

4.5.2. Rad sa Series i DataFrame objektima

Pandas biblioteka omogućava efikasan rad s podacima kroz svoje dvije osnovne strukture: **Series** i **DataFrame**. Ove strukture omogućavaju intuitivno organizovanje i analizu podataka, bilo da se radi sa jednostavnim nizovima ili složenim tabelama. U nastavku ćemo detaljno razraditi kako se kreiraju, inicijalizuju i kako se upravlja ovim objektima, uz praktične primjere koji pokazuju njihovu primjenu u svakodnevnim analizama.

4.5.2.1. Kreiranje Series objekta

Series je osnovna struktura podataka u Pandas-u, koja predstavlja jednodimenzionalni niz podataka povezanih s indeksima. Pored direktnе inicijalizacije koja je pokazana u opisu objekta Series, postoji više načina za inicijalizaciju Seriesa kroz druge tipove podataka. U nastavku ćemo prikazati neke od standardnih metoda kreiranja Series objekata, koje često srećemo u praktičnoj primjeni Python-a.

1. Kreiranje iz liste

```
import pandas as pd

# Kreiranje Series iz liste
lista = [10, 20, 30, 40]
serija_iz_liste = pd.Series(lista, index=["Q1", "Q2", "Q3", "Q4"])
print(serija_iz_liste)
```

Q1	10
Q2	20
Q3	30
Q4	40
dtype:	int64

2. Kreiranje iz numpy niza

```
import numpy as np
import pandas as pd

# Kreiranje Series iz numpy niza
niz = np.array([1.1, 2.2, 3.3, 4.4])
serija_iz_niza = pd.Series(niz, index=["A", "B", "C", "D"])
print(serija_iz_niza)
```

A	1.1
B	2.2
C	3.3
D	4.4
dtype:	float64

3. Kreiranje iz rječnika

```
import pandas as pd

# Kreiranje Series iz rječnika
podaci = {"Jan": 100, "Feb": 150, "Mar": 200}
serija_iz_rjecnika = pd.Series(podaci)
print(serija_iz_rjecnika)
```

Jan	100
Feb	150
Mar	200
dtype:	int64

4.5.2.2. Kreiranje DataFrame objekata

DataFrame je dvodimenziona struktura slična tabeli, koja omogućava organizaciju podataka u redove i kolone. Može se kreirati iz različitih izvora podataka, uključujući:

- **Liste** – Podaci organizovani u obliku lista mogu se koristiti za kreiranje tabela.
- **Rječnici (dict)** – Svaki ključ rječnika predstavlja ime kolone, a vrijednosti su podaci u toj koloni.

- **CSV ili Excel datoteke** – Podaci se mogu direktno učitati iz eksternih datoteka.
- **Baze podataka** – Podaci iz SQL upita mogu se direktno učitati u DataFrame.
- **API izvora** – Podaci se mogu preuzeti sa interneta koriteći konekciju na API servis i pretvoriti u DataFrame.

1. Kreiranje iz lista

```
import pandas as pd

# Kreiranje DataFrame iz lista
podaci = [["BiH", 3.3], ["Hrvatska", 4.0], ["Srbija", 7.0]]
dataframe_iz_liste = pd.DataFrame(podaci, columns=["Država", "Populacija"])
print(dataframe_iz_liste)

Država  Populacija
0      BiH        3.3
1  Hrvatska      4.0
2     Srbija      7.0
```

2. Kreiranje iz rječnika

```
import pandas as pd

# Kreiranje DataFrame iz rječnika
podaci = {
    "Država": ["BiH", "Hrvatska", "Srbija"],
    "BDP": [20000, 55000, 48000]
}
dataframe_iz_rjecnika = pd.DataFrame(podaci)
print(dataframe_iz_rjecnika)

Država    BDP
0      BiH  20000
1  Hrvatska  55000
2     Srbija  48000
```

3. Učitavanje iz CSV/Excel datoteka

```
import pandas as pd

# Učitavanje DataFrame iz CSV datoteke
dataframe_iz_csv = pd.read_csv("podaci.csv")

# Učitavanje DataFrame iz Excel datoteke
# Za ovo je potrebna instalacija dodatka: pip install openpyxl
dataframe_iz_excel = pd.read_excel("podaci.xlsx")
```

4. Preuzimanje podataka iz baze podataka (MySQL)

Podaci iz MySQL baze mogu se direktno učitati u Pandas DataFrame korištenjem biblioteke `pymysql` i metode `read_sql` iz Pandas-a. Prije nego što se podaci preuzmu, potrebno je osigurati odgovarajuće pristupne parametre (kredencijale), koji omogućavaju povezivanje s bazom podataka. U sljedećem primjeru navedeni su

osnovni parametri za pristup (host, korisničko ime, lozinka i naziv baze). Da bi primjer bio funkcionalan, ove parametre treba zamijeniti stvarnim vrijednostima specifičnim za vašu bazu podataka.

Modul `pymysql` je eksterni modul i neophodno ga je instalirati u Python okruženje, gdje se pokreće skripta za komunikaciju sa MySQL bazom. Ukoliko koristimo neku drugu bazu, neophodno je instalirati odgovarajući modula za tu bazu. Komanda za instalaciju modula je:

```
pip install pymysql
```

Primjer upotrebe `pymysql` modula za preuzimanje podataka u DataFrame.

```
import pandas as pd
import pymysql

# Povezivanje s MySQL bazom podataka
konekcija = pymysql.connect(
    host="localhost",
    user="vaše_korisničko_ime",
    password="vaša_lozinka",
    database="ime_baze"
)

# Izvršavanje SQL upita
query = "SELECT * FROM tabela_podaci"
dataframe_iz_baze = pd.read_sql(query, konekcija)

# Zatvaranje konekcije
konekcija.close()

print(dataframe_iz_baze)
```

5. Preuzimanje podataka sa API URL-a

Preuzimanje podataka sa API URL-a omogućava direktni pristup dinamičkim informacijama, koje se redovno ažuriraju na internetu. Ovo je korisno u slučajevima kada se radi sa servisima, koji pružaju podatke o vremenu, finansijskim tržištima, populacijskim statistikama ili bilo kojim drugim domenima dostupnim putem API-ja.

Pandas DataFrame može biti kreiran konvertovanjem odgovora API-ja, često u formatu JSON, u tabularni oblik. Za ovaj proces najčešće se koristi Python-ova biblioteka `requests`.

Koraci za preuzimanje podataka sa API URL-a:

- 1) Slanje zahtjeva API-ju: Pomoću funkcije `requests.get` šalje se zahtjev ka URL-u API-ja.
- 2) Pristup JSON podacima: Odgovor API-ja se često vraća u JSON formatu, a koji je u Python-y prezentovan kao Riječnik (dict).

- 3) Konverzija u DataFrame: JSON podaci se pretvaraju u Pandas DataFrame radi daljnje analize i manipulacije.

Primjer koda sa preuzimanjem podataka sa API servisa.

```

import pandas as pd
import requests

# Definisanje URL-a API-ja
url = "https://opendata.kosava.cloud/1/measurements/last_hour"

# Slanje zahtjeva API-ju
response = requests.get(url)

# Provjera uspješnosti zahtjeva
if response.status_code == 200:
    # Pristup JSON podacima
    response_data = response.json()

    # Konverzija u DataFrame
    dataframe_iz_api = pd.DataFrame(response_data)
    # Podešavanje proširenog prikaza podataka
    pd.set_option("display.width", 1000)
    print(dataframe_iz_api)
else:
    print(f"Neuspješan zahtjev! Kod greške: {response.status_code}")

[366 rows x 8 columns]

```

k_date	k_time	k_network_id	k_network_code	k_station_id	k_station_code	k_component_id	k_aop_value
0	2025-01-05	14:00:00		1	RS100A	29	RS1048A 8 6.468790
1	2025-01-05	14:00:00		3	RS300A	80	RS3023A 5 27.111500
2	2025-01-05	14:00:00		1	RS100A	33	RS1052A 1 17.750892
3	2025-01-05	14:00:00		1	RS100A	33	RS1052A 5 122.848333
4	2025-01-05	14:00:00		3	RS300A	80	RS3023A 6001 22.464500
..
361	2025-01-05	14:00:00		1	RS100A	26	RS1045A 10 0.261617
362	2025-01-05	14:00:00		1	RS100A	26	RS1045A 8 8.939619
363	2025-01-05	14:00:00		1	RS100A	29	RS1048A 1 10.328579
364	2025-01-05	14:00:00		1	RS100A	12	RS1027A 10 0.680495
365	2025-01-05	14:00:00		1	RS100A	29	RS1048A 10 0.698474

4.5.2.3. Pristup i manipulacija podacima

Pandas omogućava fleksibilan i intuitivan pristup podacima, bilo da se radi o pristupanju pojedinačnih redova i kolona ili o izmjenama na postojećim podacima. Koristeći metode poput `iloc` za pozicijski pristup i `loc` za imenovani pristup, može se lako izdvojiti, ažurirati, dodati ili ukloniti podatke. Ove funkcionalnosti su ključne za analizu podataka i pripremu tabela za dalju obradu. U nastavku su prikazani primjeri osnovnih metoda za rad s podacima u Pandas-u.

Pristup redovima ili kolonama pomoću indeksa pozicije - `iloc`

Omogućava pozicioni pristup podacima unutar DataFrame-a ili Series objekta. Koristi se za izdvajanje redova, kolona ili njihovih kombinacija prema njihovom redoslijedu.

Sintaksa:

```
dataframe.iloc[row_indexer, column_indexer]
```

Gdje je:

- `row_indexer` – Indeks ili opseg redova koje se žele izdvojiti.
- `column_indexer` – Indeks ili opseg kolona koje se žele izdvojiti.

Primjer:

```
import pandas as pd

# Kreiranje DataFrame-a
data = {
    "Država": ["BiH", "Hrvatska", "Srbija"],
    "BDP": [20000, 55000, 48000]
}
df = pd.DataFrame(data)

# 1. Dohvaćanje prvog reda
print("Prvi red:")
print(df.iloc[0])

# 2. Dohvaćanje vrijednosti u drugom redu i prvoj koloni
#     \n služi za prelazak u novi red prilikom štampanja rezultata
print("\nVrijednost u drugom redu i prvoj koloni:")
print(df.iloc[1, 0])

# 3. Dohvaćanje svih redova za prvu kolonu
print("\nSvi redovi za prvu kolonu:")
print(df.iloc[:, 0])

# 4. Dohvaćanje svih redova za prve dvije kolone
print("\nSvi redovi za prve dvije kolone:")
print(df.iloc[:, :2])

# 5. Dohvaćanje drugog i trećeg reda
print("\nDrugi i treći red:")
print(df.iloc[1:3])
```

```
Prvi red:
Država      BiH
BDP        20000
Name: 0, dtype: object

Vrijednost u drugom redu i prvoj
koloni:
Hrvatska

Svi redovi za prvu kolonu:
0      BiH
1    Hrvatska
2    Srbija
Name: Država, dtype: object
```

```
Svi redovi za prve dvije kolone:
Država      BDP
0      BiH  20000
1    Hrvatska  55000
2    Srbija  48000

Drugi i treći red:
Država      BDP
1    Hrvatska  55000
2    Srbija  48000
```

Pristup redovima ili kolonama pomoću imena - `loc`

Omogućava pristup redovima i kolonama DataFrame-a ili Series-a na osnovu njihovih imena (oznake). Ovo je korisno za rad s imenovanim indeksima ili kolonama.

Sintaksa:

```
dataframe.loc[row_labels, column_labels]
```

Gdje je:

- `row_labels` – Oznaka ili skup oznaka redova koji se žele dohvatiti.
- `column_labels` – Oznaka ili skup oznaka kolona koje se žele dohvatiti.

Primjer:

```
import pandas as pd

# Kreiranje DataFrame-a
data = {
    "Država": ["BiH", "Hrvatska", "Srbija"],
    "BDP": [20000, 55000, 48000]
}
df = pd.DataFrame(data, index=["A", "B", "C"])

# 1. Dohvaćanje reda sa oznakom "A"
print("\nRed 'A':")
print(df.loc["A"])

# 2. Dohvaćanje specifične vrijednosti (red "B", kolona "BDP")
print("\nVrijednost u redu 'B' i koloni 'BDP':")
print(df.loc["B", "BDP"])

# 3. Dohvaćanje specifičnih redova
print("\nRedovi 'A' i 'C':")
print(df.loc[["A", "C"]])

# 4. Dohvaćanje svih redova za kolonu "Država"
print("\nSve vrijednosti za kolonu 'Država':")
print(df.loc[:, "Država"])

# 5. Specifični redovi i kolone
print("\nRedovi 'A' i 'B', kolone 'Država' i 'BDP':")
print(df.loc[["A", "B"], ["Država", "BDP"]])
```

```
Red 'A':
Država      BiH
BDP        20000
Name: A, dtype: object
```

```
Vrijednost u redu 'B' i koloni 'BDP':
55000
```

```

Redovi 'A' i 'C':
Država      BDP
A           BiH  20000
C           Srbija 48000

Sve vrijednosti za kolonu 'Država':
A           BiH
B           Hrvatska
C           Srbija
Name: Država, dtype: object

Redovi 'A' i 'B', kolone 'Država' i 'BDP':
Država      BDP
A           BiH  20000
B           Hrvatska 55000

```

Dodavanje nove kolone

Nova kolona se dodaje tako što se definiše ime kolone i dodijeli joj niz vrijednosti. Niz vrijednosti mora biti iste dužine, kao postojeći broj redova u DataFrame-u. Ako neka vrijednost nije poznata, može se unijeti kao `None` ili `NaN`, što označava nedostajuće podatke.

```

import pandas as pd

# Kreiranje DataFrame-a
data = {
    "Država": ["BiH", "Hrvatska", "Srbija"],
    "BDP": [20000, 55000, 48000]
}
df = pd.DataFrame(data)

# Dodavanje nove kolone 'Populacija'
df["Populacija"] = [3.3, 4.0, 7.0]
print("\nDodana kolona 'Populacija':")
print(df)

# Dodavanje nove kolone s poznatim i nedostajućim vrijednostima
df["Populacija"] = [3.3, 4.0, None]
print("\nDodana kolona za NaN 'Populacija':")
print(df)

```

	Država	BDP	Populacija
0	BiH	20000	3.3
1	Hrvatska	55000	4.0
2	Srbija	48000	7.0

	Država	BDP	Populacija
0	BiH	20000	3.3
1	Hrvatska	55000	4.0
2	Srbija	48000	NaN

Dodavanje novog reda

Dodavanje novog reda u DataFrame se izvodi pomoću funkcija `concat` ili u starijim verzijama Pandas < 1.4.0 moguće je korisiti i funkciju `append`. Funkcija `append` se ne korisiti u novim verzijama Pandas-a. Funkcija `concat` omogućava kombinovanje

DataFrame-ova ili dodavanje novih redova. Podaci koji se žele dodati trebaju biti organizovani u DataFrame ili Series objektu.

Sintaksa:

```
pd.concat(objs, axis=0, join='outer', ignore_index=False,  
          keys=None, levels=None, names=None, verify_integrity=False,  
          sort=False, copy=True)
```

Gdje je:

- **objs** – Lista DataFrame-ova ili Series objekata koji se žele spojiti.
- **axis:**
 - **0** za spajanje redova (podrazumevana vrijednost).
 - **1** za spajanje kolona.
- **ignore_index** – Ako je **True**, stvara novi indeks za spojeni DataFrame.
- **join** – Način spajanja (**'inner'** za presjek, **'outer'** za uniju).
- **keys** – Dodaje ključ u hijerarhijskom indeksu za diferencijaciju podataka.
- **copy** – Određuje da li će se podaci kopirati iz originalnog DataFrame-a prilikom spajanja. **False** se može koristiti radi postizanja boljih performansi na velikim skupovima podataka.

Primjer:

```
import pandas as pd  
  
# Kreiranje početnog DataFrame-a  
data = {  
    "Država": ["BiH", "Hrvatska", "Srbija"],  
    "BDP": [20000, 55000, 48000]  
}  
df = pd.DataFrame(data)  
  
# 1. Dodavanje novog reda za 'Slovenija' sa svim podacima  
novi_red_slovenija = pd.DataFrame({"Država": ["Slovenija"], "BDP": [60000]})  
df = pd.concat([df, novi_red_slovenija], ignore_index=True)  
  
print("\nDataFrame nakon dodavanja 'Slovenija':")  
print(df)  
  
# 2. Dodavanje novog reda za 'Crna Gora' s nedostajućim podacima  
novi_red_crna_gora = pd.DataFrame({"Država": ["Crna Gora"], "BDP": [None]})  
df = pd.concat([df, novi_red_crna_gora], ignore_index=True)  
  
print("\nDataFrame nakon dodavanja 'Crna Gora' s nedostajućim podacima:")  
print(df)
```

	Država	BDP
0	BiH	20000
1	Hrvatska	55000
2	Srbija	48000
3	Slovenija	60000

```
DataFrame nakon dodavanja 'Crna Gora' s nedostajućim podacima:
   Država      BDP
0      BiH    20000
1  Hrvatska    55000
2    Srbija    48000
3  Slovenija    60000
4  Crna Gora     None
```

Brisanje kolone i reda

Brisanje podataka u Pandas DataFrame-u vrši se pomoću metode `drop`. Parametar `axis` definiše da li se žele obrisati redove ili kolone.

Sintaksa:

```
df.drop(labels, axis=0/1, inplace=False)
```

Gdje je:

- `labels` – Ime kolone ili indeks reda koji treba obrisati.
- `axis`:
 - `0` za brisanje reda.
 - `1` za brisanje kolone.
- `inplace` (podrazumjевано `False`) – Ako je `True`, promjene se primjenjuju na postojeći DataFrame, bez kreiranja novog objekta.

Primjer:

```
import pandas as pd

# Kreiranje DataFrame-a
data = {
    "Država": ["BiH", "Hrvatska", "Srbija"],
    "BDP": [20000, 55000, 48000]
}
df = pd.DataFrame(data)

# Brisanje kolone "BDP"
df_bez_kolone = df.drop("BDP", axis=1)

print("\nDataFrame nakon brisanja kolone 'BDP':")
print(df_bez_kolone)

# Brisanje reda za "Hrvatska"
df_bez_reda = df.drop(1, axis=0)

print("\nDataFrame nakon brisanja reda za 'Hrvatska':")
print(df_bez_reda)

DataFrame nakon brisanja kolone 'BDP':
   Država
0      BiH
```

```

1 Hrvatska
2 Srbija

DataFrame nakon brisanja reda za 'Hrvatska':
Država      BDP
0      BiH  20000
2    Srbija  48000

```

Ažuriranje postojećih podataka

Ažuriranje postojećih podataka omogućava izmjenu specifičnih vrijednosti unutar kolona, redova ili pojedinačnih celija koristeći `loc` ili direktni pristup. Ovo je korisno za masovna ažuriranja ili ciljanu promjenu specifičnih elemenata.

Može se ažurirati:

- Može se ažurirati cijela kolona jednostavnom matematičkom operacijom.
- Pojedinačne celije ili specifični redovi se mogu ažurirati koristeći metodu `loc` za precizno pozicioniranje.

Primjer ažuriranja cijele kolone.

```

import pandas as pd

# Ažuriranje vrijednosti u koloni
data = {
    "Država": ["BiH", "Hrvatska", "Srbija"],
    "BDP": [20000, 55000, 48000]
}
dataframe = pd.DataFrame(data)

# Povećanje BDP-a za 10%
dataframe["BDP"] = dataframe["BDP"] * 1.1
print(dataframe)

      Država      BDP
0      BiH  22000.0
1  Hrvatska  60500.0
2    Srbija  52800.0

```

Primjer ažuriranje specifične vrijednosti koristeći `loc` metodu.

```

import pandas as pd

# Ažuriranje vrijednosti u koloni
data = {
    "Država": ["BiH", "Hrvatska", "Srbija"],
    "BDP": [20000, 55000, 48000]
}

# Ažuriranje vrijednosti za specifičnu državu
dataframe.loc[dataframe["Država"] == "BiH", "BDP"] = 25000

```

```
print(dataframe)
```

	Država	BDP
0	BiH	25000.0
1	Hrvatska	60500.0
2	Srbija	52800.0

4.5.3. Učitavanje podataka

Jedna od najvažnijih funkcionalnosti biblioteke Pandas je njena sposobnost za učitavanje podataka iz različitih izvora i snimanje u različitim formatima. Podaci koji se učitavaju iz tabele u CSV, Excel ili JSON formatima, kao i iz baza podataka, direktno se preslikavaju u DataFrame strukture. DataFrame organizuje podatke u kolonama i redovima, što omogućava efikasnu obradu i analizu. Ova fleksibilnost omogućava jednostavno integriranje Pandas DataFrame-ova u složene tokove podataka.

Za potrebe ilustracije, koristićemo tabelu koje predstavljaju prodaju raznih proizvoda po danima. Struktura podataka je prikazana u sljedećoj tabeli.

Tabela 4.2. Tabela prodaje proizvoda

Datum	Proizvod	Količina	Cijena	Ukupno
2024-02-01	Hljeb	50	1.20	60.00
2024-02-01	Mlijeko	30	0.90	27.00
2024-02-01	Jabuka	20	1.50	30.00
2024-02-02	Hljeb	30	1.20	36.00
2024-02-02	Mlijeko	50	0.90	45.00

Za potrebe sljedećih primjera, podatke iz ove tabele ćemo snimiti u fajlove, koji odgovaraju prilagođenim formatima CSV, Excel i JSON.

4.5.3.1. Učitavanje podataka iz CSV datoteke

Za obradu podataka iz CSV formata koristi se funkcija `pd.read_csv()` iz Pandas biblioteke, koja omogućava jednostavno konvertovanje CSV datoteka u Pandas DataFrame. Prilikom korištenja ove funkcije, važno je pravilno navesti putanju do fajla (bilo relativnu ili absolutnu) i odrediti simbol delimitera vrijednosti u CSV fajlu ako se razlikuje od podrazumjevane zapete (,). Takođe, ukoliko CSV datoteka sadrži zaglavje (header), ključno je pravilno specificirati red s nazivima kolona, kako bi se podaci ispravno organizovali unutar DataFrame-a.

Prilikom korištenja funkcije `pd.read_csv()`, ako se `header` ne navede, Pandas će podrazumjevano koristiti `header='infer'` i prepostaviti da prvi red u datoteci sadrži nazive kolona, a u suprotnom, koristi `header=None` i dodjeljuje numeričke indekse kolonama.

Sintaksa:

```
pd.read_csv(io, sep=",", header='infer', names=None, index_col=None  
, usecols=None, dtype=None, engine=None, skiprows=None, nrows=None)
```

Gdje je:

- `io` – Putanja do CSV datoteke ili objekat kao što je URL.
- `sep` – Separator u datoteci (podrazumjevano zapeta `,`).
- `header` – Red koji sadrži nazive kolona (podrazumjevano `infer`).
- `names` – Lista naziva kolona ako nije specificirano u datoteci.
- `index_col` – Kolona koja treba da bude korištena kao indeks.
- `usecols` – Lista kolona koje treba učitati.
- `dtype` – Tip podataka za svaku kolonu.
- `skiprows` – Broj redova koje treba preskočiti na početku datoteke.
- `nrows` – Broj redova koje treba pročitati.

Primjer:

```
import pandas as pd  
  
# Učitavanje podataka iz CSV datoteke s određenim kolonama  
csv_path = "prodaja.csv"  
dataframe_csv = pd.read_csv(csv_path, sep=",", nrows=100)  
print(dataframe_csv)
```

	Datum	Proizvod	Količina	Cijena	Ukupno
0	2024-02-01	Hljeb	50	1.2	60.0
1	2024-02-01	Mlijeko	30	0.9	27.0
2	2024-02-01	Jabuka	20	1.5	30.0
3	2024-02-02	Hljeb	30	1.2	36.0
4	2024-02-02	Mlijeko	50	0.9	45.0

4.5.3.2. Učitavanje podataka iz Excel datoteke

Funkcija `read_excel()` iz Pandas biblioteke omogućava direktno učitavanje podataka iz Excel radnih listova i njihovu konverziju u DataFrame. Za korištenje ove funkcije neophodna je instalacija dodatnih paketa, poput `openpyxl`. Takođe, važno je pravilno navesti putanju do fajla, kao i specificirati ime radnog lista, koji sadrži podatke, koje želimo da učitamo.

Sintaksa:

```
pd.read_excel(io, sheet_name=0, header=0, names=None,  
index_col=None, usecols=None, dtype=None, engine=None,  
skiprows=None, nrows=None)
```

Gdje je:

- `io` – Putanja do Excel datoteke ili objekat kao što je URL.
- `sheet_name` – Ime ili indeks lista koje treba učitati (podrazumevano `0`).
- `header` – Red koji sadrži nazive kolona.
- `names` – Lista naziva kolona ako nije specificirano u datoteci.
- `index_col` – Kolona koja treba da bude korištena kao indeks.
- `usecols` – Lista kolona koje treba učitati.
- `dtype` – Tip podataka za svaku kolonu.
- `skiprows` – Broj redova koje treba preskočiti na početku datoteke.
- `nrows` – Broj redova koje treba pročitati.

Primjer:

```
import pandas as pd  
  
# Učitavanje podataka iz Excel datoteke sa specifičnog lista  
excel_path = "prodaja.xlsx"  
dataframe_excel = pd.read_excel(excel_path, sheet_name="Sheet1", usecols="A:C",  
nrows=50)  
print(dataframe_excel)  
  
 Datum Proizvod Količina Cijena Ukupno  
0 2024-02-01 Hljeb 50 1.2 60  
1 2024-02-01 Mlijeko 30 0.9 27  
2 2024-02-01 Jabuka 20 1.5 30  
3 2024-02-02 Hljeb 30 1.2 36  
4 2024-02-02 Mlijeko 50 0.9 45
```

4.5.3.3. Učitavanje podataka iz JSON datoteke

Funkcija `read_json` iz Pandas biblioteke omogućava učitavanje podataka iz JSON datoteka i njihovu konverziju u `DataFrame` ili `Series`, u zavisnosti od vrijednosti parametra `typ`. Pri radu s JSON formatom važno je pravilno razumjeti strukturu podataka i koristiti odgovarajući `orient` parametar, koji definiše način na koji su podaci organizovani u JSON-u.

JSON podaci organizovani kao dvodimenzioni niz (npr. lista rječnika) lako se pretvaraju u DataFrame korišćenjem `orient="records"`, dok formati sa ključevima kao kolonama i nizovima kao vrijednostima koriste `orient="columns"`. Ostale vrijednosti `orient` (poput `index` ili `split`) omogućavaju rad s prilagođenim strukturama JSON-a.

Sintaksa:

```
pd.read_json(io,orient=None,typ='frame',dtype=None,  
convert_axes=None,convert_dates=True)
```

Gdje je:

- **io** – Putanja do JSON datoteke ili objekat kao što je URL.
- **orient** – Format ulaznih podataka (**records**, **split**, **index**, itd.).
- **typ** – Tip strukture podataka (DataFrame ili Series).
- **dtype** – Tip podataka za svaku kolonu.
- **convert_dates** – Ako je **True**, konvertuje datume u odgovarajuće formate.

Primjer:

```
import pandas as pd  
  
# Učitavanje podataka iz JSON datoteke  
json_path = "podaci.json"  
dataframe_json = pd.read_json(json_path, orient="records")  
print(dataframe_json)
```

	Datum	Proizvod	Količina	Cijena	Ukupno
0	2024-02-01	Hljeb	50	1.2	60
1	2024-02-01	Mlijeko	30	0.9	27
2	2024-02-01	Jabuka	20	1.5	30
3	2024-02-02	Hljeb	30	1.2	36
4	2024-02-02	Mlijeko	50	0.9	45

4.5.3.4. Učitavanje podataka iz baze podataka

Za učitavanje podataka iz baze podataka u Pandas DataFrame koristi se funkcija `pd.read_sql()`. Ključni korak je uspostavljanje konekcije s bazom koristeći biblioteku poput `sqlalchemy`, koja omogućava komunikaciju između Pandas-a i baze. Važno je napomenuti da je za korištenje sqlalchemy potrebno prethodno instalirati ovu biblioteku. Nakon uspostavljanja konekcije, moguće je izvršavati SQL upite, koje će funkcije konvertovati rezultate direktno u DataFrame.

Ova funkcija je potpuno nezavisna od tipa baze podataka, jer može učitavati podatke iz bilo koje baze koju podržava sqlalchemy. Konektivnost s bazom obezbjeđuje upravo sqlalchemy, dok je za rad s različitim bazama potrebno, pored sqlalchemy, instalirati i odgovarajući drajver za konkretnu bazu podataka.

Sintaksa:

```
pd.read_sql(sql,con,index_col=None,coerce_float=True,  
params=None)
```

Gdje je:

- `sql` – SQL upit ili ime tabele.
- `con` – Konekcija s bazom podataka (npr. kreirana pomoću `sqlalchemy.create_engine`).
- `index_col` – Kolona koja treba da bude korištena kao indeks.
- `coerce_float` – Ako je `True`, pretvara numeričke vrijednosti u `float` gdje je primjenjivo.
- `params` – Parametri za SQL upit (npr. za pripremljene izjave).

Primjer:

```
import pandas as pd
import sqlalchemy

# Kreiranje konekcije sa bazom podataka
engine = sqlalchemy.create_engine("mysql+pymysql://user:password@localhost/baza")

# Izvršavanje SQL upita
dataframe_sql = pd.read_sql("SELECT * FROM tabela", engine)
print(dataframe_sql)
```

4.5.4. Snimanje podataka

Pandas omogućava snimanje podataka direktno iz DataFrame-a u različite formate, što olakšava njihovu dalju obradu ili dijeljenje. Za potrebe ilustracije primjera, prethodno ćemo formirati DataFrame iz rječnika (`dict`), koji će se koristiti za konverziju u različite formate fajlova.

U nastavku je definisana promjenjiva `podaci`, koja će se koristiti u svim primjerima konverzije. Kako bi primjeri ostali čitljivi i fokusirani, ova promjenjiva neće biti ponavljana u svakom primjeru. Ako želite da isprobate primjere ili ih prilagodite, potrebno je da kopirate i izvršite liniju koda s definisanom promjenjivom `podaci`, kako bi osigurali dostupne podatke za konverziju.

```
podaci = {
    "Datum": ["2024-02-01", "2024-02-01", "2024-02-01", "2024-02-02", "2024-02-02"],
    "Proizvod": ["Hljeb", "Mlijeko", "Jabuka", "Hljeb", "Mlijeko"],
    "Količina": [50, 30, 20, 30, 50],
    "Cijena": [1.2, 0.9, 1.5, 1.2, 0.9],
    "Ukupno": [60, 27, 30, 36, 45]
}
```

4.5.4.1. Snimanje podataka u CSV format

Funkcija `to_csv()` iz Pandas biblioteke omogućava snimanje podataka iz DataFrame-a u CSV format, pružajući fleksibilnost u prilagođavanju izlazne datoteke.

Prilikom snimanja podataka u CSV format važno je obratiti pažnju na nekoliko ključnih parametara. Parametar `index` omogućava kontrolu pojavljivanja indeksa redova. Ako ne želite da se indeksi pojavljuju na početku svake linije u izlaznoj datoteci, potrebno je postaviti ovu opciju na `False`. Takođe, simbol koji se koristi kao separator vrijednosti u datoteci (`sep`) može se prilagoditi, pri čemu je podrazumjevani separator zapeta (,), što se može mijenjati u skladu s regionalnim standardima. Na kraju, za pravilno snimanje datoteka koje sadrže latinična ili cirilična slova, preporučuje se korištenje kodiranje `utf-8`.

Sintaksa:

```
pd.DataFrame.to_csv(io, sep=",", index=True,  
header=True, columns=None, mode="w", encoding=None)
```

Gdje je:

- `io` – Putanja do datoteke u koju se snimaju podaci.
- `sep` – Separator u datoteci (podrazumevano zapeta ,).
- `index` – Ako je `True`, dodaje kolonu s indeksom u izlaznu datoteku.
- `header` – Ako je `True`, dodaje nazine kolona u izlaz.
- `columns` – Lista kolona koje treba snimiti.
- `mode` – Način otvaranja datoteke (`w` za pisanje, `a` za dodavanje).
- `encoding` – Kodiranje koje se koristi za snimanje datoteke.

Primjer:

```
import pandas as pd  
  
# Promjenjiva podaci za konverziju u CSV, dodati promjenjivu ispod ove linije  
  
# Konvertovanje u DataFrame  
df = pd.DataFrame(podaci)  
  
# Snimanje podataka u CSV fajl  
df.to_csv("prodaja_izlaz.csv", index=False, sep=",", encoding="utf-8")  
  
Izlaz je datoteka prodaja_izlaz.csv koja sadrži podatke iz promjenjive podaci
```

4.5.4.2. Snimanje podataka u Excel format

Funkcija `to_excel()` iz Pandas biblioteke omogućava snimanje podataka iz DataFrame-a u Excel format, pružajući fleksibilne opcije za prilagođavanje izlazne datoteke.

Pri korištenju funkcije `to_excel()` važno je obratiti pažnju na nekoliko ključnih parametara. Parametar `sheet_name` omogućava imenovanje radnog lista u Excel datoteci. Ako ovaj parametar nije naveden, podrazumjevano ime lista će biti `"Sheet1"`. Imenovanje radnog lista posebno je korisno za organizaciju podataka, naročito kada Excel fajl sadrži više radnih listova.

Parametar `engine` definiše mehanizam za pisanje u Excel format. Ako nije eksplicitno naveden, Pandas automatski bira odgovarajući engine, poput `openpyxl` za `.xlsx` fajlove. Međutim, ako odgovarajući engine nije instaliran, funkcija će dati grešku.

Sintaksa:

```
pd.DataFrame.to_excel(io,sheet_name="Sheet1",
index=True,header=True,columns=None,engine=None, encoding=None)
```

Gdje je:

- `io` – Putanja do Excel datoteke ili ExcelWriter objekat.
- `sheet_name` – Naziv lista u kojem se podaci snimaju (podrazumjevano `"Sheet1"`).
- `index` – Ako je `True`, dodaje kolonu s indeksom u izlaz.
- `header` – Ako je `True`, dodaje nazine kolona u izlaz.
- `columns` – Lista kolona koje treba snimiti.
- `engine` – Mehanizam za pisanje u Excel (npr. `openpyxl`).
- `encoding` – Kodiranje za snimanje datoteke.

Primjer:

```
import pandas as pd

# Promjenjiva podaci za konverziju u CSV, dodati promjenjivu ispod ove linije

# Konvertovanje u DataFrame
df = pd.DataFrame(podaci)
# Snimanje podataka u Excel datoteku
df.to_excel("prodaja.xlsx",index=False,sheet_name="Prodaja",engine="openpyxl")
```

Izlaz je datoteka `prodaja_izlaz.xlsx` koja sadrži podatke iz promjenjive podaci

4.5.4.3. Snimanje podataka u JSON format

Funkcija `to_json()` iz Pandas biblioteke omogućava snimanje podataka iz DataFrame-a u JSON format uz kontrolu organizacije izlaznih podataka. Parametar `orient` definiše strukturu izlaza, gdje se najčešće koristi opcija `"records"` za konverziju svakog reda u zaseban JSON objekat, što olakšava integraciju s

aplikacijama i API-jevima. Ova funkcija pruža fleksibilnost za generisanje čitljivih JSON fajlova spremnih za dalju upotrebu.

Sintaksa:

```
pd.DataFrame.to_json(io,orient="records",
                     date_format=None,double_precision=10,force_ascii=True,
                     lines=False)
```

Parametri:

- **io** – Putanja do JSON datoteke ili objekat za pisanje.
- **orient** – Format izlaznih podataka (**records**, **split**, **index**, itd.).
- **date_format** – Format za datumske vrijednosti (npr. **epoch**, **iso**).
- **double_precision** – Broj decimalnih mesta za numeričke vrijednosti.
- **force_ascii** – Ako je **True**, osigurava ASCII kodiranje za izlaz.
- **lines** – Ako je **True**, snima JSON u NDJSON formatu (jedan red po zapisu).

Primjer:

```
import pandas as pd

# Promjenjiva podaci za konverziju u CSV, dodati promjenjivu ispod ove linije

# Konvertovanje u DataFrame
df = pd.DataFrame(podaci)
# Snimanje podataka u JSON datoteku
df.to_json("prodaja_izlaz.json", orient="records")
```

Izlaz je datoteka prodaja_izlaz.json koja sadrži podatke iz promjenjive podaci

4.5.4.4. Snimanje podataka u MySQL bazu

Funkcija **to_sql()** iz Pandas biblioteke omogućava direktno snimanje podataka iz DataFrame-a u MySQL bazu podataka. Ključni korak za uspješno korištenje ove funkcije je pravilno ustavljanje konekcije s bazom podataka, što smo već obradili prilikom čitanja podataka iz baze koristeći **sqlalchemy**. Konekcija omogućava komunikaciju između Pandas-a i baze podataka, a kreira se pomoću funkcije **sqlalchemy.create_engine**.

Sintaksa:

```
df.to_sql(name,con,schema=None,if_exists='fail',      index=True,
          index_label=None, chunksize=None, dtype=None)
```

Gdje je:

- `name` – Ime tabele u bazi podataka.
- `con` – Konekcija s bazom podataka (npr. kreirana pomoću `sqlalchemy.create_engine`).
- `schema` – Shema baze podataka (opciono).
- `if_exists` – Akcija ako tabela već postoji (`fail`, `replace`, `append`).
- `index` – Ako je `True`, dodaje kolonu s indeksom u tabelu.
- `chunksize` – Broj redova koji se snimaju u jednom potezu (opciono).
- `dtype` – Tip podataka za kolone.

Primjer:

```
import pandas as pd
import sqlalchemy

# Promjenjiva prodaja s podacima, dodati promjenjivu ispod ove linije

# Kreiranje DataFrame-a iz podataka
df = pd.DataFrame(prodaja)

# Kreiranje konekcije s MySQL bazom pomoću sqlalchemy
engine =
    sqlalchemy.create_engine("mysql+pymysql://username:password@localhost/baza")

# Snimanje podataka u MySQL tabelu
df.to_sql(name="prodaja", con=engine, if_exists="replace", index=False)
```

4.5.5. Analiza i manipulacija podacima

Pandas biblioteka nudi širok spektar alata za analizu i manipulaciju podataka. Ova poglavlja obuhvataju osnovne i napredne tehnike filtriranja, grupisanja, spajanja i obrade podataka, uz praktične primjere koji ilustruju njihovu primjenu.

4.5.5.1. Filtriranje i selekcija

Filtriranje i selekcija podataka omogućavaju izdvajanje specifičnih redova i kolona na osnovu definisanih uslova. Ova tehnika se često koristi za pripremu podataka za analizu.

Filtriranje redova u Pandas DataFrame-u obavlja se primjenom logičkih uslova direktno na DataFrame. Prepostavimo da imamo tabelu podataka o prodaji proizvoda i želimo izdvojiti samo one redove gdje je količina veća od 30. Sljedeći kod prikazuje filtrirane podatke po traženom uslovu (`kolicina>30`).

```
import pandas as pd

# Kreiranje DataFrame-a
podaci = {
    "Proizvod": ["Hljeb", "Mlijeko", "Jabuka", "Hljeb", "Mlijeko"],
```

```

    "Količina": [50, 30, 20, 30, 50],
    "Cijena": [1.2, 0.9, 1.5, 1.2, 0.9]
}
dataframe = pd.DataFrame(podaci)

# Filtriranje redova
filtrirani_podaci = dataframe[dataframe["Količina"] > 30]
print(filtrirani_podaci)

  Proizvod Količina Cijena
0   Hljeb        50     1.2
4  Mlijeko       50     0.9

```

Pandas omogućava jednostavno izdvajanje jedne ili više kolona iz DataFrame-a. Novi DataFrame formiramo tako što postojećem Datafremu proslijedimo listu kolona, koje želimo da kopiramo u novi DataFrame.

```

import pandas as pd

# Kreiranje DataFrame-a
podaci = {
    "Proizvod": ["Hljeb", "Mlijeko", "Jabuka", "Hljeb", "Mlijeko"],
    "Količina": [50, 30, 20, 30, 50],
    "Cijena": [1.2, 0.9, 1.5, 1.2, 0.9]
}
dataframe = pd.DataFrame(podaci)
# Izdvajanje kolona "Proizvod" i "Cijena"
izabrane_kolone = dataframe[["Proizvod", "Cijena"]]
print(izabrane_kolone)

  Proizvod Cijena
0   Hljeb     1.2
1  Mlijeko    0.9
2   Jabuka    1.5
3   Hljeb     1.2
4  Mlijeko    0.9

```

4.5.5.2. Grupisanje i agregacija

Grupisanje i agregacija omogućavaju analizu podataka po kategorijama, kao što su izračunavanje prosjeka, suma i drugih statističkih pokazatelja za određene grupe podataka.

Funkcija `df.groupby()` grupiše podatke prema jednoj ili više kolona, a zatim omogućava primjenu agregatnih funkcija.

Sintaksa:

```
df.groupby(by, as_index=True).aggregate_function()
```

Gdje je:

- **by** – Naziv kolone ili lista kolona prema kojima se vrši grupisanje.
- **as_index** – Ako je `True`, koristi vrijednosti grupe kao indekse u rezultatu.

Sljedeći primjer pokazuje grupisanje podataka prema proizvodima i računanje prosječne cijene i ukupne količine.

```
import pandas as pd

# Kreiranje DataFrame-a
podaci = {
    "Proizvod": ["Hljeb", "Mlijeko", "Jabuka", "Hljeb", "Mlijeko"],
    "Količina": [50, 30, 20, 30, 50],
    "Cijena": [1.2, 0.9, 1.5, 1.2, 0.9]
}
df = pd.DataFrame(podaci)
# Grupisanje podataka prema proizvodu
agregirani_podaci = df.groupby("Proizvod").agg(
    Prosječna_cijena=("Cijena", "mean"),
    Ukupna_količina=("Količina", "sum")
)
print(agregirani_podaci)

      Prosječna_cijena  Ukupna_količina
Proizvod
Hljeb              1.2                  80
Jabuka             1.5                  20
Mlijeko            0.9                  80
```

4.5.5.3. Spajanje podataka

Pandas biblioteka omogućava spajanje podataka iz više tabela koristeći funkcije `merge`, `concat` i `join`. Ove funkcije pružaju fleksibilne opcije za integraciju podataka iz različitih izvora, omogućavajući efikasnu analizu i manipulaciju.

1) Funkcija `pd.merge()`

Funkcija `merge` omogućava spajanje dva DataFrame-a na osnovu jedne ili više zajedničkih kolona. Ova funkcija se može koristiti na dva načina: direktno iz Pandas modula (`pd.merge`) ili kao metoda jednog od DataFrame-ova (`df.merge`).

Sintaksa za `pd.merge`:

```
pd.merge(left, right, how="inner", on=None)
```

Sintaksa za `df.merge`:

```
df.merge(right, how="inner", on=None, left_on=None, right_on=None,
suffixes=("_x", "_y"))
```

Gdje je:

- `left, right` – DataFrame-ovi koji se spajaju (u slučaju `pd.merge`) ili samo `right` u slučaju `df.merge`.
- `how` – Način spajanja (`inner, outer, left, right`). Podrazumevano je `inner`.
- `on` – Kolona ili lista kolona prema kojoj se vrši spajanje. Ako nije navedeno, Pandas koristi zajedničke kolone.
- `left_on, right_on` – Kolone prema kojima se vrši spajanje u lijevom, odnosno desnom DataFrame-u (korisno kada se nazivi kolona razlikuju).
- `suffixes` – Dodaci za imenovanje kolona koje se preklapaju (npr. `_x` i `_y`).

Kada se `merge` poziva direktno iz Pandas modula, oba DataFrame-a se eksplicitno navode kao argumenti, a ako se koristi kao metoda jednog od DataFrame-ova, drugi DataFrame se prosleđuje kao argument.

Primjer:

```
import pandas as pd

# Prvi DataFrame
proizvodi = pd.DataFrame({
    "Proizvod": ["Hljeb", "Mlijeko", "Jabuka"],
    "Cijena": [1.2, 0.9, 1.5]
})

# Drugi DataFrame
kategorije = pd.DataFrame({
    "Proizvod": ["Hljeb", "Mlijeko", "Jabuka"],
    "Kategorija": ["Pekara", "Mlijecni proizvodi", "Voće"]
})

# Spajanje koristeći Pandas funkciju (pd.merge)
spojeni_podaci_pd = pd.merge(proizvodi, kategorije, on="Proizvod", how="inner")
print("Rezultat spajanja koristeći pd.merge:")
print(spojeni_podaci_pd)

# Spajanje koristeći DataFrame metodu (df.merge)
spojeni_podaci_df = proizvodi.merge(kategorije, on="Proizvod", how="inner")
print("\nRezultat spajanja koristeći df.merge:")
print(spojeni_podaci_df)
```

```
Rezultat spajanja koristeći pd.merge:
  Proizvod  Cijena      Kategorija
0   Hljeb     1.2        Pekara
1  Mlijeko     0.9  Mlijecni proizvodi
2   Jabuka     1.5          Voće
```

```
Rezultat spajanja koristeći df.merge:
  Proizvod  Cijena      Kategorija
0   Hljeb     1.2        Pekara
1  Mlijeko     0.9  Mlijecni proizvodi
2   Jabuka     1.5          Voće
```

U kojim slučajevima se koristi funkcija direktno ili na DataFrame:

- `pd.merge` – Korisno kada su DataFrame-ovi nezavisni ili kada se želi jasno navesti ova DataFrame-a u funkciji.
- `df.merge` – Praktično kada se želi direktno povezati jedan DataFrame s drugim, posebno kada postoji glavni DataFrame.

Bez obzira na način pozivanja, funkcija `merge` omogućava moćno i fleksibilno spajanje podataka, čak i kada postoje razlike u strukturi između tabela.

2) Funkcija `concat()`

Funkcija `concat` omogućava spajanje DataFrame-ova duž definisane ose, bilo po redovima ili kolonama. U Pandas-u, `concat` je dostupna isključivo kao funkcija modula (npr. `pd.concat`) i ne može se pozvati direktno kao metoda na DataFrame objektu, za razliku od funkcije `merge`. Dizajnirana je da radi s više DataFrame-ova ili Series objekata, što je čini pogodnom za spajanje lista objekata. Upravo zbog ove osobine funkcioniše na nivou Pandas modula, pružajući fleksibilnost u kombinovanju podataka iz različitih izvora.

Sintaksa:

```
pd.concat(objs, axis=0, join="outer", ignore_index=False)
```

Gdje je:

- `objs` – Lista DataFrame-ova koje treba spojiti.
- `axis` – Osa duž koje se vrši spajanje (`0` za redove, `1` za kolone).
- `join` – Tip spajanja (`outer` ili `inner`).
- `ignore_index` – Ako je `True`, resetuje indekse u rezultatu.

Primjer spajanje dva DataFrame-a po redovima.

```
import pandas as pd

# Prvi DataFrame
prodaja_januar = pd.DataFrame({
    "Proizvod": ["Hljeb", "Mlijeko"],
    "Količina": [50, 30]
})

# Drugi DataFrame
prodaja_februar = pd.DataFrame({
    "Proizvod": ["Jabuka", "Hljeb"],
    "Količina": [20, 40]
})

# Spajanje podataka
spojeni_podaci = pd.concat([prodaja_januar, prodaja_februar], axis=0,
ignore_index=True)
print(spojeni_podaci)
```

	Proizvod	Količina
0	Hljeb	50
1	Mlijeko	30
2	Jabuka	20
3	Hljeb	40

3) Funkcija `join()`

Funkcija `join` koristi se za spajanje DataFrame-ova na osnovu indeksa. Za razliku od funkcije `concat`, koja je dizajnirana za spajanje liste objekata, `join` funkcioniše koristeći jedan DataFrame kao osnovu za spajanje, čime se prirodno koristi kao metoda DataFrame-a (`df.join`). Ova funkcija nije dostupna direktno u Pandas modulu (nema `pd.join`). Njena primjena je posebno praktična za integraciju podataka iz DataFrame-ova, koji dijele iste indekse, omogućavajući efikasno spajanje podataka na osnovu zajedničke strukture.

Sintaksa:

```
df.join(dataframe2, how="left", lsuffix="", rsuffix="")
```

Gdje je:

- `how` – Način spajanja (`left`, `right`, `inner`, `outer`).
- `lsuffix`, `rsuffix` – Dodaci za imenovanje kolona koje se preklapaju.

Primjer spajanje podataka na osnovu indeksa.

```
import pandas as pd
# Prvi DataFrame
cijene = pd.DataFrame({
    "Cijena": [1.2, 0.9, 1.5]
}, index=["Hljeb", "Mlijeko", "Jabuka"])

# Drugi DataFrame
kategorije = pd.DataFrame({
    "Kategorija": ["Pekara", "Mlijecni proizvodi", "Voće"]
}, index=["Hljeb", "Mlijeko", "Jabuka"])

# Spajanje podataka
spojeni_podaci = cijene.join(kategorije)
print(spojeni_podaci)
```

	Cijena	Kategorija
Hljeb	1.2	Pekara
Mlijeko	0.9	Mlijecni proizvodi
Jabuka	1.5	Voće

4.5.5.4. Rad s nedostajućim podacima

Pandas biblioteka nudi moćne alate za upravljanje nedostajućim vrijednostima (`NaN`), što je česta situacija pri radu s podacima. Nedostajuće vrijednosti mogu uticati na tačnost analiza i modela, zbog čega je njihovo pravilno rukovanje ključno u procesima analize podataka. Pandas omogućava jednostavno pronalaženje, zamjenu i uklanjanje `NaN` vrijednosti korištenjem različitih funkcija i metoda.

Jedna od najkorisnijih funkcija za rad s nedostajućim podacima je `fillna()`, koja omogućava popunjavanje `NaN` vrijednosti definisanim vrijednostima, statističkim mjerama (npr. srednja vrijednost, medijana) ili posebnim kategorijama. Ova funkcija pruža fleksibilnost u prilagođavanju podataka specifičnim zahtjevima analize, čime se osigurava cjelovitost i dosljednost datasetova.

U nastavku će biti prikazana sintaksa i primjeri korištenja funkcije `fillna()` za različite scenarije.

Sintaksa:

```
DataFrame.fillna(value=None, method=None, axis=None,  
inplace=False, limit=None)
```

Gdje je:

- `value` – Vrijednost ili mapa vrijednosti koje će zamijeniti `NaN`. Može biti pojedinačna vrijednost (npr. broj ili string) ili rječnik koji specificira vrijednosti po kolonama.
- `method` – Metoda za popunjavanje (`ffill` za propagaciju unaprijed, `bfill` za propagaciju unazad).
- `axis` – Osa duž koje se primjenjuje popunjavanje (`0` za redove, `1` za kolone).
- `inplace` – Ako je `True`, zamjene se vrše na postojećem DataFrame-u, bez kreiranja novog.
- `limit` – Maksimalan broj zamjena koje će biti izvršene.

Primjer:

```
import pandas as pd  
  
# Zamjena nedostajućih vrijednosti  
podaci = {  
    "Proizvod": ["Hljeb", "Mlijeko", None],  
    "Količina": [50, None, 20],  
    "Cijena": [1.2, 0.9, None]  
}  
dataframe = pd.DataFrame(podaci)  
dataframe_filled = dataframe.fillna({  
    "Proizvod": "Nepoznato",  
    "Količina": 0,  
    "Cijena": dataframe["Cijena"].mean()  
})  
print(dataframe_filled)
```

	Proizvod	Količina	Cijena
0	Hljeb	50.0	1.20
1	Mlijeko	0.0	0.90
2	Nepoznato	20.0	1.05

Napomena:

- Ako se koristi `value`, svaki `NaN` u DataFrame-u zamjenjuje se specificiranim vrijednošću.
- Ako se koristi `method`, popunjavanje se vrši na osnovu propagacije vrijednosti.

4.5.5.5. Rad sa datumima i vremenom

Pandas biblioteka pruža mogućnosti za rad s vremenskim podacima, što je ključno za analizu podataka vezanih za datume i vrijeme, poput vremenskih serija, istorijskih podataka ili praćenja trendova. Mogućnosti uključuju konverziju podataka u datumske formate, manipulaciju datumima, filtriranje po vremenskim intervalima i izvođenje analiza zasnovanih na vremenskim serijama.

Jedna od ključnih funkcija za rad s datumima u Pandas-u je `to_datetime()`, koja omogućava pretvaranje podataka iz različitih formata (stringova, brojeva) u standardizovane datumske objekte koje Pandas prepoznaje.

Sintaksa:

```
pd.to_datetime(arg,format=None,errors='raise',
               dayfirst=False,yearfirst=False,utc=None,exact=True, unit=None,
               origin='unix')
```

Gdje je:

- `arg` – Podaci koji se konvertuju (npr. string, lista, niz, ili DataFrame kolona).
- `format` – String koji definiše format datuma (npr. `%Y-%m-%d` za "2024-01-01").
- `errors` – Postavlja način rukovanja greškama:
 - `'raise'` (podrazumjевано) – Daje grešku za nevalidne podatke.
 - `'coerce'` – Zamjenjuje nevalidne vrijednosti s `NaT`.
 - `'ignore'` – Ignoriše greške bez promjena.
- `dayfirst` – Ako je `True`, podrazumjeva da dan dolazi prije mjeseca (korisno za evropske formate).
- `yearfirst` – Ako je `True`, podrazumeva da godina dolazi prije mjeseca i dana.
- `utc` – Ako je `True`, konvertuje datume u UTC vremensku zonu.

Primjer konverzija stringova u datumski format i filtriranje podataka po datumu.

```

import pandas as pd
# Kreiranje DataFrame-a s datumima
podaci = {
    "Datum": ["2024-01-01", "2024-02-02", "2024-02-03"],
    "Proizvod": ["Hljeb", "Mlijeko", "Jabuka"],
    "Količina": [50, 30, 20]
}
dataframe = pd.DataFrame(podaci)
dataframe["Datum"] = pd.to_datetime(dataframe["Datum"])

# Filtriranje podataka za februar 2024.
filtrirani_podaci = dataframe[dataframe["Datum"].dt.month == 2]
print(filtrirani_podaci)

```

Datum	Proizvod	Količina
1 2024-02-02	Mlijeko	30
2 2024-02-03	Jabuka	20

4.5.6. Vizualizacija podataka koristeći Pandas

Vizualizacija podataka predstavlja ključni dio analize, pri čemu Python modul Matplotlib služi kao osnova za kreiranje grafičkih prikaza. Pandas, kao moćna biblioteka za rad s podacima, koristi integraciju s Matplotlib-om, omogućavajući jednostavno i direktno kreiranje grafičkih prikaza iz DataFrame-a ili Series objekata.

Ova integracija pruža značajne prednosti, uključujući:

- 1) **Jednostavnost korištenja:** Grafici se kreiraju pozivom jedne funkcije na DataFrame ili Series objektu, bez potrebe za dodatnim definisanjem podataka za x i y ose.
- 2) **Efikasnost:** Podaci se ne moraju prethodno formatirati ili pripremati za Matplotlib, jer Pandas automatski interpretira strukturu podataka.
- 3) **Brza analiza:** Mogućnost vizualizacije direktno iz DataFrame-a omogućava analitičarima da brzo pregledaju trendove i obrasce u podacima.
- 4) **Fleksibilnost:** Iako su osnovni grafici jednostavnii za kreiranje, dodatna prilagođavanja su moguća kroz direktnu integraciju s Matplotlib-om.

Da bi se koristili grafici u Pandas skriptama, potrebno je osigurati nekoliko ključnih koraka. Pandas koristi biblioteku Matplotlib, kao osnovu za vizualizaciju, pa je prvo potrebno instalirati Matplotlib ako nije već instalirana. Ovo se može postići korištenjem komande `pip install matplotlib`. Nakon instalacije, potrebno je importovati potrebne module, uključujući Pandas i Matplotlib, koristeći naredbu `import pandas as pd` i `import matplotlib.pyplot as plt`.

U nastavku ćemo se fokusirati na osnovne tehnike vizualizacije koristeći Pandas, uključujući osnovne metode za crtanje grafika, njihove prednosti i primjene.

4.5.6.1. Osnovni grafik u Pandas-u

Pandas omogućava vizualizaciju putem metode `plot`, koja je dostupna za DataFrame i Series objekte, a koristi Matplotlib za kreiranje grafičkih prikaza. Takođe, važno je osigurati da su podaci pripremljeni u odgovarajućem formatu, poput DataFrame-a ili Series-a i da sadrže numeričke vrijednosti, koje su pogodne za vizualizaciju. Ovi koraci omogućavaju glatku integraciju između Pandas-a i Matplotlib-a, čineći vizualizaciju podataka efikasnom i jednostavnom.

Sintaksa za funkciju `plot` u Pandas-u:

```
df.plot(x=None,y=None,kind='line',ax=None,subplots=False,  
sharex=None,sharey=False,use_index=True,title=None,  
grid=None,legend=True,style=None,figsize=None,logx=False,  
logy=False,loglog=False)
```

Gdje je:

- `x` – Kolona koja se koristi za x-osu. Ako nije navedena, koristi se indeks DataFrame-a.
- `y` – Kolona ili lista kolona koje se koriste za y-osu.
- `kind`: – Tip grafika koji želite da kreirate. Podržani tipovi su:
 - `'line'` (linijski grafik, podrazumjевано),
 - `'bar'` (stubičasti grafik),
 - `'barh'` (horizontalni stubičasti grafik),
 - `'hist'` (histogram),
 - `'box'` (box plot),
 - `'scatter'` (scatter plot),
 - `'area'` (površinski grafik),
 - `'pie'` (grafik u obliku pite).
- `ax` – Matplotlib Axes objekat na kojem se crta grafik. Ako nije naveden, Pandas kreira novi Axes objekat.
- `subplots` – Ako je `True`, kreira zaseban grafik za svaku kolonu.
- `sharex`, `sharey` – Ako je `True`, dijeljenje x- ili y-ose među subplotovima.
- `title` – Naslov grafika.
- `grid` – Ako je `True`, dodaje mrežu na grafik.
- `legend` – Ako je `True`, prikazuje legendu.
- `figsize` – Dimenzije grafika kao tuple (`širina, visina`) (npr. `(10, 6)`).
- `logx`, `logy`, `loglog` – Ako je `True`, koristi logaritamsku skalu za x, y ili obje ose.

Napomena:

- Funkcija automatski bira x i y podatke ako nisu eksplisitno navedeni.
- Podaci u DataFrame-u moraju biti numerički ili kategorizovani tako da odgovaraju odabranom tipu grafika.

U nastavku su dva primjera koja pokazuju upotrebu funkcije `plot` uz automatsko prepoznavanje tipa grafika na osnovu podataka u DataFrame-u.

Primjer linijski grafik za vremensku seriju.

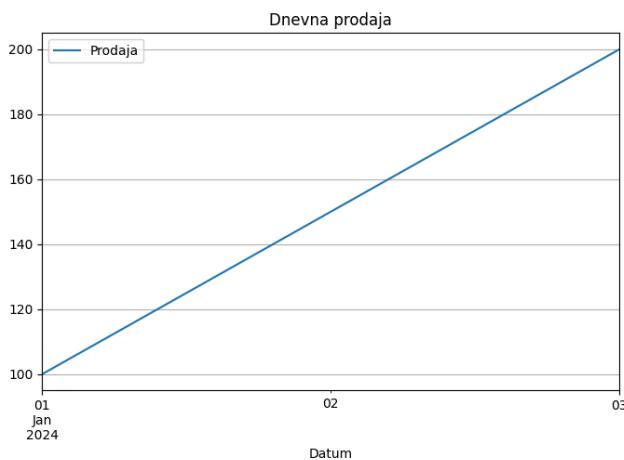
```
import pandas as pd

# Priprema podataka
data = {
    "Datum": ["2024-01-01", "2024-01-02", "2024-01-03"],
    "Prodaja": [100, 150, 200]
}
df = pd.DataFrame(data)
df["Datum"] = pd.to_datetime(df["Datum"])

# Postavljanje "Datum" kao indeksa
df.set_index("Datum", inplace=True)

# Crtanje linijskog grafika
df.plot(y="Prodaja", title="Dnevna prodaja", figsize=(8, 5), grid=True)
```

<Axes: title={'center': 'Dnevna prodaja'}, xlabel='Datum'>



Kada se koristi funkciju `plot` u Pandas-u, grafik se automatski prikazuje na ekranu, ali Python interpretator dodatno ispisuje tekstualnu reprezentaciju objekta `AxesSubplot` (npr. `<Axes: title={'center': 'Dnevna prodaja'}, xlabel='Datum'>`). Ovaj tekst predstavlja referencu na grafik koji je kreiran. Da bi se izbjeglo ispisivanje ovog dodatnog teksta i fokusiralo samo na prikaz grafika, preporučuje se korištenje funkcije `plt.show()`. Ova funkcija jasno definiše trenutak prikazivanja grafika i osigurava pravilno ponašanje u različitim okruženjima, poput terminala ili Jupyter Notebook-a, čineći kod čitljivijim i profesionalnijim.

Primjer automatski generisanog grafika sa više kolona.

```
import pandas as pd
```

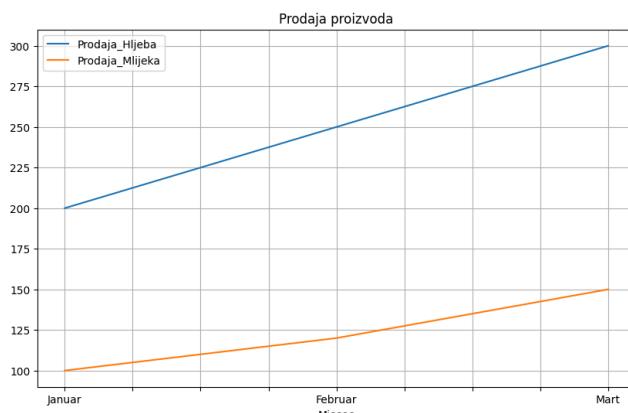
```

import matplotlib.pyplot as plt

# Priprema podataka
data = {
    "Mjesec": ["Januar", "Februar", "Mart"],
    "Prodaja_Hljeba": [200, 250, 300],
    "Prodaja_Mlijeka": [100, 120, 150]
}
df = pd.DataFrame(data)

# Crtanje grafika za više kolona
df.plot(x="Mjesec", title="Prodaja proizvoda", figsize=(10, 6), grid=True)
plt.show()

```



Na ovom grafiku Pandas je automatski prepoznao numeričke kolone u DataFrame-u ([Prodaja_Hljeba](#) i [Prodaja_Mlijeka](#)) i prikazao ih kao zasebne serije na istom grafiku. Osa x koristi kolonu "[Mjesec](#)", koja je eksplisitno definisana kao kategorijalna varijabla, dok osa y prikazuje vrijednosti prodaje za svaki proizvod. Naslov grafika "[Prodaja proizvoda](#)" dodat je kao parametar, a mreža je omogućena za bolju preglednost podataka. Svaka serija je predstavljena različitom bojom, što olakšava poređenje trendova prodaje hljeba i mlijeka tokom mjeseci.

4.5.6.2. Integracija s Matplotlib

Pandas grafici se mogu dodatno prilagoditi korištenjem funkcionalnosti Matplotlib-a, čime se omogućava kombinacija jednostavnosti Pandas-ovih metoda i preciznosti Matplotlib podešavanja. Kada pozovete metodu `plot` iz Pandas-a, ona automatski kreira osnovni grafik i vraća Matplotlib objekt ([Axes](#)), koji predstavlja osnovnu strukturu grafika. Ovaj objekt omogućava dalju manipulaciju i prilagođavanje grafika pomoću Matplotlib metoda.

Ovaj pristup omogućava da osnovno kreiranje grafika bude brzo i jednostavno, dok se dodatna prilagođavanja, poput izmjene naslova, označavanja osa, podešavanja mreže ili boja, mogu izvršiti direktno na vraćenom [Axes](#) objektu.

Kombinacija ove dvije biblioteke omogućava efikasnu izradu vizualizacija, gdje Pandas preuzima posao pripreme podataka i osnovnog crtanja, dok Matplotlib omogućava detaljno oblikovanje.

Primjer kombinacija Pandas i Matplotlib prilagodavanja.

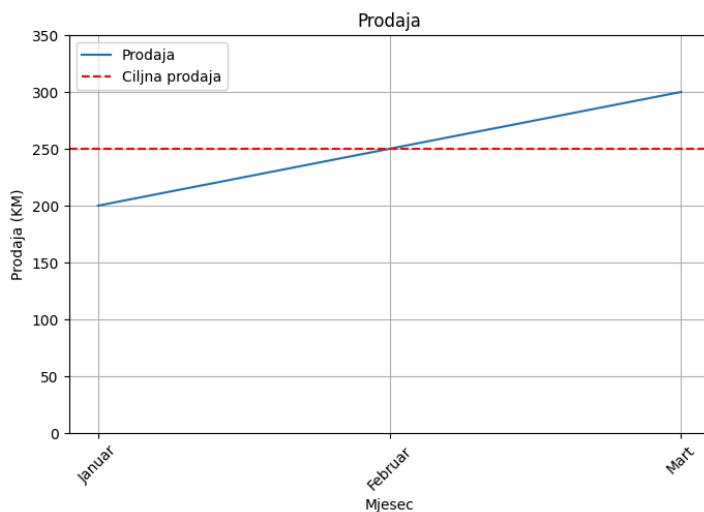
```
import pandas as pd
import matplotlib.pyplot as plt

# Priprema podataka
data = {
    "Mjesec": ["Januar", "Februar", "Mart"],
    "Prodaja": [200, 250, 300]
}
df = pd.DataFrame(data)

# Kreiranje osnovnog grafika s Pandas-om
graf = df.plot(x="Mjesec", y="Prodaja", kind="line", title="Prodaja", figsize=(8, 5), grid=True)

# Dodatno prilagođavanje koristeći Matplotlib
graf.set_xlabel("Mjesec")
graf.set_ylabel("Prodaja (KM)")
graf.set_xticks(range(len(df)))
graf.set_xticklabels(df["Mjesec"], rotation=45)
graf.set_ylim(0, 350)
graf.axhline(y=250, color="red", linestyle="--", label="Ciljna prodaja")
graf.legend(loc="upper left")

# Prikaz grafika
plt.show()
```



Primjer prikazuje kombinaciju osnovnog crtanja grafika pomoću `df.plot` i dodatnih prilagođavanja koristeći Matplotlib. Osnovni grafik kreiran je sa zadatim naslovom, veličinom i mrežom. Dodatno su prilagođeni nazivi osa, postavljeni prilagođeni

tickovi za x-osi s rotiranim nazivima mjeseci, ograničen opseg y-ose, te dodana horizontalna linija, koja označava ciljni nivo prodaje.

4.5.6.3. Tipovi grafika

Pandas pruža mogućnost kreiranja različitih tipova grafika osim podrazumjevanog linijskog grafika. Ovi tipovi grafika se specificiraju putem parametra `kind` unutar metode `plot()`. Svaki tip grafika ima svoje specifičnosti i koristi se za vizualizaciju podataka na različite načine.

Bar grafik

Stubičasti grafik se koriste za prikazivanje kategorizovanih podataka. Vertikalni stubičasti grafik koriste `kind='bar'`, dok horizontalni koriste `kind='barh'`.

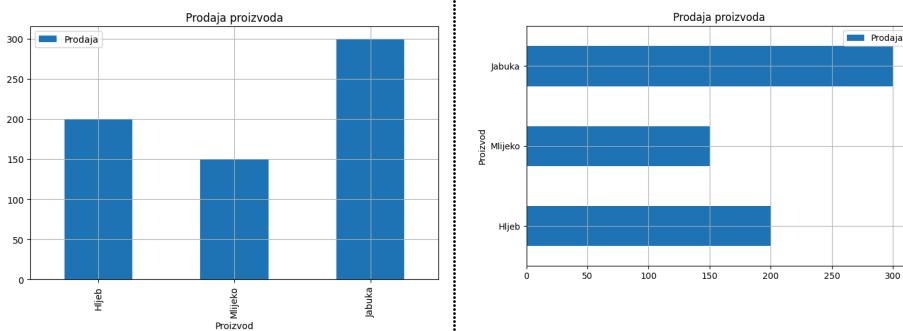
Primjer:

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

# Priprema podataka
data = {
    "Proizvod": ["Hljeb", "Mlijeko", "Jabuka"],
    "Prodaja": [200, 150, 300]
}
df = pd.DataFrame(data)

# Stubičasti grafik vertikalni
df.plot(x="Proizvod", y="Prodaja", kind="bar", title="Prodaja proizvoda",
grid=True, figsize=(8, 5))
plt.show()

# Stubičasti grafik horizontalni
df.plot(x="Proizvod", y="Prodaja", kind="barh", title="Prodaja proizvoda",
grid=True, figsize=(8, 5))
plt.show()
```



Histogram grafik

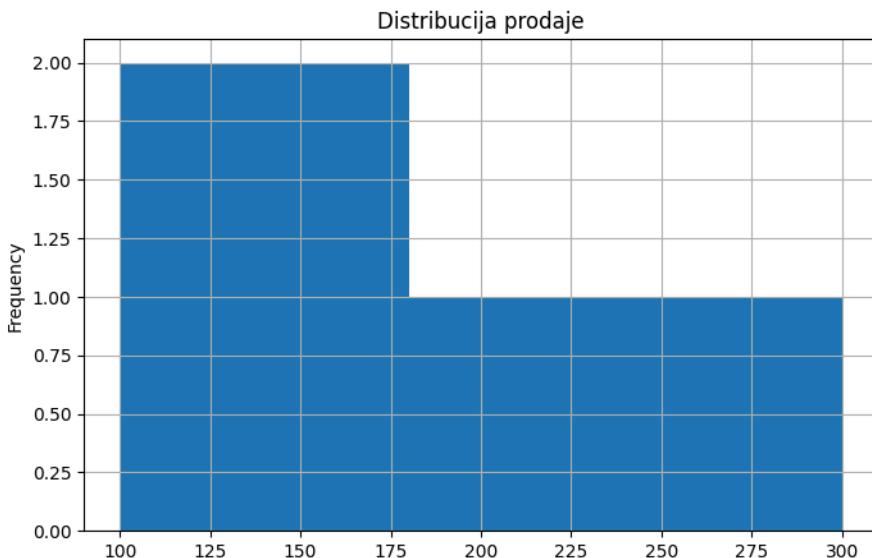
Histogram se koristi za prikazivanje distribucije podataka tako što dijeli podatke u intervale (binove) i prikazuje učestalost pojavljivanja vrijednosti unutar tih intervala. U Pandas-u se histogram kreira pomoću `kind="hist"` u funkciji `plot`, pri čemu ključni parametar `bins` određuje broj intervala, koji se koristi za grupisanje podataka.

U ekonomiji se histogrami često koriste za analizu raspodjele dohotka, troškova, cijena proizvoda ili drugih kvantitativnih varijabli, omogućavajući identifikaciju obrazaca, poput simetrije, disperzije ili koncentracije podataka u određenim vrijednostima.

Primjer histogram pokazuje kako su vrijednosti prodaje raspoređene u definisanim intervalima, što pomaže u razumijevanju njihovog ukupnog obrasca.

```
import pandas as pd
import matplotlib.pyplot as plt
# Priprema podataka
data = {
    "Prodaja": [100, 150, 200, 150, 100, 250, 300]
}
df = pd.DataFrame(data)

# Histogram
df["Prodaja"].plot(kind="hist", title="Distribucija prodaje", bins=5, grid=True,
figsize=(8, 5))
plt.show()
```



Boxplot grafik

Boxplot, poznat i kao „kutija s brkovima“¹⁶, koristi se za prikazivanje raspodjele podataka, njihovih medijana, kvartila i potencijalnih ekstremnih vrijednosti. Ovaj grafik vizualno sumira centralne tendencije, raspon i varijacije podataka u jednoj varijabli. U Pandas-u se kreira pomoću `kind="box"` u funkciji `plot`.

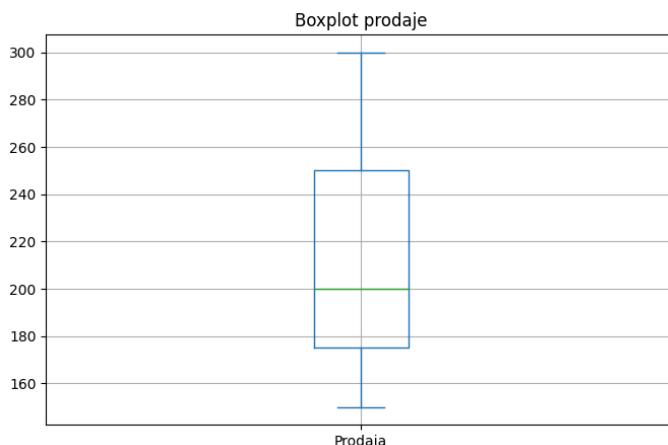
U ekonomiji, boxploti se često koriste za analizu distribucije plata, profita, troškova ili bilo koje druge varijable, koja zahtijeva identifikaciju ekstremnih vrijednosti i poređenje među grupama.

Primjer koji jasno prikazuje medijanu, interkvartilni raspon (IQR) i potencijalne ekstremne vrijednosti.

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

# Priprema podataka
data = {
    "Proizvod": ["Hljeb", "Mlijeko", "Jabuka"],
    "Prodaja": [200, 150, 300]
}
df = pd.DataFrame(data)

# Boxplot
df[["Prodaja"]].plot(kind="box", title="Boxplot prodaje", grid=True, figsize=(8, 5))
plt.show()
```



Density grafik

Density plot, poznat i kao grafik gustine, koristi se za vizualizaciju procjene distribucije podataka pomoću glatke krive. Za razliku od histograma, koji prikazuje

¹⁶ Termin „kutija s brkovima“ (eng. boxplot) potiče od grafičkog izgleda ovog tipa grafika. Naime, centralni dio grafika, pravougaonik ili „kutija“, predstavlja interkvartilni raspon (interquartile range, IQR), tj. opseg između prvog (Q1) i trećeg kvartila (Q3). Iz „kutije“ se protežu linije koje predstavljaju raspon podataka unutar definisanih granica, a te linije su figurativno nazvane „brkovima“.

podatke grupisane u binove, density plot koristi *Kernel Density Estimation (KDE)* za stvaranje kontinuirane krive, koja bolje odražava obrazce u podacima. Ovaj tip grafika omogućava jednostavno poređenje distribucija više varijabli. U Pandas-u se kreira korištenjem `kind="kde"` ili `kind="density"` u funkciji `plot`.

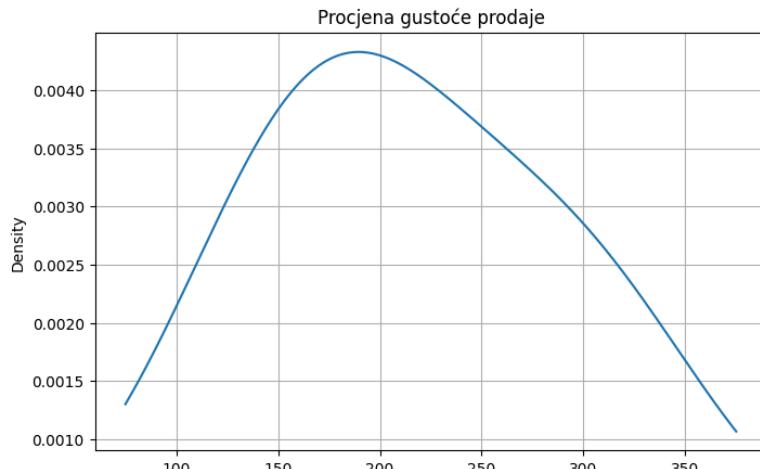
Density plot se često koristi za analizu distribucije plata, troškova, cijena proizvoda ili kamatnih stopa, omogućavajući vizualno prepoznavanje modalnosti (npr. unimodalne ili multimodalne distribucije) i koncentracije podataka.

Primjer grafičkog prikaza procjene gustine prodaje proizvoda.

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

# Priprema podataka
data = {
    "Proizvod": ["Hljeb", "Mlijeko", "Jabuka"],
    "Prodaja": [200, 150, 300]
}

# Density grafik
df["Prodaja"].plot(kind="kde", title="Procjena gustoće prodaje", grid=True,
figsize=(8, 5))
plt.show()
```



Scatter grafik

Scatter grafik koristi se za prikazivanje odnosa između dvije varijable, gdje svaka tačka na grafiku predstavlja par vrijednosti iz dataset-a. Jedna varijabla se prikazuje na x-osi, a druga na y-osi. Ovaj grafik je koristan za identifikaciju obrazaca, povezanosti ili potencijalnih korelacija između varijabli. U Pandas-u se kreira korištenjem `kind="scatter"` u funkciji `plot`.

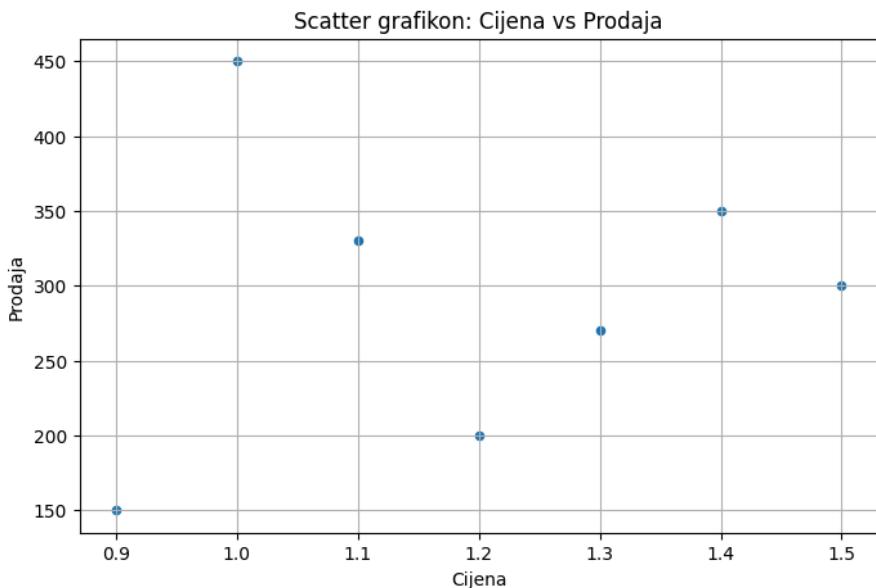
Scatter plot se često koristi za analizu odnosa između cijena i prodaje proizvoda, prihoda i troškova, zaposlenosti i BDP-a ili drugih ekonomskih varijabli. Pomaže u prepoznavanju trendova i anomalija, kao i procjeni linearne ili nelinearne korelacijske.

Primjer koji grafički prikazuje međusobni odnos prodaje i cijene.

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

# Priprema podataka
data = {
    "Cijena": [1.2, 0.9, 1.5, 1.1, 1.0, 1.4, 1.3],
    "Prodaja": [200, 150, 300, 330, 450, 350, 270]
}
df = pd.DataFrame(data)

# Scatter grafik
df.plot(x="Cijena", y="Prodaja", kind="scatter", title="Scatter grafik: Cijena vs Prodaja", grid=True, figsize=(8, 5))
plt.show()
```



Hexbin grafik

Hexbin grafik se koristi za prikazivanje gustine tačaka na grafiku, gdje se podaci grupišu u heksagonalne ćelije umjesto pojedinačnih tačaka. Ovaj grafik je posebno koristan kada se radi s velikim setovima podataka, jer omogućava bolji pregled gustine podataka, umjesto preklapanja tačaka koje je često prisutno na scatter graficima.

U Pandas-u se kreira korištenjem `kind="hexbin"` u funkciji `plot`. Hexbin grafik zahtijeva dvije numeričke varijable, koje se koriste za x i y osi i parametar `gridsize` koji određuje veličinu heksagonalnih ćelija.

Hexbin grafik se koristi za analizu gustine podataka u korelacijama između varijabli, poput odnosa cijena i količina, prihoda i troškova ili zaposlenosti i BDP-a. Omogućava vizualnu procjenu gdje se podaci najčešće pojavljuju.

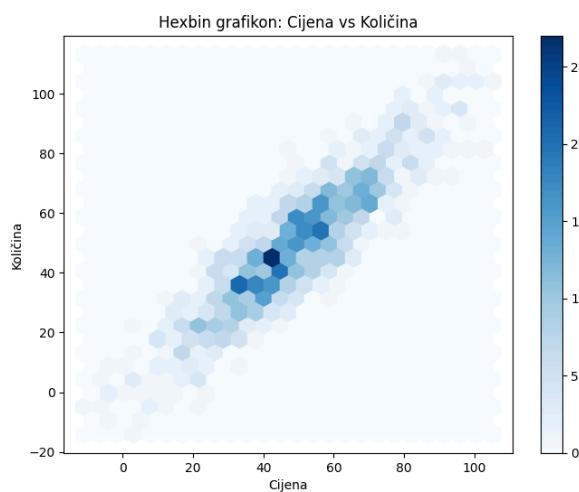
Primjer sa slučajno generisanim setovima podataka.

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

# Priprema podataka
np.random.seed(0)
x = np.random.randn(1000) * 20 + 50 # Nasumične cijene
y = x + np.random.randn(1000) * 10 # Nasumične količine s korelacijom

df = pd.DataFrame({"Cijena": x, "Količina": y})

# Hexbin grafik
df.plot(kind="hexbin", x="Cijena", y="Količina", gridsize=25, cmap="Blues",
        title="Hexbin grafik: Cijena vs Količina", figsize=(8, 6))
plt.show()
```



Pie grafik

Pie grafik koristi se za vizualizaciju procentualne raspodjele podataka, gdje svaki segment grafika predstavlja udio određene kategorije u ukupnom zbiru. Ovaj tip grafika je koristan za prikazivanje relativnih udjela i proporcija, čineći ga intuitivnim i jednostavnim za interpretaciju.

U Pandas-u se kreira korištenjem `kind="pie"` u funkciji `plot`. Da bi se kreirao pie grafik, podaci trebaju biti numerički i povezani s odgovarajućim kategorijama.

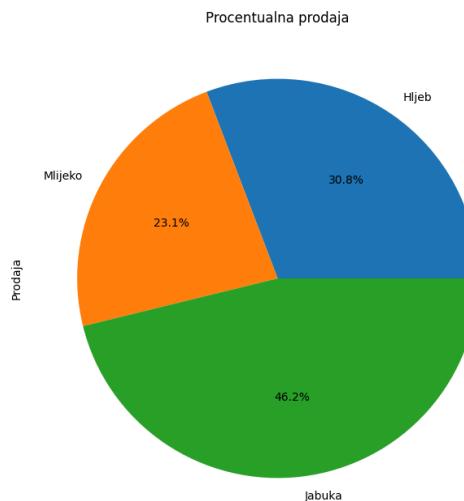
Pie grafik se koristi za prikazivanje udjela kategorija u ukupnom prihodu, troškovima, tržišnom udjelu, potrošnji i drugim sličnim analizama. Omogućava brzu vizualnu procjenu dominantnih kategorija.

Primjer raspodjele prodaje po proizvodima.

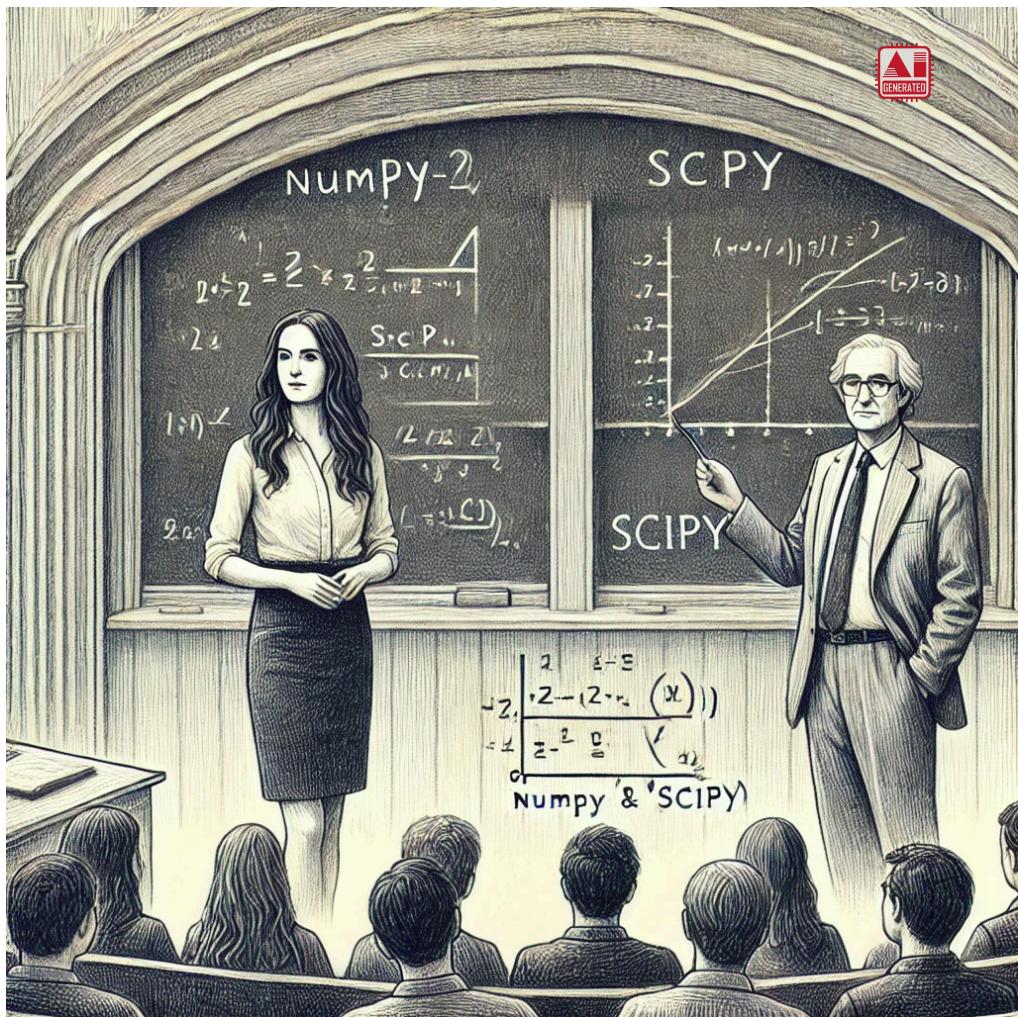
```
import pandas as pd
import matplotlib.pyplot as plt

# Priprema podataka
data = {
    "Proizvod": ["Hljeb", "Mlijeko", "Jabuka"],
    "Prodaja": [200, 150, 300]
}
df = pd.DataFrame(data)

# Pie grafik
df.set_index("Proizvod")["Prodaja"].plot(kind="pie", title="Procentualna prodaja",
autopct="%1.1f%%", figsize=(8, 8))
plt.show()
```



5. PYTHON I MATEMATIKA



"The purpose of computing is insight, not numbers."

"Svrha računarstva je uvid, a ne brojevi."

— Richard Hamming¹⁷

¹⁷ Richard Hamming (1915-1998), bio je američki matematičar i pionir računarstva, poznat po doprinosima teoriji informacija, kodiranju i razvoju algoritama. Njegov rad na Hammingovim kodovima i analizi grešaka postavio je temelje za modernu digitalnu komunikaciju i računarstvo. Izvor: Knjiga "The Art of Doing Science and Engineering"

Ekonomija se oslanja na brojne matematičke i statističke metode, kako bi analizirala podatke i donosila informisane zaključke. Korištenjem ovih metoda može se preciznije opisati i predvidjeti ponašanje tržišta, izračunati različiti finansijski pokazatelji ili izvršiti kompleksne analize troškova i prihoda. Uvođenje matematičkih i statističkih koncepata u ekonomske analize povećava tačnost modela i pomaže u boljem razumijevanju odnosa među varijablama.

Ključne teme koje će biti obrađene u ovom poglavlju su:

- **Matrice:** Ključ za razumijevanje linearog algebarskog pristupa u ekonomiji, kao što je analiza input-output modela ili rješavanje sistema linearnih jednačina.
- **Funkcije:** Upotreba različitih funkcija (linearnih, kvadratnih, eksponencijalnih, logaritamskih) u objašnjenu ekonomske dinamike.
- **Nizovi:** Aritmetički i geometrijski nizovi te njihova granična vrijednost, s primjenama u modelima rasta, diskontovanja i finansijskoj matematici.
- **Diferencijalni račun:** Primjena izvoda i diferencijala u ekonomskim analizama (marginalne funkcije, elastičnost, optimizacija).
- **Integralni račun:** Obračun površina, kumulativnih veličina i njihova veza s ekonomskim pokazateljima.

5.1. Matrice

Linearna algebra predstavlja jednu od temeljnih grana matematike, koja se bavi proučavanjem vektora, matrica i linearnih transformacija. Njena važnost se ogleda u širokom spektru primjena: od fizike, inženjerstva i računarstva, pa sve do ekonomije i društvenih nauka. U ekonomiji se matrice često koriste u modelima input-output analize, pri obradi i analizi statističkih podataka, te u finansijskom modeliranju.

U ovom poglavlju fokusiraćemo se na osnovne pojmove o matricama, njihovim tipovima i osnovnim operacijama (sabiranje, množenje i transponovanje). Nakon matematičkog uvoda, prikazaćemo i kako se te iste operacije mogu jednostavno realizovati u programskom jeziku Python, koristeći biblioteke poput **NumPy** za efikasne matematičke operacije i rad s nizovima, **SciPy** za napredne algoritme, optimizaciju i statistiku, te **matplotlib** za vizualizaciju.

5.1.1. Osnovni pojmovi o matricama

Definicija matrice: Matrica se u matematici definiše kao pravougaona (ili kvadratna) tabela brojeva poredanih u redove i kolone. Formalno, ako je data matrica A dimenzija $m \times n$, to znači da A ima m redova i n kolona. Element matrice A koji se nalazi u i -tom redu i j -toj koloni obično se označava kao a_{ij} .

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

Gdje je:

- m = broj redova
- n = broj kolona
- a_{ij} = element u i -tom redu i j -toj koloni

Ovo predstavlja matematičku formulaciju matrice, odnosno apstraktni koncept koji se može grafički prikazati, a do kog se dolazi sintezom jednostavnih početnih prepostavki i informacija. Da bi taj grafički prikaz pojednostavili i učinili ga razumljivijim, možemo početi od dvije praktične tabele, koje opisuju svakodnevnu situaciju. Uzmimo na primjer, prodaju različitih proizvoda na različitim prodajnim mjestima tokom dvije uzastopne godine.

Posmatrajmo kompaniju "ABC" koja ima četiri proizvoda (P_1, P_2, P_3, P_4) i tri prodajna mesta (M_1, M_2, M_3). Napraviti ćemo tabelu koja prikazuje količinu prodaje za 2023. godinu, a zatim i tabelu za 2024. godinu.

Tabela 5.1. Prodaje kompanije “ABC” za 2023. godinu

2023	M ₁	M ₂	M ₃
P ₁	10	5	12
P ₂	7	2	9
P ₃	13	14	6
P ₄	8	1	4

- Redovi (4 reda) predstavljaju proizvode P₁, P₂, P₃, P₄.
- Kolone (3 kolone) predstavljaju prodajna mesta M₁, M₂, M₃.
- Svaka ćelija u tabeli pokazuje koliko je komada nekog proizvoda prodano na određenom mjestu u 2023. godini.

U nastavku ćemo ovu tabelu 5.1. označiti sa slovom A.

- Broj redova je m = 4 (pošto imamo 4 proizvoda).
- Broj kolona je n = 3 (pošto imamo 3 prodajna mesta).

U matematičkom smislu, ovo je matrica dimenzija 4×3 i možemo je zapisati kao:

$$A = \begin{bmatrix} 10 & 5 & 12 \\ 7 & 2 & 9 \\ 13 & 14 & 6 \\ 8 & 1 & 4 \end{bmatrix}$$

Element a_{ij} predstavlja količinu prodaje proizvoda *i* na mjestu *j*. (a₁₁=10, a₄₃=4)

Tabela 5.2. Prodaje kompanije “ABC” za 2024. godinu

2024	M ₁	M ₂	M ₃
P ₁	12	6	15
P ₂	6	3	10
P ₃	15	12	8
P ₄	9	2	5

Tabelu 5.2. ćemo označiti sa B, i nju takođe možemo prikazati kao matricu 4x3

$$B = \begin{bmatrix} 12 & 6 & 15 \\ 6 & 3 & 10 \\ 15 & 12 & 8 \\ 9 & 2 & 5 \end{bmatrix}$$

Kako se od “tabele” dolazi do “matrice”

Matrica se može posmatrati kao pravougaona tabela brojeva raspoređenih u redove i kolone. Drugim riječima, kada kažemo da je matrica A dimenzija $m \times n$, mislimo na skup brojeva raspoređena u m redova i n kolona. Primjer za to možemo vidjeti u tabelama prodaje za 2023. i 2024. godinu, gdje imamo četiri reda (broj proizvoda) i tri kolone (broj prodajnih mjesta), što znači da svaka od tih tabela odgovara matrici dimenzija 4×3 . U konkretnom slučaju, označili smo matricu za 2023. godinu kao A , a matricu za 2024. godinu kao B , pri čemu obje imaju formu 4×3 .

U matrici A , element a_{11} predstavlja vrijednost u prvom redu i prvoj koloni, što u našem primjeru znači “prodaja proizvoda P_1 na mjestu M_1 tokom 2023. godine”. Uopšteno, a_{ij} označava “količinu proizvoda P_i prodatu na mjestu M_j ”. Na taj način, svaka celija matrice ima jasno značenje, a čitava tabela (odnosno matrica) pruža pregled svih podataka raspoređenih u označene redove i kolone.

Dimenzije i oznake matrica

Dimenzije i oznake predstavljaju dva ključna aspekta, koji nam pomažu da jasno definisemo svaku matricu. Dimenziju matrice obično iskazujemo parom brojeva $(m \times n)$, gdje je m broj redova, a n broj kolona. Na taj način, kada kažemo da je matrica A dimenzija 3×5 , odmah znamo da ona ima tri reda i pet kolona.

Pored dimenzija, važno je обратити pažnju i na oznake matrica. Najčešće se upotrebljavaju slova iz početnog dijela alfabeta (A, B, C) ili iz kraja alfabeta (X, Y, Z), zavisno od konteksta. U algebarskim, finansijskim ili ekonomskim modelima, ove oznake često imaju specifično značenje: na primjer, A može predstavljati matricu koeficijenata u jednom sistemu linearnih jednačina, dok X označava vektor nepoznatih veličina.

Zašto nam je važno da “mislimo” u matricama.

Matrice nam omogućavaju jednostavniju primjenu matematičkih operacija, jer možemo sabirati dvije matrice A i B i tako dobiti, recimo, ukupnu prodaju za 2023. i 2024. godinu element-po-element. Na isti način, lako možemo oduzimati, množiti, transponovati i primjenjivati različite matematičke postupke, koji se često koriste u poslovnim i analitičkim situacijama. Osim toga, razmišljanje u formi matrica olakšava sistematizaciju podataka i time ih čini pregleđnijim za dalju analizu. U ekonomiji, statistici ili mašinskom učenju, rad s matricama pomaže da se efikasno upravlja velikim količinama informacija.

Posmatrajmo jednostavnu operaciju sabiranja dvije tabele odnosno operaciju sabiranja dvije matrice. Ako želimo dobiti ukupnu prodajuza dvije godine (2023 + 2024) po proizvodu i prodajnom mjestu, tu operaciju možemo matematički iskazati kao:

$$C = A + B$$

$$C = \begin{bmatrix} 10 & 5 & 12 \\ 7 & 2 & 9 \\ 13 & 14 & 6 \\ 8 & 1 & 4 \end{bmatrix} + \begin{bmatrix} 12 & 6 & 15 \\ 6 & 3 & 10 \\ 15 & 12 & 8 \\ 9 & 2 & 5 \end{bmatrix} = \begin{bmatrix} 22 & 11 & 27 \\ 14 & 4 & 19 \\ 28 & 26 & 14 \\ 17 & 2 & 9 \end{bmatrix}$$

C će opet biti matrica (4x3), gdje je svaka celija zbir odgovarajućih celija iz matrice **A** i iz matrice **B**. Naprimjer $c_{11}=a_{11}+b_{11}=10+12=22$.

Tabela 5.3. Ukupna prodaja za 2023. i 2024. godinu

Ukupno	M ₁	M ₂	M ₃
P ₁	10+12	5+6	12+15
P ₂	7+6	2+3	9+10
P ₃	13+15	14+12	6+8
P ₄	8+9	1+2	4+5

Isti primjer ćemo uraditi u Python okruženju koristeći biblioteku NumPy, koja je specijalizovana za složena izračunavanja i rad s matricama.

```
import numpy as np

# Matrica A (2023)
A = np.array([
    [10, 5, 12],
    [7, 2, 9],
    [13, 14, 6],
    [8, 1, 4]
])

# Matrica B (2024)
B = np.array([
    [12, 6, 15],
    [6, 3, 10],
    [15, 12, 8],
    [9, 2, 5]
])

# Sabiranje A + B
C = A + B

print("Zbir matrica A i B (ukupna prodaja 2023 + 2024):\n", C)
```

Zbir matrica A i B (ukupna prodaja 2023 + 2024):

$$\begin{bmatrix} 22 & 11 & 27 \\ 13 & 5 & 19 \\ 28 & 26 & 14 \\ 17 & 3 & 9 \end{bmatrix}$$

Rezultat će biti nova matrica (ista veličina 4x3) koja za svaku celiju sadrži zbir prodaja iz 2023. i 2024. godine.

5.1.2. Tipovi matrica

Tipovi matrica obuhvataju nekoliko karakterističnih oblika, koji se često javljaju u matematici i njenim primjenama. U nastavku ćemo objasniti pojам i značaj kvadratnih, pravougaonih (opštih), dijagonalnih i jediničnih (identičnih) matrica, uz kratke primjere za svaki od ovih oblika.

Kvadratna matrica ima isti broj redova i kolona, odnosno $m=n$. Kao primjer, možemo uzeti matricu

$$\begin{bmatrix} 2 & -1 \\ 4 & 0 \end{bmatrix},$$

koja je dimenzija 2×2 . Budući da se broj redova poklapa s brojem kolona, ova matrica spada u kvadratne. Kvadratne matrice su bitne jer omogućavaju definisanje važnih pojmova poput determinante i inverzne matrice, što je od presudnog značaja pri rješavanju sistema linearnih jednačina i analiziranju različitih transformacija.

Pravougaona (opšta) matrica posjeduje različit broj redova i kolona, dakle $m\neq n$. Kao primjer, možemo navesti matricu

$$\begin{bmatrix} 3 & 5 & -2 \\ 1 & 0 & 7 \end{bmatrix},$$

koja ima dimenziju 2×3 . Ovakve matrice vrlo često nalazimo u praksi, recimo prilikom prikazivanja tabelarnih podataka, gdje se broj "redova" i "kolona" ne mora nužno podudarati.

Dijagonalna matrica predstavlja poseban oblik kvadratne matrice u kojoj su svi elementi izvan dijagonale jednaki nuli. Kao primjer, možemo uzeti

$$\begin{bmatrix} 5 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & -1 \end{bmatrix},$$

gdje su jedino elementi na glavnoj dijagonali različiti od nule, dok su svi ostali elementi nule. Zahvaljujući ovoj strukturi, dijagonalne matrice omogućavaju veoma jednostavne računarske operacije poput množenja i invertovanja.

Jedinica (jedinična, identična) matrica jeste poseban slučaj dijagonalne matrice, kod koje su svi elementi na glavnoj dijagonali jednaki 1, a svi ostali elementi jednaki 0. Ova matrica se najčešće označava kao **I** i ima oblik:

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix}$$

Kada ovakvom matricom množimo bilo koju drugu matricu (odgovarajućih dimenzija), dobijamo ponovo tu istu matricu, što jedinicu (identičnu) matricu čini fundamentalnim gradivnim elementom u linearnoj algebri.

5.1.3. Osnovne operacije nad matricama

Matematičke operacije s matricama u određenoj se mjeri razlikuju od uobičajenih operacija sabiranja, oduzimanja, množenja i dijeljenja primijenjenih na obične brojeve. Razlog za to leži u samoj prirodi matrice kao višedimenzione strukture. U nastavku ćemo prikazati osnovne operacije i određene transformacije matrica, uključujući primjere kako se izvode operacije između dviju matrica te operacije između matrice i skalara. Cilj ovih prikaza jeste da se demonstriraju mogućnosti upotrebe matrica u linearnoj algebri, kao i način rješavanju problema u ekonomiji koristeći Python okruženje i njegove alate za rad sa matricama.

5.1.3.1. Sabiranje i oduzimanje matrica

Operacije sabiranja i oduzimanja matrica možemo posmatrati kao komplementarne. Obje se izvode element-po-element i zahtijevaju da matrice budu iste dimenzije ($m \times n$). Neka su, dakle, date dvije matrice **A** i **B** dimenzija $m \times n$. Tada za sabiranje dobijamo matricu **C=A+B**, pri čemu se svaki element računa kao:

$$c_{ij} = a_{ij} + b_{ij}, \quad \forall i = 1, 2, \dots, m, \quad j = 1, 2, \dots, n$$

dok se za oduzimanje definiše **C=A-B** uz formulu:

$$c_{ij} = a_{ij} - b_{ij}, \quad \forall i = 1, 2, \dots, m, \quad j = 1, 2, \dots, n$$

Primjena u ekonomiji: Ako posmatramo dvije različite tabele s ekonomskim pokazateljima (npr. matrične prikaze input-output odnosa u različitim godinama),

njihovim sabiranjem možemo dobiti zbirnu tabelu, koja pokazuje kumulativne input-output odnose između grana privrede. Kada bismo, na primjer, željeli da uvidimo promjenu između dvije takve tabele (npr. porast ili pad pokazatelja u jednoj godini u odnosu na drugu), mogli bismo to da učinimo njihovim oduzimanjem.

Sljedeći primjer prikazuje sabirane i oduzimanje matrica.

```
import numpy as np

# Definisanje matrica A i B (dimenzija 2x2 radi jednostavnosti)
A = np.array([
    [1, 2],
    [3, 4]
])

B = np.array([
    [5, 6],
    [7, 8]
])

# Zbir matrica A + B
C = A + B

# Razlika matrica A - B
D = A - B

print("Matrica A:\n", A)
print("Matrica B:\n", B)

print("Zbir matrica A + B:\n", C)
print("Razlika matrica A - B:\n", D)
```

Matrica A: [[1 2] [3 4]]	Matrica B: [[5 6] [7 8]]
--------------------------------	--------------------------------

Zbir matrica A + B: [[6 8] [10 12]]	Razlika matrica A - B: [[-4 -4] [-4 -4]]
---	--

5.1.3.2. Sabiranje (oduzimanje) matrice i skalara

Matematički, operacija sabiranja matrice i realnog broja nije definisana u standardnoj teoriji linearne algebре. Matrica je skup uredenih brojeva organizovanih u redove i kolone, dok je skalar (realan broj) pojedinačna vrijednost. Da bi operacija sabiranja bila matematički ispravna, obje strane moraju imati iste dimenzije. Drugim riječima, da bismo postigli ekvivalentno sabiranje, realan broj α mora biti predstavljen kao matrica iste dimenzije kao matrica A , gdje svaki element te nove matrice ima vrijednost α .

Neka je A matrica dimenzija $m \times n$, čiji su elementi a_{ij} , i neka je α skalar. Matrica α' , koja odgovara skalaru α , definisana je kao:

$$\alpha' = \begin{bmatrix} \alpha & \alpha & \cdots & \alpha \\ \alpha & \alpha & \cdots & \alpha \\ \vdots & \vdots & \ddots & \vdots \\ \alpha & \alpha & \cdots & \alpha \end{bmatrix},$$

Tada je sabiranje matrice A i matrice α' formalno dato kao:

$$A + \alpha' = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} + \begin{bmatrix} \alpha & \alpha & \cdots & \alpha \\ \alpha & \alpha & \cdots & \alpha \\ \vdots & \vdots & \ddots & \vdots \\ \alpha & \alpha & \cdots & \alpha \end{bmatrix} = \begin{bmatrix} a_{11} + \alpha & a_{12} + \alpha & \cdots & a_{1n} + \alpha \\ a_{21} + \alpha & a_{22} + \alpha & \cdots & a_{2n} + \alpha \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} + \alpha & a_{m2} + \alpha & \cdots & a_{mn} + \alpha \end{bmatrix}$$

Elementi rezultujuće matrice C = A + α' definišu se kao:

$$c_{ij} = a_{ij} + \alpha, \quad \forall i \in \{1, 2, \dots, m\}, j \in \{1, 2, \dots, n\}$$

Iako sabiranje matrice i skalara nije standardno u linearном algebarskom smislu, u programerskom kontekstu ovo se pojednostavljuje i umjesto da od skalara α kreiramo matricu α' , to automatski uradi modul unutar Pythona. Matrica nije matematički objekat u klasičnom smislu, već je predstavljena kao niz sa dvije dimenzije. Skalarni broj u ovom kontekstu nije eksplicitno pretvoren u matricu iste dimenzije, već se koristi *broadcasting*.

Broadcasting je mehanizam koji omogućava NumPy-u da automatski proširi dimenzije skalara (ili niza s manjim dimenzijama) kako bi se uklopile u operaciju s nizom viših dimenzija. U ovom slučaju, kada dodajemo broj α matrici A, NumPy tretira α kao da je "razvučen" u matricu iste dimenzije kao A, omogućavajući sabiranje svakog elementa A[i][j] s α .

Drugim riječima, svaki element matrice A uvećava se ili umanjuje se za α . Ovo može biti korisno, na primjer, kada je potrebno "podesiti" sve vrijednosti u tabeli za neki fiksni faktor (indeksacija, podešavanje zbog inflacije i sl.).

Sljedeći primjer opisuje sabiranje i oduzimanje matrice i skalara u Pythonu.

```
import numpy as np

# Definisanje matrica A i B (dimenzija 2x2 radi jednostavnosti)
A = np.array([
    [1, 2],
    [3, 4]
])
# Skalat alpha
alpha = 10

# Sabiranje matrice A sa skalarom
```

```

A_plus_alpha = A + alpha

# Oduzimanje matrice A sa skalarom
A_minus_alpha = A - alpha

print("Matrica A:\n", A)
print(f"Matrica A + {alpha}:\n", A_plus_alpha)
print(f"Matrica A - {alpha}:\n", A_minus_alpha)

Matrica A:
[[1 2]
 [3 4]]
Matrica A + 10:
[[11 12]
 [13 14]]
Matrica A - 10:
[[-9 -8]
 [-7 -6]]

```

5.1.3.3. Množenje matrica

Za množenje matrica mora biti ispunjen uslov da je broj kolona prve matrice jednak broju redova druge matrice.

Ako je A dimenzija $m \times n$ i B dimenzija $n \times p$, tada je produkt $C = A \cdot B$ dimenzija $m \times p$. Elementi matrice C izračunavaju se kao:

$$c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}$$

Primjena u ekonomiji: U Leontijevim input-output modelima, matrice inputa i outputa se množe s vektorima cijena i količina da bi se utvrdile različite raspodjele resursa i efekti na različite sektore privrede.

5.1.3.4. Množenje matrice i skalara

Osim množenja dviju matrica, često se u praksi susreće i množenje matrice nekim skalarom (brojem). Neka je A matrica dimenzija $(m \times n)$, a α realan (ili kompleksan) broj. Tada je proizvod αA ponovo matrica $(m \times n)$ čiji je svaki element jednak α -put uvećanom odgovarajućem elementu matrice A . Matematički, to zapisujemo kao:

$$(\alpha A)_{ij} = \alpha \cdot a_{ij}$$

U praktičnom smislu, svaka celija matrice A se množi istim brojem α . Ovo je korisno, na primjer, kada želimo skalirati sve vrijednosti u tabeli za jedan isti faktor (npr. prilagođavanje cijena zbog inflacije, konverzija valute, promjena vrijednosti PDV-a i sl.).

Sljedeći primjer prikazuje operacije množenja nad matricama.

```
import numpy as np

# Definišemo matricu A (dimenzija 2x3)
A = np.array([
    [1, 2, 3],
    [4, 5, 6]
])

# Definišemo matricu B (dimenzija 3x2) - kako bi se mogla množiti sa A
B = np.array([
    [7, 8],
    [9, 10],
    [11, 12]
])

# Množenje matrica A i B
# Očekivana dimenzija rezultujuće matrice: (2x2)
C = np.dot(A, B)

print("Matrica A:\n", A)
print("Matrica B:\n", B)
print("Rezultat množenja A * B (matrica C):\n", C)

# Definisanje skalara
alpha = 3

# Množenje matrice A skalarom alpha
A_skalar = alpha * A

print("Originalna matrica A:\n", A)
print("Skalar alpha =", alpha)
print("Proizvod alpha * A:\n", A_skalar)
```

Matrica A: [[1 2 3] [4 5 6]] Matrica B: [[7 8] [9 10] [11 12]] Rezultat množenja A * B (matrica C): [[58 64] [139 154]]	Originalna matrica A: [[1 2 3] [4 5 6]] Skalar alpha = 3 Proizvod alpha * A: [[3 6 9] [12 15 18]]
---	--

5.1.3.5. Transponovanje matrice

Transponovanje matrice A (označava se najčešće kao A^T) vrši se tako da se redovi zamijene kolonama, odnosno:

$$(A^T)_{ij} = A_{ji}$$

Ako je A dimenzija $m \times n$, tada je A^T dimenzija $n \times m$.

Primjena: Transponovanje se često primjenjuje prilikom prilagođavanja formata podataka, npr. kada je potrebno vektore kolona pretvoriti u vektore redova (ili obratno) radi lakše analize ili kada algoritmi zahtijevaju određeni oblik ulaznih podataka.

Primjer:

```
import numpy as np

# Primjer kreiranja matrica A i B
A = np.array([
    [1, 2, 3],
    [4, 5, 6]
])

# Transponovanje matrice
A_transponovana = A.T

print("Matrica A:\n", A)
print("Dimenzija matrica A:", A.shape)

print("Transponovana matrica A:\n", A_transponovana)
print("Dimenzija Transponovana matrica A:", A_transponovana.shape)
```

Matrica A:
[[1 2 3]
 [4 5 6]]
Dimenzija matrica A: (2, 3)
Transponovana matrica A:
[[1 4]
 [2 5]
 [3 6]]
Dimenzija Transponovana matrica A: (3, 2)

Objašnjenje:

- `np.array([...])` kreira NumPy niz koji se može tretirati kao matrica.
- `A.shape` vraća dimenzije matrice.
- `A.T` ili `A.transpose()` vraća transponovanu matricu.

5.1.4. Napredne operacije sa matricama

Napredne operacije s matricama omogućavaju rješavanje složenijih matematičkih problema, posebno u oblasti linearne algebre i ekonomskih modela. Ove operacije prevazilaze osnovne zadatke poput sabiranja, oduzimanja ili množenja i bave se transformacijama, koje zahtijevaju dublje razumijevanje osobine matrica. Među ključnim naprednim operacijama su izračunavanje determinante, inverzne matrice, ranga matrice i dijeljenja matrica.

Dijeljenje matrica, iako je osnovna operacija, zahtijeva inverznu matricu. Definiše se kao množenje matrice A inverznom matricom B, pri čemu B mora biti kvadratna i imati determinantu različitu od nule. Ova operacija je ključna u ekonomskim

analizama, poput rješavanja sistema linearnih jednačina i modeliranja input-output odnosa, što čini razumijevanje naprednih operacija temeljem primjene linearne algebre.

5.1.4.1. Determinanta

Definicija: Za kvadratnu matricu A dimenzija $n \times n$, determinanta $\det(A)$ (ili $|A|$) predstavlja realan ili kompleksan broj (zavisno od elemenata matrice), ako je ispunjen uslov da je:

$$\det(A) \neq 0$$

Tada za matricu A kažemo da je invertibilna, odnosno da posjeduje inverznu matricu.

U slučaju da je $\det(A)=0$, matrica nije invertibilna, odnosno nema jedinstveno rješenje kada se primijeni na sistem jednačina.

U sistemima linearnih jednačina oblika $Ax=b$, ako je $\det(A) \neq 0$, tada postoji jedinstveno rješenje $x=A^{-1}b$.

U ekonomsko-matematičkim modelima, to znači da je sistem dovoljno "stabilan" i da se mogu izračunati jedinstveni koeficijenti ili veličine, recimo, u modelu ponuda-potražnja ili input-output modelu.

Primjer provjere determinante:

```
import numpy as np

# Definišemo kvadratnu matricu A (3x3)
A = np.array([
    [2, 1, 0],
    [-1, 3, 1],
    [0, 2, 1]
])
# Računanje determinante
detA = np.linalg.det(A)

print("Matrica A:\n", A)
print("Determinanta matrice A:\n", detA)
# Provjeravamo da li je matrica invertibilna
if abs(detA) > 1e-9: # U praksi se često koristi neka mala granica za "nulu"
    print("Matrica A je invertibilna (det != 0).")
else:
    print("Matrica A nije invertibilna (det = 0).")

Matrica A:
[[ 2  1  0]
 [-1  3  1]
 [ 0  2  1]]
Determinanta matrice A:
3.00000000000001
Matrica A je invertibilna (det != 0).
```

U računarima prilikom matematičkih operacija zaokruživanje može uzrokovati da nula izgleda kao vrlo mali broj, pa se pri provjeri $\det(A) \neq 0$ koristi tolerancija (npr. $1e-9$). Ovo sprječava greške kod ekstremno malih ili negativnih vrijednosti, koje ne utiču na invertibilnost matrice.

Sa $\text{abs}(\det A)$ osiguravamo da se matrica proglaši invertibilnom sve dok je njena determinanta, po apsolutnoj vrijednosti, veća od vrlo malog praga i nije praktično nula.

5.1.4.2. Inverzna matrica

Definicija i uslovi za postojanje: Neka je A kvadratna matrica dimenzija $n \times n$. Inverzna matrica A^{-1} (ako postoji) definisana je relacijom:

$$A \cdot A^{-1} = A^{-1} \cdot A = I$$

gdje je I jedinična matrica dimenzija $n \times n$.

Osnovni uslovi da bi matrica A imala inverznu:

- 1) Mora biti kvadratna (istи broj redova i kolona: $n \times n$).
- 2) Njena determinanta mora biti različita od nule ($\det(A) \neq 0$).

Primjena u ekonomskoj input-output analizi:

U Leontijevim input-output modelima često se javlja sistem linearnih jednačina oblika:

$$x = Ax + f$$

gdje x predstavlja vektor ukupne proizvodnje po sektorima, A je matrica tehničkih koeficijenata (koja opisuje međusektorsku potrošnju), a f je vektor finalne potražnje. Preuređivanjem dobijamo:

$$x - Ax = f \Rightarrow (I - A)x = f$$

Da bi riješili x po sektorima, koristi se inverzna matrica:

$$x = (I - A)^{-1}f$$

Ovo predstavlja suštinu input-output analiza u ekonomiji, gdje $(I - A)$ mora biti inverzibilna matrica.

Primjer računanja inverzne matrice upotrebom modula NumPy.

```
import numpy as np
# Definišemo kvadratnu matricu A (npr. 3x3)
```

```

A = np.array([
    [0.7, 0.1, 0.1],
    [0.2, 0.8, 0.0],
    [0.1, 0.1, 0.9]
])

# Računamo njenu inverznu matricu
A_inv = np.linalg.inv(A)

print("Matrica A:\n", A)
print("Inverzna matrica A^-1:\n", A_inv)

# Primjer input-output (I - A)^-1
I = np.eye(3)
I_minus_A = I - A
I_minus_A_inv = np.linalg.inv(I_minus_A)

print("Matrica I - A:\n", I_minus_A)
print("Inverzna matrica (I - A)^-1:\n", I_minus_A_inv)

```

Matrica A:
[[0.7 0.1 0.1]
[0.2 0.8 0.]
[0.1 0.1 0.9]]

Inverzna matrica A^-1:
[[1.5 -0.16666667 -0.16666667]
[-0.375 1.29166667 0.04166667]
[-0.125 -0.125 1.125]]

Matrica I - A:
[[0.3 -0.1 -0.1]
[-0.2 0.2 0.]
[-0.1 -0.1 0.1]]

Inverzna matrica (I - A)^-1:
[[-1.80143985e+16 -1.80143985e+16 -1.80143985e+16]
[-1.80143985e+16 -1.80143985e+16 -1.80143985e+16]
[-3.60287970e+16 -3.60287970e+16 -3.60287970e+16]]

U ovakvim ekonomskim modelima, x dobijamo množenjem $(I-A)^{-1}$ sa vektorom finalne potražnje f.

5.1.4.3. Rang matrice

Definicija: Rang matrice A (dimenzija $m \times n$) predstavlja maksimalan broj linearno nezavisnih redova (ili kolona) te matrice. Ako rang matrice A iznosi r, to znači da je najveći broj međusobno nezavisnih jednačina (ili vektora) jednak r.

Rang se može definisati na više načina, ali suština je:

- Ako je rang jednak broju kolona ($r=n$), tada su sve kolone linearno nezavisne.
- Ako je rang jednak broju redova ($r=m$), tada su svi redovi linearно nezavisni.

Važnost ranga u ekonomsko-matematičkim modelima

Prilikom formulisanja sistema linearnih jednačina (npr. u modeliranju ponude i potražnje, input-output analiza, optimizacija resursa), rang govori o tome koliko nezavisnih jednačina zapravo imamo.

- Ako je rang manji od broja nepoznatih, sistem vjerovatno nema jedinstveno rješenje (previše nepoznatica, a premalo nezavisnih uslova).
- Ako je rang jednak broju nepoznatih (i matrica je kvadratna), tada postoji mogućnost za jedinstveno rješenje (ukoliko je determinanta različita od nule).

U NumPy modulu, rang matrice se računa upotrebom funkcije linalg koja ima sintaksu:

```
numpy.linalg.matrix_rank(A, tol=None, hermitian=False)
```

gdje je:

- **A** (array_like) – Matrica čiji rang se računa.
- **tol** (float, optional) – Opcioni prag ispod kojeg će se singularne vrijednosti smatrati nulama. Ako nije zadat, automatski se određuje na osnovu dimenzija matrice i najveće singularne vrijednosti.
- **hermitian** (bool, optional) – Ako je True, matrica se tretira kao Hermitovska (simetrična ako su svi elementi realni), što može poboljšati preciznost.

Funkcija vraća vrijednost ranga matrice, tj. broj njenih linearно nezavisnih redova ili kolona.

U sljedećem primjeru provjeravamo rang dvije različite matrice.

```
import numpy as np

# Primjer 1: Matrica A koja je punog ranga (3x3)
A = np.array([
    [1, 2, 3],
    [0, 1, 4],
    [2, 0, 1]
])

# Primjer 2: Matrica B koja ima zavisne redove (2x3)
B = np.array([
    [1, 2, 3],
    [2, 4, 6]
])

rank_A = np.linalg.matrix_rank(A)
rank_B = np.linalg.matrix_rank(B)

print("Rang matrice A:", rank_A)
print("Rang matrice B:", rank_B)
```

```
Rang matrice A: 3
Rang matrice B: 1
```

Objašnjenje:

- Matrica A (3x3) može imati rang 3, ako su svi redovi ili kolone linearne nezavisne.
- Matrica B (2x3) ima drugi red koji očigledno ima dvostrukе vrijednosti prvog reda ($2 \times (1,2,3) = (2,4,6)$), pa će rang biti 1, a ne 2.

5.1.4.4. Dijeljenje matrice matricom

U klasičnom smislu, dijeljenje se u aritmetici definiše kao množenje s recipročnom vrijednošću, na primjer:

$$\frac{x}{y} = x \times \frac{1}{y} \quad \forall y \neq 0$$

Međutim, kada je riječ o matricama, ne postoji direktno „dijeljenje“ u stilu aritmetike realnih brojeva. Umjesto toga, ako želimo „podijeliti“ matricu A drugom matricom B, to radimo množenjem s inverznom matricom B^{-1} , pod uslovom da ta inverzna matrica uopšte postoji (dakle, da je B kvadratna i njena determinanta različita od nule).

U simboličkom obliku, „dijeljenje“ A matricom B zapisuje se kao:

$$A \div B \equiv A \times B^{-1}$$

- Preduslov: B mora biti invertibilna ($\det(B) \neq 0$).
- Izvođenje: Umjesto $A \div B$, računamo $A \cdot B^{-1}$.

Neka su A i B matrice dimenzija 2×2 :

$$\mathbf{A} = \begin{pmatrix} 2 & 3 \\ 4 & 1 \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

Ako se želi izračunati „ $A \div B$ “, prvo se računa B^{-1} . U Pythonu se ovo može uraditi na sljedeći način:

```
import numpy as np

A = np.array([[2, 3],
              [4, 1]])
B = np.array([[1, 2],
              [3, 4]])

B_inv = np.linalg.inv(B)      # Inverz matrice B
A_div_B = A.dot(B_inv)        # "Dijeljenje" A matricom B

print("Matrica B^-1:\n", B_inv)
print("Rezultat A / B = A * B^-1:\n", A_div_B)
```

```

Matrica B^-1:
[[ -2.   1. ]
 [ 1.5 -0.5]]
Rezultat A / B = A * B^-1:
[[ 0.5  0.5]
 [-6.5  3.5]]

```

5.1.4.5. Dijeljenje matrice skalarom

Dijeljenje matrice skalarom svodi se na množenje matrice recipročnom vrijednošću tog skalara. Neka je A matrica dimenzija $(m \times n)$, a α skalar različit od nule. Tada je rezultat dijeljenja matrice A skalarom α definisan kao:

$$C = \frac{A}{\alpha} = A \cdot \frac{1}{\alpha}$$

Elementi rezultujuće matrice C dobijaju se tako što se svaki element matrice A podijeli skalarom α :

$$c_{ij} = \frac{a_{ij}}{\alpha}, \quad \text{za sve } i = 1, \dots, m \text{ i } j = 1, \dots, n$$

Ova operacija je korisna kada želimo smanjiti ili normalizovati sve vrijednosti u matrici za isti faktor, što je česta potreba u ekonomiji, statistici i naučnim istraživanjima.

Primjer dijeljenja matrice sa skalarom.

```

import numpy as np
# Definišemo matricu A
A = np.array([
    [2, 4, 6],
    [8, 10, 12]
])
alpha = 2      # Definišemo skalar
C = A / alpha # Dijeljenje matrice A skalarom alpha
print("Originalna matrica A:\n", A)
print(f"Matrica A podijeljena skalarom {alpha}:\n", C)

Originalna matrica A:
[[ 2  4  6]
 [ 8 10 12]]
Matrica A podijeljena skalarom 2:
[[1. 2. 3.]
 [4. 5. 6.]]

```

Ova operacija praktično skalira vrijednosti matrice, proporcionalno smanjujući svaki element. Važno je napomenuti da mora biti $\alpha \neq 0$, pošto dijeljenje nulom nije definisano.

5.2. Sistemi linearnih jednačina

Sistemi linearnih jednačina predstavljaju skupove jednačina u kojima su nepoznate veličine međusobno povezane na linearan način. To znači da svaka nepoznata ima svoj koeficijent (ponder), a veze između nepoznatih predstavljaju odnose, koji se mogu izraziti kao zbir proizvoda koeficijenata i nepoznatih. Ovi sistemi su ključni alati u matematičkom modeliranju, jer omogućavaju analizu i kvantifikaciju odnosa između varijabli u mnogim naučnim disciplinama, uključujući ekonomiju.

Posmatrajmo sljedeći sistem od dvije linearne jednačine i u njemu izdvojene osnovne komponente, koje čine sistem linearnih jednačina.

$$2x + 3y = 8$$

$$x + 4y = 3$$

Matrica koeficijenata: Ovo je pravougaona tabela brojeva, koja sadrži koeficijente ispred nepoznatih u svakoj jednačini. Za posmatrani sistem matrica koeficijenata bi bila:

$$\mathbf{A} = \begin{bmatrix} 2 & 3 \\ 1 & -4 \end{bmatrix}$$

Vektor nepoznatih: Ovo je kolona koja sadrži sve nepoznate veličine sistema. U gornjem primjeru, vektor nepoznatih veličina je:

$$\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}$$

Vektor slobodnih članova: Ovo je kolona koja sadrži konstante na desnoj strani jednačina u sistemu, tj. vrijednosti nezavisne od nepoznatih. U gornjem primjeru, to je:

$$\mathbf{b} = \begin{bmatrix} 8 \\ 3 \end{bmatrix}$$

Sistem linearnih jednačina tada se zapisuje u matričnom obliku:

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$$

$$\begin{bmatrix} 2 & 3 \\ 1 & -4 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 8 \\ 3 \end{bmatrix}$$

Ovaj sistem linearnih jednačina se jednostavno rješava upotrebom modula NumPy i funkcije `np.linalg.solve()`, kao u sljedećem primjeru koda.

```
import numpy as np

# Definisanje matrice koeficijenata A i vektora slobodnih članova b
A = np.array([
    [2, 3],
    [1, -4]
])
b = np.array([8, 3])

# Rješavanje sistema koristeći np.linalg.solve
solution = np.linalg.solve(A, b)

# Ispis rješenja
print("Rješenje sistema:")
print(f"x = {solution[0]:.2f}, y = {solution[1]:.2f}")

Rješenje sistema:
x = 3.73, y = 0.18
```

Funkcija `np.linalg.solve` iz NumPy biblioteke koristi se za rješavanje sistema linearnih jednačina oblika $A \cdot x = b$. Funkcija rješava sistem koristeći *LU dekompoziciju*¹⁸, što je efikasno za kvadratne matrice.

Sintaksa

```
numpy.linalg.solve(a, b)
```

Gdje je:

- **a** – Matrica koeficijenata A koja mora biti kvadratna ($n \times n$) i inverzna.
- **b** – Vektor ili matrica slobodnih članova b. Dimenzija mora biti kompatibilna sa matricom A, tj. broj redova u b mora biti jednak broju redova u matrici A.

Funkcija vraća vektor (ili matricu) rješenja x takav da je $x = A^{-1} \cdot b$

Ukoliko neki od definisanih uslova nije uspinjen funkcija će vratiti jednu od sljedećih grešaka:

- `LinAlgError: Singular matrix` – Ako je A singularna (determinanta jednaka nuli).
- `LinAlgError: Last 2 dimensions of the array must be square` – Ako A nije kvadratna matrica.

¹⁸

LU dekompozicija je metoda za faktorizaciju kvadratne matrice A u proizvod dviju matrica: donje trougaone matrice L i gornje trougaone matrice U , takve da: $A = L \cdot U$.

Iako funkcija `np.linalg.solve` pruža sve što je potrebno za efikasno rješavanje sistema linearnih jednačina, u praksi se često ne zahtijeva dodatno poznavanje metoda poput Cramerovog pravila, Kroneker-Kapelijeve teoreme ili analize homogenih sistema. NumPy automatski obavlja sve potrebne provjere i proračune, kako bi odredio rješenje, omogućavajući brzo i pouzdano rješavanje.

U praksi, `np.linalg.solve` je osnovni alat za brzo rješavanje sistema, ali ćemo u nastavku prikazati kako se teoreme poput Kroneker-Kapelijeve i analiza homogenih sistema mogu koristiti za:

- Dokazivanje uslova rješivosti: Korišćenjem ranga matrice A i proširene matrice $(A \mid b)$.
- Analizu homogenih sistema: Identifikovanje trivijalnih i netrivijalnih rješenja, posebno korisno u ekonomiji kod modeliranja zatvorenih sistema bez eksternih uticaja.

5.2.1. Kroneker-Kapelijeva teorema

Sistem linearnih jednačina je osnovni alat za analizu odnosa između različitih varijabli. Kroneker-Kapelijeva¹⁹ teorema daje jasne uslove za određivanje da li sistem ima rješenje, koliko rješenja ima i da li je rješenje jednoznačno ili beskonačno. Ova teorema je ključna u ekonomiji, gdje se često analiziraju modeli sa višestrukim međuzavisnostima, poput tržišne ravnoteže i input-output analiza.

Pretpostavimo da postoji sistem linearnih jednačina, koji se može zapisati u matričnom obliku:

$$A \cdot x = b$$

gdje su:

- A – matrica koeficijenata dimenzija $m \times n$,
- x – vektor nepoznanica dimenzija $n \times 1$,
- b – vektor slobodnih članova dimenzija $m \times 1$.

Da bismo dokazali da sistem ima rješenje, formiramo proširenu matricu:

$$(A \mid b)$$

koja se sastoji od matrice A i vektora b dodanog kao posljednja kolona, dimenzija $m \times (n+1)$.

¹⁹ U literaturi je ime ove teoreme prisutno u varijantama „Kroneker-Kapeli“, „Kroniker-Kapeli“ ili „Rouché–Capelli“, zavisno od izvora. Suština teoreme ostaje ista – ključni su rang matrice koeficijenata A i rang proširene matrice $(A \mid b)$.

Da bismo odredili da li postoje rješenja sistema linearnih jednačina po teoremi potrebno je prvo da odredimo rang matrice A i rang priširene matrice ($A|b$).

Uslovi postojanja rješenja po Króneker-Kapelijevoj teoremi:

- Ako je $\text{rank}(A) < \text{rank}(A|b)$, sistem je kontradiktoran i nema rješenje.
- Ako je $\text{rank}(A) = \text{rank}(A|b) < n$ (gdje je n broj nepoznatih), sistem ima beskonačno mnogo rješenja. To znači da postoji sloboda u odabiru određenih varijabli, dok druge zavise od njih.
- Ako je $\text{rank}(A) = \text{rank}(A|b) = n$, sistem ima jedinstveno rješenje. Ovo je situacija u kojoj postoji tačno jedan skup vrijednosti za sve nepoznate.

Sljedeći primjer opisuje praktičnu upotrebe Króneker-Kapelijevoj teoreme. Python skripta obrađuje sva tri slučaja definisana teoremom. Da bi smo testirali svaki od slučajeva, potrebno je mijenjati vrijednosti za matrice A i b.

```
import numpy as np

# Definišemo matricu A i vektor b
A = np.array([
    [2, 1],
    [1, 3]
])
b = np.array([8, 13])

# Provjera ranga
rank_A = np.linalg.matrix_rank(A)
rank_A_b = np.linalg.matrix_rank(np.column_stack((A, b)))

print("Rang matrice A:", rank_A)
print("Rang proširene matrice (A|b):", rank_A_b)

# Analiza rješivosti
if rank_A < rank_A_b:
    print("Sistem nema rješenja.")
elif rank_A == rank_A_b and rank_A < A.shape[1]:
    print("Sistem ima beskonačno mnogo rješenja.")
    # Generisanje jedne solucije
    particular_solution = np.linalg.lstsq(A, b, rcond=None)[0]
    print("Jedno moguće rješenje:", particular_solution)
else:
    print("Sistem ima jednoznačno rješenje.")
    x = np.linalg.solve(A[:rank_A, :rank_A], b[:rank_A])
    print("Jedinstveno rješenje:", x)
```

A = np.array([[2, 1], [1, 3]]) b = np.array([8, 13])	Rang matrice A: 2 Rang proširene matrice (A b): 2 Sistem ima jednoznačno rješenje. Jedinstveno rješenje: [2.2 3.6]
--	---

A = np.array([[1, 2, 3], [2, 4, 6]])	Rang matrice A: 1 Rang proširene matrice (A b): 2 Sistem nema rješenja.
--	---

b = np.array([5, 11])	
A = np.array([[1, 2, 3], [2, 4, 6]]) b = np.array([5, 10])	Rang matrice A: 1 Rang proširene matrice (A b): 1 Sistem ima beskonačno mnogo rješenja. Jedno moguće rješenje: [0.35714286 0.71428571 1.07142857]

Objašnjenje:

Funkcija `np.linalg.matrix_rank` je korištena za određivanje ranga matrice A i proširene matrice (A|b), kako bismo analizirali rješivost sistema. Dok je funkcija `np.linalg.solve` korištena za računanje rješenja sistema linearnih jednačina, nakon potvrde da je sistem konzistentan.

Kada postoji beskonačno mnogo rješenja korištena je funkcija `np.linalg.lstsq` za pronalaženje jednog partikularnog rješenja sistema $A \cdot x=b$, koje zadovoljava sve jednačine.

5.2.2. Homogenost sistema

Homogeni sistemi linearnih jednačina predstavljaju poseban slučaj sistema u kojem je vektor slobodnih članova jednak nuli:

$$A \cdot x = 0$$

Onda bi kvadratni sistem linearnih jednačina imao oblik:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{14}x_n = 0, \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{24}x_n = 0, \\ \dots \\ a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n = 0. \end{cases}$$

Ovi sistemi su značajni u matematičkoj analizi jer uvijek imaju barem jedno trivijalno rješenje, gdje su sve nepoznanice jednake nuli, tj. $(x_1, x_2, \dots, x_n) = (0, 0, \dots, 0)$. Prema Kramerovoj teoremi, trivijalno rješenje je jedino moguće rješenje sistema, ako je $\det(A) \neq 0$, što ukazuje da je matrica A regularna. Međutim, ako je $\det(A) = 0$, matrica A je singularna, pa sistem ima beskonačno mnogo netrivijalnih rješenja.

Primjer analize homogenosti sistema na osnovu matrice A.

```
import numpy as np

# Definišemo matricu koeficijenata A - ovdje mijenjamo ulaznu matricu
A = np.array([
    [2, 1],
    [1, 3]
])

# Provjera ranga matrice A
rank_A = np.linalg.matrix_rank(A)
n = A.shape[1] # Broj nepoznanica

# Izračunavanje determinante ako je matrica kvadratna
if A.shape[0] == A.shape[1]:
    det_A = np.linalg.det(A)
    print(f"Determinanta matrice A: {det_A:.2f}")
else:
    det_A = None
    print("Determinanta nije definisana (matrica nije kvadratna).")

print("\nMatrica A:")
print(A)
print("Rang matrice A:", rank_A)

# Analiza sistema
if rank_A == n:
    print("Sistem ima samo trivijalno rješenje (x = 0).")
else:
```

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

```
Determinanta matrice A: 5.00
Matrica A:
[[2 1]
 [1 3]]
Rang matrice A: 2
Sistem ima samo trivijalno rješenje (x = 0).
```

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

```
Determinanta matrice A: 0.00
Matrica A:
[[1 2 3]
 [2 4 6]
 [2 4 6]]
Rang matrice A: 1
Sistem ima beskonačno mnogo netrivijalnih rješenja.
```

Funkcije `np.linalg.det(A)` se koristi za računanjem determinante matrice A, ali samo ako je matrica kvadratna. Ako A nije kvadratna matrica, ispisuje se poruka da determinanta nije definirana.

Logičko odlučivanje:

- Ako je rang matrice A jednak broju kolona ($\text{rank}(A)=n$), sistem ima samo trivijalno rješenje.

- Ako je rang manji od broja kolona ($\text{rank}(A) < n$), sistem ima beskonačno mnogo netrivijalnih rješenja.

5.2.3. Model tržišne ravnoteže

U realnom poslovnom okruženju, često imamo više povezanih proizvoda čije ponude i potražnje međusobno utiču jedna na drugu. Primjer za to može biti domaćinstvo koje koristi električnu energiju i prirodni gas za zagrijavanje. Poskupljenje gasa može djelimično povećati potražnju za električnom energijom (i obrnuto), a istovremeno ponude ovih energetika zavise od različitih troškova proizvodnje, transporta i državnih regulativa.

Studija slučaja: Tržišna ravnoteža dva povezana dobra

Posmatrajmo malo tržište na kojem djeluju sljedeći faktori:

1. **Električna energija (dobre M1):**
 - **Potražnja** se smanjuje s porastom cijene električne energije, ali se malo povećava s rastom cijene gase (jer neki potrošači prelaze na grijanje na električnu energiju ako gas poskupi).
 - **Ponuda** zavisi od troškova proizvodnje u lokalnim elektranama, pa raste sa cijenom električne energije ali ne zavisi direktno od cijene gase (pretpostavimo da električna energija dolazi iz, recimo, hidro i termoelektrana koje ne koriste gas).
2. **Prirodni gas (dobre M2):**
 - **Potražnja** se smanjuje s rastom cijene gase, ali djelimično zavisi i od cijene električne energije (ako električna energija poskupi, neki potrošači se vraćaju na gas).
 - **Ponuda** se povećava s rastom cijene gase, jer je isplativije uvoziti ili proizvoditi više, ali ne zavisi direktno od cijene električne energije.

Želimo odrediti **ravnotežne cijene** (P_1, P_2) i **količine** (Q_1, Q_2) za oba dobra.

Da bi smo odreditili traženu ravnotežu cijene i količine potrebno je da definisemo matematičke modele, koji odgovaraju traženom poslovnom modelu koji smo predstavili. U tom cilju ćemo definisati linearnu funkciju potražnje i ponude, kao i uslove tržišne ravnoteže. Ovaj primjer ćemo obraditi u Pythonu koristeći funkcije za rješavanje sistema linearnih jednačina.

Linearne funkcije potražnje

Neka su **P1** i **P2** cijene električne energije i prirodnog gasea. Potražnja za električnom energijom Q_d_1 i prirodnim gasom Q_d_2 definišemo ovako:

$$\begin{cases} Qd_1 = 50 - 2P1 + 0.5P2, \\ Qd_2 = 30 - 1.5P2 + 0.2P1 \end{cases}$$

Gdje je:

- **50 i 30** – bazna potražnja kad su cijene niske.
- **-2P1** – Ako cijena električne energije (P1) raste, potražnja opada.
- **+0.5P2** – Ako cijena gasa (P2) raste, dio potražnje prelazi na električnu energiju (subtitucija).
- **-1.5P2** – Potražnja za gasom opada s rastom cijene gasa.
- **+0.2P1** – Ako električna energija poskupi, neki korisnici prelaze na gas.

Linearne funkcije ponude

Ponuda Qs_1 (električne energije) i Qs_2 (gasa) mogu izgledati ovako:

$$\begin{cases} Qs_1 = -10 + 2P1, \\ Qs_2 = 5 + 2.5P2. \end{cases}$$

Gdje je:

- **-10 i 5** – mogu biti posljedica fiksnih troškova ili minimalnih potrebnih uslova).
- **2P1** – Ponuda el. energije raste 2 jedinice za svaki rast cijene el. energije za 1.
- **2.5P2** – Ponuda gasa raste 2.5 jedinica za svaki rast cijene gasa za 1.

Uslovi tržišne ravnoteže

Ravnoteža znači da je **ponuda jednaka potražnji** za svako dobro:

$$\begin{cases} Qd_1 = Qs_1, \\ Qd_2 = Qs_2. \end{cases}$$

Zamijenimo Qd_1 i Qs_1 , Qd_2 i Qs_2 :

$$\begin{cases} 50 - 2P1 + 0.5P2 = -10 + 2P1, \\ 30 - 1.5P2 + 0.2P1 = 5 + 2.5P2. \end{cases}$$

Odnosno:

$$\begin{cases} -4P1 + 0.5P2 = -60, \\ 0.2P1 - 1P2 = -25. \end{cases}$$

Ovo je sistem od dvije linearne jednačine s dvije nepoznate (P1, P2).

U nastavku slijedi skripta u Pythonu koja formira ovaj sistem u matričnom obliku i pronalazi (P1, P2).

```
import numpy as np

# Koeficijenti iz (1') i (2'):
# (1') -4P1 + 0.5P2 = -60
# (2') 0.2P1 - 1P2 = -25

A = np.array([
    [-4, 0.5],
    [0.2, -1]
])
b = np.array([-60, -25])

# Rješavanje sistema
sol = np.linalg.solve(A, b)
P1, P2 = sol[0], sol[1]

print("Ravnotežna cijena struje (P1):", P1)
print("Ravnotežna cijena gasa (P2):", P2)

# Računanje i ispis ravnotežnih količina
Q1 = 50 - 2*P1 + 0.5*P2 # ili: -10 + 2*P1
Q2 = 30 - 1.5*P2 + 0.2*P1 # ili: 5 + 2.5*P2

print("Ravnotežna količina struje (Q1):", Q1)
print("Ravnotežna količina gasa (Q2):", Q2)
```

Ravnotežna cijena struje (P1): 18.58974358974359
Ravnotežna cijena gasa (P2): 28.71794871794872
Ravnotežna količina struje (Q1): 27.179487179487175
Ravnotežna količina gasa (Q2): -9.358974358974361

- Ako je P1 (električna energija) iznad izračunate ravnoteže, može doći do viška ponude električne energije ili pad potražnje.
- Ako je P2 ispod ravnotežne, može se desiti nestašica ili pritisak na rast cijena u narednom periodu.

Donosioци odluka (npr. energetske kompanije, vlada, uvoznici gase, proizvođači električne energije) mogu koristiti ove modele za **planiranje** kapaciteta, **procjenu rizika** od kolebanja cijena i **simulaciju** efekata promjena regulativa ili poreza.

Studija slučaja: Tri povezana energenta i različiti troškovi

Slijedi složeniji primjer, gdje su uključujeni i dodatni parametri. Ovaj primjer će i dalje biti linearan (radi jednostavnosti rješavanja), ali će imati više ulaznih varijabli, čime ćemo simulirati realniju situaciju.

Analiza malog tržišta na kome djeluju tri osnovna energenta:

1. Električna energija (E)
2. Prirodni gas (G)
3. Pelet (P)

Poslovni model

1. **Potražnja** svakog energenta zavisi od:
 - Cijene samog energenta (P_E , P_G , P_P).
 - Cijena ostalih energenata (npr. ako gas poskupi, potražnja za peletom i električnom energijom raste).
 - Dodatni vanjski faktor (ekološke subvencije, kvalitet grijanja, navike korisnika), koje ovdje možemo pojednostaviti u linearne konstante.
2. **Ponuda** svakog energenta zavisi od:
 - Troškova proizvodnje ili nabavke.
 - Cijene tog energenta (marža, profitni motiv).
 - Tehničkih ograničenja (npr. maksimalni kapacitet proizvodnje električne energije, ograničenja uvoza gasa i sl.).

U nastavku je kreiran konkretan sistem koji obuhvata tri ponude i tri potražnje (dakle šest linearnih jednačina). Ovaj zadatak se rješava kao jedan sistem s nepoznatim cijenama (P_1 , P_2 , P_3) i količinama (Q_1 , Q_2 , Q_3). Da bi zadržali da sistem bude linearan, razdvaja se ponuda i potražnja tako da se izraze u terminima cijena, a zatim se kaže da je $Qd_i = Qs_i$ za svaki energet i .

Funkcija potražnja za energentima

Neka (P_1 , P_2 , P_3) su **cijene**:

- P_1 – cijena električne energije (E)
- P_2 – cijena gasa (G)
- P_3 – cijena peleta (P)

Potražnja (Q_d):

$$\begin{cases} Qd_1 = 60 - 2P_1 + 0.4P_2 + 0.3P_3, & \text{(električna energija)} \\ Qd_2 = 40 + 0.3P_1 - 2P_2 + 0.2P_3, & \text{(gas)} \\ Qd_3 = 30 + 0.3P_1 + 0.2P_2 - 1.5P_3. & \text{(pelet)} \end{cases}$$

Gdje je:

- **Qd1** – Bazna potražnja je 60 (ako su sve cijene niske), a za svaki porasta cijene električne energije od 1 EUR/MWh porasta cijene električne energije,

tražnja opada za 2 jedinice. Ako gas i pelet poskupe, dio potražnje se „prelije“ na električnu energiju (koeficijenti +0.4 i +0.3).

- **Qd2** – Bazna potražnja za gas je 40, ali raste ako električna energija poskupi (+0.3P1) i opada s vlastitom cijenom (-2P2), dok poskupljenje peleta (+0.2P3) malo utiče na tražnju gasa.
- **Qd3** – Za pelet počinjemo s baznih 30 jedinica, raste ako poskupi električna energija ili gas (+0.3P1, +0.2P2), a opada jače s rastom vlastite cijene (-1.5P3).

Funkcija ponuda za energente

Ponuda (Qs):

$$\begin{cases} Qs_1 = -10 + 3P1, & \text{(struja, E)} \\ Qs_2 = 5 + 2.5P2, & \text{(gas, G)} \\ Qs_3 = 2 + 2P3. & \text{(pelet, P)} \end{cases}$$

Gdje je:

- **Qs1** – Bazni trošak -10 (npr. minimalni troškovi rada elektrana). Za svaki rast od 1 EUR/MWh cijene električne energije, dodatno se isplati proizvesti 3 jedinice električne energije.
- **Qs2** – Bazna ponuda gasa 5 (domaća proizvodnja ili minimalni uvoz). Rast cijene gasa za 1 EUR/MWh poveća ponudu za 2.5 jedinica.
- **Qs3** – Bazna ponuda peleta je 2, a raste za 2 jedinice kad se cijena peleta poveća za 1 EUR/MWh (npr. više proizvođača ulazi na tržiste).

Uslovi tržišne ravnoteže

Da bi tržiste bilo u ravnoteži, **ponuda = potražnja** za svaki energet:

$$\begin{cases} Qd_1 = Qs_1, \\ Qd_2 = Qs_2, \\ Qd_3 = Qs_3. \end{cases}$$

ovo znači:

$$\begin{cases} 60 - 2P1 + 0.4P2 + 0.3P3 = -10 + 3P1, \\ 40 + 0.3P1 - 2P2 + 0.2P3 = 5 + 2.5P2, \\ 30 + 0.3P1 + 0.2P2 - 1.5P3 = 2 + 2P3. \end{cases}$$

Odnosno:

$$\begin{cases} 5P_1 - 0.4P_2 - 0.3P_3 = 70, \\ 0.3P_1 - 4.5P_2 + 0.2P_3 = -35, \\ 0.3P_1 + 0.2P_2 - 3.5P_3 = -28. \end{cases}$$

Ovo je sistem od tri linearne jednačine sa tri nepoznate (P_1, P_2, P_3).

Nakon što izračunamo (P_1, P_2, P_3), mogu se dobiti ravnotežne količine (Q_1, Q_2, Q_3) unošenjem tih cijena nazad u potražnju ili ponudu.

Ispod slijedi skripta koja formira i rješava opisani sistem, te na kraju ispisuje i ravnotežne količine.

```
import numpy as np

# 1) Formiranje matrice koeficijenata i vektora slobodnih članova
A = np.array([
    [ 5,     -0.4,   -0.3],
    [ 0.3,   -4.5,    0.2 ],
    [ 0.3,     0.2,   -3.5]
])
b = np.array([70, -35, -28])

# 2) Rješavanje linearnih jednačina
sol = np.linalg.solve(A, b)
P1, P2, P3 = sol
print(f"Ravnotežna cijena struje (P1) = {P1:.2f}")
print(f"Ravnotežna cijena gasa (P2)   = {P2:.2f}")
print(f"Ravnotežna cijena peleta (P3) = {P3:.2f}")

# 3) Izračunavanje ravnotežnih količina
Qd1 = 60 - 2*P1 + 0.4*P2 + 0.3*P3
Qd2 = 40 + 0.3*P1 - 2*P2 + 0.2*P3
Qd3 = 30 + 0.3*P1 + 0.2*P2 - 1.5*P3

print(f"Ravnotežna količina struje (Q1)  = {Qd1:.2f}")
print(f"Ravnotežna količina gasa (Q2)   = {Qd2:.2f}")
print(f"Ravnotežna količina peleta (Q3) = {Qd3:.2f}")
```

```
Ravnotežna cijena struje (P1) = 15.33
Ravnotežna cijena gasa (P2)   = 9.24
Ravnotežna cijena peleta (P3) = 9.84
Ravnotežna količina struje (Q1)  = 35.99
Ravnotežna količina gasa (Q2)   = 28.09
Ravnotežna količina peleta (Q3) = 21.68
```

5.2.4. Input-Output Analiza

Input-output analiza se koristi za proučavanje mnogih oblasti u ekonomiji. U nastavku će se obraditi primjer međusektorskih odnosa u privredi. Izvršit će se analizira tri sektora: industrija, poljoprivreda i usluge u jednoj ekonomiji. Svaki sektor koristi resurse iz drugih sektora za proizvodnju svojih finalnih proizvoda. Cilj je odrediti

obim proizvodnje svakog sektora, kako bi se zadovoljila tražnja svih sektora unutar ekonomije.

Neka x_1, x_2, x_3 predstavljaju ukupnu proizvodnju sektora industrije, poljoprivrede i usluga, redom. Svaki sektor koristi dio svoje proizvodnje za međusobno snabdijevanje, dok ostatak ide za finalnu potrošnju (f_1, f_2, f_3).

Imamo sljedeće podatke:

- **Industrija** koristi 40% svoje proizvodnje za industriju, 10% za poljoprivredu i 20% za usluge.
- **Poljoprivreda** koristi 20% svoje proizvodnje za industriju, 30% za poljoprivredu i 10% za usluge.
- **Usluge** koriste 10% svoje proizvodnje za industriju, 20% za poljoprivredu i 40% za usluge.

Dodatno:

- Finalna potrošnja je $f_1=50$ (industrija), $f_2=30$ (poljoprivreda), i $f_3=40$ (usluge).

Matematički model

Sistem input-output analize može se zapisati kao:

$$\mathbf{x} = \mathbf{A} \cdot \mathbf{x} + \mathbf{f},$$

gdje je:

- ukupna proizvodnja sektora,

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

- matrica tehničkih koeficijenata (procenti međusobne potrošnje),

$$\mathbf{A} = \begin{bmatrix} 0.4 & 0.1 & 0.2 \\ 0.2 & 0.3 & 0.1 \\ 0.1 & 0.2 & 0.4 \end{bmatrix}$$

- finalna potrošnja.

$$\mathbf{f} = \begin{bmatrix} 50 \\ 30 \\ 40 \end{bmatrix}$$

Preuređivanjem:

$$\mathbf{x} - \mathbf{A} \cdot \mathbf{x} = \mathbf{f},$$

ovo možemo zapisati kao:

$$(\mathbf{I} - \mathbf{A}) \cdot \mathbf{x} = \mathbf{f},$$

gdje je \mathbf{I} jedinična matrica.

Korišćenje funkcije `np.linalg.solve` možemo direktno riješiti dobijeni sistem linearnih jednačina bez potrebe da eksplicitno izračunavamo inverznu matricu $(\mathbf{I} - \mathbf{A})^{-1}$. Ovo je efikasnije i numerički tačniji način, jer se izbjegavaju potencijalne greške zaokruživanja, koje mogu nastati prilikom računanja inverzne matrice.

```
import numpy as np

# Definicija matrice tehničkih koeficijenata A
A = np.array([
    [0.4, 0.1, 0.2],
    [0.2, 0.3, 0.1],
    [0.1, 0.2, 0.4]
])

I = np.eye(3)          # Jedinična matrica I
f = np.array([50, 30, 40]) # Vektor finalne potrošnje f
I_minus_A = I - A      # Matrica (I - A)

# Rješavanje sistema (I - A) * x = f
x = np.linalg.solve(I_minus_A, f)

# Ispis rezultata
print("Ukupna proizvodnja sektora:")
print(f"Industrija: {x[0]:.2f}")
print(f"Poljoprivreda: {x[1]:.2f}")
print(f"Usluge: {x[2]:.2f}")
```

Ukupna proizvodnja sektora:
Industrija: 141.46
Poljoprivreda: 100.98
Usluge: 123.90

5.3. Funkcije

Funkcije su jedan od osnovnih matematičkih alata za modeliranje realnih pojava, a posebno su korisne u ekonomiji pri opisivanju ponude, potražnje, troškova, prihoda i drugih pokazatelja. U nastavku ćemo se fokusirati na najčešće tipove funkcija, koji se primjenjuju u ekonomskim analizama: linearne, kvadratne, eksponencijalne i logaritamske funkcije.

5.3.1. Linearne funkcije

Linearne funkcije su osnovni alati u ekonomskim analizama zbog svoje jednostavnosti i široke primjene. Njihova opšta forma je:

$$f(x) = a + bx$$

gdje je:

- a : konstanta, definiše presijecanje y-ose (vrijednost funkcije kada je $x=0$),
- b : koeficijent nagiba, koji pokazuje kako se $f(x)$ mijenja kada x poraste za 1 jedinicu.

Linearne funkcije koriste se za modeliranje osnovnih odnosa u ekonomiji. Najčešći primjeri su:

Funkcija potražnje:

$$D = a - bP$$

gdje je:

- D – potražnja (količina proizvoda koju potrošači žele kupiti),
- P – cijena proizvoda,
- a – maksimalna potražnja kada je cijena $P=0$,
- $-b$ – koeficijent koji pokazuje koliko potražnja opada kada cijena poraste za 1 jedinicu.

Funkcija ponude:

$$S=c+dP$$

gdje je:

- S – ponuda (količina proizvoda koju proizvođači žele ponuditi),
- P – cijena proizvoda,
- c – početna ponuda kada je cijena $P=0$,

- d – koeficijent koji pokazuje koliko ponuda raste kada cijena poraste za 1 jedinicu.

Tumačenje parametara:

- **Presretanje (a, c)** – Predstavlja početnu vrijednost funkcije, tj. potražnju ili ponudu kada je cijena $P=0$.
- **Nagib ($-b, d$)** – Određuje brzinu promjene funkcije. Veći apsolutni nagib označava veću osjetljivost potražnje/ponude na promjenu cijene.

Sljedeći primjer pokazuje modeliranje ponude i potražnje. Neka je:

- Funkcija potražnje: $D=100-2P$,
- Funkcija ponude: $S=-20+3P$.

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

# Definisanje cijena (P)
P = np.linspace(-20, 100) # Vrijednosti cijena od -10 do 100

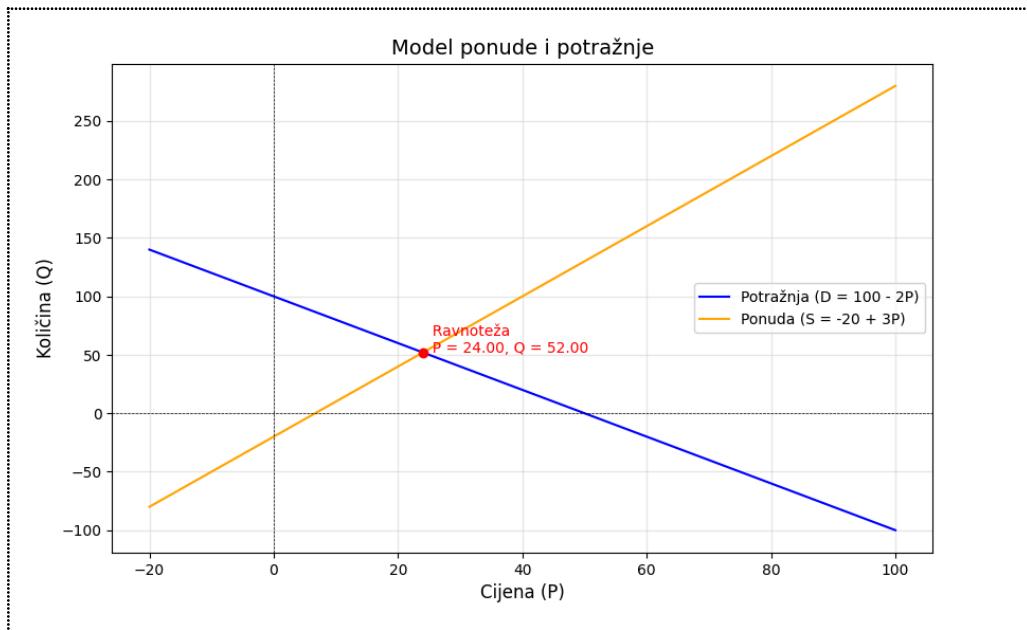
# Funkcija potražnje i ponude
D = 100 - 2 * P # Funkcija potražnje
S = -20 + 3 * P # Funkcija ponude

# Pronalazak tačke ravnoteže (gdje su D = S)
ravnoteza_cijene = (100 + 20) / (2 + 3)
ravnoteza_kolicine = 100 - 2 * ravnoteza_cijene

# Grafički prikaz
plt.figure(figsize=(10, 6))
plt.plot(P, D, label='Potražnja (D = 100 - 2P)', color='blue')
plt.plot(P, S, label='Ponuda (S = -20 + 3P)', color='orange')

# Oznaka ravnoteže
plt.scatter(ravnoteza_cijene, ravnoteza_kolicine, color='red', zorder=5)
plt.text(ravnoteza_cijene, ravnoteza_kolicine,
         f' Ravnoteža\n P = {ravnoteza_cijene:.2f}, Q = {ravnoteza_kolicine:.2f}',
         fontsize=10, color='red')

# Naslovi i osi
plt.title('Model ponude i potražnje', fontsize=14)
plt.xlabel('Cijena (P)', fontsize=12)
plt.ylabel('Količina (Q)', fontsize=12)
plt.axhline(0, color='black', linewidth=0.5, linestyle='--')
plt.axvline(0, color='black', linewidth=0.5, linestyle='--')
plt.grid(alpha=0.3)
plt.legend()
plt.show()
```



Kod generiše grafik koji prikazuje funkcije potražnje i ponude, kao i tačku u kojoj se one sijeku (ravnotežu):

- Ravnotežna cijena: P=24.
- Ravnotežna količina: Q=52.

Funkcija `P=np.linspace(-20, 100)`, predstavlja x-osu u grafiku, a definisani parametri određuju granične vrijednosti (-20...100).

Funkcije `plt.scatter()` definišu crveni marker, koji označava tačku ravnoteže na grafiku, dok `plt.text()` dodaje tekst s vrijednostima ravnoteže (P, Q).

Dodavanje virtualne x-y ose unutar grafika koja je pozicionirana na x=0 i y=0, pomaže u prikazu kretanja funkcija prije i poslije nulte tačke. Ovo se postiže upotrebom funkcija `plt.axhline()` i `plt.axvline()`.

5.3.2. Kvadratna funkcija

Kvadratna funkcija ima opšti oblik:

$$f(x) = ax^2 + bx + c$$

gdje je:

- a – koeficijent kvadratnog člana (ax^2), određuje zakrivljenost grafika,
- b – koeficijent linearne funkcije (bx), utiče na nagib,
- c – konstanta koja određuje presjecanje na y-osi.

Grafik ove funkcije je **parabola**:

- Ako je $a > 0$, parabola je otvorena prema gore.
- Ako je $a < 0$, parabola je otvorena prema dole.

U ekonomiji, kvadratna funkcija se često koristi za modeliranje **funkcije troškova**, jer omogućava prikazivanje promjena ukupnih i marginalnih troškova s obimom proizvodnje (Q).

Opšti oblik kvadratne funkcije troškova:

$$C(Q) = aQ^2 + bQ + c$$

gdje je:

- $C(Q)$ – ukupni troškovi,
- a – koeficijent koji opisuje rastuće granične troškove (MC),
- b – koeficijent linearnih troškova,
- c – fiksni troškovi koji ne zavise od obima proizvodnje (Q).

Ako je $a > 0$ funkcija opisuje ekonomsku neefikasnost na višim nivoima proizvodnje.

Pored osnovne funkcije troškova, u ekonomskim analizama zanimljivi su i derivati ove funkcije a to su:

Granični troškovi (MC): granični troškovi rastu linearno s povećanjem proizvodnje.

$$MC = \frac{dC(Q)}{dQ} = 2aQ + b$$

Prosječni troškovi (AC): opisuju troškove po jedinici proizvodnje.

$$AC = \frac{C(Q)}{Q}$$

Primjer kvadratne funkcije troškova $C(Q)$ prikazane na desnoj strani sa skaliranjem y-ose i graničnih troškova (MC) i prosječnih troškova (AC) prikazanih na lijevoj strani y-ose.

Pretpostavimo da je funkcija troškova izražena matematičkom formulom: $C(Q)=2xQ^2+3xQ+1000$, gdje su 1000 fiksni troškovi.

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

# Parametri kvadratne funkcije troškova
a = 2      # Koeficijent kvadratnog člana
b = 3      # Koeficijent linearног člana
c = 1000   # Fiksni troškovi

Q = np.linspace(1, 100, 100)      # Količina proizvodnje (Q)
C_Q = a * Q**2 + b * Q + c      # Funkcija ukupnih troškova (C(Q))
MC = 2 * a * Q + b              # Granični troškovi (MC)
AC = C_Q / Q                   # Prosječni troškovi (AC)

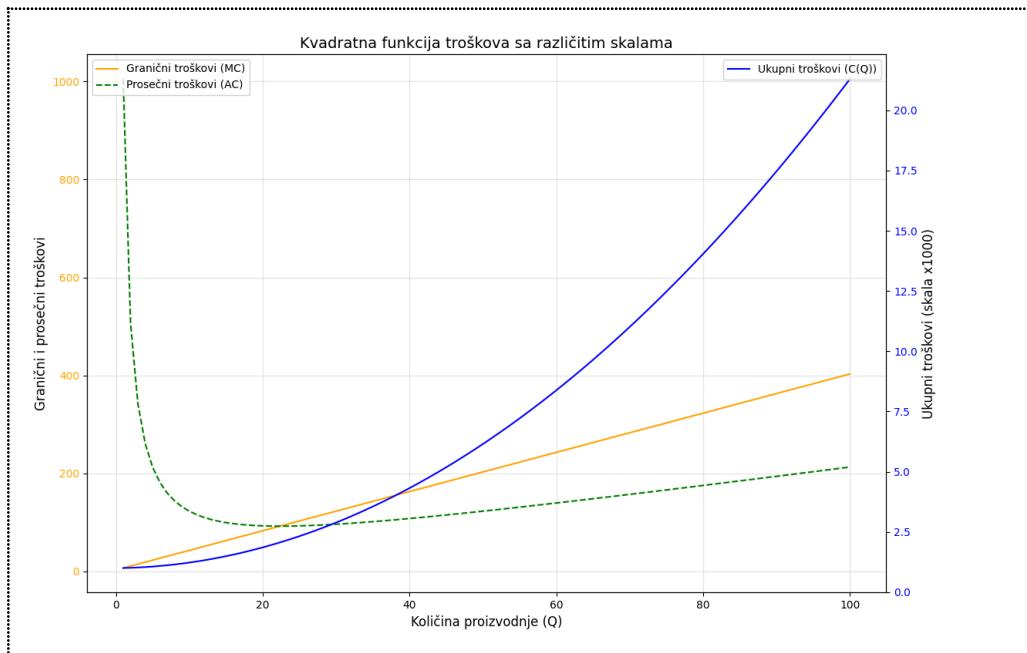
# Grafički prikaz
fig, ax1 = plt.subplots(figsize=(12, 8))

# Levi y-osa: Granični i prosečni troškovi
ax1.plot(Q, MC, label='Granični troškovi (MC)', color='orange')
ax1.plot(Q, AC, label='Prosečni troškovi (AC)', color='green', linestyle='--')
ax1.set_xlabel('Količina proizvodnje (Q)', fontsize=12)
ax1.set_ylabel('Granični i prosečni troškovi', fontsize=12)
ax1.tick_params(axis='y', labelcolor='orange')
ax1.legend(loc='upper left')
ax1.grid(alpha=0.3)

# Desni y-osa: Ukupni troškovi
ax2 = ax1.twinx()  # Deljena x-osa
ax2.plot(Q, C_Q / 1000, label='Ukupni troškovi (C(Q))', color='blue')
ax2.set_ylabel('Ukupni troškovi (skala x1000)', fontsize=12)
ax2.tick_params(axis='y', labelcolor='blue')
ax2.legend(loc='upper right')

# Naslov
plt.title('Kvadratna funkcija troškova sa različitim skalama', fontsize=14)

# Prikaz grafika
plt.tight_layout()
plt.show()
```



Objašnjenje korištenih funkcija za generisanje dvije y-ose:

- Razdvajanje osa postignuto je korišćenjem funkcije `twinx()`, koja omogućava dodavanje sekundarne y-ose na istom grafiku. Primarna osa (`ax1`) koristi se za prikaz graničnih (`MC`) i prosječnih troškova (`AC`), dok sekundarna osa (`ax2`) prikazuje ukupne troškove (`C(Q)`).
- Funkcija skaliranja ostvarena je dijeljenjem ukupnih troškova sa 1000 unutar izraza za njihovu vrijednost ($C(Q)/1000$). Ovom metodom obezbeđuje se da funkcije različitih opsega budu čitljive na istom grafiku.
- Oznake i legende za primarnu i sekundarnu osu prilagođene su pomoću funkcija `set_ylabel()` i `tick_params()`, čime se obezbeđuje vizuelna razlika između njih. Ovaj pristup omogućava pregled svih funkcija u kontekstu različitih dimenzija troškova na jednom grafiku.

5.3.3. Eksponencijalne funkcije

Eksponencijalne funkcije igraju ključnu ulogu u ekonomiji kada se analiziraju procesi, koji uključuju kontinuirani rast ili pad, poput rasta investicija, kapitalizacije kamate ili modela populacijskog rasta. Eksponencijalna funkcija ima oblik:

$$f(x) = A \cdot e^{kx}$$

gdje su:

- A – početna vrijednost funkcije (količina, investicija, kapital, itd.),
- k – stopa rasta (ili pada),
- e – Eulerov broj (≈ 2.718).
- Ako je $k > 0$: Funkcija eksponencijalno raste.
- Ako je $k < 0$: Funkcija eksponencijalno opada.

Eksponencijalne funkcije su široko primjenjive u ekonomiji za modeliranje rasta, troškova i dinamike u finansijskim i proizvodnim sistemima. U nastavku je pokazan model rasta investicije, kao oblike eksponencijalnih funkcija.

Model rasta investicija

Vrijednost investicije raste eksponencijalno tokom vremena kada se kamate kontinuirano kapitaliziraju. Ovo je korisno za:

- Dugoročno planiranje investicija.
- Predviđanje rasta kapitala u zavisnosti od kamatne stope.

Sljedeća matematička formula prestavlja model:

$$V(t) = V_0 \cdot e^{rt}$$

gdje je:

- V_0 – početna vrijednost investicije,
- r – kontinuirana kamatna stopa,
- t – vrijeme.

Sljedeći primer prikazuje rast investicija tokom vremena prema eksponencijalnom modelu.

Početna investicija iznosi 10.000, a kontinuirana kamatna stopa je 5% godišnje. Potrebno je pokazati rast vrijednosti investicije tokom 10 godina.

```

import numpy as np
import matplotlib.pyplot as plt

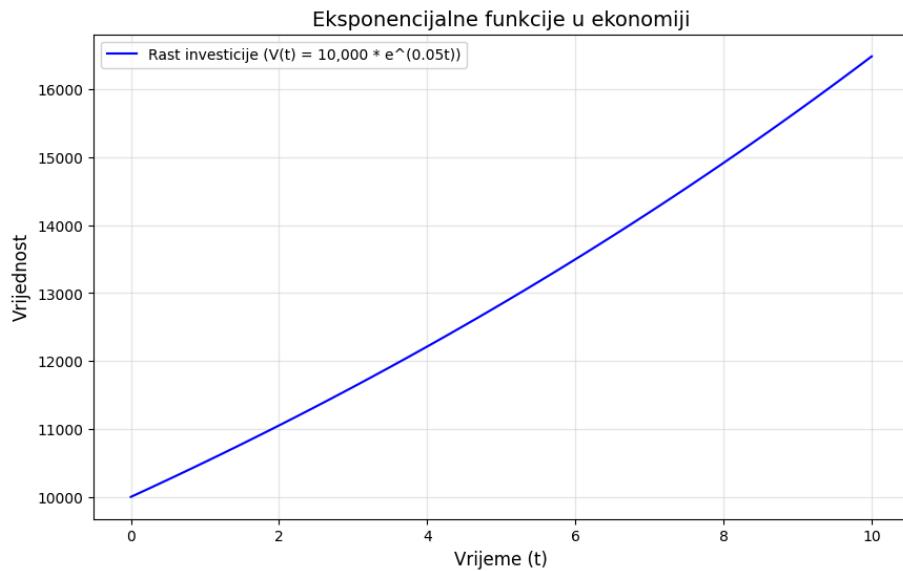
# Parametri za različite funkcije
t = np.linspace(0, 10) # Vrijeme (godine)

# Model rasta investicija
V0 = 10000 # Početna investicija
r = 0.05 # Kontinuirana kamatna stopa
V_t = V0 * np.exp(r * t)

# Grafički prikaz
plt.figure(figsize=(10, 6))
plt.plot(t, V_t, label='Rast investicije (V(t) = 10,000 * e^(0.05t))', color='blue')

# Dodavanje naslova i oznaka
plt.title("Eksponencijalne funkcije u ekonomiji", fontsize=14)
plt.xlabel("Vrijeme (t)", fontsize=12)
plt.ylabel("Vrijednost", fontsize=12)
plt.legend()
plt.grid(alpha=0.3)
plt.show()

```



5.3.4. Logaritamske funkcije

Logaritamske funkcije imaju značajnu primjenu u ekonomiji zbog svoje sposobnosti da linearizuju proporcionalne promjene i olakšaju analizu složenih odnosa između ekonomskih promjenljivih. Korišćenje logaritama omogućava detaljnu analizu promjena cijena i količina, pružajući jednostavniji uvid u elastičnost potražnje i ponude.

Logaritamska transformacija je posebno korisna za analizu trendova u podacima sa velikim varijacijama. Smanjuje uticaj ekstremnih vrijednosti i omogućava bolju interpretaciju dugoročnih obrazaca, čineći je ključnim alatom u ekonomskim istraživanjima.

U teoriji potrošača, logaritamske funkcije se često koriste za opisivanje korisnosti, pri čemu zavisnost korisnosti od količine dobara prati obrazac opadajućeg graničnog užitka — svaka dodatna jedinica dobra donosi sve manje zadovoljstva.

Logaritamska funkcija ima opšti oblik:

$$f(x) = a + b \ln(x)$$

gdje je:

- **a** – konstanta, određuje početnu vrijednost funkcije,
- **b** – koeficijent logaritamskog člana, opisuje stepen promjene funkcije u odnosu na ulaz,
- **ln(x)** – prirodni logaritam varijable x, definisan samo za $x > 0$.

U nastavku je pokazana primjena logaritamskih funkcija u prikazu mjere elastičnosti.

Koncept elastičnosti

Elastičnost je ekomska mjera, koja opisuje osjetljivost jedne promjenljive na promjene u drugoj promjenljivoj. Kada se govori o potražnji i cijeni, **elastičnost potražnje** pokazuje kako se tražena količina (Q) mijenja u odnosu na promjenu cijene (P).

Opšta formula za cjenovnu elastičnost potražnje je:

$$\varepsilon = \frac{\Delta Q/Q}{\Delta P/P}$$

gdje je:

- ε – koeficijent elastičnosti potražnje,
- ΔQ – promjena tražene količine,

- Q – početna tražena količina,
- ΔP – promjena cijene,
- P – početna cijena.

Opšta formula ima nekoliko praktičnih nedostataka:

- **Asimetrija:** Rezultati zavise od početne tačke (Q_1, P_1) i krajnje tačke (Q_2, P_2), što može dovesti do različitih vrijednosti elastičnosti za isti par promjena.
- **Ograničena primjena:** Kada se radi sa velikim skupovima podataka ili kontinuiranim promjenama, osnovna formula postaje nepraktična za precizne analize.

Zbog ovih ograničenja, koristi se pristup baziran na logaritamskim funkcijama, koji eliminiše problem asimetrije i omogućava rad sa kontinuiranim promjenama.

Nakon logaritamskih transformacija početne opšte formule u linearni model u log-log²⁰ prostoru možemo pretpostaviti da je odnos između Q i P opisan funkcijom:

$$\ln(Q) = a + b \ln(P)$$

tada je elastičnost potražnje (ε) direktno jednaka koeficijentu b :

$$\varepsilon = b$$

U nastavku je Python skripta, koja računa elastičnost i crta logaritamski grafik modela potražnje.

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

# Podaci o cijenama (P) i količinama (Q)
cijene = np.array([10, 12, 15, 18, 20, 25]) # Cijene
kolicine = np.array([100, 90, 75, 60, 50, 40]) # Tražene količine

# Logaritamska transformacija
log_cijene = np.log(cijene)
log_kolicine = np.log(kolicine)

# Izračunavanje koeficijenta elastičnosti
b, a = np.polyfit(log_cijene, log_kolicine, 1) # Prvi element (b) je nagib
# (elastičnost)

# Ispis rezultata
print(f"Logaritamska funkcija: ln(Q) = {a:.2f} + {b:.2f} ln(P)")
print(f"Koeficijent elastičnosti potražnje (b): {b:.2f}")

# Vizualizacija
```

²⁰ Log-log prostor je koordinatni sistem u kojem su i x-osi i y-osi prikazane u logaritamskoj skali. Drugim riječima, umjesto stvarnih vrijednosti promenljivih xxx i yyy , koristi se prirodni logaritam $\ln(x)$ i $\ln(y)$ za skaliranje osa.

```

plt.figure(figsize=(10, 6))

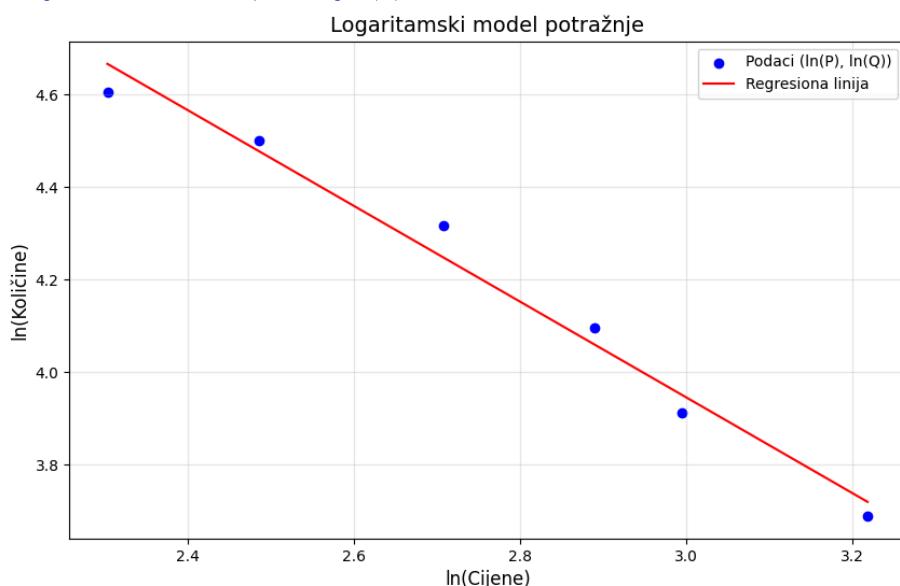
# Scatter plot podataka
plt.scatter(log_cijene, log_kolicine, color="blue", label="Podaci ( $\ln(P)$ ,  $\ln(Q)$ )")

# Regresiona linija
predvidjene_kolicine = a + b * log_cijene
plt.plot(log_cijene, predvidjene_kolicine, color="red", label="Regresiona linija")

# Dodavanje naslova i oznaka
plt.title("Logaritamski model potražnje", fontsize=14)
plt.xlabel("ln(Cijene)", fontsize=12)
plt.ylabel("ln(Količine)", fontsize=12)
plt.legend()
plt.grid(alpha=0.3)
plt.show()

```

Logaritamska funkcija: $\ln(Q) = 7.05 + -1.03 \ln(P)$
 Koeficijent elastičnosti potražnje (b): -1.03



Tumačenje rezultata:

- Elastičnost potražnje (b) je -1.03, što ukazuje na elastičnu potražnju, odnosno svaki rast cijene za 1% smanjuje potražnju za približno 1.03%.

Objašnjenje koda:

- 1) **Logaritamska transformacija** – Podaci o cijenama i količinama se transformišu prirodnim logaritmima pomoću funkcije `np.log`.
- 2) **Linearna regresija sa numpy** – Funkcija `np.polyfit` koristi se za linearnu regresiju, gdje se računa koeficijent nagiba (b) i presretanje (a).

- 3) **Koefficijent elastičnosti** – Direktno dobijamo vrijednost b iz linearne modela u log-log prostoru.
- 4) **Grafički prikaz** – Prikazuju se tačke podataka i odgovarajuća regresiona linija u log-log koordinatnom sistemu.

5.3.5. Primjer optimizacije profita

Optimizacija profita je ključna komponenta ekonomskog odlučivanja u preduzećima. Cilj je maksimizirati razliku između ukupnog prihoda (TR) i ukupnih troškova (TC):

$$\text{Profit}(\Pi) = TR - TC$$

Gdje je:

- Ukupni prihod (TR) – Definiše se kao proizvod prodajne cijene (P) i količine (Q): $TR = P \cdot Q$.
- Ukupni trošak (TC) – Obuhvata fiksne i varijabilne troškove: $TC = FC + VC(Q)$, gdje FC označava fiksne troškove, a $VC(Q)$ varijabilne troškove, koji zavise od količine.

Optimizacija profita zahtijeva nalaženje količine (Q^*) pri kojoj je razlika između TR i TC najveća. Ovo se postiže izjednačavanjem graničnog prihoda ($MR = \partial TR / \partial Q$) i graničnog troška ($MC = \partial TC / \partial Q$).

Posmatrajmo primjer proizvođača koji prodaje proizvod po ceni $P=130-0.5Q$, gdje je Q količina proizvoda. Troškovna funkcija je $TC=50Q+1000$, gdje su FC=1000 (fiksni troškovi), a $VC=50Q$ (varijabilni troškovi linearno zavise od Q). Koristeći prethodno formula za računanje ukupnih prihoda i troškova, dolazi se do početnih jednačina za optimizaciju profita.

- Ukupni prihod: $TR = P \cdot Q = (130 - 0.025Q) \cdot Q$.
- Ukupni trošak: $TC = 50Q + 1000$.
- Optimizacija: Maksimizacija profita nalaženjem količine Q^* koja maksimizira $\Pi(Q)$.

U nastavku je Python skripta, koja na osnovu ulanih podataka i definisanih funkcija računa optimalni profit u funkciji količine proizvoda.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize_scalar

# Parametri
fiksni_troskovi = 1000
varijabilni_trosak_po_jedinici = 50

# Funkcije
```

```

def ukupni_prihod(Q):
    P = 130 - 0.25 * Q # cijena kao funkcija količine
    return P * Q

def ukupni_trosak(Q):
    return fiksni_troskovi + varijabilni_trosak_po_jedinici * Q

def profit(Q):
    return ukupni_prihod(Q) - ukupni_trosak(Q)

# Optimizacija
rezultat = minimize_scalar(lambda Q: -profit(Q), bounds=(0, 300), method='bounded')
optimalna_kolicina = rezultat.x
maksimalni_profit = -rezultat.fun

# Prikaz rezultata
Q = np.linspace(0, 300)
TR = ukupni_prihod(Q)
TC = ukupni_trosak(Q)
P = profit(Q)

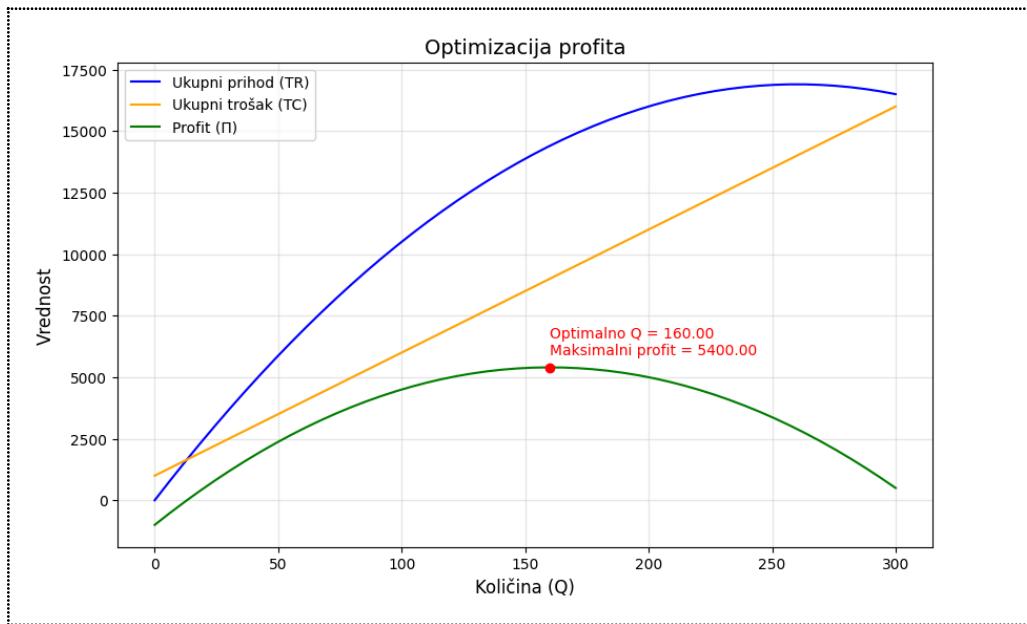
plt.figure(figsize=(10, 6))

# Grafovi
plt.plot(Q, TR, label="Ukupni prihod (TR)", color='blue')
plt.plot(Q, TC, label="Ukupni trošak (TC)", color='orange')
plt.plot(Q, P, label="Profit (\u03a0)", color='green')

# Optimalna tačka
plt.scatter(optimalna_kolicina, maksimalni_profit, color='red', zorder=5)
plt.text(optimalna_kolicina, maksimalni_profit + 500,
        f"Optimalno Q = {optimalna_kolicina:.2f}\nMaksimalni profit = {maksimalni_profit:.2f}",
        fontsize=10, color='red')

# Dodavanje naslova i oznaka
plt.title("Optimizacija profita", fontsize=14)
plt.xlabel("Količina (Q)", fontsize=12)
plt.ylabel("Vrednost", fontsize=12)
plt.legend()
plt.grid(alpha=0.3)
plt.show()

```



Objašnjenje:

- Funkcija `ukupni_prihod(Q)` računa ukupni prihod $TR=130-0.025Q$
- Funkcija `ukupni_trosak(Q)` Računa ukupne troškove $TC=50Q+1000$.
- Funkcija `profit(Q)` računa profit kao razliku između ukupnog prihoda i ukupnih troškova.
- Funkcija `minimize_scalar` minimizuje vrijednost date funkcije. Pošto se želi maksimizacija profita, optimizuje se negativna vrijednost profita ($-\text{profit}(Q)$). Definišu se granice za Q kao $(0, 300)$, što znači da se proizvodnja ograničava na ovaj opseg.
- Rezultati optimizacije:
 - `optimalna_kolicina(Q*)`: Količina pri kojoj je profit najveći.
 - `maksimalni_profit`: Maksimalni ostvareni profit.

5.4. Nizovi

U ekonomiji se vrlo često susrećemo s pojmom niza. Neki tipični primjeri uključuju vremenske serije (mjesečni ili godišnji podaci o troškovima, zaradi, investicijama) i različite oblike rasta ili opadanja tokom više perioda. Dva osnovna oblika niza su **aritmetički** i **geometrijski**. U ovom poglavlju objasnićemo njihovu definiciju i primjenu, te ćemo demonstrirati kako se mogu lako generisati i analizirati u Pythonu.

5.4.1. Aritmetički niz

Definicija: Aritmetički niz je slijed brojeva (a_n), definisan tako da je razlika između dva uzastopna člana konstantna. Ako označimo prvi član niza sa a_1 , a razliku sa d , tada se članovi niza računaju kao:

$$a_1, \quad a_2 = a_1 + d, \quad a_3 = a_1 + 2d, \quad \dots, \quad a_n = a_1 + (n - 1)d$$

Primjena u vremenskim serijama (uzastopni porast troškova)

Troškovi neke kompanije mogu rasti za konstantan iznos svake godine (npr. zbog inflacije, plata, fiksnih ulaganja). U tom slučaju, ako su troškovi u prvoj godini C_1 , a godišnji porast troškova iznosi Δ , onda bismo imali sekvencu $C_1, C_1+\Delta, C_1+2\Delta, \dots$

Skripta u Python-u koja demonstrira prethodni primjer i korištenje NumPy funkcije [arange\(\)](#).

```
import numpy as np

# Parametri aritmetičkog niza
a1 = 1000 # pocetni trosak
d = 200   # godisnji porast
n = 10    # broj godina

# Generisemo niz tako da svaki naredni clan raste za d
# Napomena: np.arange(start, stop, step) kreira niz [start, start+step, ...
stop-step]
troskovi_aritmeticki = np.arange(a1, a1 + n*d, d)

print("Aritmetički niz troškova:")
print(troskovi_aritmeticki)
```

Aritmetički niz troškova:
[1000 1200 1400 1600 1800 2000 2200 2400 2600 2800]

5.4.2. Geometrijski niz

Definicija: Geometrijski niz je slijed brojeva (g_n) kod kojeg je odnos dva uzastopna člana konstantan. Ako označimo prvi član niza sa g_1 , a taj odnos sa r , tada se članovi niza računaju kao:

$$g_1, \quad g_2 = g_1 \cdot r, \quad g_3 = g_1 \cdot r^2, \quad \dots, \quad g_n = g_1 \cdot r^{n-1}$$

Primjena u ekonomiji kroz rast investicija uz konstantan prinos

Ukoliko se neka investicija vrednuje po konstantnoj godišnjoj kamatnoj stopi (prinosu) r (izraženoj, npr., kao $1 + \text{kamatna stopa}$), tada je rast investicije prema geometrijskom nizu. Ako na početku imamo ulaganje I_1 , nakon n godina investicija će biti:

$$I_n = I_1 \cdot r^{n-1}$$

Realizacija primjera rasta investicije u Python-u.

```
import numpy as np

# Parametri geometrijskog niza
g1 = 1000    # početno ulaganje
r = 1.05      # godišnja kamatna stopa / prinos od 5%
n = 10        # broj godina

# Generisanje geometrijskog niza koristeći np.arange()
investicije_geo = g1 * (r ** np.arange(n))

# Ispis rezultata
print("Geometrijski niz investicija:")
print(investicije_geo)
```

Geometrijski niz investicija:
[1000. 1050. 1102.5 1157.625 1215.50625
1276.2815625 1340.09564063 1407.10042266 1477.45544379 1551.32821598]

5.4.3. Grafički prikaz aritmetičkog i geometriskog niza

Posmatrajmo primjer aritmetičkog niza rasta troškova predstavljenog formulom:

$$a_n = 1000 + (n-1) \cdot 50$$

gdje je početni trošak 1000 jedinica, godišnji porast troškova je 50 jedinica i period u kome se posmatra investicija je 10 godina. Ovaj niz biće predstavljen na grafiku zajedno sa geometrijskim nizom rasta investicije, predstavljen je formulom:

$$g_n = 1000 \cdot 1.05^{n-1}$$

gdje je početno ulaganje 1000 jedinica, godišnja kamata/prinos 5% (1.05) i period povrata investicije 10 godina.

Primjer skripte u Python-u koja generiše ova dva niza i crta grafika sa dobijenim vrijenostima.

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

# Parametri
n = 10
a1, d = 1000, 50 # aritmeticki niz
g1, r = 1000, 1.05 # geometrijski niz

# Generisemo nizove
niz_arit = [a1 + i*d for i in range(n)]
niz_geo = [g1 * (r**i) for i in range(n)]

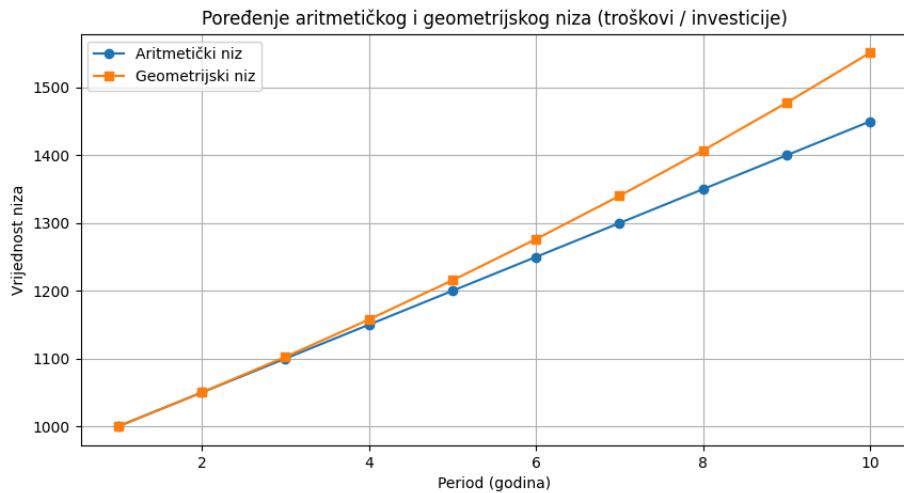
# Kreiramo x-osi za prikaz, tj. 'godine' [1,2,...,n]
godine = np.arange(1, n+1)

plt.figure(figsize=(10,5))

# Prikaz aritmetickog niza
plt.plot(godine, niz_arit, marker='o', label="Aritmetički niz")

# Prikaz geometrijskog niza
plt.plot(godine, niz_geo, marker='s', label="Geometrijski niz")

plt.title("Poređenje aritmetičkog i geometrijskog niza (troškovi / investicije)")
plt.xlabel("Period (godina)")
plt.ylabel("Vrijednost niza")
plt.legend()
plt.grid(True)
plt.show()
```



5.4.4. Granična vrijednost niza

Granična vrijednost niza opisuje ponašanje članova niza, kada broj članova raste beskonačno. Ovo je ključan koncept u matematici i ekonomiji jer pomaže u analizi dugoročnih trendova i predikcija.

Pojam granice: Niz (a_n) ima graničnu vrijednost L ako se članovi niza sve više približavaju L kako n raste. Formalna definicija glasi:

$$\lim_{n \rightarrow \infty} = L$$

ako za svako $\varepsilon > 0$ postoji indeks N takav da je za sve $n > N$ ispunjeno:

$$|a_n - L| < \varepsilon$$

Koncept granice se koristi u ekonomiji za analizu veličina, koje imaju tendenciju stabilizacije ili dugoročnog rasta/pada.

- **Aritmetički niz** sa konstantnim pozitivnim rastom ($d > 0$) nema konačnu granicu, već raste prema beskonačnosti (∞).
- **Geometrijski niz** ima različito ponašanje, zavisno o odnosu r :
 - Ako je $|r| < 1$, niz konvergira ka 0.
 - Ako je $|r| > 1$, niz raste prema beskonačnosti (∞).
 - Ako je $r = 1$, niz je konstantan i nema promjena.

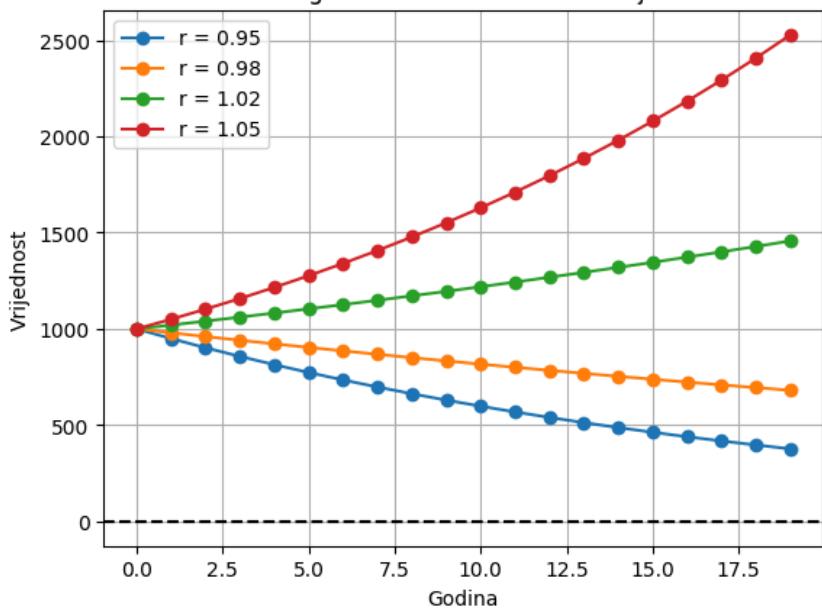
Posmatrajmo primjer gdje neka ekonomija raste po stopi $r < 1$ (npr. 0.95), veličina poput BDP-a konvergira ka 0, što je znak dugoročnog opadanja. S druge strane, ako je $r > 1$, BDP raste eksponencijalno, ali takav rast često nije održiv. U nastavku je primjer dugoročnih trendova realizovan u Python-u sa grafičkim prikazima.

```
# Primjer dugoročnog trenda
g1 = 1000      # Početna vrijednost (npr. početni prihod)
r_trendovi = [0.95, 0.98, 1.02, 1.05] # Stope rasta (pad i rast)
n = 20         # Broj godina

# Generisanje nizova
for r in r_trendovi:
    niz = g1 * (r ** np.arange(n))
    plt.plot(np.arange(n), niz, marker='o', label=f'r = {r}')

plt.title("Dugoročni trendovi u ekonomiji")
plt.xlabel("Godina")
plt.ylabel("Vrijednost")
plt.axhline(y=0, color='black', linestyle='--')
plt.legend()
plt.grid(True)
plt.show()
```

Dugoročni trendovi u ekonomiji



5.5. Diferencijalni račun

Diferencijalni račun jedan je od osnovnih matematičkih alata za analizu ekonomske dinamike. U ekonomiji se često koriste pojmovi poput marginalnih funkcija (npr. marginalni trošak, marginalni prihod) i brzina promjene različitih varijabli (npr. potražnje, ponude, proizvodnje i sl.). Cilj ovoga poglavlja je upoznati se s osnovnim konceptima granica i neprekidnosti funkcije te sa derivacijama funkcija, kao i s njihovom primjenom u ekonomiji i primjerima u Pythonu.

5.5.1. Granične vrijednosti i neprekidnost funkcije

Pojam granice funkcije predstavlja jedan od temeljnih koncepta matematičke analize i ima ključno mjesto u analizi ekonomskih modela. Granica funkcije se formalno definiše kao ponašanje funkcije $f(x)$ kada se nezavisna varijabla x približava nekoj vrijednosti a . Izraz:

$$\lim_{x \rightarrow a} f(x) = L$$

znači da se vrijednost funkcije $f(x)$ može učiniti proizvoljno bliskom vrijednosti L , pod uslovom da je x dovoljno blizu a . Ovo omogućava analizu ponašanja funkcije čak i u slučajevima kada ona nije definisana u tački a .

Važnost granice leži u njenoj intuitivnoj upotrebi za opisivanje promjena. Na primjer, granica može pomoći u razumjevanju što se događa s vrijednošću funkcije u ekstremnim situacijama, poput beskonačnog povećanja potražnje ili ekstremnih cijena. U ekonomiji, analiziranje granica omogućava bolje razumijevanje kako tržište reaguje na promjene u ključnim parametrima poput potražnje, ponude ili troškova.

Neprekidnost funkcije je blisko povezana s granicama i odnosi se na svojstvo funkcije da ne pravi „skokove“ kada se nezavisna varijabla mijenja. Formalno, funkcija $f(x)$ je neprekidna u tački a , ako su ispunjeni sljedeći uslovi:

- funkcija je definisana u tački a ,
- granica $\lim_{x \rightarrow a} f(x)$ postoji, i
- granica je jednaka vrijednosti funkcije u toj tački, odnosno $f(a) = \lim_{x \rightarrow a} f(x)$.

Neprekidnost funkcije osigurava da male promjene u ulaznim vrijednostima rezultiraju malim promjenama u izlaznim vrijednostima.

U ekonomiji, neprekidnost je ključna za modeliranje jer mnoge stvarne ekonomske pojave, poput odnosa ponude i potražnje, ponašanja potrošača ili troškova proizvodnje, ne smiju pokazivati diskontinuitete. Na primjer, ukoliko cijena proizvoda zavisi od količine proizvodnje, očekuje se da ta veza bude glatka i neprekidna. Diskontinuiteti u

funkcijama cijena ili troškova bi sugerisali da se nagli skokovi događaju bez ekonomski opravdanih razloga, što bi otežalo modeliranje i donošenje odluka.

Kako bi kroz primjer dodatno objasnili graničnu vrijednost i neprekidnost funkcije, pomatraćemo funkciju $f(x)=x^2$. Analiziraćemo ponašanje ove funkcije dok se x približava $a=2$. Ova funkcija je neprekidna, pa bi trebala zadovoljiti uslov da $\lim_{x \rightarrow a} f(x) = f(a)$.

```
import numpy as np

# Funkcija koju analiziramo
def f(x):
    return x**2

# Vrijednost prema kojoj se x približava
a = 2

# Generisanje niza vrijednosti x koji se približavaju a sa obje strane
x_values = np.array([a - 0.1, a - 0.01, a - 0.001, a + 0.001, a + 0.01, a + 0.1])
f_values = f(x_values)

# Ispis rezultata
print(f"Ispitivanje granice funkcije f(x) = x^2 dok se x približava {a}:")
for x, fx in zip(x_values, f_values):
    print(f"x = {x:.6f}, f(x) = {fx:.6f}")

# Provjera vrijednosti funkcije u tački a
print(f"\nFunkcija u tački x = {a}: f({a}) = {f(a)}")
```

Ispitivanje granice funkcije $f(x) = x^2$ dok se x približava 2:
x = 1.900000, f(x) = 3.610000
x = 1.990000, f(x) = 3.960100
x = 1.999000, f(x) = 3.996001
x = 2.001000, f(x) = 4.004001
x = 2.010000, f(x) = 4.040100
x = 2.100000, f(x) = 4.410000

Funkcija u tački x = 2: f(2) = 4

Zaključak

- Kako se x približava $a=2$ sa obje strane ($x \rightarrow 2^-$ i $x \rightarrow 2^+$), vrijednosti $f(x)$ postaju sve bliže 4.
- Funkcija je neprekidna jer $\lim_{x \rightarrow 2} f(x) = f(2) = 4$.

Sljedeći Python kod se nastavlja na prethodni primjer i grafički predstavlja funkciju $f(x)=x^2$, u posmatranoj tački $a=2$

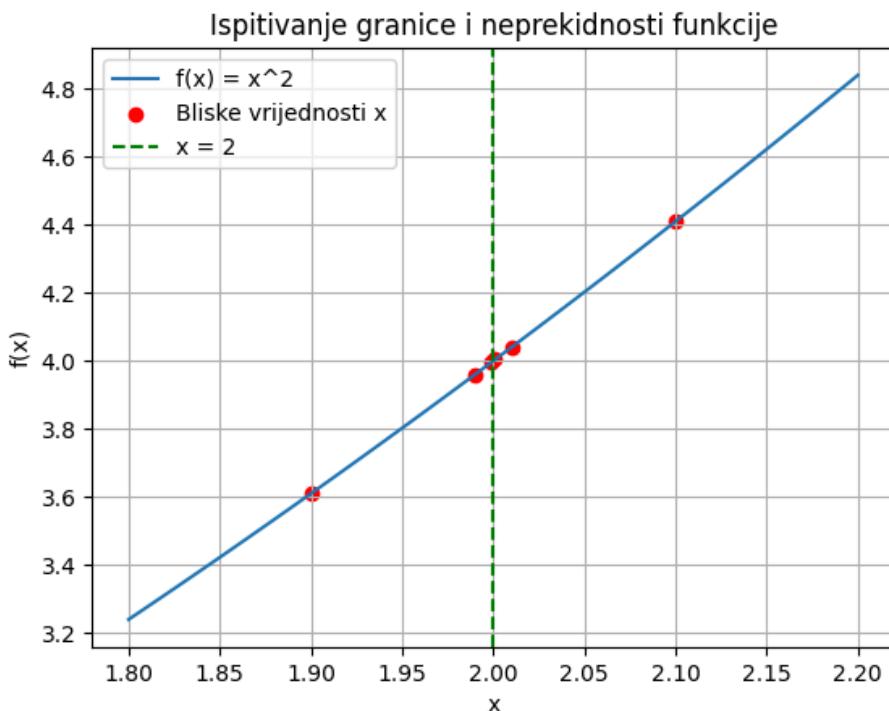
```
import matplotlib.pyplot as plt

# Generisanje tačaka za crtanje funkcije
x_plot = np.linspace(1.8, 2.2)
y_plot = f(x_plot)
```

```

# Crtanje funkcije
plt.plot(x_plot, y_plot, label="f(x) = x^2")
plt.scatter(x_values, f_values, color='red', label="Bliske vrijednosti x")
plt.axvline(a, color='green', linestyle='--', label=f"x = {a}")
plt.title("Ispitivanje granice i neprekidnosti funkcije")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.legend()
plt.grid()
plt.show()

```



U prethodnom primjeru smo analizirali neprekidnu funkciju $f(x)=x^2$, koja je definisana i glatka za sve vrijednosti x . Međutim, ako zamjenimo kvadratnu funkciju s $f(x)=x^2+1/x$, dobijamo funkciju koja nije definisana u tački $x=0$ zbog dijeljenja s nulom. Ova funkcija pokazuje diskontinuitet i ima ponašanje koje divergira u beskonačnost kada se x približava nuli.

Prilikom analize takve funkcije, $f(x)$ raste prema $+\infty$ kada se x približava nuli s pozitivne strane ($x \rightarrow 0^+$), dok opada prema $-\infty$ kada se x približava nuli s negativne strane ($x \rightarrow 0^-$). Ovo ponašanje omogućava jasno sagledavanje kako funkcija divergira zavisno o smjeru približavanja nuli.

Da bi se izbjegla greška uzrokovana dijeljenjem s nulom u praktičnoj implementaciji, funkciju $f(x)$ prilagođavamo skriptu tako da numerički simulira granicu funkcije bez

računanja za $x=0$. Na ovaj način, se pruža pregled i numerička analiza kako se $f(x)$ ponaša u blizini tačke diskontinuiteta.

Primjer koda koji obrađuje ovu funkciju, kao i grafički prikaz ponašanja u blizini $x=0$.

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

# Funkcija koju analiziramo
def f(x):
    with np.errstate(divide='ignore', invalid='ignore'):
        return np.where(x == 0, float('inf'), x**2 + 1/x)

# Generisanje vrijednosti x koje se približavaju 0 sa obje strane
x_left = np.linspace(-2, -0.000001) # Lijeva strana ( $x \rightarrow 0^-$ )
x_right = np.linspace(0.000001, 2) # Desna strana ( $x \rightarrow 0^+$ )

# Računanje vrijednosti funkcije
f_left = f(x_left)
f_right = f(x_right)

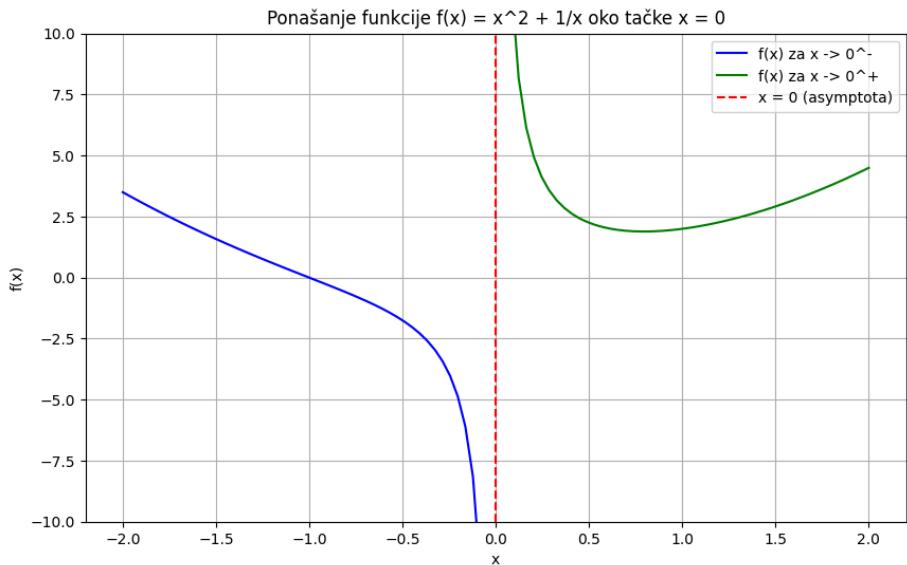
# Kombinovanje vrijednosti za prikaz
x_values = np.concatenate((x_left, x_right))
f_values = f(x_values)

# Grafički prikaz funkcije
plt.figure(figsize=(10, 6))
plt.plot(x_left, f_left, label="f(x) za  $x \rightarrow 0^-$ ", color="blue")
plt.plot(x_right, f_right, label="f(x) za  $x \rightarrow 0^+$ ", color="green")

# Oznake na grafiku
plt.axvline(x=0, color='red', linestyle='--', label="x = 0 (asimptota)")
plt.title("Ponašanje funkcije  $f(x) = x^2 + 1/x$  oko tačke  $x = 0$ ")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.ylim(-10, 10) # Ograničenje na y-osi za bolji pregled
plt.legend()
plt.grid()

# Prikaz grafa
plt.show()

# Ispis numeričkih vrijednosti za x blizu 0
print("Numeričke vrijednosti funkcije f(x) blizu x = 0:")
for x, fx in zip(np.concatenate((x_left[-5:], x_right[:5])),
                  np.concatenate((f_left[-5:], f_right[:5]))):
    print(f"x = {x:.6f}, f(x) = {fx:.6f}")
```



Numeričke vrijednosti funkcije $f(x)$ blizu $x = 0$:

```

x = -0.163266, f(x) = -6.098310
x = -0.122450, f(x) = -8.151610
x = -0.081634, f(x) = -12.243192
x = -0.040817, f(x) = -24.497746
x = -0.000001, f(x) = -1000000.000000
x = 0.000000, f(x) = 1000000.000000
x = 0.040816, f(x) = 24.501607
x = 0.081633, f(x) = 12.256650
x = 0.122449, f(x) = 8.181654
x = 0.163265, f(x) = 6.151652

```

5.5.2. Izvod funkcija

Izvod funkcije predstavlja jedan od najvažnijih matematičkih alata, koji se koriste za analizu promjena, optimizaciju procesa i donošenje odluka. Njegova suština leži u mogućnosti da se ispita brzina kojom se jedna veličina mijenja u zavisnosti od druge. U ekonomiji, izvod se često primjenjuje za analizu marginalnih vrijednosti, kao što su marginalni trošak, prihod i korisnost, kao i za razumijevanje elastičnosti, odnosno relativnih promjena u varijablama.

Pored tehničke primjene, ovaj alat pruža i dublje razumijevanje prirode ekonomskih pojava. Pomaže u analizi odnosa između ekonomskih varijabli, predviđanju tržišnih trendova i identifikovanju optimalnih strategija.

Definicija izvoda: Izvod funkcije $f(x)$ u tački x formalno se definiše kao:

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$$

Suštinski, izvod predstavlja „nagib” tangente na grafiku funkcije u tački x . Drugim riječima, izvod pokazuje brzinu promjene vrijednosti funkcije $f(x)$ u odnosu na promjenu x .

Kao primjer posmatrajmo funkciju ukupnog troška $C(x)$, gdje x označava količinu proizvedenih jedinica. Izvod $C'(x)$ daje marginalni trošak, odnosno dodatni trošak proizvodnje jedne dodatne jedinice.

$$C'(x) = \frac{\Delta C}{\Delta x}$$

Kada $\Delta x \rightarrow 0$, izvod precizno izražava promjenu troška za vrlo malu promjenu u proizvodnji.

Izvod omogućava odgovore na ključna pitanja u ekonomiji, kao što su:

- 1) Koliko će se trošak povećati proizvodnjom dodatne jedinice?
Odgovor na ovo pitanje daje **marginalni trošak**, koji je prvi izvod funkcije ukupnog troška.
- 2) Koliko dodatnog prihoda će donijeti prodaja jedne dodatne jedinice proizvoda?
Ovo se analizira kroz **marginalni prihod**, koji je prvi izvod funkcije ukupnog prihoda.
- 3) Kako se tražnja mijenja u zavisnosti od cijene?
Elastičnost potražnje, zasnovana na derivacijama, mjeri relativne promjene u tražnji uslijed promjene cijene.

Primjer s funkcijom ukupnog troška $C(x)=3x^2+5x+200$. Koristi se modul SciPy i funkcija `derivative` za računanje marginalnog troška M .

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.misc import derivative

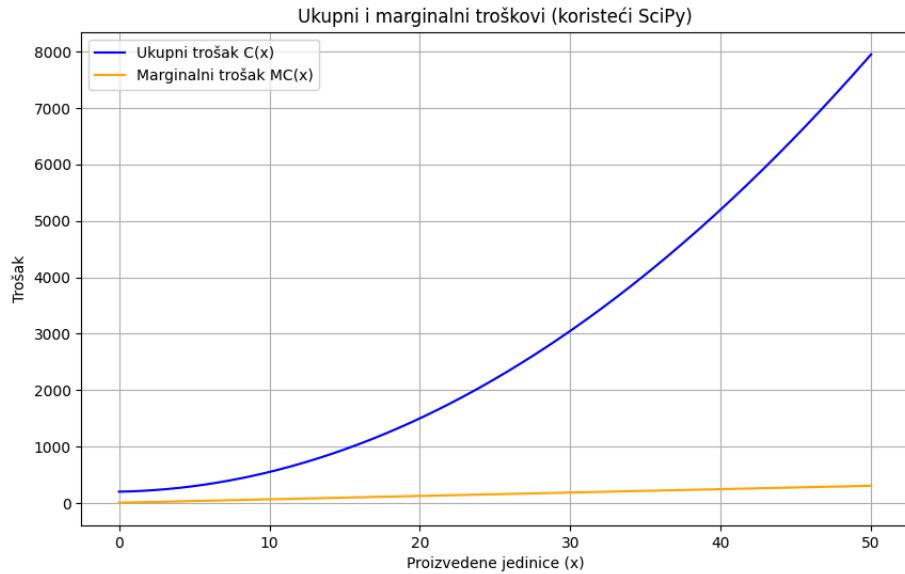
# Definicija funkcije ukupnog troška
def C(x):
    return 3 * x**2 + 5 * x + 200

# Numeričko računanje marginalnog troška koristeći SciPy
x_values = np.linspace(0, 50, 500)
mc_values = [derivative(C, x, dx=1e-6) for x in x_values] # dx je mali razmak

# Generisanje vrijednosti funkcije troškova
c_values = C(x_values)

# Grafički prikaz
plt.figure(figsize=(10, 6))
plt.plot(x_values, c_values, label="Ukupni trošak C(x)", color='blue')
plt.plot(x_values, mc_values, label="Marginalni trošak MC(x)", color='orange')
plt.title("Ukupni i marginalni troškovi (koristeći SciPy)")
```

```
plt.xlabel("Proizvedene jedinice (x)")
plt.ylabel("Trošak")
plt.legend()
plt.grid(True)
plt.show()
```



5.6. Integralni račun

Integrali su matematički koncept koji omogućava izračunavanje ukupnih ili akumuliranih vrijednosti iz funkcija, koje zavise od promjenjivih veličina. Njihova primjena nije ograničena na čisto matematičke probleme već se proteže i na praktične oblasti, uključujući ekonomiju, gdje omogućavaju analizu različitih ekonomskih pojava i procesa. Kroz integrale možemo sagledati ukupne efekte promjena, poput akumulacije troškova, prihoda ili koristi, u vremenu ili drugim kontekstima.

U ekonomiji, integrali se koriste za razumijevanje i kvantifikaciju kontinuiranih procesa. Na primjer, izračunavanje ukupnog troška proizvodnje kada je poznata funkcija marginalnog troška zahtjeva upotrebu određenog integrala. Slično, akumulirani prihod kroz određeni vremenski period može se modelirati pomoću integracije marginalnog prihoda. Pored toga, koncept integracije omogućava analizu diskontovanog novčanog toka, gdje se uzimaju u obzir vremenska vrijednost novca i dugoročne investicije.

Definicija integrala: Određeni integral funkcije $f(x)$ na intervalu $[a,b]$ definisan je kao:

$$\int_a^b f(x) dx$$

U suštini, integral predstavlja površinu ispod grafika funkcije $f(x)$ između tačaka a i b .

Neodređeni integral funkcije, ili primitivna funkcija, predstavlja obrnuti proces izvoda:

$$F(x) = \int f(x) dx$$

Gdje je $F'(x)=f(x)$.

U nastavku je Python skripta, koja grafički prikazuje površinu ispod funkcije $f(x)=3x^2+5x+200$ i računa određeni integral funkcije u granicama $[2,4]$.

Za implementaciju je korišćen modul SymPy, koji omogućava rad sa simboličkim matematičkim izrazima. Modul pruža funkcije poput `symbols` za definisanje simboličkih varijabli i `integrate` za analitičko izračunavanje integrala. Funkcija `lambdify` je korišćena za konverziju simboličke funkcije $f(x)$ u numeričku funkciju, ovo omogućava njeno direktno korišćenje u bibliotekama poput NumPy-a za generisanje podataka i vizualizaciju.

Ovaj primjer pokazuje kako se kombinuje matematička analiza i programiranje za vizualizaciju i rješavanje matematičkih problema.

```

import numpy as np
import matplotlib.pyplot as plt
from sympy import symbols, integrate, lambdify

# Definišemo simboličku varijablu i funkciju
x = symbols('x')
f = 3 * x**2 + 5 * x + 200

# Izračunavanje određenog integrala na intervalu [2, 4]
area = integrate(f, (x, 2, 4))
print(f"Određeni integral funkcije na intervalu [2, 4]: {area}")

# Konverzija simboličke funkcije u numeričku
f_numeric = lambdify(x, f, 'numpy')

# Generisanje podataka za grafiku
x_values = np.linspace(1, 5, 500)
y_values = f_numeric(x_values)

# Granice integracije
x_fill = np.linspace(2, 4, 500)
y_fill = f_numeric(x_fill)

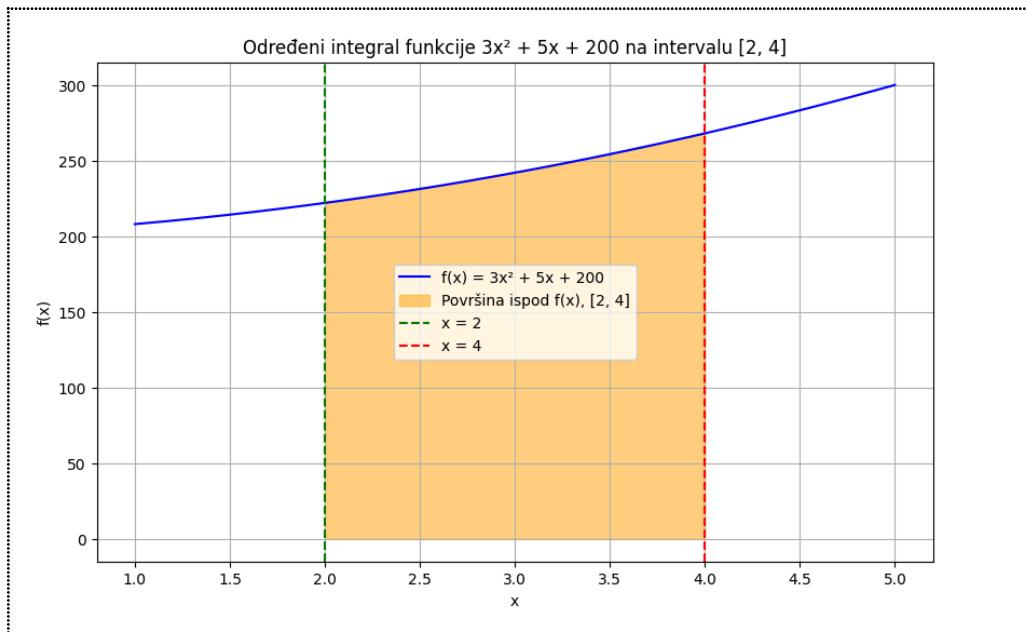
# Crtanje grafika
plt.figure(figsize=(10, 6))
plt.plot(x_values, y_values, label=f"f(x) = 3x2 + 5x + 200", color='blue')
plt.fill_between(x_fill, y_fill, color='orange', alpha=0.5, label="Površina ispod f(x), [2, 4]")

# Oznake granica
plt.axvline(x=2, color='green', linestyle='--', label="x = 2")
plt.axvline(x=4, color='red', linestyle='--', label="x = 4")

# Dodavanje naslova i oznaka
plt.title("Određeni integral funkcije 3x2 + 5x + 200 na intervalu [2, 4]")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.legend()
plt.grid(True)
plt.show()

```

Određeni integral funkcije na intervalu [2, 4]: 486



5.6.1. Primjer računanja neto viška dobiti

Pretpostavimo da imamo dva investiciona plana, svaki sa svojim modelom generisanja profita u zavisnosti od broja godina x :

1. Prvi plan: Profit se generiše prema funkciji $P_1(x)=x^2+50$.
2. Drugi plan: Profit se generiše prema funkciji $P_2(x)=300+4x$.

Želimo odgovoriti na sljedeća pitanja:

1. Za koliko godina će drugi plan postati profitabilniji od prvog (tj. kada će se profiti izjednačiti)?
2. Kolika je neto višak dobiti ako se uloži u drugi plan do tačke presjeka?

Matematičko rješenje

1) Tačka presjeka (vrijeme susreta):

Tačka presjeka se pronađe rješavanjem jednačine: $P_1(x)=P_2(x)$, odnosno:
 $x^2+50=300+4x$

2) Neto višak dobiti:

Neto višak dobiti drugog plana u odnosu na prvi izračunava se integracijom razlike između funkcija $P_2(x)-P_1(x)$ na intervalu od 0 do tačke presjeka x :

$$\Delta = \int_0^x (P_2(x) - P_1(x))$$

Implementacija rješenja u Python-u sa grafičkim prikazom je prikazana u nastavku.

```

import sympy as sp
import numpy as np
import matplotlib.pyplot as plt

# Definišemo simboličku varijablu
x = sp.symbols('x')

# Funkcije profita za oba plana
P1 = x**2 + 70
P2 = 150 + 3 * x

# 1. Tačka presjeka
intersection = sp.solve(P1 - P2, x)
intersection_point = [float(sol) for sol in intersection if sol > 0][0] # Uzimamo
prvu pozitivnu tačku
print(f"Tačka presjeka: {intersection_point:.2f} godina")

# 2. Neto višak dobiti
profit_difference = P2 - P1
net_profit = sp.integrate(profit_difference, (x, 0, intersection_point))
print(f"Neto višak dobiti drugog plana: {net_profit:.2f}")

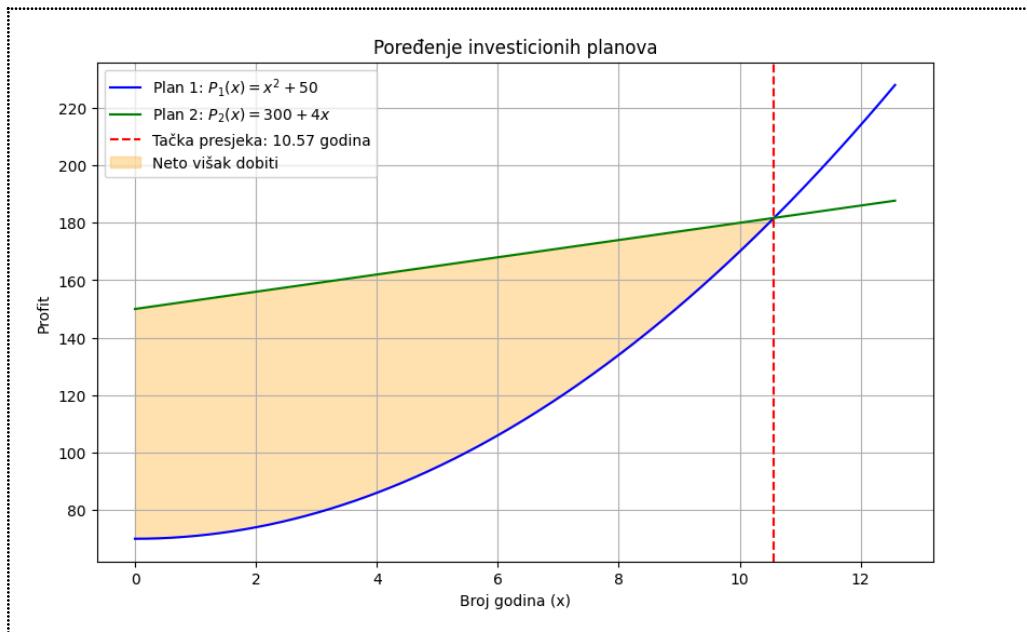
# Konverzija simboličkih funkcija u numeričke
P1_numeric = sp.lambdify(x, P1, modules='numpy')
P2_numeric = sp.lambdify(x, P2, modules='numpy')

# Generisanje podataka za grafiku
x_vals = np.linspace(0, intersection_point + 2, 500)
P1_vals = P1_numeric(x_vals)
P2_vals = P2_numeric(x_vals)

# Grafički prikaz
plt.figure(figsize=(10, 6))
plt.plot(x_vals, P1_vals, label="Plan 1: $P_1(x) = x^2 + 50$", color="blue")
plt.plot(x_vals, P2_vals, label="Plan 2: $P_2(x) = 300 + 4x$", color="green")
plt.axvline(intersection_point, color="red", linestyle="--", label=f"Tačka presjeka:
{intersection_point:.2f} godina")
plt.fill_between(x_vals, P2_vals, P1_vals, where=(x_vals <= intersection_point),
color="orange", alpha=0.3, label="Neto višak dobiti")
plt.title("Poređenje investicionih planova")
plt.xlabel("Broj godina (x)")
plt.ylabel("Profit")
plt.legend()
plt.grid(True)
plt.show()

Tačka presjeka: 10.57 godina
Neto višak dobiti drugog plana: 619.54

```



5.6.2. Primjer računanja prosječne akumulirane dobiti

Pretpostavimo da je ukupna dobit kroz vrijeme t (u milionima BAM) data funkcijom:

$$P(t) = (t + 4) \cdot \sqrt{t}$$

Želimo izračunati prosječnu akumulaciju dobiti za $t=2$, $t=5$ i $t=10$ godina.

Matematičko rješenje

Prosječna dobit u intervalu od $t=0$ do $t=T$ se računa kao:

$$P_{\text{avg}} = \frac{1}{T} \int_0^T P(t) dt$$

Gdje je:

- $\int P(t) dt$ predstavlja ukupnu akumuliranu dobit u tom periodu.

Realizacija ovog primjera u Python okruženu prikazana je u nastavku.

```
import sympy as sp

# Definisanje simboličkih varijabli i funkcije dobiti
t = sp.symbols('t')
P = (t + 4) * sp.sqrt(t)

# Prosječna dobit
def prosjecna_dobit(T):
    ukupna_dobit = sp.integrate(P, (t, 0, T)) # Izračunavanje integrala
    prosjecna_dobit = ukupna_dobit / T # Podjela sa T
    return ukupna_dobit, prosjecna_dobit

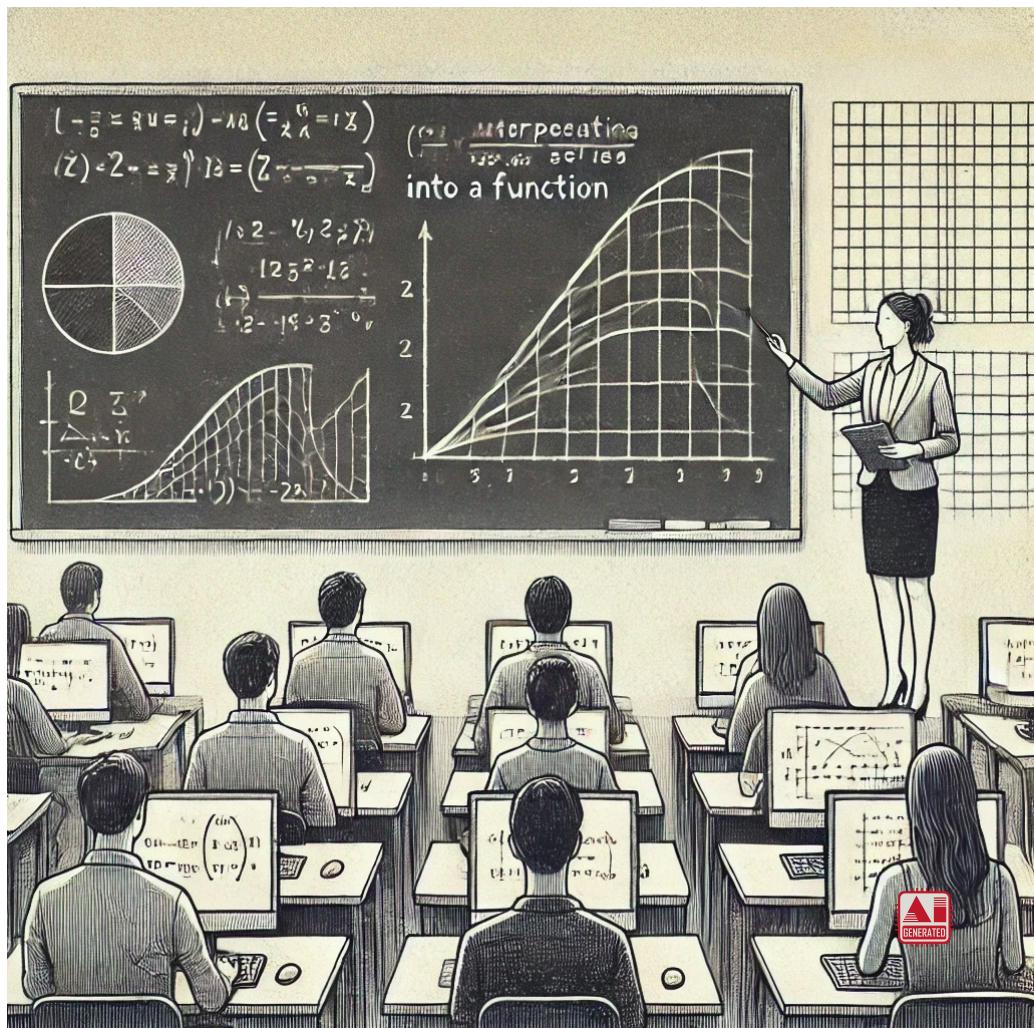
# Vrijednosti za T
vrijednosti_T = [2, 5, 10]
for T in vrijednosti_T:
    ukupna, prosjecna = prosjecna_dobit(T)
    print(f"Za T = {T} godina:")
    print(f"  Ukupna akumulacija dobiti: {ukupna:.2f} miliona BAM")
    print(f"  Prosječna godišnja dobit: {prosjecna:.2f} miliona BAM\n")
```

```
Za T = 2 godina:
  Ukupna akumulacija dobiti: 9.81 miliona BAM
  Prosječna godišnja dobit: 4.90 miliona BAM

Za T = 5 godina:
  Ukupna akumulacija dobiti: 52.17 miliona BAM
  Prosječna godišnja dobit: 10.43 miliona BAM

Za T = 10 godina:
  Ukupna akumulacija dobiti: 210.82 miliona BAM
  Prosječna godišnja dobit: 21.08 miliona BAM
```

6. PYTHON I STATISTKA



"Najveća vrijednost slike je kada nas prisili da primjetimo ono što nikada nismo očekivali da ćemo vidjeti."

"The greatest value of a picture is when it forces us to notice what we never expected to see."

– John Tukey²¹

²¹ Džon Tuki, poznati statističara, izjava iz njegove knjige Exploratory Data Analysis (1977), koja naglašava moć vizualizacije statističkih podataka u otkrivanju neočekivanih uvida. Ovo je posebno važno za ekonomsku analizu, gdje se statistika koristi za prepoznavanje trendova, anomalija i skrivenih obrazaca, čime se omogućava donošenje boljih odluka.

Statistika je moćan alat u ekonomiji, koji omogućava sistematsko prikupljanje, detaljnu analizu i preciznu interpretaciju podataka, koji oslikavaju različite ekonomske procese i pojave. Primjenom statističkih metoda, ekonomisti mogu donositi utemeljene zaključke zasnovane na podacima, identifikovati važne obrazce i trendove, te predvidjeti buduća kretanja u različitim segmentima ekonomije. Takođe, statistika pruža osnovu za razumijevanje i kvantifikaciju kompleksnih odnosa između ekonomske varijabli, čime postaje neophodna za donošenje ispravnih odluka u poslovanju, upravljanju i ekonomskim politikama.

Jedan od ključnih alata koji značajno unapređuje primjenu statistike u ekonomiji je **Python** – programski jezik poznat po jednostavnosti i fleksibilnosti. Zahvaljujući bibliotekama poput **NumPy**, **SciPy** i **Pandas**, Python omogućava brzu obradu velikih skupova podataka, napredne statističke analize i kreiranje vizualizacija za bolje razumijevanje rezultata. Ovi alati ekonomistima nude brojne prednosti, uključujući automatizaciju procesa, povećanje preciznosti proračuna i lakšu analizu kompleksnih odnosa među varijablama.

Korišćenjem **NumPy**, ekonomisti mogu efikasno rukovati numeričkim podacima i izvoditi osnovne statističke proračune, poput računanja prosjeka, standardne devijacije i korelacije. Sa **SciPy**, mogu se primijeniti naprednije statističke metode, uključujući testiranje hipoteza i regresione analize, dok **Pandas** omogućava rad sa tabelarnim podacima na način koji je intuitivan i prilagođen ekonomskim potrebama. Sve ove biblioteke omogućavaju brzu i tačnu analizu podataka, čak i kada se radi o velikim setovima podataka, koji bi bili izazovni za obradu tradicionalnim metodama.

Ključne teme koje će biti obrađene u ovom poglavlju su:

- **Osnovni statistički pojmovi i mjere:** Kako kvantifikovati podatke i razumjeti njihovu raspodjelu koristeći mjere centralne tendencije i disperzije.
- **Vizualizacija podataka:** Upotreba grafika za preglednu prezentaciju ekonomske informacije.
- **Statističko zaključivanje:** Primjena hipoteza i testova za dovođenje odluka na osnovu podataka.
- **Primjena u ekonomiji:** Kako statistika omogućava analizu tržišta, praćenje ekonomske indikatora i dovođenje poslovnih i strateških odluka.

6.1. Osnove statistike

Statistika je naučna disciplina koja se bavi prikupljanjem, organizacijom, analizom, interpretacijom i prezentacijom podataka. Njena osnovna svrha je da omogući donošenje zaključaka na osnovu podataka i pruži alate za kvantifikaciju neizvjesnosti. U ekonomiji, statistika omogućava razumijevanje i objašnjavanje ekonomskih pojava, donošenje ispravnih odluka i optimizaciju ekonomskih procesa.

6.1.1. Skup podataka i vrste podataka

Prvi korak u svakoj statističkoj analizi je razumijevanje prirode podataka, koji se analiziraju. Skup podataka predstavlja organizovanu zbirku vrijednosti, koje opisuju određene pojave, procese ili varijable. Kvalitet analize zavisi od toga koliko su podaci pravilno prikupljeni, kategorizovani i obrađeni. U ekonomiji, podaci dolaze iz različitih izvora, poput anketa, tržišnih istraživanja, državnih statistika ili finansijskih izveštaja i često su osnova za donošenje poslovnih i političkih odluka.

Podaci se prvo dijele na **kvalitativne** i **kvantitativne** u zavisnosti od toga što opisuju i kako se mijere.

Kvalitativni podaci (opisni podaci)

Ovi podaci izražavaju kvalitativne karakteristike, koje ne mogu biti kvantifikovane brojevima. Kvalitativni podaci se često koriste za kategorizaciju ili identifikaciju specifičnih osobina, a obično se izražavaju kao riječi, simboli ili etikete.

Primjeri kategorija kvalitativnih podataka:

- Kategorije proizvoda: Luksuzni, osnovni, tehnološki, ekološki.
- Tip tržišta: Savršena konkurencija, monopol, oligopol, monopson.
- Zadovoljstvo potrošača: Potpuno zadovoljan, umjereni zadovoljan, neutralan, nezadovoljan, izuzetno nezadovoljan.

Podjela kvalitativnih podataka:

- Nominalni podaci: Kategorije koje nemaju prirodan redoslijed (npr. vrste industrija: poljoprivreda, proizvodnja, usluge).
- Ordinalni podaci: Kategorije koje imaju prirodan redoslijed (npr. kreditni rejting: „loš“, „prosječan“, „odličan“).

Kvantitativni podaci (numerički podaci)

Kvantitativni podaci izražavaju količinu ili broj i uvijek se mijere numerički. Ovi podaci omogućavaju precizne proračune i analize, koje su ključne za donošenje odluka u ekonomiji.

Primjeri kvantitativnih podataka:

- Prihodi kompanija u dolarima.
- Stopa nezaposlenosti izražena u procentima.
- Broj zaposlenih u različitim sektorima.

Podjela kvantitativnih podataka:

- Intervalni podaci. Po definiciji intervalni podaci imaju fiksani razmak između vrijednosti, ali nemaju apsolutnu nulu (tj. nula ne znači "odsustvo vrijednosti"). Odnosi između vrijednosti nisu smislena mjera. Na primjer temperatura u Celzijusima ili Farenhajtima (0°C ne znači "bez temperature") ili vrijeme na satu (razlika između 2 h i 4 h je dva sata, ali 4 h nije "dvostruko više" od 2 h).
- Racionalni podaci. Po definiciji racionalni podaci imaju apsolutnu nulu (nula znači "odsustvo vrijednosti") i odnosi među vrijednostima imaju smisla. Na primjer prihodi kompanija (prihod od 0 dolara znači da kompanija nije zaradila ništa) ili cijene proizvoda (proizvod od 100 dolara koštira dvostruko više od proizvoda od 50 dolara).

Kvantitativni podaci se dalje dijele na **diskretne i kontinuirane**, u zavisnosti od toga da li mogu poprimiti samo određene vrijednosti ili bilo koju vrijednost unutar određenog intervala.

Diskretni podaci

Diskretni podaci predstavljaju brojeve, koji se mogu prebrojati. Oni imaju konačan ili prebrojiv broj mogućih vrijednosti, što znači da ne mogu poprimiti decimalne ili razlomljene vrijednosti.

Primjeri diskretnih podataka:

- Broj prodanih proizvoda u određenom periodu.
- Broj zaposlenih u kompaniji.
- Broj transakcija na tržištu.

Kontinuirani podaci

Kontinuirani podaci mogu poprimiti bilo koju vrijednost unutar određenog intervala i često uključuju decimalne vrijednosti. Oni se mjere, a ne broje, što ih čini korisnim za analizu veličina koje su neprekidne.

Primjeri kontinuiranih podataka:

- Prihodi kompanija izraženi u milionima dolara.
- Vrijeme potrebno za isporuku proizvoda.
- Inflacija izražena sa preciznošću do dva decimalna mesta.

Slijede primjeri Python skripte koja definiše i prikazuje različite vrste podataka, zajedno sa njihovom klasifikacijom u skladu sa podjelom na kvalitativne i kvantitativne, te diskretnе i kontinuirane.

```
# Definicija podataka prema tipu

# Kvalitativni podaci
kvalitativni_podaci = {
    "Kategorije proizvoda": ["Lukuzni", "Osnovni", "Tehnološki", "Ekološki"],
    "Tip tržišta": ["Savršena konkurencija", "Monopol", "Oligopol", "Monopson"],
    "Zadovoljstvo potrošača": ["Potpuno zadovoljan", "Umjereni zadovoljan", "Nezadovoljan"]
}

# Kvantitativni podaci
kvantitativni_podaci = {
    "Prihodi kompanija (u milionima USD)": [1.2, 3.5, 7.8, 15.0],
    "Stopa nezaposlenosti (%)": [4.5, 7.8, 10.2, 3.4],
    "Broj zaposlenih": [120, 250, 340, 80]
}

# Diskretni i kontinuirani podaci
diskretni_podaci = {
    "Broj zaposlenih": [120, 250, 340, 80],
    "Broj prodanih proizvoda": [500, 1000, 750, 1200]
}

kontinuirani_podaci = {
    "Prihodi kompanija (u milionima USD)": [1.2, 3.5, 7.8, 15.0],
    "Stopa nezaposlenosti (%)": [4.5, 7.8, 10.2, 3.4],
    "Prosječna plata (u hiljadama USD)": [25.3, 30.1, 27.8, 29.5]
}

# Prikaz podataka
print("Kvalitativni podaci:")
for kategorija, vrijednosti in kvalitativni_podaci.items():
    print(f" {kategorija}: {vrijednosti}")

print("\nKvantitativni podaci:")
for kategorija, vrijednosti in kvantitativni_podaci.items():
    print(f" {kategorija}: {vrijednosti}")

print("\nDiskretni podaci:")
for kategorija, vrijednosti in diskretni_podaci.items():
    print(f" {kategorija}: {vrijednosti}")

print("\nKontinuirani podaci:")
for kategorija, vrijednosti in kontinuirani_podaci.items():
    print(f" {kategorija}: {vrijednosti}")
```

6.1.2. Distribucija frekvencija

Distribucija frekvencija je metoda za organizaciju podataka, tako da prikazuje koliko često se pojedine vrijednosti ili opsezi vrijednosti pojavljuju u skupu. Ona je posebno korisna kod velikih skupova podataka, gdje pojedinačne vrijednosti nisu lako uočljive.

Elementi distribucije frekvencija:

- Intervali – Podaci se grupišu u klase ili intervale, koje pokrivaju cijeli raspon vrijednosti.
- Frekvencija – Broj podataka koji pripadaju svakom intervalu.
- Kumulativna frekvencija – Zbir frekvencija svih intervala do određenog trenutka.

Odredivanje intervala

Širina intervala (h) i broj klasa (k) određuju se prema opsegu podataka (R) koristeći formulu:

$$h = \frac{R}{k},$$

Za R , za koje vrijedi da je $R=X_{\max}-X_{\min}$, gdje je X_{\max} maksimalna vrijednost u intervalu, a X_{\min} minimalna vrijednost u intervalu.

Broj klasa se određuje prema Sturgesovom pravilu:

$$k = 1 + 3.322 \cdot \log_{10}(N)$$

Gdje je N broj podataka.

Sljedeći primjer u Python-u pokazuje kako se primjenjuje distribucija frekvencija nad nizom podataka.

```
import numpy as np
import pandas as pd

# Skup podataka
data = [2, 5, 8, 10, 12, 15, 18, 20, 22, 25, 28, 30]

# Parametri
X_min, X_max = min(data), max(data)
R = X_max - X_min
k = int(1 + 3.322 * np.log10(len(data))) # Broj klasa
h = np.ceil(R / k) # Širina intervala, zaokružena na viši cijeli broj

# Formiranje intervala
bins = np.arange(X_min, X_max + h, h)
labels = [f"{int(bins[i])}-{int(bins[i+1])-1}" for i in range(len(bins)-1)]
freq_table = pd.cut(data, bins=bins, labels=labels).value_counts().sort_index()

# Prikaz tabele
print("Tabela frekvencijske distribucije:")
print(freq_table)
```

Tabela frekvencijske distribucije:

2-8	2
9-15	3
16-22	3
23-29	3

Objašnjenje:

- `bins = np.arange(X_min, X_max + h, h)` - u ovom dijelu koda definišemo intervale. Funkciju `np.arange` iz biblioteke NumPy služi za generisanje granica intervala (poznatih kao "bins"). Granice se generišu od minimalne vrijednosti (X_{\min}) do maksimalne vrijednosti (X_{\max}) uz korak h , koji predstavlja širinu intervala. Parametar `X_max+h` se koristi da se maksimalna vrijednost (X_{\max}) povećava za h kako bi se osiguralo da gornja granica posljednjeg intervala uključuje najveću vrijednost iz skupa podataka.
- `labels` - Ova linija kreira čitljive oznake za intervale (klase) koje će se prikazati u tabeli distribucija frekvencija. Koristeći "granične vrijednosti" iz `bins`, generiše se string oznaka za svaki interval. `bins[i+1]-1`. Na ovaj način se smanjuje gornja granica intervala za 1 kako bi se osigurala ispravna inkluzivna donja granica i ekskluzivna gornja granica.
- `freq_table` - Brojanje frekvencija po intervalima. Funkcija `pd.cut` grupiše vrijednosti iz skupa podataka (`data`) u definisane intervale (`bins`) i dodjeljuje im oznake (`labels`). Ovo mapira svaku vrijednost iz skupa na odgovarajući interval. Funkcija `.value_counts()` broji koliko vrijednosti iz skupa podataka pripada svakom intervalu. Funkcija `.sort_index()` sortira rezultate prema redoslijedu intervala, osiguravajući da se prikazuju u ispravnom redoslijedu (od najmanjeg prema najvećem).

6.1.3. Mjere distribucije podataka (Kvantili)

Mjere distribucije podataka predstavljaju skup statističkih alata, koji omogućavaju dublje razumijevanje položaja vrijednosti unutar raspodjele podataka. Za razliku od mjera centralne tendencije, koje se fokusiraju na opisivanje središnje vrijednosti skupa, mjere distribucije pružaju uvid u način na koji su podaci raspoređeni, bilo da se radi o njihovoj koncentraciji, rasipanju ili ekstremima.

Kvantili, kao ključne mjere distribucije, dijele podatke na jednake dijelove, čime omogućavaju analizu položaja vrijednosti unutar skupa u odnosu na cijekupnu distribuciju. Ovi alati su neophodni u situacijama kada je potrebno procijeniti relativni položaj određene vrijednosti ili analizirati obrasce i razlike unutar podataka.

Mjere distribucije imaju široku primjenu u ekonomiji i društvenim naukama, gdje se često koriste za analizu prihoda, cijena, tržišnih performansi i drugih ključnih ekonomskih pokazatelja. Korišćenjem kvantila, moguće je precizno segmentirati podatke i donositi ispravne zaključke o njihovoj raspodjeli.

Kvantili su vrijednosti koje dijele skup podataka na q jednakih dijelova, gdje svaki dio obuhvata $1/q$ ukupnog broja podataka. Kvantil reda k (Q_k) definisan je kao vrijednost ispod koje se nalazi q/k svih podataka, gdje je $k \in \{1, 2, \dots, q-1\}$.

Specijalni slučajevi kvantila

- **Kvartili:** Dijele podatke na četiri jednakna dijela:
 - Q1 Prvi kvartil (25% podataka ispod ove vrijednosti),
 - Q2: Drugi kvartil (medijana, 50% podataka ispod ove vrijednosti),
 - Q3: Treći kvartil (75% podataka ispod ove vrijednosti).
- **Decili:** Dijele podatke na deset jednakih dijelova:
 - D1: Prvi decil (10% podataka ispod ove vrijednosti),
 - D2: Drugi decil (20% podataka ispod ove vrijednosti),
 - ...
 - D9: Deveti decil (90% podataka ispod ove vrijednosti).
- **Percentili:** Dijele podatke na stotinu jednakih dijelova:
 - P1: Prvi percentil (1% podataka ispod ove vrijednosti),
 - P50: Pedeseti percentil (medijana, 50% podataka ispod ove vrijednosti),
 - P99: Devedeset deveti percentil (99% podataka ispod ove vrijednosti).

Kvantili, poput kvartila, decila i percentila, dijele podatke na jednake dijelove kako bi se bolje razumjela njihova raspodjela. Međutim, vrijednosti poput Q4 (četvrti kvartil), D10 (deseti decil) i P100 (stoti percentil) se ne navode jer predstavljaju maksimum skupa podataka.

Funkcije za računanje kvantila u Pythonu

U modulima NumPy i Pandas koriste se sljedeće funkcije:

- `np.quantile(a, q, axis=None, interpolation='linear')`
- `np.percentile(a, q, axis=None, interpolation='linear')`
- `pd.Series.quantile(q, interpolation='linear')`

Gdje je:

- `a` – Niz podataka.
- `q` – Kvantil (vrijednosti između 0 i 1 za `quantile` ili između 0 i 100 za `percentile`).
- `interpolation='linear'` – Određuje način na koji se izračunava kvantil kada položaj kvantila pada između dvije tačke u skupu podataka.

Primjer računanja kvantila u Python-u upotrebom modula NumPy i Pandas.

```
import numpy as np
import pandas as pd

# Skup podataka
data = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

# Kvartili koristeći NumPy
Q1_numpy = np.quantile(data, 0.25) # Prvi kvartil
```

```

Q2_numpy = np.quantile(data, 0.50) # Medijana
Q3_numpy = np.quantile(data, 0.75) # Treći kvartil

# Decili koristeći NumPy
D1_numpy = np.quantile(data, 0.10) # Prvi decil
D9_numpy = np.quantile(data, 0.90) # Deveti decils

# Percentili koristeći NumPy
P25_numpy = np.percentile(data, 25) # 25. percentil
P50_numpy = np.percentile(data, 50) # 50. percentil (medijana)
P75_numpy = np.percentile(data, 75) # 75. percentil

# Kvartili koristeći Pandas
data_series = pd.Series(data)
Q1_pandas = data_series.quantile(0.25)
Q2_pandas = data_series.quantile(0.50)
Q3_pandas = data_series.quantile(0.75)

# Prikaz rezultata
print("Kvartili (NumPy): Q1 =", Q1_numpy, ", Q2 =", Q2_numpy, ", Q3 =", Q3_numpy)
print("Decili (NumPy): D1 =", D1_numpy, ", D9 =", D9_numpy)
print("Percentili (NumPy): P25 =", P25_numpy, ", P50 =", P50_numpy, ", P75 =", P75_numpy)
print("Kvartili (Pandas): Q1 =", Q1_pandas, ", Q2 =", Q2_pandas, ", Q3 =", Q3_pandas)

Kvartili (NumPy): Q1 = 32.5 , Q2 = 55.0 , Q3 = 77.5
Decili (NumPy): D1 = 19.0 , D9 = 91.0
Percentili (NumPy): P25 = 32.5 , P50 = 55.0 , P75 = 77.5
Kvartili (Pandas): Q1 = 32.5 , Q2 = 55.0 , Q3 = 77.5

```

Objašnjenje:

Niz ulaznih podataka `data=[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]` je već sortiran, što je ključno za tačno izračunavanje kvantila. Kvantilna vrijednost računa se prema poziciji:

$$k = q \cdot (n-1)$$

Gdje je:

- q – Kvantil (npr. 0.25 za Q1),
- n – Broj elemenata ($n=10$).

Na primjer, za Q1 ($q=0.25$) vrijedi sljedeće: $k=0.25 \cdot (10-1)=0.25 \cdot 9=2.25$

Ova pozicija je između 3. i 4. elementa niza, pa se rezultat interpolira (podrazumijevano linearno) na vrijednost između 30 i 40.

6.1.4. Mjere centralne tendencije

Mjere centralne tendencije predstavljaju način da se opiše srednja vrijednost ili tipična vrijednost skupa podataka. One pomažu u razumijevanju kako su podaci grupisani i gdje se nalaze njihove ključne vrijednosti. Glavne mjere centralne tendencije su:

- Aritmetička sredina
- Geometrijska sredina
- Harmonijska sredina
- Medijana
- Modus

Ove mjere su osnova za statističku analizu i primjenjuju se u mnogim oblastima, uključujući ekonomiju, gdje se koriste za analizu prihoda, troškova, cijena i drugih ključnih pokazatelja.

6.1.4.1. Aritmetička sredina

Aritmetička sredina predstavlja mjeru centralne tendencije, koja se često koristi u statistici za predstavljanje srednje vrijednosti skupa podataka. Za razliku od pozicionih srednjih vrijednosti, aritmetička sredina je pouzdanija kada se koristi za kvantifikaciju centralne tendencije uz ispunjavanje određenih statističkih pretpostavki. Njena vrijednost obuhvata sve pojedinačne vrijednosti niza podataka i definiše se na osnovu sljedeće formule:

$$\bar{X} = \frac{\sum X_i}{N}$$

Gdje je:

- \bar{X} – Aritmetička sredina,
- X_i – Vrijednosti posmatranog niza podataka,
- N – Ukupan broj vrijednosti u posmatranom skupu.

Za ispravno računanje aritmetičke sredine neophodno je da podaci budu kvantitativne prirode, odnosno predstavljati numeričke vrijednosti mjerene na intervalnoj ili racionalnoj skali, jer samo takvi podaci omogućavaju sabiranje i dijeljenje potrebne za računanje srednje vrijednosti. Takođe, podaci treba da imaju normalnu distribuciju u posmatranom skupu, što osigurava reprezentativnost aritmetičke sredine. Konačno, u slučaju frekvencijskih raspodjela, svi intervali moraju biti zatvoreni i jednake širine, jer prisustvo otvorenih ili nejednakih intervala može narušiti tačnost računanje srednje vrijednosti.

Karakteristike aritmetičke sredine:

- Zbir odstupanja pojedinačnih vrijednosti od aritmetičke sredine uvijek je jednak nuli. Ovo osigurava da pozitivna i negativna odstupanja od sredine budu u ravnoteži.

$$\sum(X_i - \bar{X}) = 0$$

- Promjena bilo koje vrijednosti u skupu direktno utiče na vrijednost aritmetičke sredine, jer ona zavisi od svih pojedinačnih vrijednosti.
- Aritmetička sredina je podložna uticaju ekstremnih vrijednosti. U prisustvu izuzetno velikih ili malih vrijednosti, sredina može prestati da bude reprezentativna za većinu podataka u skupu.

Prednost aritmetička sredina se ogleda u činjenici da uzima u obzir sve vrijednosti unutar skupa podataka, što je čini reprezentativnom mjerom centralne tendencije za homogen skup, dok njeno jednostavno računanje i jedinstvenost omogućavaju primjenu u različitim analitičkim kontekstima. Ipak, postoje i određeni nedostaci, koji se javljaju u slučajevima prisustva ekstremnih vrijednosti, koje mogu značajno narušiti njenu reprezentativnost. Takođe, u slučajevima distribucija frekvencija sa otvorenim intervalima ili kod veoma velikih negrupsiranih skupova, postupak izračunavanja postaje složeniji i manje precizan.

Funkcije u Pythonu koje računaju aritmetičku sredinu:

- NumPy: `np.mean(niz)`
- Pandas: `pd.mean(niz)`

Primjer upotrebe funkcija za računanje aritmetičke sredine

```
import numpy as np
import pandas as pd

data = [10, 20, 30, 40]

mean_numpy = np.mean(data)          # Aritmetička sredina koristeći NumPy
mean_pandas = pd.Series(data).mean() # Aritmetička sredina koristeći Pandas

print("Aritmetička sredina (NumPy):", mean_numpy)
print("Aritmetička sredina (Pandas):", mean_pandas)
```

Aritmetička sredina (NumPy): 25.0
Aritmetička sredina (Pandas): 25.0

6.1.4.2. Geometrijska sredina

Geometrijska sredina je mjera centralne tendencije, koja se koristi za niz pozitivnih vrijednosti, gdje su važni odnosi i stope promjene između elemenata skupa podataka.

Izračunava se kao n-ti korijen proizvoda svih vrijednosti u skupu. Matematički, geometrijska sredina se izražava formulom:

$$G = \sqrt[n]{X_1 \cdot X_2 \cdot \dots \cdot X_n}$$

Gdje je:

- G – Geometrijska sredina,
- X_1, X_2, \dots, X_n – Vrijednosti u skupu podataka,
- n – Broj elemenata u skupu.

Geometrijska sredina je neophodna mjera u ekonomiji, kada se analiziraju proporcije, stope rasta, ili relativne promjene. Njena primjena u izračunavanju stope rasta, inflacije, prinosa i efikasnosti čini je osnovnim alatom za donošenje značajnih ekonomskih odluka.

Za pravilnu primjenu geometrijske sredine, neophodno je da svi elementi u skupu podataka budu pozitivni, budući da geometrijska sredina nije definisana za negativne brojeve ili nulu. Ova mjera centralne tendencije koristi se u situacijama gdje su stope promjene ili proporcionalni odnosi ključni za analizu, čineći je posebno pogodnom za skupove podataka sa multiplikativnim karakterom, gdje odnosi između vrijednosti igraju važniju ulogu od njihovih apsolutnih iznosa.

Karakteristike geometrijske sredine

- Za razliku od aritmetičke sredine, geometrijska sredina ublažava uticaj izuzetno velikih ili malih vrijednosti.
- Idealna je za podatke koji predstavljaju proporcionalne promjene, kao što su godišnje stope rasta.
- Geometrijska sredina se može analizirati koristeći logaritme, jer je proizvod elemenata ekvivalentan sumi logaritama. To znači da je suma razlika $\log(X_i)$ i $\log(G)$ uvijek jednaka nuli, što je ključna karakteristika geometrijske sredine.

$$\sum_{i=1}^n (\log(X_i) - \log(G)) = 0$$

Funkcije u Python-u za računanje geometrijske sredine:

- SciPy: `scipy.stats.gmean(niz)`
- NumPy: `np.prod(niz) ** (1 / len(niz))`, alternativno računanje

Primjer računanja geometrijske sredine u Python-u.

```
import numpy as np
from scipy.stats import gmean
```

```

# Podaci
data = [4, 8, 16]

gmean_scipy = gmean(data) # koristeći SciPy
gmean_numpy = np.prod(data) ** (1 / len(data)) # koristeći NumPy

# Razlike (logaritamski pristup za analizu odstupanja)
log_values = np.log(data) # Logaritmi pojedinačnih vrijednosti
log_mean = np.log(gmean_scipy) # Logaritamska vrijednost geometrijske sredine
log_differences = log_values - log_mean # Razlike između logaritama
sum_of_log_differences = np.sum(log_differences) # Suma logaritamskih razlika

# Prikaz rezultata sa zaokruživanjem
print(f"Geometrijska sredina (SciPy): {gmean_scipy:.2f}")
print(f"Geometrijska sredina (NumPy): {gmean_numpy:.2f}")
print(f"Logaritamske vrijednosti (log(Xi)): {log_values}")
print(f"Logaritamska sredina (log(G)): {log_mean:.2f}")
print(f"Logaritamske razlike (log(Xi) - log(G)): {log_differences}")
print(f"Suma logaritamskih razlika: {sum_of_log_differences:.2f}")

```

```

Geometrijska sredina (SciPy): 8.00
Geometrijska sredina (NumPy): 8.00
Logaritamske vrijednosti (log(Xi)): [1.38629436 2.07944154 2.77258872]
Logaritamska sredina (log(G)): 2.08
Logaritamske razlike (log(Xi) - log(G)): [-0.69314718  0.          0.69314718]
Suma logaritamskih razlika: 0.00

```

6.1.4.3. Harmonijska sredina

Harmonijska sredina je mjera centralne tendencije, koja se koristi kada su podaci izraženi kao inverzne vrijednosti, poput brzine, cijene po jedinici ili efikasnosti. Matematički, harmonijska sredina se definije kao recipročna vrijednost prosjeka recipročnih vrijednosti elemenata skupa podataka.

Formula za harmonijsku sredinu:

$$H = \frac{n}{\sum_{i=1}^n \frac{1}{X_i}}$$

Gdje je:

- H – Harmonijska sredina,
- X_i – Vrijednosti u skupu podataka,
- n – Ukupan broj elemenata u skupu.

Uslov za primjenu harmonijske sredine je da svi elementi u skupu moraju biti pozitivni, jer harmonijska sredina nije definisana za nulu ili negativne vrijednosti.

Karakteristike harmonijske sredine:

- Harmonijska sredina daje veću težinu manjim vrijednostima u skupu, čineći je korisnom za analize gdje su manje vrijednosti značajnije.
- Idealna je za analize brzine, cijene po jedinici ili drugih slučajeva gdje je važan recipročni odnos.
- Prisustvo negativnih ili nultih vrijednosti onemogućava izračunavanje harmonijske sredine.

Funkcija u Pythonu koja direktno račun geometrijsku sredinu:

- SciPy: `sc.stats.hmean(a, axis=0, dtype=None)`

Gdje je:

- `a` – Lista pozitivnih vrijednosti za koje se računa harmonijska sredina.
- `axis` – Osa duž koje se računa sredina. Podrazumijevano je `axis=0` (računanje duž prve ose).
- `dtype` – Tip podataka koji se koristi pri računanu (opciono).

Pretpostavimo primjer u kome imamo različite količine proizvoda po različitim cijenama (paket proizvoda sa različitim masama i cijenama po kilogramu). U nastavku je Python skripta, koja računa prosječnu cijenu po jedinici na bazi harmonijske sredine.

```
import scipy.stats as sc

cijena_po_kg = [10, 15, 20] # Cijene po kilogramu (KM/kg)
kolicine = [5, 10, 15]      # Količine u kilogramima

# Ukupna cijena za svaki paket
Ukupna_cijena = [price * quantity for price, quantity in zip(cijena_po_kg,
kolicine)]

prosjecna_cijena = sc.hmean(cijena_po_kg) # Harmonijska sredina cijena po kilogramu

# Prikaz rezultata
print(f"Cijene po kilogramu (€/kg): {cijena_po_kg}")
print(f"Količine (kg): {kolicine}")
print(f"Ukupne cijene (€): {Ukupna_cijena}")
print(f"Harmonijska sredina cijena po kg (€/kg): {prosjecna_cijena:.2f}")

Cijene po kilogramu (€/kg): [10, 15, 20]
Količine (kg): [5, 10, 15]
Ukupne cijene (€): [50, 150, 300]
Harmonijska sredina cijena po kg (€/kg): 13.85
```

Objašnjenje:

- `zip(cijena_po_kg, kolicine)`. Kombinuje dvije liste (`cijena_po_kg` i `kolicine`) u parove elemenata. Ovo znači da svaki element iz prve liste

(`cijena_po_kg`) bude uparen s odgovarajućim elementom iz druge liste (`kolicine`). Primjer: [(10, 5), (15, 10), (20, 15)]

- `price * quantity for price, quantity in zip(...)`. Ovo generiše ukupnu cijenu za svaki paket množenjem cijene po kilogramu (`price`) i količine (`quantity`). Ovaj je informativni podataka u ovom primjeru.
- `sc.hmean(cijena_po_kg)`. Računa harmonijsku sredinu cijena.

6.1.4.4. Medijana

Medijana je mjera centralne tendencije, koja predstavlja srednju vrijednost skupa podataka kada su podaci poredani po veličini. Ona dijeli skup podataka na dva jednakih dijela: 50% vrijednosti je manje ili jednako medijani, dok je preostalih 50% vrijednosti veće ili jednako medijani.

Medijana je posebno korisna kada skup podataka sadrži ekstremne vrijednosti, jer je manje osjetljiva na njih u poređenju s aritmetičkom sredinom.

Medijana se određuje na osnovu broja elemenata u skupu:

- **Ako je broj elemenata neparan (n):**

$$M = X_{\frac{n+1}{2}}$$

- **Ako je broj elemenata paran (n):**

$$M = \frac{X_{\frac{n}{2}} + X_{\frac{n}{2}+1}}{2}$$

Medijana je aritmetička sredina dvije srednje vrijednosti.

Gdje je:

- M – Srednja vrijednost (medijana),
- X_i – Vrijednosti u skupu podataka poredane po veličini.
- n – Ukupan broj elemenata u skupu.

Za pravilnu primjenu medijane, podaci moraju biti kvantitativni ili ordinalni, što znači da se medijana može izračunati za numeričke ili rangirane podatke, dok nije primjenjiva na nominalne kategorije. Takođe, podaci moraju biti poredani u rastućem ili opadajućem redoslijedu, kako bi se ispravno odredila njihova srednja vrijednost.

Karakteristike medijane:

- Medijana nije osjetljiva na ekstremne vrijednosti, što je čini idealnom za skupove sa ekstremnim vrijednostima.

- Medijana je poziciona mjera, jer zavisi od redoslijeda vrijednosti, a ne od njihovih apsolutnih veličina.
- Medijana bolje opisuje centralnu vrijednost nesimetričnih raspodjela nego aritmetička sredina.

Funkcije u Pythonu:

- NumPy: `np.median(a, axis=None, out=None)`

Gdje je:

- `a` – Skup podataka (array ili lista).
- `axis` – Osa duž koje se računa medijana (opciono).
- `out` – Niz u kojem će se sačuvati rezultat (opciono).
- Pandas: `pd.Series.median(axis=0, skipna=True)`

Gdje je:

- `axis` – Osa (`0` za kolone, `1` za redove, ako se koristi DataFrame).
- `skipna` – Preskakanje `NaN`, bez unešene vrijednosti za elemenat niza (podrazumijevano je `True`).

Prepostavimo da imamo podatke o prihodima domaćinstava u KM, gdje ekstremne vrijednosti značajno utiču na aritmetičku sredinu. U nastavku je primjer urađen u Pythonu.

```
import numpy as np
import pandas as pd

# Podaci o prihodima domaćinstava (u KM)
prihodi = [1200, 1500, 1800, 3000, 8000] # Ekstremna vrijednost: 8000

# Medijana koristeći NumPy
median_numpy = np.median(prihodi)

# Medijana koristeći Pandas
median_pandas = pd.Series(prihodi).median()
# Prikaz rezultata
print("Prihodi domaćinstava (u eurima):", prihodi)
print(f"Medijana (NumPy): {median_numpy}")
print(f"Medijana (Pandas): {median_pandas}")

Prihodi domaćinstava (u eurima): [1200, 1500, 1800, 3000, 8000]
Medijana (NumPy): 1800.0
Medijana (Pandas): 1800.0
```

6.1.4.5. Modus

Modus je mjera centralne tendencije, koja predstavlja najčešće ponavljaju vrijednost u skupu podataka. Modus je posebno koristan za kvalitativne podatke, gdje opisuje kategoriju s najvećom frekvencijom, ali se primjenjuje i na kvantitativne podatke.

Za razliku od aritmetičke ili medijane, skup podataka može imati:

- **Jedan modus** (unimodalni skup),
- **Više modusa** (multimodalni skup),
- **Bez modusa** (skup gdje se nijedna vrijednost ne ponavlja).

Ako serija podataka sadrži samo jednu vrijednost obilježja s najvećom frekvencijom, tada je riječ o unimodalnoj seriji. U slučaju da postoje dvije ili više vrijednosti s jednakom maksimalnom frekvencijom, serija se naziva bimodalnom ili multimodalnom.

Modus ne može biti određen kada sve vrijednosti obilježja imaju međusobno jednake frekvencije. Za serije podataka s atributivnim obilježjima, modus se određuje na isti način kao i za numerička obilježja. Na veličinu modusa ne utiču promjene frekvencija sve dok najveća frekvencija ostaje nepromijenjena. Ipak, kod malih uzoraka modus nije uvijek najpogodnija mjera centralne tendencije.

Karakteristike modusa

- Jednostavnost: Modus je lako razumjeti i izračunati, posebno za kategoriskske podatke.
- Primjena na distribuciju frekvencija: Pogodan je za skupove podataka gdje su vrijednosti grupisane u intervale.
- Nije jedinstven: Skup podataka može imati jedan modus, više njih ili nijedan.
- Neosjetljiv na ekstremne vrijednosti: Za razliku od aritmetičke sredine, modus nije pod uticajem ekstremnih vrijednosti.

Funkcije u Pythonu:

- SciPy: `sc.stats.mode(a, axis=0, nan_policy='propagate', keepdims=False)`

Gdje je:

- `a` – Skup podataka (array ili lista).
- `axis` – Osa duž koje se računa modus.
- `nan_policy` – Postupanje sa NaN vrijednostima ('`propagate`', '`om`')
- `pdims=True` – Osigurava da `mode()` vraća rezultat kao niz
- Pandas: `pd.Series.mode(dropna=True)`

Gdje je:

- **dropna** – Da li treba isključiti **NaN** vrijednosti (podrazumijevano **True**).

U sljedećem primjeru pretpostavimo da imamo podatke o najčešće prodavanim veličinama proizvoda u kilogramima.

```
import numpy as np
import pandas as pd
from scipy.stats import mode

# Podaci o veličinama proizvoda u kilogramima
sizes = [1, 2, 2, 3, 3, 3, 4, 5]

mode_scipy = mode(sizes, keepdims=True) # Modus koristeći SciPy
mode_pandas = pd.Series(sizes).mode() # Modus koristeći Pandas

# Prikaz rezultata
print("Veličine proizvoda (kg):", sizes)
print(f"Modus (SciPy): Vrijednost = {mode_scipy.mode[0]}, Frekvencija = {mode_scipy.count[0]}")
print(f"Modus (Pandas): Vrijednost(i) = {mode_pandas.tolist()}")
```

```
Veličine proizvoda (kg): [1, 2, 2, 3, 3, 3, 4, 5]
Modus (SciPy): Vrijednost = 3, Frekvencija = 3
Modus (Pandas): Vrijednost(i) = [3]
```

Tabela 6.1. Uporedni pregled funkcija po modulima

Mjera	NumPy	Pandas	SciPy
Aritmetička sredina	<code>numpy.mean()</code>	<code>Series.mean()</code>	Oslanja se na NumPy
Geometrijska sredina	-	-	<code>scipy.stats.gmean()</code>
Harmonijska sredina	-	-	<code>scipy.stats.hmean()</code>
Medijana	<code>numpy.median()</code>	<code>Series.median()</code>	Oslanja se na NumPy
Modus	-	<code>Series.mode()</code>	<code>scipy.stats.mode()</code>

6.1.5. Mjere rasipanja (disperzije)

Mjere disperzije kvantifikuju rasipanje ili varijabilnost podataka u odnosu na centralnu vrijednost. Dok mjere centralne tendencije opisuju "središte" podataka, mjere disperzije pružaju uvid u to koliko su podaci homogeni ili heterogeni.

Mjere disperzije su ključne za razumijevanje pouzdanosti i konzistentnosti podataka:

- Omogućavaju preciznu procjenu odstupanja vrijednosti u skupu podataka.
- Pružaju informacije o volatilnosti ili stabilnosti ekonomskih pokazatelja.
- Koriste se za procjenu rizika u investicijama, analizu troškova i prihoda, te predviđanje tržišnih trendova.

Mjere centralne tendencije i mjere disperzije zajedno pružaju kompletну sliku o podacima. Na primjer aritmetička sredina može biti ista za dva skupa podataka, ali ako su mjere disperzije različite, distribucija podataka je potpuno drugačija. Disperzija osigurava kontekst za interpretaciju srednje vrijednosti.

Mjere disperzije su neophodne za analizu ekonomskih varijabli kao što su:

- Fluktuacije cijena na tržištu.
- Procjena rizika investicija i stabilnosti prihoda.
- Analiza volatilnosti ekonomskih pokazatelja, poput inflacije i nezaposlenosti.

6.1.5.1. Apsolutne mjere disperzije

Apsolutne mjere disperzije kvantifikuju odstupanja vrijednosti u skupu podataka u odnosu na centralnu vrijednost ili međusobno odstupanje podataka. Ove mjere izražavaju apsolutne vrijednosti rasipanja podataka i pružaju osnovu za procjenu varijabilnosti unutar skupa.

U apsolutne mjere disperzije spadaju:

- **Razmak (Range)** – Razlika između najveće i najmanje vrijednosti u skupu podataka.
- **Interkvartalna razlika (Interquartile Range)** – Razlika između trećeg kvartala (Q3) i prvog kvartala (Q1), koja obuhvata srednjih 50% podataka.
- **Varijansa (Variance)** – Prosječna kvadratna odstupanja vrijednosti od aritmetičke sredine.
- **Standardna devijacija (Standard Deviation)** – Kvadratni korijen varijanse, koji se koristi za procjenu prosječnog odstupanja vrijednosti.

Razmak (Range)

Razmak (range) predstavlja najjednostavniju mjeru disperzije, koja se definiše kao razlika između najveće i najmanje vrijednosti u skupu podataka. Matematički se izražava formulom:

$$R = X_{\max} - X_{\min}$$

Gdje je:

- R – Razmak (range),
- X_{\max} – Najveća vrijednost u skupu,
- X_{\min} – Najmanja vrijednost u skupu.

Prednosti razmaka leže u njegovoj jednostavnosti, jer se lako razumije i izračunava kao razlika između najveće i najmanje vrijednosti. Pruža brz uvid u raspon podataka i efikasan je za preliminarnu analizu varijabilnosti.

Nedostaci razmaka uključuju osjetljivost na ekstremne vrijednosti, zbog čega može iskriviti sliku rezultata. Takođe, ne pruža informacije o raspodjeli podataka između minimuma i maksimuma i nije pogodan za asimetrične distribucije.

Primjena razmaka u ekonomiji uključuje analizu raspona cijena proizvoda, razlike između najmanjih i najvećih plata, te minimalne i maksimalne vrijednosti deviznih kurseva u određenom periodu, što pomaže u brzoj procjeni tržišnih uslova.

Primjer za izračunavanje razmaka.

```
import numpy as np
import pandas as pd

# Skup podataka: Cijene proizvoda u eurima
prices = [10, 15, 20, 25, 40]

range_numpy = np.max(prices) - np.min(prices) # NumPy
range_pandas = pd.Series(prices).max() - pd.Series(prices).min() # Pandas

# Prikaz rezultata
print("Cijene proizvoda (EUR):", prices)
print(f"Razmak (NumPy): {range_numpy}")
print(f"Razmak (Pandas): {range_pandas}")

Cijene proizvoda (EUR): [10, 15, 20, 25, 40]
Razmak (NumPy): 30
Razmak (Pandas): 30
```

Interkvartilni razmak

Interkvartilni razmak (Interquartile Range - IQR) je mjera disperzije koja predstavlja razliku između trećeg kvartila (Q_3) i prvog kvartila (Q_1). Ova mjeru obuhvata srednjih 50% podataka i pruža uvid u varijabilnost podataka eliminiranjem uticaja ekstremnih vrijednosti.

Formula za interkvartilni razmak:

$$IQR = Q_3 - Q_1$$

Gdje je:

- Q_1 – Prvi kvartil, odnosno medijana donje polovine podataka.
- Q_3 – Treći kvartil, odnosno medijana gornje polovine podataka.
- IQR – Razlika između Q_3 i Q_1 , koja mjeri širinu interkvartilnog opsega.

Prednosti interkvartilnog razmaka su njegova otpornost na ekstremne vrijednosti, jer analizira samo srednjih 50% podataka, te sposobnost da realističnije opiše varijabilnost u asimetričnim raspodjelama. Njegova interpretacija je jednostavna, jer jasno prikazuje širinu opsega u kojem se nalazi polovina vrijednosti.

Nedostaci interkvartilnog razmaka uključuju ignorisanje krajnjih vrijednosti, čime se mogu izgubiti značajne informacije. Kod manjih uzoraka, kvartili su manje precizni, a za izračunavanje Q_1 i Q_3 podaci moraju biti sortirani, što povećava složenost analize kod većih skupova.

Interkvartilni razmak je široko primjenjiv u ekonomiji za analize, koje zahtijevaju eliminaciju uticaja ekstremnih vrijednosti, kao što su:

- Analiza prihoda: Procjena varijabilnosti u prihodima različitih domaćinstava ili sektora.
- Procjena cijena: Analiza cijena proizvoda ili usluga na tržištu, fokusirajući se na srednju vrijednost distribucije.
- Identifikacija ekstremnih vrijednosti: Korištenje IQR-a za prepoznavanje i eliminaciju izuzetno niskih ili visokih vrijednosti u podacima.

Python kod za izračunavanje interkvartilnog razmaka.

```
import numpy as np
import pandas as pd

# Skup podataka: Prihodi domaćinstava u hiljadama eura
incomes = [20, 25, 28, 30, 35, 40, 45, 50, 55, 100]

# Izračunavanje kvartila koristeći NumPy
Q1_numpy = np.percentile(incomes, 25) # Prvi kvartil (25. percentil)
Q3_numpy = np.percentile(incomes, 75) # Treći kvartil (75. percentil)
IQR_numpy = Q3_numpy - Q1_numpy
```

```

# Izračunavanje kvartila koristeći Pandas
incomes_series = pd.Series(incomes)
Q1_pandas = incomes_series.quantile(0.25) # Prvi kvartil
Q3_pandas = incomes_series.quantile(0.75) # Treći kvartil
IQR_pandas = Q3_pandas - Q1_pandas

# Prikaz rezultata
print("Prihodi domaćinstava (hiljade EUR):", incomes)
print(f"Interkvartilni razmak (NumPy): {IQR_numpy}")
print(f"Interkvartilni razmak (Pandas): {IQR_pandas}")

```

Prihodi domaćinstava (hiljade EUR): [20, 25, 28, 30, 35, 40, 45, 50, 55, 100]
 Interkvartilni razmak (NumPy): 20.25
 Interkvartilni razmak (Pandas): 20.25

Varijansa i standardna devijacija

Varijansa i standardna devijacija su mjere rasipanja, koje opisuju kako su podaci u skupu raspoređeni u odnosu na njihovu srednju vrijednost. Varijansa mjeri prosječnu kvadratnu udaljenost podataka od aritmetičke sredine, dok standardna devijacija predstavlja kvadratni korijen varijanse. Glavna razlika je u jedinicama: varijansa je izražena u kvadratu jedinica originalnih podataka, dok standardna devijacija zadržava iste jedinice kao i podaci, što je čini intuitivnijom za interpretaciju.

Formule za varijansu i standardnu devijaciju razlikuju se ovisno o tome da li radimo s populacijom ili uzorkom.

Varijansa

- Za populaciju:

$$\sigma^2 = \frac{\sum_{i=1}^N (X_i - \mu)^2}{N}$$

- Za uzorak:

$$s^2 = \frac{\sum_{i=1}^n (X_i - \bar{X})^2}{n-1}$$

Standardna devijacija

- Za populaciju:

$$\sigma = \sqrt{\sigma^2}$$

- Za uzorak:

$$s = \sqrt{s^2}$$

Gdje je:

- X_i – Pojedinačne vrijednosti u skupu podataka,
- μ – Aritmetička sredina populacije,
- \bar{X} – Aritmetička sredina uzorka,
- N – Veličina populacije,
- n – Veličina uzorka.

Kako bi varijansa i standardna devijacija bile smislene, moraju biti ispunjeni sljedeći uslovi:

- Kvantitativni podaci: Ove mjere primjenjuju se na numeričke podatke.
- Intervalna ili racio skala: Podaci moraju biti mjerljivi na intervalnoj ili racio skali.
- Normalna distribucija: Iako nije strogo neophodno, ove mjere su najrelevantnije za podatke s približno normalnom distribucijom.

Varijansa i standardna devijacija imaju široku primjenu u različitim područjima, uključujući ekonomiju. Neke od ključnih primjena su:

- Procjenu rizika i varijabilnosti u finansijama (npr. fluktuacija cijena akcija).
- Analizu stabilnosti sistema u proizvodnji.
- Mjerenje odstupanja od prosjeka u ekonomskim pokazateljima, poput prihoda ili inflacije.

NumPy i Pandas funkcije:

NumPy:

- Varijansa: `np.var(a, ddof=0)` za populaciju, `np.var(a, ddof=1)` za uzorak.
- Standardna devijacija: `np.std(a, ddof=0)` za populaciju, `np.std(a, ddof=1)` za uzorak.

Pandas:

- Varijansa: `pd.Series.var(ddof=1)` (uzorak), odnosno `ddof=0` za populaciju.
- Standardna devijacija: `pd.Series.std(ddof=1)` (uzorak), odnosno `ddof=0` za populaciju.

Primjer izračunavanja varijanse i standardne devijacije.

```
import numpy as np
import pandas as pd

# Skup podataka
data = [10, 20, 30, 40, 50]

# Varijansa i standardna devijacija koristeći NumPy
var_population = np.var(data) # Populacija (ddof=0 - podrazumijevano)
std_population = np.std(data) # Populacija(ddof=0 - podrazumijevano)

var_sample = np.var(data, ddof=1) # Uzorak
std_sample = np.std(data, ddof=1) # Uzorak

# Varijansa i standardna devijacija koristeći Pandas
data_series = pd.Series(data)

var_pandas_population = data_series.var(ddof=0) # Populacija (ddof=0)
std_pandas_population = data_series.std(ddof=0) # Populacija (ddof=0)

var_pandas_sample = data_series.var() # Uzorak (ddof=1 - podrazumijevano)
std_pandas_sample = data_series.std() # Uzorak (ddof=1 - podrazumijevano)

# Prikaz rezultata
print("NumPy:")
print(f"Varijansa (populacija - NumPy): {var_population}")
print(f"Standardna devijacija (populacija - NumPy): {std_population}")
print(f"Varijansa (uzorak - NumPy): {var_sample}")
print(f"Standardna devijacija (uzorak - NumPy): {std_sample}")

print("\nPandas:")
print(f"Varijansa (populacija - Pandas): {var_pandas_population}")
print(f"Standardna devijacija (populacija - Pandas): {std_pandas_population}")
print(f"Varijansa (uzorak - Pandas): {var_pandas_sample}")
print(f"Standardna devijacija (uzorak - Pandas): {std_pandas_sample}")
```

```
NumPy:
Varijansa (populacija - NumPy): 200.0
Standardna devijacija (populacija - NumPy): 14.142135623730951
Varijansa (uzorak - NumPy): 250.0
Standardna devijacija (uzorak - NumPy): 15.811388300841896
```

```
Pandas:
Varijansa (populacija - Pandas): 200.0
Standardna devijacija (populacija - Pandas): 14.142135623730951
Varijansa (uzorak - Pandas): 250.0
Standardna devijacija (uzorak - Pandas): 15.811388300841896
```

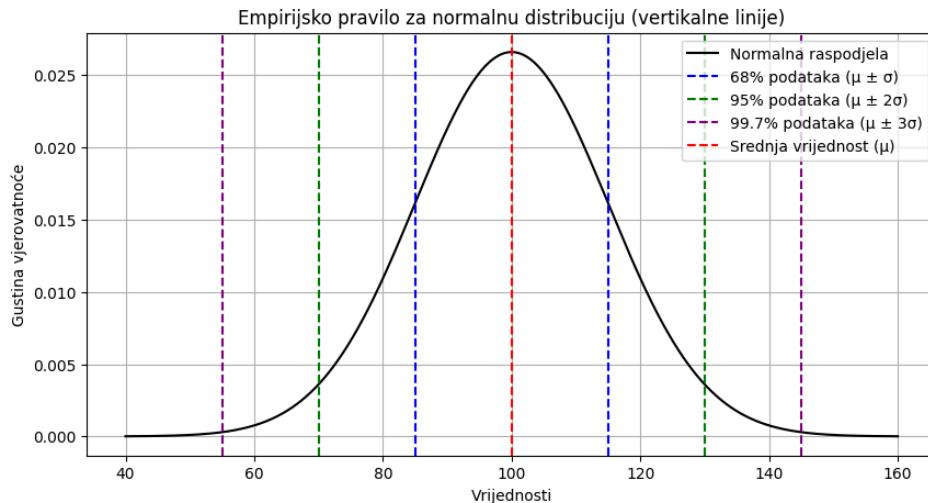
6.1.5.2. Empirijsko pravilo i Čebišovljeva teorema

Empirijsko pravilo i Čebišovljeva teorema predstavljaju dva važna koncepta u statistici, koja opisuju raspodjelu podataka u odnosu na prosječnu vrijednost i standardnu devijaciju. Dok se empirijsko pravilo primjenjuje isključivo na podatke s normalnom raspodjelom. Čebišovljeva teorema pruža širu primjenu, obuhvatajući i simetrične i asimetrične distribucije.

Empirijsko pravilo opisuje kako su podaci raspoređeni u normalnoj distribuciji. Prema ovom pravilu:

- **68%** podataka se nalazi unutar jedne standardne devijacije ($\mu \pm \sigma$) od aritmetičke sredine.
- **95%** podataka se nalazi unutar dvije standardne devijacije ($\mu \pm 2\sigma$).
- **99.7%** podataka se nalazi unutar tri standardne devijacije ($\mu \pm 3\sigma$).

Sljedeća slika prikazuje raspodjelu podataka po empirijskom pravilu.



Slika 6.1. Raspodjela podataka po empirijskom pravilu, Izvor: Autori.

Normalnu raspodjelu primjenjujemo u sljedećim slučajevima:

- Predviđanje vjerovatnoće pojavljivanja vrijednosti unutar određenog opsega.
- Analiza odstupanja od prosječne vrijednosti u populacijama s normalnom distribucijom.
- Primjena u ekonomiji za procjenu varijabilnosti cijena, zarada ili troškova.

Čebišovljeva teorema daje minimalni procenat podataka, koji se nalaze unutar određenog broja standardnih devijacija od srednje vrijednosti.

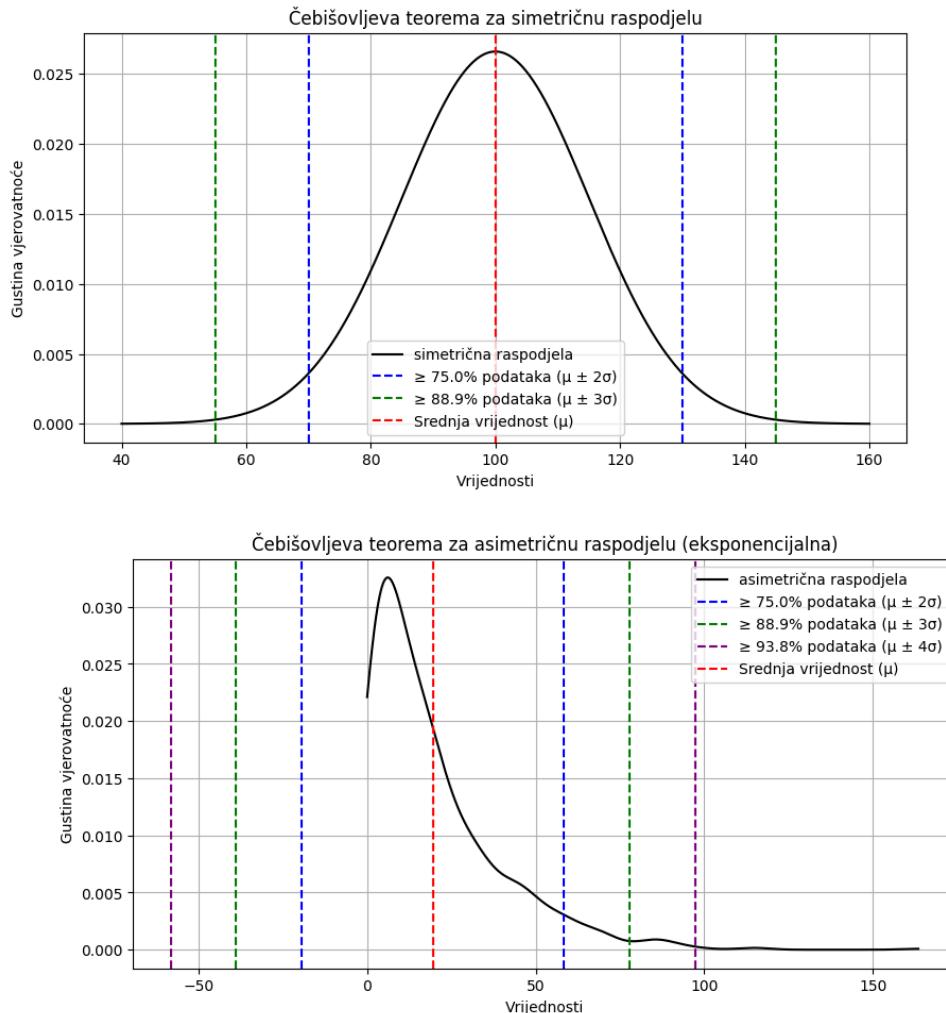
Po ovoj teoremi za bilo koju distribuciju, bez obzira na njen oblik vrijedi sljedeće:

$$P(|X - \mu| \geq k\sigma) \leq \frac{1}{k^2}$$

Gdje je:

- P – Vjerovatnoća,
- μ – Srednja vrijednost,

- σ – Standardna devijacija,
- k – Broj standardnih devijacija od srednje vrijednosti ($k > 1$).



Slika 6.2. Raspodjela podataka po Čebišovljevoj teoremi, Izvor: Autori.

Primjena Čebišovljeve teoreme u nesimetričnim distribucijama:

- Koristi se za analizu podataka u distribucijama s dugim repovima, kao što su prihodi stanovništva ili tržišna volatilnost.
- Pomaže u procjeni vjerovatnoće kada podaci značajno odstupaju od srednje vrijednosti, bez obzira na oblik raspodjele.
- Pronalazi primjenu u ekonomiji pri mjerjenju varijabilnosti u nepovoljnim uslovima, gdje raspodjela podataka nije normalna, već asimetrična ili ekstremo rasuta.

Razlika između Čebišovljeve teoreme i empirijskog pravila. Čebišovljeva teorema je univerzalnija jer se može primijeniti na bilo koju raspodjelu podataka, dok se empirijsko pravilo isključivo odnosi na normalne distribucije. Empirijsko pravilo omogućava preciznije procjene u slučajevima kada su podaci normalno raspoređeni, dok je Čebišovljeva teorema konzervativnija i garantuje samo minimalni procenat podataka unutar određenog broja standardnih devijacija.

Primjer primjene Empirijskog pravila - Analiza plata zaposlenih

U jednoj kompaniji sprovedeno je istraživanje o platama zaposlenih. Prepostavlja se da su plate normalno distribuirane sa prosječnom platom od 3000 KM i standardnom devijacijom od 500 KM.

Cilj analize je da se pomoću empirijskog pravila (pravilo 68-95-99.7) odredi interval plata u kojem se nalazi većina zaposlenih.

Zadatak:

- 1) Generisati 1000 nasumičnih plata koristeći normalnu distribuciju sa zadatim parametrima.
- 2) Ispisati prvih 10 plata iz generisanog skupa podataka.
- 3) Na osnovu empirijskog pravila izračunati intervale plata u kojima se nalazi:
 - a) 68% zaposlenih ($\mu \pm \sigma$)
 - b) 95% zaposlenih ($\mu \pm 2\sigma$)
 - c) 99.7% zaposlenih ($\mu \pm 3\sigma$)
- 4) Prikazati rezultate.

```
# PRIMJER 1: EMPIRIJSKO PRAVILO (NORMALNA DISTRIBUCIJA) - PLATE ZAPOSLENIH
import numpy as np

# Parametri distribucije (u KM)
mu_salary = 3000 # Prosječna plata u KM
sigma_salary = 500 # Standardna devijacija

# Generisanje nasumičnog skupa podataka za plate (normalna distribucija)
np.random.seed(42)
data_salary = np.random.normal(loc=mu_salary, scale=sigma_salary, size=1000)

# Intervali prema empirijskom pravilu
range_68 = (mu_salary - sigma_salary, mu_salary + sigma_salary)
range_95 = (mu_salary - 2 * sigma_salary, mu_salary + 2 * sigma_salary)
range_997 = (mu_salary - 3 * sigma_salary, mu_salary + 3 * sigma_salary)

# Prikaz rezultata
print("EMPIRIJSKO PRAVILO - PLATE ZAPOSLENIH")
print("Uzorak podataka (prvih 10 plata):", np.round(data_salary[:10], 2))
print(f"68% zaposlenih prima platu između {range_68[0]:,.2f} KM i {range_68[1]:,.2f} KM")
print(f"95% zaposlenih prima platu između {range_95[0]:,.2f} KM i {range_95[1]:,.2f} KM")
print(f"99.7% zaposlenih prima platu između {range_997[0]:,.2f} KM i {range_997[1]:,.2f} KM")
```

```

EMPIRIJSKO PRAVILO - PLATE ZAPOSLENIH
Uzorak podataka (prvih 10 plata): [3248.36 2930.87 3323.84 3761.51 2882.92 2882.93
3789.61 3383.72 2765.26
3271.28]
68% zaposlenih prima platu između 2,500.00 KM i 3,500.00 KM
95% zaposlenih prima platu između 2,000.00 KM i 4,000.00 KM
99.7% zaposlenih prima platu između 1,500.00 KM i 4,500.00 KM

```

Primjer primjene Čebišovljeve teoreme - Analiza prihoda preduzetnika

U analizi preduzetničkih prihoda utvrđeno je da raspodjela prihoda nije normalna, već da ima asimetrični raspored. U ovom slučaju, empirijsko pravilo se ne može primijeniti, pa se koristi Čebišovljeva teorema.

Cilj analize je da se utvrdi minimalan procenat preduzetnika čiji se prihodi nalaze unutar određenog broja standardnih devijacija.

Zadatak:

- 1) Generisati 1000 nasumičnih prihoda koristeći eksponencijalnu raspodjelu sa prosječnim prihodom oko 3000 KM.
- 2) Ispisati prvih 10 prihoda iz generisanog skupa podataka.
- 3) Izračunati srednju vrijednost i standardnu devijaciju generisanih prihoda.
- 4) Na osnovu Čebišovljeve teoreme izračunati intervale prihoda u kojima se nalazi najmanje:
 - a) 75% preduzetnika ($\mu \pm 2\sigma$)
 - b) 88.9% preduzetnika ($\mu \pm 3\sigma$)
 - c) 93.75% preduzetnika ($\mu \pm 4\sigma$)
- 5) Prikazati rezultate.

```

# PRIMJER 2: ČEBIŠOVLJEVA TEOREMA (ASIMETRIČNA DISTRIBUCIJA) - PRIHODI PREDUZETNIKA
import numpy as np

# Generisanje podataka za eksponencijalnu raspodjelu (asimetrična)
np.random.seed(42)
data_income = np.random.exponential(scale=3000, size=1000) # Skala određuje
prosječni prihod

# Računanje srednje vrijednosti i standardne devijacije
mu_income = np.mean(data_income)
sigma_income = np.std(data_income)

# Čebišovljeve granice za različite standardne devijacije
k_values = [2, 3, 4] # Standardne devijacije
chebyshev_ranges = [(mu_income - k * sigma_income, mu_income + k * sigma_income) for
k in k_values]
chebyshev_percentages = [1 - 1/k**2 for k in k_values]

# Prikaz rezultata
print("\nČEBIŠOVLJEVA TEOREMA - PRIHODI PREDUZETNIKA")
print("Uzorak podataka (prvih 10 prihoda):", np.round(data_income[:10], 2))
for i, k in enumerate(k_values):
    print(f"Najmanje {chebyshev_percentages[i]}% preduzetnika ima prihod"

```

```
između "
f'{chebyshev_ranges[i][0]:,.2f} KM i {chebyshev_ranges[i][1]:,.2f} KM")
```

ČEBIŠOVLJEVA TEOREMA - PRIHODI PREDUZETNIKA

Uzorak podataka (prvih 10 prihoda): [1407.8 9030.36 3950.24 2738.83 508.87 508.79
179.52 6033.69 2757.25
3693.75]

Najmanje 75.0% preduzetnika ima prihod između -2,914.62 KM i 8,749.66 KM

Najmanje 88.9% preduzetnika ima prihod između -5,830.69 KM i 11,665.73 KM

Najmanje 93.8% preduzetnika ima prihod između -8,746.76 KM i 14,581.80 KM

6.1.5.3. Relativne mjere disperzije

Relativne mjere disperzije koriste se kada želimo da uporedimo varijabilnost između dva ili više skupova podataka, koji imaju različite jedinice mjere ili različite prosječne vrijednosti.

Za razliku od apsolutnih mjera disperzije (npr. standardna devijacija), relativne mjere daju odnos varijabilnosti u odnosu na srednju vrijednost, omogućavajući direktno poređenje između različitih skupova podataka.

Dvije najvažnije relativne mjere disperzije su:

1. Koeficijent varijacije (CV) – izražava standardnu devijaciju kao procenat srednje vrijednosti.
2. Relativni razmak (Range Ratio) – opisuje odnos između maksimalne i minimalne vrijednosti u skupu.

Koeficijent varijacije (Coefficient of Variation - CV)

Koeficijent varijacije predstavlja relativnu mjeru disperzije, koja se koristi za poređenje varijabilnosti podataka različitih veličina ili različitih jedinica mjere.

Izračunava se kao odnos standardne devijacije i srednje vrijednosti, a rezultat se izražava u procentima.

$$CV = \left(\frac{\sigma}{\mu} \right) \times 100\%$$

Gdje je:

- σ – standardna devijacija
- μ – srednja vrijednost

Koeficijent varijacije je korisna statistička mjeru, jer omogućava direktno poređenje varijabilnosti podataka, koji imaju različite jedinice mjere, što ga čini posebno pogodnim za analize u kojima se upoređuju, na primjer, plate izražene u konvertibilnim markama (KM) i cijene proizvoda u američkim dolarima (USD). Njegova primjena je efikasna u analizama stabilnosti podataka, jer niža vrijednost

koeficijenta varijacije ukazuje na veću stabilnost skupa podataka, dok viša vrijednost signalizira veće rasipanje. Takođe, ova mjera se često koristi u finansijskim analizama za procjenu rizika investicija, pri čemu veći koeficijent varijacije ukazuje na veći investicioni rizik i veću neizvjesnost očekivanih povrata.

Primjer poređenja varijabilnosti plata u dvije različite industrije.

```
import numpy as np

# Plate u IT sektoru (KM)
it_salaries = [3500, 4200, 5100, 5800, 6200]
mu_it = np.mean(it_salaries)
sigma_it = np.std(it_salaries, ddof=1)
cv_it = (sigma_it / mu_it) * 100

# Plate u građevinskom sektoru (KM)
construction_salaries = [1400, 1500, 1800, 2000, 2200]
mu_construction = np.mean(construction_salaries)
sigma_construction = np.std(construction_salaries, ddof=1)
cv_construction = (sigma_construction / mu_construction) * 100

# Ispis rezultata
print(f"Koeficijent varijacije za IT sektor: {cv_it:.2f}%")
print(f"Koeficijent varijacije za građevinski sektor: {cv_construction:.2f}%")

Koeficijent varijacije za IT sektor: 22.48%
Koeficijent varijacije za građevinski sektor: 18.80%
```

Objašnjenje:

Ako je CV za IT sektor 22%, a za građevinski sektor 18%, to znači da su plate u IT sektoru varijabilnije u odnosu na njihovu prosječnu vrijednost. Manji CV u građevinskom sektoru znači da su plate stabilnije, dok veći CV u IT sektoru pokazuje veće rasipanje plata.

Relativni razmak (Range Ratio)

Relativni razmak je još jedna mjera varijabilnosti, koja pokazuje odnos između maksimalne i minimalne vrijednosti u skupu podataka.

Koristi se kada želimo da odredimo koliko puta je najveća vrijednost veća od najmanje.

$$RR = \frac{X_{max} - X_{min}}{X_{max} + X_{min}}$$

Gdje je:

- X_{max} – najveća vrijednost u skupu
- X_{min} – najmanja vrijednost u skupu

Primjena u ekonomiji

- Raspon cijena proizvoda – Upoređuje koliko je najskuplji proizvod skuplji od najjeftinijeg u istoj kategoriji.
- Platne razlike među zaposlenima – Pokazuje raspon plata u kompaniji i može ukazati na nepravedne razlike.
- Razlike u BDP-u među regijama – Koristi se za mjerenje ekonomske nejednakosti između regija.

Primjer poređenja razlike u cijenama goriva između različitih benzinskih pumpi.

```
# Cijene goriva (KM/litar) na različitim benzinskim pumpama
fuel_prices = [2.50, 2.60, 2.75, 3.10, 3.20]
```

```
# Računanje relativnog razmaka
X_max = max(fuel_prices)
X_min = min(fuel_prices)
range_ratio = (X_max - X_min) / (X_max + X_min)
```

```
# Ispis rezultata
print(f"Relativni razmak cijena goriva: {range_ratio:.2f}")
```

```
Relativni razmak cijena goriva: 0.12
```

Objašnjenje:

Ako je relativni razmak cijena goriva 0.12, to znači da je najskuplje gorivo za 12% skuplje od najjeftinijeg u skupu podataka. Ova analiza pomaže potrošačima da pronađu najpovoljnije opcije, a tržišnim analitičarima da procijene stabilnost cijena.

6.2. Osnovni pojmovi vjerovatnoće

Vjerovatnoća predstavlja koncept u statistici, koji omogućava kvantifikovanje neizvjesnosti i predviđanje budućih događaja na osnovu podataka. U ekonomiji i poslovnoj analitici vjerovatnoća se koristi za procjenu rizika, donošenje poslovnih odluka i analizu tržišnih trendova.

U nastavku će biti obrađeni osnovni pojmove vjerovatnoće, uključujući definicije, zakon velikih brojeva i ključne teoreme u statistici.

6.2.1. Pojam vjerovatnoće

Vjerovatnoća je brojčana mjera, koja izražava mogućnost da se određeni događaj dogodi. Označava se simbolom $P(A)$, gdje je A određeni događaj.

Matematički, vjerovatnoća događaja A definisana je kao:

$$P(A) = \frac{\text{broj povoljnih ishoda}}{\text{ukupan broj mogućih ishoda}}$$

Vrijednost vjerovatnoće uvijek se nalazi u intervalu od 0 do 1, gdje:

- $P(A)=0$ znači da se događaj nikada neće dogoditi
- $P(A)=1$ znači da će se događaj sigurno dogoditi
- $0 < P(A) < 1$ znači da postoji određena vjerovatnoća da se događaj dogodi.

Vjerovatnoća se u ekonomiji koristi za analizu tržišta, gdje pomaže u procjeni uspješnosti proizvoda i prilagođavanju marketinških strategija. U investicijama, omogućava procjenu vjerovatnoće ostvarivanja povrata, pomažući investitorima u upravljanju rizikom. Finansijske institucije je primjenjuju u analizi kreditnog rizika klijenata, procjenjujući njihovu sposobnost otplate kredita na osnovu finansijskih podataka.

Računanje vjerovatnoće u Pythonu se svodi na proste matematičke operacije dijeljenja. U sljedećem primjeru će biti prikazano računanje uspješnosti prodaje novog proizvoda, urađeno na bazi anketirnih korisnika.

Pretpostavimo da kompanija procjenjuje vjerovatnoću da će njen novi proizvod biti uspješan. Od 192 testiranih korisnika, 73 je izrazilo pozitivan stav.

```
# Broj povoljnih ishoda i ukupnih ishoda
povoljni_ishodi = 73
ukupni_ishodi = 192

# Izračunavanje vjerovatnoće
probability = povoljni_ishodi / ukupni_ishodi
```

```
# Prikaz rezultata
print(f"Vjerovatnoća uspjeha novog proizvoda: {probability:.2f}")

Vjerovatnoća uspjeha novog proizvoda: 0.38
```

Ovaj rezultat pokazuje da je vjerovatnoća uspjeha novog proizvoda 38%.

6.2.2. Zakon velikih brojeva

Zakon velikih brojeva predstavlja jedan od principa statistike, koji navodi da kako se broj ponovljenih eksperimenata povećava, prosječna vrijednost dobijenih rezultata sve više približava teoretskoj očekivanoj vrijednosti. Ovaj zakon objašnjava zašto dugoročna ponavljanja eksperimenta daju pouzdanije procjene stvarnih parametara populacije.

Postoje dva oblika zakona velikih brojeva:

- Slabi zakon velikih brojeva – kada broj ponavljanja eksperimenta raste, prosječna vrijednost uzorka sve više teži očekivanoj vrijednosti, ali uz mogućnost odstupanja u pojedinačnim uzorcima.
- Jaki zakon velikih brojeva – osigurava da se konvergencija ka očekivanoj vrijednosti dešava gotovo sigurno, što znači da će se, uz dovoljno ponavljanja, razlika između prosjeka uzorka i stvarne vrijednosti praktično eliminisati.

Zakon velikih brojeva ima značajnu primjenu u industriji osiguranja, jer omogućava dugoročne prognoze iznosa odštetnih zahtjeva. Kako raste broj osiguranih osoba i dobara, a pod uslovom da su svi izloženi približno jednakom riziku, smanjuje se uticaj slučajnih varijacija i osiguravajuće kuće mogu preciznije procijeniti svoje obaveze. Ipak, ovaj zakon ne omogućava predviđanje ko će konkretno podnijeti odštetni zahtjev, već samo daje procjenu ukupnih troškova na nivou velike populacije.

U oblasti dugoročnih investicija, zakon velikih brojeva pokazuje da kako se broj transakcija povećava, prosječan povrat na investiciju teži teoretskoj stopi prinosa, smanjujući efekat pojedinačnih varijacija. Takođe, u analizi tržišta, obrada velikih skupova podataka omogućava filtriranje slučajnih oscilacija, čime se dobijaju realniji trendovi i preciznije prognoze kretanja na tržištu.

U nastavku je u Pythonu obrađen primjer simulacija bacanja novčića kako bismo vidjeli kako se relativna učestalost "glave" približava teorijskoj vrijednosti od 50%.

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

# Simulacija bacanja novčića 10,000 puta
np.random.seed(42)
```

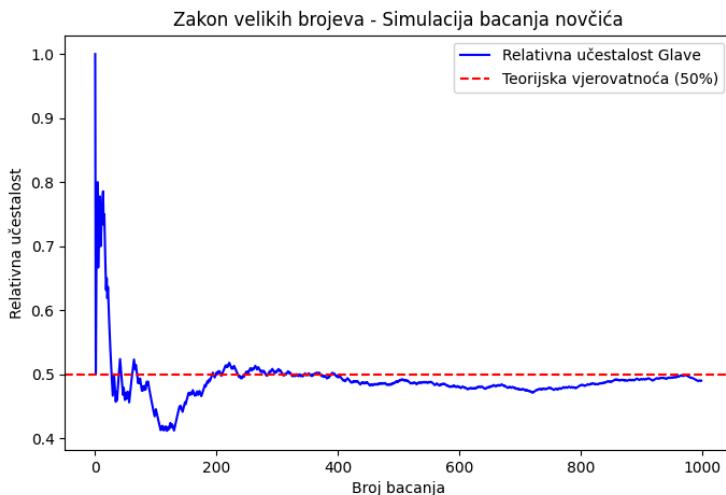
```

broj_bacanja = 10000
rezultati = np.random.choice(["Glava", "Pismo"], size=broj_bacanja)

# Računanje kumulativne frekvencije glava
glave = np.cumsum(rezultati == "Glava") / np.arange(1, broj_bacanja + 1)

# Prikaz rezultata
plt.figure(figsize=(8,5))
plt.plot(glave, label="Relativna učestalost Glave", color='blue')
plt.axhline(y=0.5, color='red', linestyle='--', label="Teorijska vjerovatnoća (50%)")
plt.xlabel("Broj bacanja")
plt.ylabel("Relativna učestalost")
plt.title("Zakon velikih brojeva - Simulacija bacanja novčića")
plt.legend()
plt.show()

```



6.2.3. Teorema o ukupnoj vjerovatnoći (Law of Total Probability)

Teorema o ukupnoj vjerovatnoći predstavlja jedan od osnovnih principa u teoriji vjerovatnoće, koji povezuje vjerovatnoću nekog događaja sa vjerovatnoćama skupa međusobno isključivih događaja. Ova teorema polazi od ideje da ako se neki događaj može desiti na više načina, tada je ukupna vjerovatnoća tog događaja jednaka zbiru vjerovatnoća svih pojedinačnih načina na koje se može ostvariti.

Matematički, teorema se izražava formulom:

$$P(A) = P(A|B_1)P(B_1) + P(A|B_2)P(B_2) + \dots + P(A|B_n)P(B_n)$$

Gdje su:

- $P(A)$ – ukupna vjerovatnoća događaja A.

- $P(A | B_i)$ – uslovna vjerovatnoća događaja A pod uslovom da se dogodio B_i .
- $P(B_i)$ – vjerovatnoća da se dogodi događaj B_i .
- n – broj mogućih načina na koje se događaj A može ostvariti.

Ova teorema je temelj za analizu vjerovatnoće u različitim oblastima, uključujući finansije, ekonomiju, statistiku i analizu podataka. Neki od primjera primjene ove teorije su:

- Analiza tržišnog učešća. Kompanije često koriste ovu teoremu kako bi izračunale ukupnu vjerovatnoću prodaje u različitim regijama. Ako postoje različita tržišta sa specifičnim vjerovatnoćama kupovine proizvoda, možemo koristiti ovu teoremu da dobijemo ukupnu stopu prodaje.
- Procjena vjerovatnoće neplaćanja kredita. Banka može koristiti ovu teoremu kako bi procijenila ukupnu vjerovatnoću da klijent neće vratiti kredit, uzimajući u obzir različite grupe klijenata sa različitim kreditnim rejtingom.

Primjer u Pythonu se takođe bazira na osnovnim matematičkim operacijama. Pretpostavimo da banka želi izračunati vjerovatnoću da klijent neće vratiti kredit na osnovu dva različita kreditna rejtinga. Ako imamo sljedeće vjerovatnoće:

- Visoko kreditno sposobni klijenti imaju malu vjerovatnoću neplaćanja (2%) i čine 80% svih klijenata.
- Klijenti sa niskim kreditnim rejtingom imaju veću vjerovatnoću neplaćanja (15%) i čine 20% svih klijenata.

```
# Vjerovatnoće neplaćanja u zavisnosti od kreditnog rejtinga
P_A_B1 = 0.02 # Vjerovatnoća neplaćanja za klijente visoko kreditno sposobane
P_B1 = 0.80 # Vjerovatnoća da je klijent visoko kreditno sposoban

P_A_B2 = 0.15 # Vjerovatnoća neplaćanja za klijente niskog rejtinga
P_B2 = 0.20 # Vjerovatnoća da je klijent niskog rejtinga

# Ukupna vjerovatnoća neplaćanja kredita
P_A = P_A_B1 * P_B1 + P_A_B2 * P_B2

# Prikaz rezultata
print(f"Ukupna vjerovatnoća da klijent neće vratiti kredit: {P_A:.3f}")
```

To znači da postoji 4.6% šanse da nasumično odabrani klijent neće vratiti kredit.

6.2.4. Bayesova teorema

Bayesova teorema je fundamentalni princip vjerovatnoće, koji omogućava računanje uslovne vjerovatnoće, odnosno vjerovatnoće da se određeni događaj A dogodi pod uslovom da se već dogodio drugi događaj B. Matematički, Bayesova formula glasi:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

Gdje je:

- $P(A|B)$ – vjerovatnoća događaja A pod uslovom da se dogodio B
- $P(B|A)$ – vjerovatnoća događaja B ako se dogodio A
- $P(A)$ – vjerovatnoća događaja A
- $P(B)$ – vjerovatnoća događaja B

U nastavku je primjer procjena vjerovatnoće uspjeha investicije u novi proizvod, gdje se vjerovatnoća računa pomoću Bayesove teoreme. Za računanje vjerovatnoće u ovom primjeru su korištene osnovne matematičke funkcije u Python-u.

Posmatrajmo kompaniju XYZ koja planira lansirati novu liniju ekoloških proizvoda, ali prije donošenja konačne odluke, želi procijeniti vjerovatnoću da li će proizvod biti uspješan na tržištu. Na osnovu podataka iz prethodnih istraživanja tržišta, znaju sljedeće:

- 60% potrošača iskazuje interesovanje za ekološke proizvode.
- 70% potrošača koji su ranije koristili slične proizvode nastavili su ih kupovati.
- 40% novih kupaca koji su ih probali ostali su lojalni brendu.

Potrebito je izračunati vjerovatnoću da će potrošač koji prvi put kupi ekološki proizvod postati dugoročni kupac.

```
# Definisane vjerovatnoće
P_L = 0.60 # Vjerovatnoća da potrošač preferira ekološke proizvode (Lojalan)
P_E_L = 0.70 # Vjerovatnoća lojalnosti ako kupac preferira ekološke proizvode
P_E_not_L = 0.30 # Vjerovatnoća lojalnosti ako kupac NE preferira ekološke proizvode
P_not_L = 1 - P_L # Vjerovatnoća da kupac ne preferira ekološke proizvode

# Računanje ukupne vjerovatnoće P(E) pomoću teoreme o ukupnoj vjerovatnoći
P_E = (P_E_L * P_L) + (P_E_not_L * P_not_L)

# Primjena Bayesove teoreme za izračunavanje P(L|E)
P_L_E = (P_E_L * P_L) / P_E
# Prikaz rezultata
print(f"Vjerovatnoća da će novi kupac postati lojalan: {P_L_E:.2f}")

Vjerovatnoća da će novi kupac postati lojalan: 0.78
```

To znači da postoji 78% šanse da novi kupac postati lojalan kupac.

6.3. Statističko zaključivanje (Inferencijalna statistika)

Inferencijalna statistika (statističko zaključivanje) predstavlja skup metoda, koje omogućavaju izvođenje zaključaka o populaciji na osnovu uzorka podataka. Dok deskriptivna statistika opisuje podatke, inferencijalna statistika omogućava testiranje hipoteza, procjenu parametara i predviđanje.

Glavna svrha inferencijalne statistike je da, na osnovu ograničenog uzorka, doneše pouzdane zaključke o cijeloj populaciji. Ovo je posebno korisno u ekonomiji, gdje je često nemoguće analizirati sve podatke (npr. prihod svih građana u zemlji), pa se koriste metode zaključivanja na osnovu reprezentativnih uzoraka.

6.3.1. Osnovni pojmovi testiranja hipoteza

6.3.1.1. Nulta i alternativna hipoteza

Testiranje hipoteza je osnovni postupak u inferencijalnoj statistici, koji se koristi za provjeru pretpostavki o populaciji.

- Nulta hipoteza (H_0) predstavlja početnu pretpostavku i obično podrazumijeva nepostojanje efekta, razlike ili povezanosti.
- Alternativna hipoteza (H_1) je suprotna tvrdnja, koja prepostavlja postojanje efekta, razlike ili povezanosti.

Primjer u ekonomiji:

- H_0 – Ne postoji razlika u prosječnim prihodima muškaraca i žena.
- H_1 – Postoji statistički značajna razlika u prosječnim prihodima muškaraca i žena.

6.3.1.2. Statistička značajnost i P-vrijednost

Statistička značajnost nam govori da li je dobijeni rezultat vjerovatno posljedica slučajnosti ili stvarnog efekta. Pritom koristimo P-vrijednost²² (p-value), koja označava vjerovatnoću da se dobijeni rezultat desio slučajno.

- Ako je P-vrijednost manja od 0.05 → odbacujemo nultu hipotezu i zaključujemo da postoji statistički značajna razlika.
- Ako je P-vrijednost veća od 0.05 → ne možemo odbaciti nultu hipotezu i smatramo da nema dovoljno dokaza za tvrdnju o razlici.

²² Slovo p u P-vrijednosti dolazi od engleske riječi "probability", što znači vjerovatnoća.

Primjer u ekonomiji:

Ako analiziramo razliku u prosječnim prihodima između dvije grupe zaposlenih i dobijemo $p=0.02$, to znači da postoji samo 2% šanse da je ova razlika nastala slučajno. Pošto je $p<0.05$, možemo reći da je razlika statistički značajna i vjerovatno odražava stvaran efekat.

6.3.1.3. Greške tipa I i tipa II

Pri testiranju hipoteza, moguće su dvije vrste grešaka:

- Greška tipa I (lažno pozitivna) – pogrešno odbacivanje nulte hipoteze (npr. zaključimo da postoji razlika u prihodima, iako je nema).
- Greška tipa II (lažno negativna) – pogrešno prihvatanje nulte hipoteze (npr. ne prepoznamo stvarnu razliku u prihodima).

Primjer:

- Greška tipa I – Kompanija zaključi da novi marketing povećava prodaju, iako zapravo nema efekta.
- Greška tipa II – Kompanija ne prepozna da bi novi marketing značajno povećao prodaju.

6.3.2. Parametarski testovi

Parametarski testovi koriste se kada su podaci normalno distribuirani i kada znamo neke karakteristike populacije. Najčešće korišćen test je Studentov t-test.

6.3.2.1. Studentov t-test

Studentov t-test koristi se za poređenje srednjih vrijednosti između jednog ili više uzoraka. Postoje tri glavne verzije:

- 1) Test za jedan uzorak – poredi srednju vrijednost uzorka sa poznatom srednjom vrijednošću populacije.
- 2) Test za dva nezavisna uzorka – poredi srednje vrijednosti između dvije nezavisne grupe.
- 3) Parni t-test – poredi srednje vrijednosti između dva zavisna (uparena) uzorka (npr. prije i poslije primjene ekonomske politike).

Funkcija `stats.ttest_ind()` iz biblioteke SciPy koristi se za nezavisni Studentov t-test, koji poredi srednje vrijednosti dvije nezavisne grupe, kako bi se utvrdilo da li između njih postoji značajna razlika.

Osnovna sintaksa:

```
t_stat,p_value = scipy.stats.ttest_ind(a,b,equal_var=True,  
alternative='two-sided')
```

Gdje je:

- **a, b** – dva uzorka podataka (list, numpy array, pandas series, itd.).
- **equal_var=True** – podrazumijeva se da obje grupe imaju jednake varijanse. Ako se varijanse razlikuju, postavlja se **equal_var=False** (Welchov t-test).
- **alternative='two-sided'** – specificira tip testa:
 - **'two-sided'** (dvostrani test) – provjerava postoji li bilo kakva razlika između grupa.
 - **'less'** – provjerava da li je srednja vrijednost prve grupe manja od druge.
 - **'greater'** – provjerava da li je srednja vrijednost prve grupe veća od druge.

Funkcija **stats.ttest_ind()** kao rezultat vraća dvije vrijednosti:

- **t_stat** – t-statistika. Ovo je vrijednost testne statistike koja pokazuje koliko su srednje vrijednosti dvije grupe različite u standardizovanim jedinicama. Veće apsolutne vrijednosti t ukazuju na veću razliku između grupa. Ako je t pozitivan, prva grupa ima veću srednju vrijednost. Ako je t negativan, druga grupa ima veću srednju vrijednost.
- **p_value** – p-vrijednost. P-vrijednost pokazuje vjerovatnoću da se dobijeni rezultat dogodio slučajno ako je nulta hipoteza tačna. Ako $p < 0.05$, razlika između grupa je statistički značajna, tj. postoji dovoljno dokaza da su grupe različite. Ako $p > 0.05$, nema dovoljno dokaza za razliku između grupa, tj. nulta hipoteza se ne odbacuje.

Sljedeći primjer u Pythonu predstavlja poređenje prosječnih prihoda između dvije grane industrije. Ovo je standardni nezavisni t-test koji podrazumijeva jednake varijanse.

```
import scipy.stats as stats  
  
# Dva skupa podataka (npr. prihodi zaposlenih u dvije industrije)  
group_A = [2500, 2600, 2450, 2700, 2800, 2650, 2550, 2750]  
group_B = [1900, 2250, 2300, 2150, 2400, 2350, 2580, 2920]  
  
# Standardni t-test (pretpostavlja jednake varijanse)  
t_stat, p_value = stats.ttest_ind(group_A, group_B)  
  
print(f"t-statistika: {t_stat:.3f}, P-vrijednost: {p_value:.3f}")  
  
t-statistika: 2.338, P-vrijednost: 0.035
```

Ako t-stat = 2.338, to znači da postoji razlika između grupa u 2.338 standardizovane jedinice. Ako p-value = 0.035, to znači da postoji 3.5% šanse da je razlika slučajna, što je statistički značajno jer je $p < 0.05$.

6.3.2.2. Varijacije Studentovog t-testa

Osim `stats.ttest_ind`, postoje još dvije varijante Studentovog t-testa:

- 1) t-test za jedan uzorak (`stats.ttest_1samp`), koja poredi srednju vrijednost uzorka sa poznatom populacijskom sredinom.

```
import scipy.stats as stats

sample = [2550, 2600, 2450, 2700, 2800, 2650]
pop_mean = 2500 # Pretpostavljena srednja vrijednost populacije

t_stat, p_value = stats.ttest_1samp(sample, pop_mean)
print(f"T-test za jedan uzorak - P-vrijednost: {p_value:.3f}")

T-test za jedan uzorak - P-vrijednost: 0.053
```

Ako p-value = 0.053, to znači da postoji 5.3% šanse da je razlika slučajna, što statistički nije značajno jer nije $p < 0.05$.

- 2) Parni t-test (`stats.ttest_rel`), koja se koristi kada imamo **zavisne uzorke** (npr. prije i poslije nekog događaja).

```
import scipy.stats as stats

# Primjer: Prihodi zaposlenih prije i poslije ekonomskih reformi
before_reform = [2500, 2600, 2450, 2700, 2800, 2650]
after_reform = [2550, 2800, 2600, 2850, 2900, 2750]

t_stat, p_value = stats.ttest_rel(before_reform, after_reform)
print(f"Parni t-test - P-vrijednost: {p_value:.3f}")

Parni t-test - P-vrijednost: 0.002
```

Ako $p < 0.05$, reforma je značajno uticala na prihode.

6.3.2.3. ANOVA test

Analiza varijanse (ANOVA - *Analysis of Variance*) je parametarski test, koji se koristi za poređenje srednjih vrijednosti više od dvije grupe. Dok Studentov t-test poredi dvije grupe, ANOVA omogućava analizu razlika između tri ili više grupa istovremeno. Glavni cilj ANOVA testa je da utvrdi da li postoji statistički značajna razlika između srednjih vrijednosti grupa ili su razlike nastale slučajno.

Ako ANOVA test pokaže statistički značajnu razliku između grupa, potrebno je dodatno istražiti, koje tačno grupe se međusobno razlikuju.

Funkcija `stats.f_oneway()` iz biblioteke SciPy koristi se za jednofaktorsku ANOVA analizu (One-Way ANOVA). Ova funkcija testira da li postoji statistički značajne razlike između srednjih vrijednosti više nezavisnih grupa.

Osnovna sintaksa ove funkcije je:

```
t_stat,p_value = scipy.stats.f_oneway(*args)
```

Gdje je:

- `*args` – liste, nizovi (`numpy array, pandas series`) koji predstavljaju različite grupe podataka. Moraju imati numeričke vrijednosti.
- Nema potrebe za definisanjem varijanse unaprijed, jer funkcija sama računa F-statistiku i P-vrijednost.

Funkcija vraća dvije vrijednosti:

- **F-statistika** – pokazuje odnos između varijacije između grupa i varijacije unutar grupa.
- **P-vrijednost** – ukazuje na statističku značajnost razlika između srednjih vrijednosti. Ako je **P < 0.05**, postoji **statistički značajna razlika** između barem dvije grupe. U slučaju da je **P > 0.05**, nema dokaza da su srednje vrijednosti grupa različite.

U sljedećem primjeru će biti analizirane razlike u prosječnim prihodima zaposlenih u tri različite industrije pomoću ANOVA testa.

```
import scipy.stats as stats

# Prihodi zaposlenih u tri različite industrije (u KM)
IT_sector = [2000, 3200, 4100, 4300, 4400, 3950, 4150]
Finance_sector = [3800, 3900, 3700, 3950, 3850, 3750, 4000]
Retail_sector = [2200, 3300, 3100, 3350, 3400, 3150, 3250]

# Izvršavanje ANOVA testa
F_stat, p_value = stats.f_oneway(IT_sector, Finance_sector, Retail_sector)

# Prikaz rezultata
print(f"F-statistika: {F_stat:.3f}")
print(f"P-vrijednost: {p_value:.3f}")
```

F-statistika: 3.634
P-vrijednost: 0.047

Budući da je P-vrijednost manja od 0.05 ($p < 0.05$), možemo zaključiti da postoji statistički značajna razlika između najmanje dvije grupe u našem skupu podataka.

Dvostruka ANOVA (*Two-Way ANOVA*) je statistički test koji omogućava istovremenu analizu uticaja dvije nezavisne varijable (faktora) na jednu zavisnu varijablu. Za razliku od jednofaktorske ANOVA analize, koja ispituje samo jedan faktor, dvostruka ANOVA omogućava identifikaciju glavnih efekata svakog faktora, kao i njihove međusobne interakcije.

U biblioteci SciPy ne postoji ugrađena funkcija za sprovodenje dvostrukih ANOVA analiza. Umjesto toga, za ovu vrstu statističkog testiranja koristi se napredniji modul [statsmodels](#), koji omogućava detaljniju analizu varijanse i regresione modele.

S obzirom na to da upotreba [statsmodels](#) modula i dvostrukih ANOVA analize prevazilazi nivo ove knjige, preporučuje se da čitalac samostalno istraži ovu temu i način njene primjene u Python okruženju.

Prednosti ANOVA testa:

- Može istovremeno uporediti više grupa, za razliku od t-testa.
- Efikasan za analizu kompleksnih podataka u ekonomiji i biznisu.
- Mogućnost analize interakcije između faktora (npr. industrija + obrazovanje).

Ograničenja:

- Prepostavlja da su podaci normalno distribuirani i da grupe imaju jednaku varijanu.
- Ne pokazuje koje grupe se razlikuju, već samo da postoji razlika (potrebni su post-hoc testovi poput Tukey testa).

6.3.3. Neparametarski testovi

Neparametarski testovi koriste se kada podaci ne zadovoljavaju prepostavku normalne distribucije ili kada su podaci rangirani umjesto numerički izraženi. Za razliku od parametarskih testova, koji se oslanjaju na srednje vrijednosti i standardne devijacije, neparametarski testovi su robustniji i primjenjuju se u slučajevima, kada su podaci asimetrični ili dolaze iz različitih mjerila.

Neparametarski testovi se često koriste u ekonomiji, sociologiji, medicini i analizi tržišta, gdje su podaci rangirani ili kategorisani.

6.3.3.1. Chi-kvadrat test

Chi-kvadrat test (χ^2 -test) koristi se za testiranje povezanosti između dvije kategorijalne varijable. On ispituje da li postoji statistički značajna veza između varijabli tako što upoređuje očekivane i stvarne frekvencije u kontingencijskim tabelama.

Matematički, Chi-kvadrat statistika se računa pomoću formule:

$$\chi^2 = \sum \frac{(O-E)^2}{E}$$

Gdje su:

- O – stvarne (posmatrane) vrijednosti
- E – očekivane vrijednosti

Ako je dobijena P-vrijednost manja od 0.05, tada možemo zaključiti da postoji statistički značajna povezanost između varijabli.

Chi-kvadrat test se koristi u različitim ekonomskim analizama:

- Analiza povezanosti obrazovanja i zaposlenja – Da li osobe s višim obrazovanjem imaju veću vjerovatnoću zaposlenja?
- Proučavanje odnosa između regije i potrošačkih preferencija – Da li stanovnici različitih regija preferiraju različite tipove proizvoda?
- Marketinške analize – Testiranje da li pol kupca utiče na izbor proizvoda.

Funkcija `stats.chi2_contingency()` iz SciPy biblioteke se koristi za sprovođenje Chi-kvadrat testa nezavisnosti.

Sintaksa ove funkcije je:

```
chi2_stat,p_value,dof,expected = scipy.stats.chi2_contingency(  
    observed,correction=True, lambda_=None)
```

Gdje je:

- `observed` – Kontingenciona tabela (numpy array ili pandas DataFrame).
- `correction=True` – Primjenjuje Yatesovu korekciju za kontinuitet kod 2×2 tabele (preporučuje se za male uzorke).
- `lambda_` – Opcioni parametar za prilagođene modifikacije Chi-kvadrat testa.

Funkcija vraća četiri vrijednosti:

- `chi2_stat` – Chi-kvadrat statistika
- `p_value` – P-vrijednost (značajnost)
- `dof` – Stepeni slobode
- `expected` – Očekivane vrijednosti u tabeli

Pretpostavimo da analiziramo povezanost između vrste plaćanja i starosne grupe kupaca u supermarketu. Podaci su prikazani u sljedećoj tabeli.

Tabela 6.2. Primjer podataka za Chi-kvadrat test

	Gotovina	Kartica	Mobilno plaćanje
Mladi (18-35)	50	100	40
Srednja dob (36-55)	30	120	50
Stariji (56+)	20	80	30

Koristeći Chi-kvadrat test i funkciju `chi2_contingency()` provjeravamo da li postoji statistički značajna povezanost između starosne grupe i načina plaćanja.

```
import scipy.stats as stats
import numpy as np

# Kreiranje kontingencione tabelle (Starosna grupa x Način plaćanja)
data = np.array([[50, 100, 40],    # Mladi (18-35)
                 [30, 120, 50],  # Srednja dob (36-55)
                 [20, 80, 30]]) # Stariji (56+)

# Izvršavanje Chi-kvadrat testa
chi2_stat, p_value, dof, expected = stats.chi2_contingency(data)

# Prikaz rezultata
print(f"Chi-kvadrat statistika: {chi2_stat:.3f}")
print(f"P-vrijednost: {p_value:.3f}")
print(f"Stepeni slobode: {dof}")
print("Očekivane vrijednosti u tabeli:")
print(expected)

Chi-kvadrat statistika: 9.840
P-vrijednost: 0.043
Stepeni slobode: 4
Očekivane vrijednosti u tabeli:
[[ 36.53846154 109.61538462 43.84615385]
 [ 38.46153846 115.38461538 46.15384615]
 [ 25.          75.          30.        ]]
```

Tumačenje rezultata:

- Ako je $p < 0.05$, znači da postoji statistički značajna povezanost između starosne grupe i načina plaćanja (u našem primjeru $p=0.043$).
- Ako je $p > 0.05$, ne postoji značajna povezanost, što znači da način plaćanja nije zavisan od starosne grupe.

6.3.3.2. Mann-Whitney u-test

Mann-Whitney u-test je neparametarski test, koji se koristi kada podaci nisu normalno distribuirani ili kada se analiziraju rangirani podaci. Ovaj test ne upoređuje srednje vrijednosti između dvije grupe, već se temelji na rangovima podataka, ispitujući da li jedna grupa ima konzistentno veće ili manje vrijednosti od druge. Koristi se kao

alternativa Studentovom t-testu kada podaci ne zadovoljavaju pretpostavku normalnosti.

Mann-Whitney u-test se široko koristi u analizi ekonomskih podataka, uključujući:

- Poređenje prihoda između dvije industrije kada podaci nisu normalno distribuirani.
- Analizu potrošačkih navika – da li kupci iz ruralnih i urbanih područja troše različite iznose?
- Analizu tržišnog udjela – testiranje razlika u prodaji između dvije konkurenčke kompanije.

Funkcija `stats.mannwhitneyu()` iz SciPy biblioteke koristi se za sprovođenje Mann-Whitney u-testa.

```
u_stat,p_value=scipy.stats.mannwhitneyu(x,y,use_continuity=True  
, alternative='two-sided', method='auto')
```

Gdje je:

- `x,y` – Dva skupa podataka (liste ili nizovi) za poređenje.
- `use_continuity=True` – Primjenjuje kontinuitetnu korekciju (korisno za male uzorke).
- `alternative='two-sided'` – Definiše vrstu testa:
 - `'two-sided'` – Testira razliku u oba smjera.
 - `'less'` – Testira da li su vrijednosti iz prve grupe značajno manje od druge.
 - `'greater'` – Testira da li su vrijednosti iz prve grupe značajno veće od druge.
- `method='auto'` – Automatski bira optimalni algoritam za izračunavanje testa.

Funkcija vraća dvije vrijednosti:

- **U-statistika** – Vrijednost Mann-Whitney u-testa (mjereno koliko su podaci u jednoj grupi veći/manji od druge).
- **P-vrijednost** – Statistička značajnost rezultata (ako $p < 0.05$, razlika između grupa je značajna).

Prepostavimo da želimo analizirati razliku u prihodima zaposlenih između IT sektora i Gradevinarstva. S obzirom na prirodu ovih industrija, očekujemo da su plate u IT sektoru veće nego u gradevinarstvu, ali želimo statistički potvrditi tu razliku pomoću Mann-Whitney u-testa.

Posmatrćemo uzorak od po 10 zaposlenih u IT sektoru i Gradevinarstvu i bilježimo njihove mjesечne prihode u KM. Podaci nisu normalno distribuirani, pa koristimo Mann-Whitney u-test umjesto klasičnog t-testa.

```

import scipy.stats as stats

# Prihodi u IT sektoru (KM)
it_salaries = [5000, 5200, 5300, 4900, 5500, 6000, 5800, 5700, 5400, 5100]

# Prihodi u Građevinarstvu (KM)
construction_salaries = [3200, 3400, 3300, 3100, 3000, 3500, 2900, 2800, 3100, 3000]

# Izvršavanje Mann-Whitney U testa
u_stat, p_value = stats.mannwhitneyu(it_salaries, construction_salaries,
alternative='greater')

# Prikaz rezultata
print(f"U statistika: {u_stat:.3f}")
print(f"P-vrijednost: {p_value:.5f}")

U statistika: 100.000
P-vrijednost: 0.00009

```

Rezultati Mann-Whitney u-testa pokazuju:

- U-statistika = 100.000 : Vrijednosti iz prve grupe (IT sektor) su značajno veće od vrijednosti iz druge grupe (Građevinarstvo).
- P-vrijednost = 0.00009 : Budući da je P-vrijednost znatno manja od 0.05, razlika između grupa je statistički značajna.

6.4. Primjena statistike u ekonomiji

Savremene ekonomiske analize nezamislive su bez statističkih metoda, koje omogućavaju kvantifikaciju nesigurnosti, mjerjenje uticaja različitih faktora na ekonomске pojave i predviđanje budućih kretanja. Deskriptivna statistika omogućava sažimanje velikih skupova podataka kroz ključne mjere centralne tendencije i disperzije, dok regresiona i korelaciona analiza omogućavaju kvantifikaciju odnosa između varijabli poput cijena i potražnje, dohotka i potrošnje ili inflacije i nezaposlenosti.

Posebno značajnu ulogu imaju ekonomski indeksi, koji omogućavaju praćenje promjena u cijenama, proizvodnji i prihodima kroz duže vremenske periode, kao i napredne metode prognoziranja, koje se zasnivaju na vremenskim serijama i ekonometrijskim modelima. Ovi alati omogućavaju ekonomistima donošenje ispravnih odluka u vezi s monetarnom i fiskalnom politikom, investicijama i tržišnim strategijama.

Kroz ovo poglavlje detaljno će biti obrađena primjena statistike u ekonomiji, pri čemu će primjeri koristiti podatke, koji su dostupni kroz online servise. Ulagani podaci mogu biti zamjenjeni sa podacima do kojih se dolazi kroz prikupljanje i obradu realnih podataka poslovanja.

6.4.1. Deskriptivna statistika u ekonomiji

Deskriptivna statistika predstavlja alat za analizu velikih skupova ekonomskih podataka. U kontekstu tržišnih analiza, makroekonomskih indikatora i praćenja poslovnih ciklusa, deskriptivna statistika omogućava donošenje kvalitetnih poslovnih i ekonomskih odluka, bolje razumijevanje ekonomskih trendova te adekvatno planiranje strategija.

U ovom poglavlju će biti detaljno razmatrane ključne primjene deskriptivne statistike u ekonomiji, zajedno s primjerima u Pythonu. Korištenje Python modula kao što su Pandas, NumPy i matplotlib omogućava brzo i efikasno prikupljanje, obradu i vizualizaciju ekonomskih podataka. U primjerima će biti korišteni podaci iz različitih online izvora (FRED, World Bank Open Data, Yahoo Finance, itd.).

6.4.1.1. Analiza tržišta

Analiza tržišta se zasniva na postupaku prikupljanja, obrade i interpretacije finansijskih podataka s ciljem prepoznavanja trendova, volatilnosti²³ i potencijalnih prilika za investiranje ili strateško planiranje u dinamičnom ekonomskom okruženju.

²³ Volatilnost se u ekonomsko-finansijskoj terminologiji koristi za označavanje neke nestabilnosti. Volatilnost je mjeru nepredvidljive promjene neke varijable u nekom razdoblju.

Izračunavanje prosječne cijene

U tržišnim analizama često je potrebno odrediti prosječnu (engl. *mean*) ili medijalnu (engl. *median*) vrijednost određenog finansijskog instrumenta (npr. akcije) ili proizvoda (npr. roba). Projek (aritmetička sredina) daje globalni parametar nivo cijena, dok medijan može bolje prikazati srednju vrijednost uz manju osjetljivost na ekstremne vrijednosti.

U sljedećem primjeru analizirane su cijene akcije kompanije Microsoft (MSFT) koristeći online servis Yahoo Finance i izračunata prosječna i medijalna cijena zatvaranja za određeni period.

Ukoliko se prvi put koristite modul `yfinance` potrebno ga je instalirati sa Python alatom `pip`, odnosno komandom `pip install yfinance`.

```
import yfinance as yf
import pandas as pd
import numpy as np

# 1. Preuzimanje podataka
msft_data = yf.download("MSFT", start="2022-01-01", end="2022-12-31")

# 2. Izdvajanje cijena zatvaranja
close_prices = msft_data['Close']

# 3. Računanje aritmetičke sredine i medijana
mean_price = np.mean(close_prices)
median_price = np.median(close_prices)

print("Prosječna (mean) cijena zatvaranja za MSFT 2022. godine:", round(mean_price, 2))
print("Medijalna (median) cijena zatvaranja za MSFT 2022. godine:",
round(median_price, 2))

[*****100*****] 1 of 1 completed
Prosječna (mean) cijena zatvaranja za MSFT 2022. godine: 268.92
Medijalna (median) cijena zatvaranja za MSFT 2022. godine: 265.9
```

Mean (projek) pokazuje kolika je bila prosječna dnevna cijena zatvaranja akcije u odabranom periodu. Median (medijan) može biti sličan projeku, ali ako postoji ekstremno visoke ili niske vrijednosti u kratkom periodu, medijan neće biti toliko zavisna od ekstrema kao mean.

U prilugu C - Modul Yahoo Finance možete pronaći detaljne opise najčešće korištenih funkcija u modulu `yfinance`.

Raspon cijena (Range, Min, Max)

Raspon cijena (engl. *range*) označava razliku između najviše i najniže zabilježene cijene. Pored samog raspona, korisno je znati i minimalnu i maksimalnu vrijednost. U nastavku je primjer, koji računa ove podake za akcije Microsoft kompanije.

```
import yfinance as yf

# Preuzimanje podataka sa Yahoo Finance
msft_data = yf.download("MSFT", start="2022-01-01", end="2022-12-31")
close_prices = msft_data['Close']

# Minimum i maksimum cijene zatvaranja
min_price = close_prices.min()
max_price = close_prices.max()

# Raspon
price_range = max_price - min_price

print("Minimalna cijena zatvaranja:", round(min_price, 2))
print("Maksimalna cijena zatvaranja:", round(max_price, 2))
print("Raspon cijena zatvaranja:", round(price_range, 2))

[*****100%*****] 1 of 1 completed
Minimalna cijena zatvaranja: Ticker
MSFT    214.25
dtype: float64
Maksimalna cijena zatvaranja: Ticker
MSFT    334.75
dtype: float64
Raspon cijena zatvaranja: Ticker
MSFT    120.5
dtype: float64
```

Stopa inflacije

Stopa inflacije u ekonomiji označava prosječan rast cijena roba i usluga tokom određenog vremenskog perioda. Najčešće se mjeri pomoću indeksa potrošačkih cijena (CPI – Consumer Price Index).

Formula za godišnju stopu inflacije, kada imamo CPI vrijednosti, može se predstaviti na sljedeći načini:

$$\text{Inflacija} = \frac{\text{CPI}_{\text{tekuga godina}} - \text{CPI}_{\text{prosla godina}}}{\text{CPI}_{\text{prosla godina}}} \times 100\%$$

Primjer računanja stope inflacije na osnovu preuzetih CPI podataka iz FRED-a online baze.

```
import datetime
import pandas_datareader.data as web
```

```

# Vremenski opseg kako bismo obuhvatili decembar 2023. i decembar 2024.
start = datetime.datetime(2022, 1, 1)
end = datetime.datetime(2025, 1, 1)

# Preuzimanje mjesecnog CPI indeksa Kine (CHNCPIALLMINMEI) sa FRED-a
df = web.DataReader("CHNCPIALLMINMEI", "fred", start, end)

# Preimenovanje kolone radi jednostavnijeg pristupa
df.rename(columns={"CHNCPIALLMINMEI": "China"}, inplace=True)

# Definisanje datuma za decembar 2023. i decembar 2024.
start_date = "2023-12-01"
end_date = "2024-12-01"

# Računanje inflacije ako postoje podaci za oba datuma
if start_date in df.index and end_date in df.index:
    cpi_start = df.loc[start_date, "China"]
    cpi_end = df.loc[end_date, "China"]
    inflacija = ((cpi_end - cpi_start) / cpi_start) * 100
    print(f"Godišnja inflacija (dec 2023. → dec 2024.): {inflacija:.2f}%")
else:
    print("Nema dostupnih podataka za decembar 2023. ili decembar 2024.")
```

Godišnja inflacija (dec 2023. → dec 2024.): 0.10%

Za čitanje podataka određene zemlje neophodno je podesiti identifikator te zemlje. Ukoliko želite da promjenite zemlju, potrebno je u funkciji `web.DataReader(key, "fred", start, end)`, promijeniti identifikator (key). Identifikator je složeni ključ od ISO koda države i ključne riječi kao na primjer:

- Germany: "DEUCPIALLMINMEI",
- United Kingdom: "GBRCPIALLMINMEI",
- China: "CHNCPIALLMINMEI"

6.4.1.2. Makroekonomski indikatori

Makroekonomski indikatori poput Bruto domaći proizvod (BDP/GDP), stope nezaposlenosti, kamatnih stopa, te drugih ključnih pokazatelja (poput indeksa industrijske proizvodnje) služe za praćenje ukupnog ekonomskog stanja.

Bruto domaći proizvod (BDP / GDP)

BDP je tržišna vrijednost svih finalnih roba i usluga proizvedenih u nekoj zemlji u određenom periodu. Praćenje BDP-a pomaže u ocjeni rasta ili aktivnosti ekonomije.

U nastavku je primjer koji preuzima kvartalne BDP (GDP) podatke za SAD od početka 2023. do kraja 2024. godine i računa kvartalne stope rasta (QoQ – *quarter-over-quarter*). U primjeru će se koristiti online servis FRED kao izvor podataka, koje ćemo preuzeti sa modulom `pandas_datareader`.

```

import datetime
import pandas_datareader.data as web

# 1. Definišemo period (od 1. januara 2023. do 1. januara 2025.)
start = datetime.datetime(2023, 1, 1)
end = datetime.datetime(2025, 1, 1)

# 2. Preuzimanje kvartalnih GDP podataka za SAD s FRED-a
gdp_data = web.DataReader("GDP", "fred", start, end)

# 3. Ispis posljednjih 5 vrijednosti radi pregleda
print("Posljednjih 5 kvartalnih vrijednosti BDP:\n", gdp_data.tail())

# 4. Računanje prosječnog kvartalnog BDP-a za odabrani period
average_gdp = gdp_data['GDP'].mean()
print("\nProsječni kvartalni BDP (u milijardama USD) za period 2023-2024:",
      round(average_gdp, 2))

# 5. Računanje stope rasta između kvartala (QoQ Growth)
#   QoQ = ((GDP_t - GDP_{t-1}) / GDP_{t-1}) * 100 (%)
gdp_data['QoQ_Growth_%'] = gdp_data['GDP'].pct_change() * 100

print("\n--- BDP i kvartalne stope rasta (QoQ) ---")
print(gdp_data)

```

Posljednjih 5 kvartalnih vrijednosti BDP:
GDP

DATE	GDP
2023-10-01	28296.967
2024-01-01	28624.069
2024-04-01	29016.714
2024-07-01	29374.914
2024-10-01	29700.580

Prosječni kvartalni BDP (u milijardama USD) za period 2023-2024: 28449.89

--- BDP i kvartalne stope rasta (QoQ) ---
GDP QoQ_Growth_%

DATE	GDP	QoQ_Growth_%
2023-01-01	27164.359	NaN
2023-04-01	27453.815	1.065573
2023-07-01	27967.697	1.871805
2023-10-01	28296.967	1.177323
2024-01-01	28624.069	1.155961
2024-04-01	29016.714	1.371730
2024-07-01	29374.914	1.234461
2024-10-01	29700.580	1.108653

Oznaka "GDP" u FRED bazi podataka predstavlja seriju podataka o Bruto domaćem proizvodu za SAD i prikazane su u milijardama američkih dolara. Prilikom analize, prosječni kvartalni BDP može se izračunati korištenjem funkcije `.mean()` nad kolonom 'GDP', što omogućava brzu procjenu centralne tendencije u posmatranom periodu.

Za razumijevanje dinamike, kvartalne stope rasta (QoQ) izračunavaju se metodom `.pct_change()`, koja određuje relativnu promjenu trenutnog BDP-a u odnosu na prethodni kvartal, prema formuli: $(\text{GDP}_{\text{trenutni}} - \text{GDP}_{\text{prethodni}}) / \text{GDP}_{\text{prethodni}}$.

Množenjem rezultata sa 100 dobija se stopa rasta u procentima, koja se zatim dodaje u novu kolonu nazvanu 'QoQ_Growth_%' kako bi se vizualno naglasio trend promjene.

Nezaposlenost

Stopa nezaposlenosti je odnos broja nezaposlenih i ukupne radne snage. Visoka stopa nezaposlenosti u jednoj zemlji može biti indikator ekonomske recesije, dok niska stopa može ukazivati na ekonomski rast i veće ekonomske aktivnosti.

Podatke o nezaposlenosti možemo preuzeti s World Bank Open Data ili Eurostata (za EU). U SAD, jedan od najkorištenijih pokazatelja je *Unemployment Rate* sa FRED-a (oznaka [UNRATE](#)). Fred sadži i podatke od nekih značajnih svjetskih ekonomija.

Primjer analize podataka o nezaposlenosti u vodećim svjetskim ekonomijama sa podacima preuzetim iz online servisa FRED.

```
import pandas_datareader.data as web
import datetime

# Definiranje zemalja i njihovih FRED oznaka
countries = {
    "SAD": "UNRATE",
    "Njemačka": "LRUN64TTDEQ156S",
    "Francuska": "LRUN64TTFRQ156S",
    "Japan": "LRUN64TTJPQ156S"
}

start = datetime.datetime(2010, 1, 1)
end = datetime.datetime(2023, 12, 31)

# Analiza po zemljama
for country, code in countries.items():
    # Preuzimanje podataka
    unratedata = web.DataReader(code, "fred", start, end)

    avg_unemployment = unratedata[code].mean() # srednja vrijednost
    max_unemployment = unratedata[code].max() # maksimalna vrijenodt

    print(f"\n--- {country} ---")
    print(f"Prosječna stopa nezaposlenosti (2010-2023): {round(avg_unemployment, 2)}%")
    print(f"Maksimalna stopa: {round(max_unemployment, 2)}%")
    print("Posljednjih 5 mjeseci:")
    print(unratedata.tail()) ## bez tail() štampa sve rezultate
```

---	SAD	---
Prosječna stopa nezaposlenosti		
(2010-2023):	5.92%	
Maksimalna stopa:	14.8%	
Posljednjih 5 mjeseci:		
	UNRATE	
DATE		
2023-08-01	3.7	
2023-09-01	3.8	
2023-10-01	3.9	

---	Francuska	---
Prosječna stopa nezaposlenosti (2010-2023):		
9.11%		
Maksimalna stopa:	10.57%	
Posljednjih 5 mjeseci:		
	LRUN64TTFRQ156S	
DATE		
2022-10-01	7.206165	
2023-01-01	7.163929	
2023-04-01	7.396268	

2023-11-01	3.7	2023-07-01	7.448564
2023-12-01	3.8	2023-10-01	7.617982
--- Njemačka --			
Prosječna stopa nezaposlenosti (2010-2023): 4.25%			
Maksimalna stopa: 7.17%			
Posljednjih 5 mjeseci: LRUN64TTDEQ156S			
DATE			
2022-10-01	3.185016	2022-10-01	2.666667
2023-01-01	3.112914	2023-01-01	2.733333
2023-04-01	3.124425	2023-04-01	2.733333
2023-07-01	3.150754	2023-07-01	2.766667
2023-10-01	3.215580	2023-10-01	2.600000
--- Japan --			
Prosječna stopa nezaposlenosti (2010-2023): 3.49%			
Maksimalna stopa: 5.37%			
Posljednjih 5 mjeseci: LRUN64TTJPQ156S			
DATE			
2022-10-01	2.666667	2023-01-01	2.733333
2023-04-01	2.733333	2023-07-01	2.766667
2023-10-01	2.600000		

Primjer analize nezaposlenosti u zemljama regionala sa podacim iz online servisa svjetske banke.

```
import pandas_datareader.wb as wb
import pandas as pd

# World Bank indikator za stopu nezaposlenosti: SL.UEM.TOTL.ZS
countries_wb = {
    "Srbija": "SRB",
    "Hrvatska": "HRV",
    "Crna Gora": "MNE",
    "Makedonija": "MKD",
    "Bosna i Hercegovina": "BIH"
}

# Definiranje perioda
start = 2010
end = 2023

# Analiza po zemljama
for country, code in countries_wb.items():
    # Preuzimanje podataka
    data = wb.download(indicator="SL.UEM.TOTL.ZS", country=code, start=start,
    end=end)

    # Provjera da li postoje podaci
    if not data.empty:
        # Konverzija podataka u numerički format
        data['SL.UEM.TOTL.ZS'] = pd.to_numeric(data['SL.UEM.TOTL.ZS'],
        errors='coerce')

        # Računanje prosječne stope nezaposlenosti
        avg_unemployment = data['SL.UEM.TOTL.ZS'].mean()

        # Ispis rezultata
        print(f"\n--- {country} ---")
        print(f"Prosječna stopa nezaposlenosti ({start}-{end}):\n{round(avg_unemployment, 2)}%")
        print("Posljednjih 5 godina:")
        print(data) ## bez tail() štampa sve rezultate
    else:
        print(f"\nNema dostupnih podataka za {country}.")
```

--- Srbija ---
 Prosječna stopa nezaposlenosti (2010-2023): 15.19%
 Posljednjih 5 godina:
 SL.UEM.TOTL.ZS

country	year	SL.UEM.TOTL.ZS
Serbia	2023	8.268
	2022	8.439
	2021	9.843
	2020	9.011
	2019	10.395
	2018	12.733
	2017	13.477
	2016	15.257
	2015	17.660
	2014	19.223
	2013	22.154
	2012	24.000
	2011	22.959
	2010	19.204

--- Hrvatska ---
 Prosječna stopa nezaposlenosti (2010-2023): 11.39%
 Posljednjih 5 godina:
 SL.UEM.TOTL.ZS

country	year	SL.UEM.TOTL.ZS
Croatia	2023	6.086
	2022	6.960
	2021	7.608
	2020	7.507
	2019	6.617
	2018	8.432
	2017	11.208
	2016	13.103
	2015	16.175
	2014	17.290
	2013	17.251
	2012	15.935
	2011	13.678
	2010	11.620

--- Crna Gora ---
 Prosječna stopa nezaposlenosti (2010-2023): 17.32%
 Posljednjih 5 godina:
 SL.UEM.TOTL.ZS

country	year	SL.UEM.TOTL.ZS
Montenegro	2023	14.621
	2022	14.851
	2021	16.543
	2020	17.878
	2019	15.128
	2018	15.190
	2017	16.082
	2016	17.732
	2015	17.555
	2014	18.054
	2013	19.585
	2012	19.808
	2011	19.759
	2010	19.649

--- Makedonija ---
 Prosječna stopa nezaposlenosti (2010-2023): 23.23%
 Posljednjih 5 godina:
 SL.UEM.TOTL.ZS

country	year	SL.UEM.TOTL.ZS
North Macedonia	2023	13.174
	2022	14.476
	2021	15.801
	2020	16.575
	2019	17.414
	2018	21.208
	2017	22.857
	2016	24.312
	2015	26.395
	2014	28.215
	2013	29.017
	2012	31.096
	2011	31.502
	2010	33.135

--- Bosna i Hercegovina ---
 Prosječna stopa nezaposlenosti (2010-2023): 21.41%
 Posljednjih 5 godina:

country	year	SL.UEM.TOTL.ZS
Bosnia and Herzegovina	2023	10.668
	2022	12.657
	2021	14.897
	2020	15.868
	2019	15.692
	2018	18.400
	2017	20.527
	2016	25.408
	2015	27.695
	2014	27.517
	2013	27.490
	2012	28.010
	2011	27.582
	2010	27.312

Sljedeći primjer koristi Eurostat online servis i u njemu analiziramo podatke za Bosnu i Hercegovinu. Podaci su razvrstani po više kategorija. Za pristup podacima Eurostata, potrebno je instalirati modul `eurostat`. Komand za instaliranje modula je `pip install eurostat`.

```

import eurostat
import pandas as pd

# 1) Preuzimanje 'wide' formata, npr. unemployment dataset 'une_rt_a'
df_wide = eurostat.get_data_df('une_rt_a', flags=False)

# 2) Kolona s geografijom se zove 'geo\\TIME_PERIOD' -> preimenujemo u 'geo'
df_wide.rename(columns={'geo\\TIME_PERIOD': 'geo'}, inplace=True)

# 3) Pronalazimo sve kolone koje predstavljaju godine
year_columns = [col for col in df_wide.columns if col.isdigit()]

# 4) Spajanje svih kolona-godina u jednu kolonu 'year'
df_long = df_wide.melt(
    id_vars=['freq', 'age', 'unit', 'sex', 'geo'],
    value_vars=year_columns,
    var_name='year',
    value_name='value'
)

# 5) Filtriranje -> Bosna i Hercegovina (kod 'BA'), godina '2023' unit PC_ACT
df_ba_2023 = df_long[
    (df_long['geo'] == 'BA') &          # kod zemlje
    (df_long['year'] == '2023') &        # godina
    (df_long['unit'] == 'PC_ACT')       # Percentage of active population
]

print("\n--- Rezultati za Bosnu i Hercegovinu i godinu 2023. ---")
print(df_ba_2023)

--- Rezultati za Bosnu i Hercegovinu i godinu 2023. ---
   freq    age   unit sex geo  year  value
46621     A  Y15-24  PC_ACT    F  BA  2023   33.2
46658     A  Y15-24  PC_ACT    M  BA  2023   28.5
46695     A  Y15-24  PC_ACT    T  BA  2023   30.1
46954     A  Y15-29  PC_ACT    F  BA  2023   27.2
46991     A  Y15-29  PC_ACT    M  BA  2023   20.6
47028     A  Y15-29  PC_ACT    T  BA  2023   23.1
47287     A  Y15-74  PC_ACT    F  BA  2023   16.7
47324     A  Y15-74  PC_ACT    M  BA  2023   10.9
47361     A  Y15-74  PC_ACT    T  BA  2023   13.2
47620     A  Y20-64  PC_ACT    F  BA  2023   16.8
47657     A  Y20-64  PC_ACT    M  BA  2023   10.6
47694     A  Y20-64  PC_ACT    T  BA  2023   13.0
47953     A  Y25-54  PC_ACT    F  BA  2023   16.9
47990     A  Y25-54  PC_ACT    M  BA  2023    9.7
48027     A  Y25-54  PC_ACT    T  BA  2023   12.6
48286     A  Y25-74  PC_ACT    F  BA  2023   15.7
48323     A  Y25-74  PC_ACT    M  BA  2023    9.5
48360     A  Y25-74  PC_ACT    T  BA  2023   12.0
48619     A  Y55-74  PC_ACT    F  BA  2023    9.7
48656     A  Y55-74  PC_ACT    M  BA  2023    8.9
48693     A  Y55-74  PC_ACT    T  BA  2023    9.2

```

Opis koda:

- `df_wide=eurostat.get_data_df('une_rt_a', flags=False)`. Preuzima set podataka une_rt_a (Unemployment rates, annual) u “wide” formatu gdje su kolone 2003, 2004, ... itd.
- `rename(columns={'geo\\TIME_PERIOD':'geo'})`. Mjenja trenutno ime kolone geo\TIME_PERIOD u geo.
- `year_columns`. Pretražuje sve kolone koje su čisto brojevi (.isdigit()), pretpostavljajući da su to godine ('2003', '2004', ..., '2023').
- `df_wide.melt`. Pretvara “wide” format (gdje su godine kolone) u “long” format gdje year postaje jedna kolona, a vrijednosti idu u kolonu value.
- Filtriranje podataka:
 - `df_long['geo'] == 'BA'` : zadržava samo redove gdje je geo (država) = BA (Bosna i Hercegovina).
 - `df_long['year'] == '2023'` : zadržava samo godinu 2023.
 - `df_long['unit'] == 'PC_ACT'` : procenti aktivne populacije.

Kamatne stope

Kamatne stope centralnih banaka (npr. Fed za SAD, ECB za Evropu i sl.) od ključnog su značaja za vođenje monetarne politike, regulaciju novčane mase i usmjeravanje privrednih tokova. Praćenje trendova kamatnih stopa pomaže u razumijevanju tokova novca, kreditiranja i investicionih odluka, te u procjeni makroekonomskih trendova poput inflacije, rasta i stabilnosti finansijskog sistema.

U nastavku slijede pojednostavljeni primjeri, koji prezentuju mogućnosti preuzimanja i analize kamatne stope iz nekoliko različitih izvora kao što su FRED, yfinance i World Bank. Ovdje su prikazani osnovni koncepti rada sa online servisima, za potrebe detaljnijih analiza ponekad ovi izvori nisu dovoljni pa je neophodno koristiti i druge izvore podatka.

Primjer praćenje Federal Funds Rate (Fed) putem FRED-a. FRED (Federal Reserve Economic Data) pruža veliki broj serija podataka, uključujući i ključnu kamatnu stopu Fed-a: Effective Federal Funds Rate (oznaka FEDFUNDS).

```
import datetime
import pandas_datareader.data as web

start = datetime.datetime(2024, 1, 1)
end = datetime.datetime(2024, 12, 31)

# Preuzimanje Effective Federal Funds Rate (FEDFUNDS) sa FRED-a
fed_funds = web.DataReader("FEDFUNDS", "fred", start, end)

# Ispis posljednjih 5 vrijednosti
print("Mjesečne vrijednosti FEDFUNDS:\n", fed_funds.tail(12)) # 12 mjeseci

# Jednostavna statistika
```

```

average_rate = fed_funds["FEDFUNDS"].mean()
print(f"\nProsječna kamatna stopa (FEDFUNDS) za 2024.: {average_rate:.2f}%")


Mjesečne vrijednosti FEDFUNDS:
    FEDFUNDS
DATE
2024-01-01      5.33
2024-02-01      5.33
2024-03-01      5.33
2024-04-01      5.33
2024-05-01      5.33
2024-06-01      5.33
2024-07-01      5.33
2024-08-01      5.33
2024-09-01      5.13
2024-10-01      4.83
2024-11-01      4.64
2024-12-01      4.48
Prosječna kamatna stopa (FEDFUNDS) za 2024.: 5.14%

```

Primjer praćenje prinosa na obveznice putem yfinance (primjer za SAD). Biblioteka yfinance nema direktnu podršku za kamatne stope centralnih banaka, ali možemo posredno pratiti kretanja kamatnih stopa posmatrajući prinose na državne obveznice (*US 10-Year Treasury Yield*). Ovo nije isto što i zvanična kamatna stopa Fed-a, ali u praksi često služi kao referenca za dugoročno kretanje kamatnih stopa.

Simbol za 10-godišnji prinos (yield) na američke državne obveznice na Yahoo Finance je ^TNX. Simbol za 30-godišnji prinos je ^TYX.

```

import yfinance as yf

# Preuzimanje podataka o 10-godišnjem prinosu na obveznice (USD)
tnx_data = yf.download("^TNX", period="1y", interval="1d")
# Ispis nekoliko posljednjih vrijednosti
print("\n--- 10-godišnji prinos na američke državne obveznice (posl. 5) ---")
print(tnx_data.tail(15))

# Brza statistika
avg_tnx = tnx_data["Close"].mean()
print(f"\nProsječni 10-godišnji prinos u posljednjih godinu dana: {avg_tnx}%")


--- 10-godišnji prinos na američke državne obveznice (posl. 8) ---
Price           Adj Close   Close   High    Low   Open Volume
Ticker          ^TNX       ^TNX     ^TNX    ^TNX   ^TNX   ^TNX
Date
2025-01-29 00:00:00+00:00  4.555   4.555   4.591   4.510   4.526   0
2025-01-30 00:00:00+00:00  4.512   4.512   4.537   4.488   4.494   0
2025-01-31 00:00:00+00:00  4.569   4.569   4.569   4.502   4.525   0
2025-02-03 00:00:00+00:00  4.543   4.543   4.557   4.462   4.506   0
2025-02-04 00:00:00+00:00  4.513   4.513   4.598   4.509   4.577   0
2025-02-05 00:00:00+00:00  4.422   4.422   4.474   4.400   4.464   0
2025-02-06 00:00:00+00:00  4.440   4.440   4.460   4.412   4.448   0
2025-02-07 00:00:00+00:00  4.487   4.487   4.513   4.398   4.444   0
Prosječni 10-godišnji prinos u posljednjih godinu dana: Ticker
^TNX        4.261333

```

Primjer upotrebe World Bank baze za kamatne stope. World Bank nudi više indikatora vezanih za kamate, poput real interest rate (FR.INR.RINR), lending interest rate (FR.INR.LEND) ili deposit interest rate (FR.INR.DPST). U zavisnosti od toga šta tražite (npr. kamatne stope na kredite, depozite, itd.), birate odgovarajući indikator. U nastavku će se korisiti pandas_datareader.wb za preuzimanje lending interest rate (FR.INR.LEND) za nekoliko zemalja (SAD, Rusija, Italija i Japan).

```
import pandas as pd
from pandas_datareader import wb

# Oznaka indikatora za Lending Interest Rate (%)
indicator = "FR.INR.LEND"

countries = ["US", "JP", "IT", "RU"] # ISO kodovi zemalja

# Preuzimanje godišnjih podataka (World Bank obično nudi godišnje vrijednosti)
df_wb = wb.download(
    indicator=indicator,
    country=countries,
    start=2013,
    end=2017
)

# Preimenovanje kolone indikatora radi čitljivosti
df_wb.rename(columns={indicator: "LendingRate"}, inplace=True)

print("\n--- Lending interest rate (%): prikaz prvih 30 redova ---")
print(df_wb.head(30))
```

```
--- Lending interest rate (%): prikaz prvih 30 redova ---
   LendingRate
country      year
Italy        2017    3.000833
              2016    3.496167
              2015    4.129250
              2014    4.866667
              2013    5.144167
Japan        2017    0.994000
              2016    1.044750
              2015    1.142833
              2014    1.219167
              2013    1.303750
Russian Federation 2017  10.560000
                  2016  12.598333
                  2015  15.716667
                  2014  11.141667
                  2013  9.466667
United States   2017  4.096667
                  2016  3.511667
                  2015  3.260000
                  2014  3.250000
                  2013  3.250000
```

6.4.1.3. Praćenje poslovnih ciklusa

Uspješna analiza poslovnih ciklusa podrazumijeva i prepoznavanje kratkoročnih fluktuacija, odnosno sezonskih komponenti. Na primjer, potražnja za odjećom, hranom ili uslugama može tokom praznika privremeno skočiti, što može zamagliti dugoročni trend.

- Trend: predstavlja glavni, opšti smjer kretanja kroz duži vremenski period (rast ili pad).
- Sezonsko stanje: ponavljajući obrasci koji se javljaju u pravilnim vremenskim intervalima (npr. povećanje prodaje pred novogodišnje praznike).
- Nasumična (rezidualna) komponenta: ono što preostane nakon izdvajanja trenda i sezonalnosti, a obuhvata kratkoročne i nepredvidive oscilacije.

Ako se želi ocijeniti dugoročni rasta ili pad (trend), privremeni skokovi u potražnji mogu dovesti do pogrešnih zaključaka. Sezonske promjene pomažu da se sagleda realno kretanje, posebno kada se želi uporediti kvartalna ili mjesecna kretanja kroz više godina.

Za analizu poslovnih ciklusa koristi se modul `statsmodels`, koji služi za statističku analizu i modeliranje vremenskih serija. Omogućava napredne metode regresije, testiranje hipoteza, dekompoziciju vremenskih serija i analizu sezonalnosti. Često se koristi za ekonometriju, finansijske analize i predviđanje.

Ovo je dosta kompleksan modul i u ovoj knjizi će biti obrađene samo osnovne funkcije. Modul je potrebno instalirati komandom: `pip install statsmodels`

U sljedećem primjeru izvršena je analiza mjesecnih podataka o maloprodaji (retail sales) iz FRED baze (u ovom slučaju američki Retail Sales, oznaka `RSXFS`).

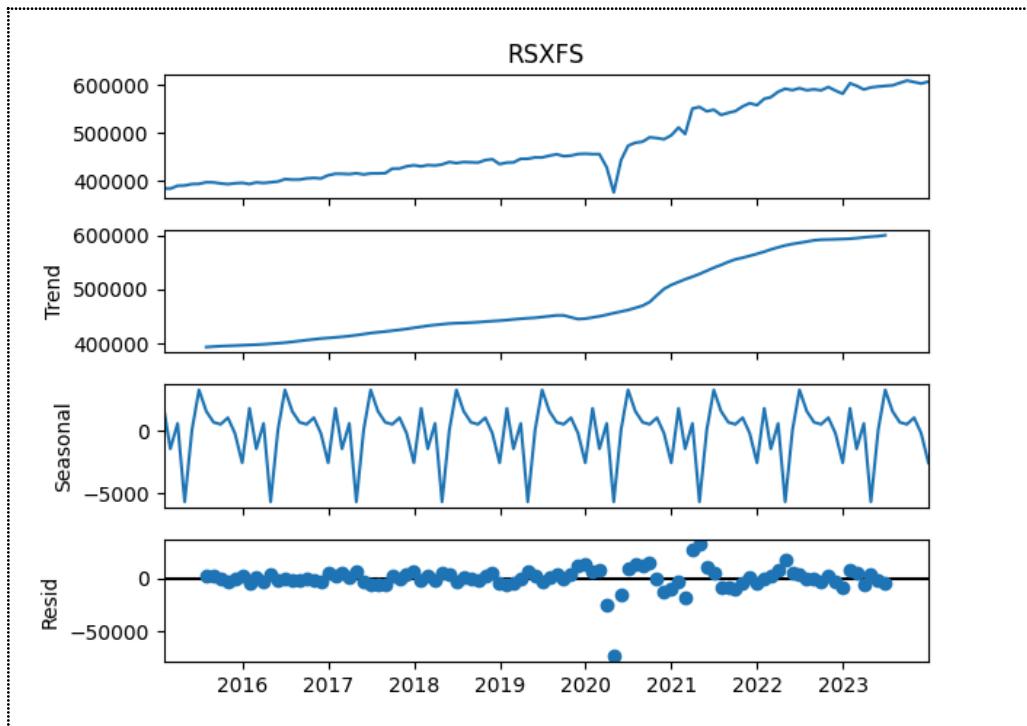
```
import datetime
import pandas_datareader.data as web
import statsmodels.api as sm
import matplotlib.pyplot as plt

# Definišemo period
start = datetime.datetime(2015, 1, 1)
end = datetime.datetime(2023, 12, 31)

# Preuzimanje mjesecnih podataka o maloprodaji (Retail Sales, 'RSXFS')
retail_data = web.DataReader("RSXFS", "fred", start, end)

# Pretvaramo u mjesecnu vremensku seriju -> prosječna vrijednost po mjesecu
retail_series = retail_data['RSXFS'].resample('M').mean()

# Dekompozicija vremenske serije (additive model)
decomposition = sm.tsa.seasonal_decompose(retail_series.dropna(), model='additive')
decomposition.plot()
plt.show()
```



Objašnjenje 4 dobijena grafika:

- Originalna serija: sirovi podaci maloprodaje.
- Trend: dugoročni pravac kretanja.
- Seasonal: sezonski obrazac koji se ponavlja tokom godine.
- Residual: preostala, neobjašnjiva komponenta (nasumični šum).

Tumačenje podataka: Dijagram sezonskog uticaja pokazuje značajnu oscilaciju. Vrijednosti se kreću između -5000 i +1000, što odgovara značajnom uticaju sezonske prodaje na ukupan tren prodaje. U ovom slučaju treba posmatrati trend prodaje u dužem vremenskom okviru.

Primjer analiza cijena nafte (WTI crude oil). Cijena nafte (WTI crude oil, simbol DCOILWTICO) može imati sezonske varijacije zbog povećane potrošnje tokom zimskih mjeseci (za grijanje) i ljetnih mjeseci (za transport). Takođe, i geopolitički faktori mogu da utiču na proizvodnju i snabdijevanje.

Iz FRED baze su preuzete mjesečne cijene i analizirani trendovi i sezonsko stanje.

```
import datetime
import pandas_datareader.data as web
import statsmodels.api as sm
import matplotlib.pyplot as plt
```

```

# Definisanje vremenskog perioda analize
start = datetime.datetime(2015, 1, 1)
end = datetime.datetime(2023, 12, 31)

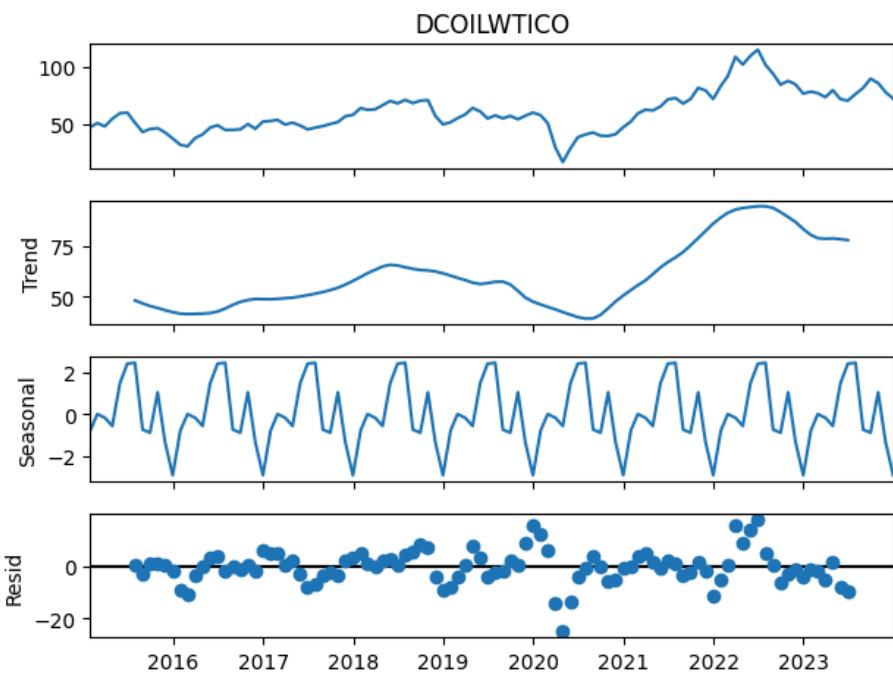
# Preuzimanje podataka o cijeni sirove nafte (West Texas Intermediate - WTI)
oil_data = web.DataReader("DCOILWTICO", "fred", start, end)

# Pretvaramo u mjesecnu vremensku seriju (prosječne cijene po mjesecu)
oil_series = oil_data['DCOILWTICO'].resample('M').mean()

# Dekompozicija vremenske serije (additive model)
decomposition = sm.tsa.seasonal_decompose(oil_series.dropna(), model='additive')

# Prikaz grafičkih rezultata
decomposition.plot()
plt.show()

```



Tumačenje rezultat: Sezonska komponenta (dijagram 3 u primjeru iznad) oscilira između -2 i 2, to ukazuje da cijena nafte nema izražene sezonske varijacije. Cijene se neznatno povećavaju i smanjuju u određenim periodima godine, ali bez jakih cikličnih obrazaca.

6.4.2. Regresiona i korelaciona analiza

Regresiona i korelaciona analiza predstavljaju osnovne statističke alate za kvantifikovanje odnosa između različitih ekonomskih varijabli. Kroz korelaciju,

dobijamo mjeru jačine i smjera međusobne povezanosti, dok nam regresija omogućava da predvidimo kako će promjena u jednoj varijabli uticati na drugu. Ove tehnike su nezamjenjive u ekonomiji počevši od mikroekonomskih pitanja (npr. odnos plate i produktivnosti) do makroekonomskih veza (npr. inflacija i nezaposlenost).

Ključne primjene:

- Procjena odnosa između varijabli: Na primjer, kako cijena proizvoda utiče na traženu količinu ili kako se trend BDP-a mijenja sa investicijama.
- Primjena u mikroekonomiji: Odnos plate i produktivnosti, troškova i profita, potrošnje i dohotka.
- Primjena u makroekonomiji: Analiza povezanosti između inflacije i nezaposlenosti, BDP-a i investicija, kamatnih stopa i rasta ekonomije.

Prvi korak u analizi podrazumijeva **proračun korelacije** između ovih varijabli, čime se dobija uvid u pravac i jačinu međusobnih odnosa. Nakon toga, primjeničemo **regresionu analizu**, kako bismo kvantifikovali konkretan međusobni uticaj posmatranih parametara. Korišćenjem OLS metode moći ćemo da procijenimo u kojoj mjeri promjene jednog parametra utiću na druge parametre, što može biti korisno za predviđanje makroekonomskih trendova i donošenje ekonomskih politika zasnovanih na podacima.

6.4.2.1. Korelaciona analiza

Korelaciona analiza se svodi na mjerjenje stepena u kojem su dvije ili više varijabli linearno povezane. Najčešće korišćena mjera je Pearsonov koeficijent korelacijske (raspon od -1 do 1), pa tako razlikujemo sljedeće korelacijske:

- Teži ka 1: jaka pozitivna korelacija (varijable rastu zajedno).
- Teži ka -1: jaka negativna korelacija (kada jedna raste, druga opada).
- Teži ka 0: nema značajne linearne povezanosti.

Primjer korelacijske matrice gdje iz FRED-a baze preuzimamo mjesecne podataka o industrijskoj proizvodnji (INDPRO), Indeksu potrošačkih cijena CPI (CPIAUCSL) i nezaposlenosti (UNRATE) u SAD, a zatim računamo korelaciju između tih varijabli.

```
import datetime
import pandas as pd
import pandas_datareader.data as web

# Definišemo period (recimo, 24 mjeseca unazad, 2021-01 do 2022-12)
start = datetime.datetime(2021, 1, 1)
end = datetime.datetime(2022, 12, 31)

# FRED kodovi za:
#   - Industrijska proizvodnja (INDPRO)
#   - CPI za sve potrošače (CPIAUCSL)
#   - Stopa nezaposlenosti (UNRATE)
```

```

fred_series = {
    "IndustrialProduction": "INDPRO",
    "CPI": "CPIAUCSL",
    "Unemployment": "UNRATE"
}

# Preuzimanje podataka sa FRED-a
data_frames = {}
for col_name, fred_code in fred_series.items():
    df_temp = web.DataReader(fred_code, "fred", start, end)
    df_temp.rename(columns={fred_code: col_name}, inplace=True)
    data_frames[col_name] = df_temp

# Spajamo sve serije u jedan DataFrame po datumu
df_real = pd.concat(data_frames.values(), axis=1)

print("\n--- Prvih 5 redova stvarnih podataka: ---")
print(df_real.head())

# Korelaciona matrica
corr_matrix = df_real.corr()
print("\n--- Korelaciona matrica: ---")
print(corr_matrix)

--- Prvih 5 redova stvarnih podataka: ---
      IndustrialProduction      CPI  Unemployment
DATE
2021-01-01           98.8135   262.518       6.4
2021-02-01           95.5072   263.583       6.2
2021-03-01           98.1929   264.910       6.1
2021-04-01           98.3317   266.752       6.1
2021-05-01           99.1867   268.452       5.8

--- Korelaciona matrica: ---
      IndustrialProduction      CPI  Unemployment
IndustrialProduction           1.000000   0.921919     -0.910725
CPI                         0.921919   1.000000     -0.939250
Unemployment                 -0.910725  -0.939250      1.000000

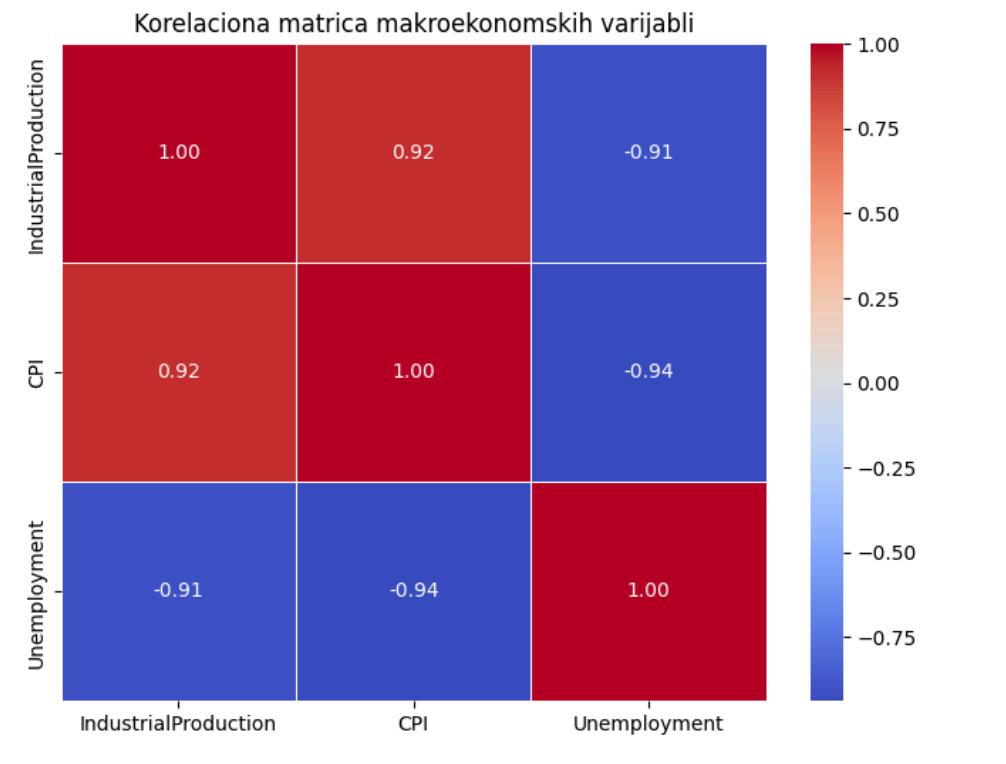
```

Tumačenje rezultata:

- IndustrialProduction – CPI = 0.921919 : Visoka pozitivna korelacija (preko 0.9). Znači da kada industrijska proizvodnja raste, istovremeno raste i CPI (indeks potrošačkih cijena). U praksi, ovo može sugerisati da povećane ekonomske aktivnosti (veća proizvodnja) idu ruku pod ruku sa rastom cijena.
- IndustrialProduction – Unemployment = -0.910725: Visoka negativna korelacija (oko -0.91). Kako industrijska proizvodnja raste, nezaposlenost pada. Suprotno, kada industrijska proizvodnja opada, nezaposlenost raste. Ovaj rezultat je u skladu s klasičnom makroekonomijom, gdje veća ekonomska aktivnost dovodi do većeg zapošljavanja.
- CPI–Unemployment=-0.939250: Takođe veoma visoka negativna korelacija. Viša inflacija (CPI) je u ovom uzorku povezana sa nižom stopom nezaposlenosti i obrnuto.

Korlaciona analiza nam kao rezultat daje korelacionu matricu, iz koje donosimo zaključke o međusobnom uticaju posmatranih parametara. Ukoliko bi grafički predstavili korelacionu matricu pomoću *Heatmap* dijagrama, mogli bi mnogo jasnije da vidimo međusobne uticaje posmatranih parametara. Sljedeći kod predstavlja nastavak prethodnog koda i crta *Heatmp* dijagram.

```
# Vizualizacija korelacije pomoću heatmap-a
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)
plt.title("Korelaciona matrica makroekonomskih varijabli")
plt.show()
```



6.4.2.2. Regresiona analiza

Regresiona analiza ide korak dalje od korelacijske analize. Dok korelacija kaže samo kolika je linearna veza, regresija služi za modeliranje odnosa između nezavisnih varijabli i zavisne varijable, uz mogućnost predviđanja.

Jednostavna linearna regresija ima oblik:

$$y = \beta_0 + \beta_1 x + \epsilon$$

gdje je:

- y – zavisna varijabla (npr. potrošnja, cijena, dohodak itd.),
- x – nezavisna varijabla (npr. dohodak, količina itd.),
- β_0 – intercept (presjek na y-osi),
- β_1 – nagib (mjeri uticaj x na y),
- ϵ – nasumična greška.

U ekonomskoj analizi često je potrebno kvantifikovati odnose između različitih makroekonomskih varijabli, kako bi se bolje razumjele međusobne povezanosti i omogućile preciznije ekonomske prognoze. Jedan od najčešće korišćenih pristupa za modelovanje ovih odnosa je OLS metoda (*Ordinary Least Squares* – Metoda najmanjih kvadrata), koji predstavlja osnovni statistički alat u linearnim regresionim analizama. Ova metoda omogućava procjenu stepena uticaja nezavisnih varijabli na zavisnu varijablu, što ga čini nezamjenjivim u ekonometrijskim istraživanjima.

Jedna od ključnih prednosti OLS metode leži u njenoj jednostavnoj interpretaciji, jer omogućava jasno razumijevanje uticaja svake nezavisne varijable na zavisnu varijablu. Takođe, koristi se za procjenu uzročnih veza između ekonomskih pokazatelja, čime pomaže u razumijevanju mehanizama, koji stoje iza promjena u ekonomskim agregatima. Na primjer, ako želimo analizirati kako promjene u industrijskoj proizvodnji utiću na stopu nezaposlenosti, OLS metoda pruža matematički okvir za precizno kvantifikovanje tog odnosa.

Primjer upotrebe OLS metode na stvarnim podacima iz FRED baze koji su početno korišteni u korelacionoj analizi, odnosno analizirajmo odnose između industrijske proizvodnje (INDPRO), indeksa potrošačkih cijena (CPIAUCSL) i stope nezaposlenosti (UNRATE), kako bismo kvantifikovali uticaj industrijske proizvodnje na nezaposlenost.

Napomena: Kod koji slijedi je nastavak na prethodna dva primjera u kojima smo na istom preuzetom setu podataka iz FRED baze uradili korelacionu analizu i crtali *Heatmap* dijagram. Ukoliko se sastave sva tri dijela koda dobija se skripta, koja radi regresionu i korelacionu analizu nad definisanim skupom podataka.

```
import statsmodels.api as sm

# Definisanje zavisne i nezavisne varijable
y = df["Unemployment"] # Stopa nezaposlenosti (zavisna varijabla)
X = df["IndustrialProduction"] # Industrijska proizvodnja (nezavisna varijabla)

# Dodavanje konstante (intercept) u model
X_with_const = sm.add_constant(X)

# Kreiranje i fitovanje OLS modela
model = sm.OLS(y, X_with_const).fit()

# Prikaz rezultata regresije
print(model.summary())
```

OLS Regression Results						
Dep. Variable:	Unemployment	R-squared:	0.799			
Model:	OLS	Adj. R-squared:	0.798			
Method:	Least Squares	F-statistic:	707.1			
Date:	Wed, 12 Feb 2025	Prob (F-statistic):	6.72e-64			
Time:	01:03:46	Log-Likelihood:	-255.46			
No. Observations:	180	AIC:	514.9			
Df Residuals:	178	BIC:	521.3			
Df Model:	1					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
const	58.7171	1.991	29.485	0.000	54.787	62.647
IndustrialProduction	-0.5317	0.020	-26.592	0.000	-0.571	-0.492
Omnibus:	9.121	Durbin-Watson:	0.218			
Prob(Omnibus):	0.010	Jarque-Bera (JB):	5.973			
Skew:	0.299	Prob(JB):	0.0505			
Kurtosis:	2.338	Cond. No.	2.65e+03			

Analiza rezultata OLS regresije:

1) Kvalitet modela (R-squared i F-statistika)

- R-squared = 0.799 : Oko 79.9% varijabilnosti u stopi nezaposlenosti može se objasniti kretanjem industrijske proizvodnje. Ovo pokazuje visok nivo prihvatljivosti modela, što znači da postoji jak odnos između ove dvije varijable.
- Adj. R-squared = 0.798 : Skoro identičan R², što znači da dodatne nezavisne varijable vjerovatno ne bi značajno poboljšale model.
- F-statistic = 707.1, Prob(F-statistic) = 6.72e-64 : Izuzetno visoka vrijednost F-statistike i p-vrijednost praktično nula (6.72e-64) potvrđuju da je model statistički značajan.

2) Koeficijenti i njihova interpretacija

Varijabla	Koeficijent (coef)	t-statistika	p-vrijednost	95% interval pouzdanosti
Intercept (const)	58.7171	29.485	0.000	[54.787, 62.647]
IndustrialProduct.	-0.5317	-26.592	0.000	[-0.571, -0.492]

- Intercept (58.7171) : Kada bi industrijska proizvodnja bila 0, očekivana stopa nezaposlenosti bila bi 58.72%. Ovo nema praktično značenje jer industrijska proizvodnja nikada nije nula, ali pokazuje osnovni nivo nezaposlenosti kada ne bi imali proizvodnju.
- Koeficijent za IndustrialProduct (-0.5317) : Svako povećanje industrijske proizvodnje za 1 indeksni poen dovodi do smanjenja nezaposlenosti za 0.53

procenatnih poena. Negativan znak koeficijenta potvrđuje inverznu vezu između industrijske proizvodnje i nezaposlenosti: kada privreda proizvodi više, više ljudi je zaposleno.

- t-statistika (-26.592) je visoko negativna, što ukazuje da je ovaj koeficijent izuzetno značajan ($p\text{-vrijednost} < 0.001$).
- Interval pouzdanosti [-0.571, -0.492] pokazuje da smo 95% sigurni da je pravi efekat u ovom opsegu.

3) Dijagnostičke statistike

- Durbin-Watson = 0.218 : Ova vrijednost je daleko od idealne vrijednosti 2, što sugerira da postoji autokorelacija. Autokorelacija znači da se greške u modelu možda ponavljaju kroz vrijeme, što može dovesti do pristrasnih procjena.
- Omnibus test ($\text{Prob} = 0.010$) i Jarque-Bera test ($\text{Prob} = 0.0505$) : Niske p-vrijednosti sugeriraju da greške vjerovatno nisu savršeno normalno distribuirane. Ovo može uticati na pouzdanost standardnih grešaka i p-vrijednosti.
- Condition Number = $2.65e+03$ (2,650): Veća vrijednost ukazuje na moguću multikolinearnost ili probleme sa skaliranjem podataka. U ovom slučaju, nije ekstremno visok broj (npr. 10,000+ bi bio alarmant), ali vrijedi dodatno provjeriti.

Na osnovu prethodne analize koja je rađena na skupu podataka dostupnih u FRED bazi, a koji opisuju indeks industrijske proizvodnje, indeksa potrošačkih cijena i stope nezaposlenosti u SAD za period 2021. do 2022. godina može se zaključiti sljedeće:

- Industrijska proizvodnja ima značajan negativan uticaj na nezaposlenost, što je u skladu sa ekonomskom teorijom.
- Model veoma dobro objašnjava podatke ($R^2 = 79.9\%$), što znači da industrijska proizvodnja zaista igra veliku ulogu u predviđanju nezaposlenosti.
- Mogući problemi uključuju autokorelaciju i blage greške modela, što može uticati na preciznost modela.
- Multikolinearnost nije veliki problem, ali vrijedi provjeriti ako dodamo još varijabli u budućem modelu.

6.4.3. Prognoziranje u ekonomiji (Forecasting)

Prognoziranje ekonomskih varijabli je složen, ali neophodan dio svake ekonomiske analize, budući da pomaže u planiranju monetarne politike, investicijama i strateškom poslovnom odlučivanju. Kao važan alat koji unapređuje kvalitet donošenja odluka, prognoziranje omogućava da se na osnovu istorijskih podataka i uočenih trendova formiraju očekivanja o budućim kretanjima ključnih pokazatelja, poput inflacije, BDP-a, kamatnih stopa, nezaposlenosti i drugih indikatora finansijskih tržišta. Iako se u praksi primjenjuju različiti pristupi, od jednostavnih metoda ekstrapolacije do složenih ekonometrijskih i AI modela, zajednički cilj je procijeniti vjerovatni ishod budućih događaja i unaprijed se pripremiti za promjene u ekonomskom okruženju.

U osnovi, metode prognoziranja podijeljene su na:

- **Vremenske serije**, koje analiziraju evoluciju jedne ili više varijabli kroz duži vremenski period,
- **Jednostavne metode ekstrapolacije** koje se oslanjaju na osnovne istorijske trendove,
- **Složenije ekonometrijske modele** poput ARIMA ili različitih regresionih modela i neuronskih mreža, koji pokušavaju objasniti međuzavisnost varijabli i na taj način dati preciznije predviđanje.

U nastavku će kroz primjere u Pythonu biti opisana svaka metoda posebno, tako što se prilagodi kontekst ekonomskom prognoziranju.

6.4.3.1. Jednostavne metode ekstrapolacije

Kada je potrebno brzo procijeniti budući trend ekonomске varijable, analitičari često koriste jednostavne metode ekstrapolacije, poput pokretnog prosjeka, eksponencijalnog poliranja i linearne ekstrapolacije. Ove metode su brze za implementaciju, ne zahtijevaju složene statističke modele i pogodne su za kratkoročna predviđanja, posebno kada su podaci stabilni i bez izražene sezonalnosti. Rezultati su intuitivni i lako razumljivi, što ih čini korisnim za donosioce odluka.

Međutim, jednostavne metode imaju i nedostatke. One ne uzimaju u obzir nagle promjene u ekonomskim uslovima, ne prepoznaju ciklične obrasce i nisu pogodne za dugoročne predikcije. Zbog toga se ekstrapolacija često koristi kao prvi korak u analizi, nakon čega slijedi validacija rezultata i primjena složenijih modela, poput ARIMA ili mašinskog učenja, kako bi se dobole tačnije i pouzdanoje procjene. Ova kombinacija omogućava bolje razumijevanje trendova i donošenje ispravnih odluka.

Pokretni prosjek (Moving Average - MA)

Pokretni prosjek se izračunava kao prosječna vrijednost varijable u određenom vremenskom periodu (npr. 3, 6 ili 12 mjeseci). Ova metoda omogućava poravnanje (peglanje) podataka i uklanjanje kratkoročnih fluktuacija.

U Pandas modulu pokretni prosjek se može izračunati pomoću metode `pd.rolling().mean()`.

U nastavku je primjer analiza cijene nafte (WTI) koristeći pokretni prosjek iz FRED baze. Na dijagramu će biti prikazan stvarne cijene, kao i vrijednosti koje su obrađene upotrebom pokretnog prosjeka za vremenske periode 3 i 12 mjeseci.

```
import datetime
import pandas as pd
import pandas_datareader.data as web
import matplotlib.pyplot as plt

# Definišemo period analize
start = datetime.datetime(2015, 1, 1)
end = datetime.datetime(2024, 12, 31)

# Preuzimanje mjesečnih podataka o cijeni sirove nafte (WTI) iz FRED baze
oil_data = web.DataReader("DCOILWTICO", "fred", start, end)

# Izračunavanje 3-mjesečnog i 12-mjesečnog pokretnog prosjeka
oil_data['MA_3'] = oil_data['DCOILWTICO'].rolling(window=3).mean()
oil_data['MA_12'] = oil_data['DCOILWTICO'].rolling(window=12).mean()

# Prikaza podataka
print(oil_data.head(100))

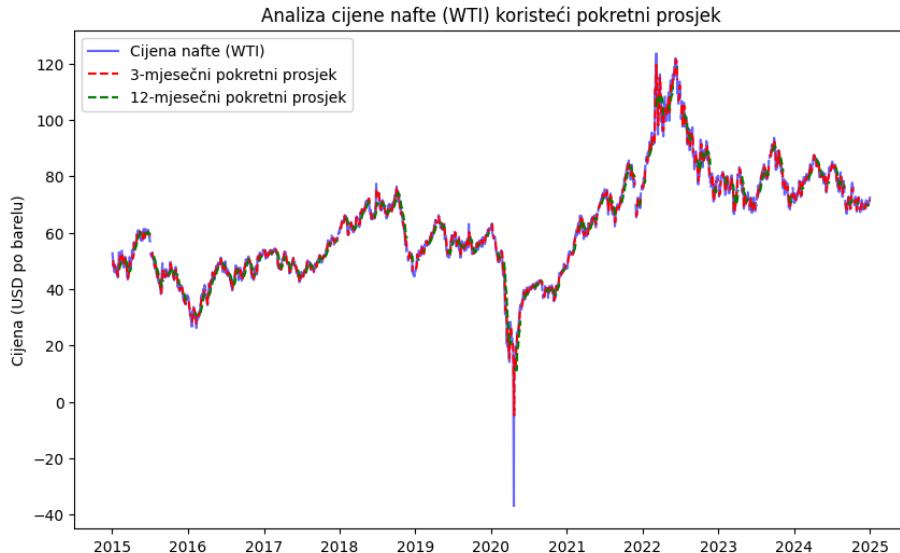
# Isrtavanje rezultata
plt.figure(figsize=(10,6))
plt.plot(oil_data.index, oil_data['DCOILWTICO'], label='Cijena nafte (WTI)', color='blue', alpha=0.6)
plt.plot(oil_data.index, oil_data['MA_3'], label='3-mjesečni pokretni prosjek', color='red', linestyle='--')
plt.plot(oil_data.index, oil_data['MA_12'], label='12-mjesečni pokretni prosjek', color='green', linestyle='--')

plt.title("Analiza cijene nafte (WTI) koristeći pokretni prosjek")
plt.ylabel("Cijena (USD po barelu)")
plt.legend()
plt.show()
```

	DCOILWTICO	MA_3	MA_12
DATE			
2015-01-01	NaN	NaN	NaN
2015-01-02	52.72	NaN	NaN
2015-01-05	50.05	NaN	NaN
2015-01-06	47.98	50.250000	NaN
2015-01-07	48.69	48.906667	NaN
...
2015-05-14	59.89	60.370000	59.686667
2015-05-15	59.73	60.040000	59.785000

2015-05-18	59.44	59.686667	59.770000
2015-05-19	57.30	58.823333	59.620000
2015-05-20	58.96	58.566667	59.623333

[100 rows x 3 columns]



Eksponencijalno poliranje (Exponential Smoothing - ES)

Eksponencijalno poliranje je napredniji oblik pokretnog prosjeka, koji pridaje veću težinu novijim podacima, čime bolje prati promjene u trendu. Ova metoda je pogodna kada se želi dati prioritet novim informacijama, ali i dalje zadržati istorijske obrasce.

Eksponencijalno poliranje može se koristiti pomoću funkcije `SimpleExpSmoothing()` Simple Exponential Smoothing (SES) iz `statsmodels` modela.

Primjer analiza indeksa potrošačkih cijena (CPI) koristeći eksponencijalno poliranje podataka iz FRED baze.

```
from statsmodels.tsa.holtwinters import SimpleExpSmoothing
import pandas as pd
import matplotlib.pyplot as plt
import pandas_datareader.data as web
import datetime

# Definišemo period analize
start = datetime.datetime(2015, 1, 1)
end = datetime.datetime(2024, 12, 31)

# Preuzimanje podataka o CPI (Indeks potrošačkih cijena) iz FRED baze
cpi_data = web.DataReader("CPIAUCSL", "fred", start, end)
```

```

# Eksplisitno definijemo frekvenciju podataka (mjesečna frekvencija)
cpi_data = cpi_data.asfreq('MS') # 'MS' = Month Start

# Kreiranje modela eksponencijalnog poliranje (alpha = 0.2)
model = SimpleExpSmoothing(cpi_data['CPIAUCSL'])
fit = model.fit(smoothing_level=0.2, optimized=False)

# Dodavanje izglađene serije u DataFrame
cpi_data['Exp_Smooth'] = fit.fittedvalues

# Prikaz podataka
print(cpi_data)

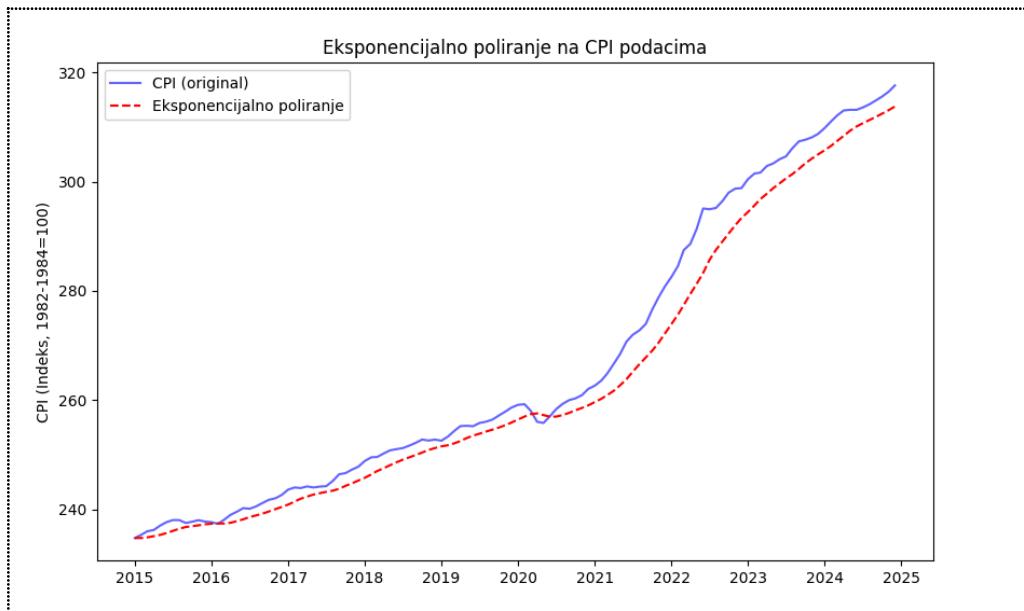
# Isrtavanje rezultata
plt.figure(figsize=(10,6))
plt.plot(cpi_data.index, cpi_data['CPIAUCSL'], label='CPI (original)', color='blue',
alpha=0.6)
plt.plot(cpi_data.index, cpi_data['Exp_Smooth'], label='Eksponencijalno poliranje',
color='red', linestyle='--')

plt.title("Eksponencijalno poliranje na CPI podacima")
plt.ylabel("CPI (Indeks, 1982-1984=100)")
plt.legend()
plt.show()

```

	CPIAUCSL	Exp_Smooth
DATE		
2015-01-01	234.747	234.747000
2015-02-01	235.342	234.747000
2015-03-01	235.976	234.866000
2015-04-01	236.222	235.088000
2015-05-01	237.001	235.314800
...
2024-08-01	314.131	311.258612
2024-09-01	314.851	311.833090
2024-10-01	315.564	312.436672
2024-11-01	316.449	313.062137
2024-12-01	317.603	313.739510

[120 rows x 2 columns]



Na dobijenom dijagramu:

- Plava linija prikazuje originalne podatke o CPI.
- Crvena linija prikazuje eksponencijalno izglađenu verziju, koja daje veću težinu novijim podacima.

Linearna ekstrapolacija (Trend Projection)

Ako vremenska serija pokazuje stalan trend, možemo koristiti jednostavnu linearu regresiju za ekstrapolaciju budućih vrijednosti. Ova metoda se često koristi kada nema izraženog skokova i kada želimo okvirnu procjenu budućih trendova.

Primjer prognoze nezaposlenosti u SAD koristeći linearu regresiju iz FRED baze.

```

import numpy as np
from sklearn.linear_model import LinearRegression

# Preuzimanje podataka o nezaposlenosti (UNRATE) iz FRED baze
unemp_data = web.DataReader("UNRATE", "fred", start, end)

# Provjera da li su podaci učitani ispravno
print("Originalni CPI podaci:")
print(unemp_data.head()) # Ispis prvih nekoliko redova

# Konverzija datuma u numeričke vrijednosti (potrebno za regresiju)
unemp_data['Time'] = np.arange(len(unemp_data))

# Kreiranje linearog regresionog modela
X = unemp_data[['Time']]
y = unemp_data['UNRATE']

```

```

model = LinearRegression()
model.fit(X, y)

# Ekstrapolacija za narednih 12 mjeseci
future_X = np.arange(len(unemp_data), len(unemp_data) + 12).reshape(-1, 1)
future_predictions = model.predict(future_X)

# Ispis ekstrapoliranih podataka
print("\nEkstrapolirane vrijednosti CPI za narednih 12 mjeseci od",
      extrapolation_start_date.date(), ":")
print(future_predictions)

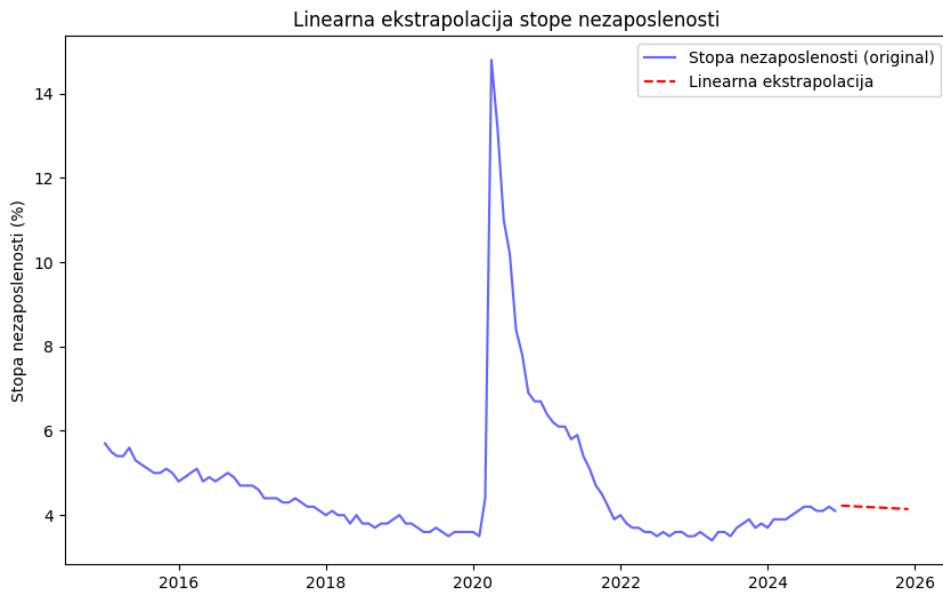
# IsCRTavanje rezultata
plt.figure(figsize=(10,6))
plt.plot(unemp_data.index, unemp_data['UNRATE'], label='Stopa nezaposlenosti (original)', color='blue', alpha=0.6)
plt.plot(pd.date_range(start=end, periods=12, freq='ME'), future_predictions, label='Linearna ekstrapolacija', color='red', linestyle='--')

plt.title("Linearna ekstrapolacija stope nezaposlenosti")
plt.ylabel("Stopa nezaposlenosti (%)")
plt.legend()
plt.show()

```

Originalni CPI podaci:
UNRATE
DATE
2015-01-01 5.7
2015-02-01 5.5
2015-03-01 5.4
2015-04-01 5.4
2015-05-01 5.6

Ekstrapolirane vrijednosti CPI za narednih 12 mjeseci od 2025-01-01 :
[4.2259944 4.2184213 4.21084821 4.20327511
4.19570202 4.18812892
4.18055583 4.17298273 4.16540963 4.15783654
4.15026344 4.14269035]



6.4.3.2. Vremenske serije

Vremenske serije omogućavaju praćenje i predviđanje ključnih makroekonomskih i finansijskih pokazatelja kroz vrijeme. Uobičajeni primjeri obuhvataju kvartalne vrijednosti BDP-a, mjesecni indeks potrošačkih cijena (CPI), nedjeljne izvještaje o cijenama goriva ili dnevna kretanja cijena akcija. Ključni cilj analize vremenskih serija je razumjeti obrasce poput **trenda** (dugoročnog kretanja), **sezonalnosti** (periodičnih oscilacija unutar godine) i **cikličnosti** (dugoročnih poslovnih ciklusa). Na osnovu uočenih obrazaca i statističkih testova, razvijaju se modeli za ekstrapolaciju budućih vrijednosti.

Neki od modela vremenskih serija koje se mogu koristiti u analizi vremenskih serija su:

ARIMA (*Autoregressive Integrated Moving Average*): Koristi se za modeliranje jedne vremenske serije analiziranjem autokorelacija i uklanjanjem sezonalnosti i trenda. U `statsmodels` modelu, koristi se funkcija `sm.tsa.ARIMA()` koja omogućava definisanje parametara autoregresije (p), integracije (d) i pokretnog prosjeka (q). ARIMA je korisna kada želimo filtrirati trend i sezonalnost kako bismo dobili preciznija predviđanja.

VAR (*Vector Autoregression*): Model VAR proširuje koncept autoregresije tako da obuhvata više međusobno povezanih vremenskih serija. Ova metoda je pogodna kada analiziramo više varijabli, koje istovremeno utiču jedna na drugu, npr. inflacija, nezaposlenost i kamatne stope. U `statsmodels` modelu, koristi se funkcija `sm.tsa.VAR()`. VAR omogućava analizu međusobnih efekata između više ekonomskih pokazatelja i često se koristi u makroekonomiji.

Sezonski modeli: SARIMA, TBATS, Holt-Winters. Kada postoji sezonski trend u podacima (npr. povećana potrošnja krajem godine), koriste se proširenja ARIMA modela. U `statsmodels` modelu, sezonske komponente se dodaju kroz funkciju `sm.tsa.SARIMAX()`, koja omogućava uključivanje sezonskih parametara.

Napredne tehnike: Neuronske mreže (RNN, LSTM), mašinsko učenje i hibridni pristupi. U poslednje vrijeme, sa razvojem veštačke inteligencije, vremenske serije sve češće analiziraju se korišćenjem rekurentnih neuronskih mreža (RNN, LSTM) i metoda mašinskog učenja poput XGBoost-a. Ovi modeli omogućavaju složeniju analizu i bolju predikciju, posebno kada su podaci nelinearni i kompleksni. U `statsmodels` modelu nema ugrađene podrške za neuronske mreže, ali se za to mogu koristiti biblioteke `TensorFlow`, `Keras` i `scikit-learn`.

Kroz sljedeći primjer se može vidjeti kako se, uz svega nekoliko linija koda, može preuzeti mjesecna serija iz FRED baze i pokrenuti osnovni ARIMA model prognoze, čime se demonstrira pristupačnost i moć koju Python i ekonometrijske biblioteke donose stručnjacima u sferi ekonomije i finansija.

Posmatrajmo pojednostavljeni primjer korišćenja ARIMA modela za mjesecne vrijednosti CPI (indeksa potrošačkih cijena) u SAD, preuzetog iz FRED baze (oznaka [CPIAUCSL](#)).

```
import datetime
import pandas as pd
import pandas_datareader.data as web
import statsmodels.api as sm
import matplotlib.pyplot as plt

# Definišemo vremenski period i preuzimamo podatke iz FRED baze
start = datetime.datetime(2015, 1, 1)
end = datetime.datetime(2024, 12, 31)

# Preuzimanje mjesecnog CPI (sezonski prilagođen) iz FRED baze
cpi_data = web.DataReader("CPIAUCSL", "fred", start, end)

# Reindeksiranje na mjesecnom nivou
cpi_data.index = pd.DatetimeIndex(cpi_data.index).to_period('M').to_timestamp('M')

# ARIMA model (p,d,q) = (1,1,1) - primjerna postavka; u praksi se radi analiza
# ACF/PACF
model = sm.tsa.ARIMA(cpi_data["CPIAUCSL"], order=(1,1,1))
results = model.fit()

print("\n--- Rezultati ARIMA(1,1,1) modela ---")
print(results.summary())

# Kreiramo prognozu za narednih 36 mjeseci
forecast_steps = 36
forecast_obj = results.get_forecast(steps=forecast_steps)

# 'predicted_mean' je serija sa indeksom datuma
forecast_mean = forecast_obj.predicted_mean

# Intervali pouzdanosti (npr. 95%)
confidence_int = forecast_obj.conf_int()

print("\nPrognoza CPI za narednih 36 mjeseci (predicted_mean):\n", forecast_mean)
print("\nIntervali pouzdanosti:\n", confidence_int)

# Isrtavamo istorijske podatke i prognozu na istom dijagramu, u bojama
plt.figure(figsize=(10, 6))

# Istorijski CPI (plava linija)
cpi_data["CPIAUCSL"].plot(label='Istorijski CPI', color='blue')

# ARIMA prognoza (crvena isprekidana linija)
forecast_mean.plot(label='ARIMA prognoza', color='red', linestyle='--')

# Opcionalno: prikaz intervala pouzdanosti zasjenjenim oblastima
plt.fill_between(
    confidence_int.index,
    confidence_int.iloc[:, 0],
    confidence_int.iloc[:, 1],
    color='pink', alpha=0.3,
    label='95% interval pouzdanosti'
)
```

```

plt.title("ARIMA prognoza CPI-a (CPIAUCSL)")
plt.ylabel("CPI (Indeks, 1982-1984=100)")
plt.legend()
plt.show()

```

```

--- Rezultati ARIMA(1,1,1) modela ---
SARIMAX Results
=====
Dep. Variable:          CPIAUCSL    No. Observations:             120
Model:                 ARIMA(1, 1, 1)    Log Likelihood:         -116.567
Date:                 Wed, 12 Feb 2025   AIC:                   239.135
Time:                 11:22:39        BIC:                   247.472
Sample:                01-31-2015      HQIC:                  242.521
                           - 12-31-2024
Covariance Type:            opg
=====
              coef    std err      z      P>|z|      [0.025      0.975]
ar.L1      0.9662    0.022    44.618      0.000      0.924      1.009
ma.L1     -0.6520    0.069    -9.499      0.000     -0.787     -0.517
sigma2     0.4109    0.041    10.100      0.000      0.331      0.491
=====
Ljung-Box (L1) (Q):      4.67  Jarque-Bera (JB):       29.67
Prob(Q):                  0.03  Prob(JB):           0.00
Heteroskedasticity (H):   3.41  Skew:              -0.58
Prob(H) (two-sided):      0.00  Kurtosis:          5.15
=====
```

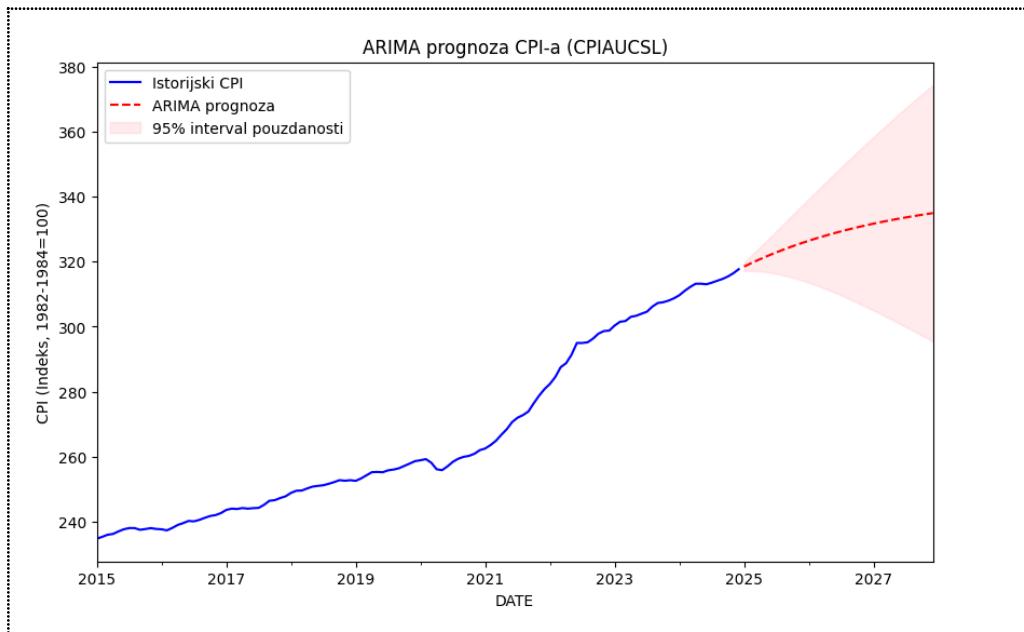
Prognoza CPI za narednih 36 mjeseci
(predicted_mean):

2025-01-31	318.507241
2025-02-28	319.301651
2025-03-31	320.069171
2025-04-30	320.810711
2025-05-31	321.527152
2025-06-30	322.219342
2025-07-31	322.888103
2025-08-31	323.534228
2025-09-30	324.158482
2025-10-31	324.761606
2025-11-30	325.344315
2025-12-31	325.907300
2026-01-31	326.451229
2026-02-28	326.976748
2026-03-31	327.484478
2026-04-30	327.975022
2026-05-31	328.448962
2026-06-30	328.906860
2026-07-31	329.349259
2026-08-31	329.776683
2026-09-30	330.189640
2026-10-31	330.588619
2026-11-30	330.974093
2026-12-31	331.346519
2027-01-31	331.706339
2027-02-28	332.053980
2027-03-31	332.389854
2027-04-30	332.714359
2027-05-31	333.027880
2027-06-30	333.330789
2027-07-31	333.623445
2027-08-31	333.906195
2027-09-30	334.179374
2027-10-31	334.443307
2027-11-30	334.698306
2027-12-31	334.944673

Freq: ME, Name: predicted_mean, dtype:
float64

Intervali pouzdanosti:

	lower CPIAUCSL	upper CPIAUCSL
2025-01-31	317.250931	319.763551
2025-02-28	317.227005	321.376297
2025-03-31	317.164929	322.973413
2025-04-30	317.042633	324.578790
2025-05-31	316.858308	326.195996
2025-06-30	316.614680	327.824005
2025-07-31	316.315635	329.460572
2025-08-31	315.965237	331.103219
2025-09-30	315.567395	332.749569
2025-10-31	315.125763	334.397448
2025-11-30	314.643718	336.044911
2025-12-31	314.124365	337.690235
2026-01-31	313.570557	339.331901
2026-02-28	312.984920	340.968575
2026-03-31	312.369871	342.599084
2026-04-30	311.727639	344.222405
2026-05-31	311.060285	345.837639
2026-06-30	310.369717	347.444003
2026-07-31	309.657703	349.040815
2026-08-31	308.925886	350.627480
2026-09-30	308.175794	352.203486
2026-10-31	307.408850	353.768388
2026-11-30	306.626379	355.321807
2026-12-31	305.829620	356.863419
2027-01-31	305.019728	358.392950
2027-02-28	304.197786	359.910174
2027-03-31	303.364805	361.414903
2027-04-30	302.521733	362.906985
2027-05-31	301.669457	364.386303
2027-06-30	300.808810	365.852768
2027-07-31	299.940572	367.306317
2027-08-31	299.065477	368.746913
2027-09-30	298.184212	370.174536
2027-10-31	297.297423	371.589191
2027-11-30	296.405716	372.990895
2027-12-31	295.509663	374.379684



Tumačenje rezultata: Rezultat predstavlja procjenu ARIMA(1,1,1) modela na osnovu 120 mjesечnih posmatranja za CPIAUCSL (Indeks potrošačkih cijena - CPI). Ovaj model koristi autoregresivni (AR) i pokretni prosjek (MA) komponente sa jednom diferencijom ($d=1$), što znači da se analiza zasniva na promjenama u CPI umjesto na njegovim apsolutnim vrijednostima.

1) Opšti pokazatelji ARMA modela

Tabela 6.3. Pregled opštih pokazatelja ARMA modela

Pokazatelj	Vrijednost	Objašnjenje
Broj posmatranja (No. Observations)	120	Model koristi 120 mjesечnih posmatranja (10 godina podataka).
Log Likelihood	-116.567	Indikator kako dobro model "pranja" uz podatke; veće vrijednosti znače bolje prilagođenje.
AIC (Akaike Information Criterion)	239.135	Manje vrijednosti AIC ukazuju na bolji model (manji broj slobodnih parametara uz dobro prilagođenje).
BIC (Bayesian Information Criterion)	247.472	Slično kao AIC, ali penalizuje složenije modele više nego AIC.
HQIC (Hannan-Quinn Information Criterion)	242.521	Srednja verzija između AIC i BIC; niže vrijednosti ukazuju na bolji model.

Manje vrijednosti AIC i BIC ukazuju na bolji model, ali ih treba uporediti s alternativnim modelima (npr. ARIMA(2,1,1), ARIMA(1,1,2), SARIMA itd.).

2) Tumačenje koeficijenata ARMA modela

- ar.L1 = 0.9662 : Pokazuje visok stepen autoregresije, što znači da CPI veoma zavisi od prethodnih vrijednosti (što je očekivano kod inflacije).
- ma.L1 = -0.6520 : Negativna vrijednost pokazuje da postoji značajna korekcija serije kroz vremenske periode. Drugim riječima, šokovi u podacima se relativno brzo smiruju.
- sigma² = 0.4109 : Varijansa rezidualnih vrijednosti, što znači da model dobro hvata varijabilnost serije.

P-vrijednosti su sve ispod 0.05, što znači da su svi parametri **statistički značajni** i ne možemo ih odbaciti.

3) Dijagnostičke statistike ARMA modela

Tabela 6.4. Pregles dijagnostičkih pokazatelja ARMA modela

Test	Vrijednost	Interpretacija
Ljung-Box (L1) (Q-statistika)	4.67	Testira da li su zaostaci nezavisni; niže vrijednosti su poželjne.
Prob(Q)	0.03	Ako je ispod 0.05, postoji određena autokorelacija (ne potpuno bijeli šum).
Jarque-Bera (JB) test	29.67	Test normalnosti reziduala; ako je vrijednost visoka, podaci nisu savršeno normalni.
Prob(JB)	0.00	Pošto je ispod 0.05, reziduali nisu savršeno normalno distribuirani.
Heteroskedastičnost (H-test)	3.41	Testira varijabilnost kroz vrijeme; visoke vrijednosti ukazuju na varijabilnu volatilnost podataka.
Prob(H) (two-sided)	0.00	Ako je ispod 0.05, postoji heteroskedastičnost (što nije idealno).
Skewness (asimetrija)	-0.58	Blaga negativna asimetrija (lagani rep na lijevoj strani).
Kurtosis	5.15	Vrijednost veća od 3 ukazuje na "oštiji" vrh distribucije podataka.

- Ljung-Box test (Q-statistika): Pokazuje postojanje autokorelacije u rezidualima (što može ukazivati na potrebu za složenijim modelom).
- Jarque-Bera test: Indikator da reziduali nisu savršeno normalno distribuirani, što može biti problem za precizna predviđanja.
- Heteroskedastičnost (H-test): Postoji varijabilnost volatilnosti, što znači da promjene u CPI nisu ravnomjerno raspoređene kroz vrijeme (moguće šokove treba dalje analizirati).

4) Glavni zaključci ARMA modela

Model je dobro prilagođen podacima, ali postoje određeni problemi, koji bi mogli uticati na njegovu tačnost u dugoročnim prognozama.

Prednosti:

- Visok koeficijent autoregresije ($ar.L1=0.9662$). Podaci su stabilni i predvidivi.
- Svi koeficijenti su statistički značajni ($P < 0.05$).
- Model pokazuje dobru procjenu trenda CPI-a.

Nedostaci:

- Postoji blaga autokorelacija. Možda bi trebalo testirati ARIMA(2,1,1) ili SARIMA model.
- Reziduali nisu savršeno normalni. Možda postoje ekstremni skokovi u podacima (npr. ekonomski šokovi).
- Postoji heteroskedastičnost. Možda bi trebalo koristiti GARCH model za bolje modeliranje volatilnosti.

Preporuka:

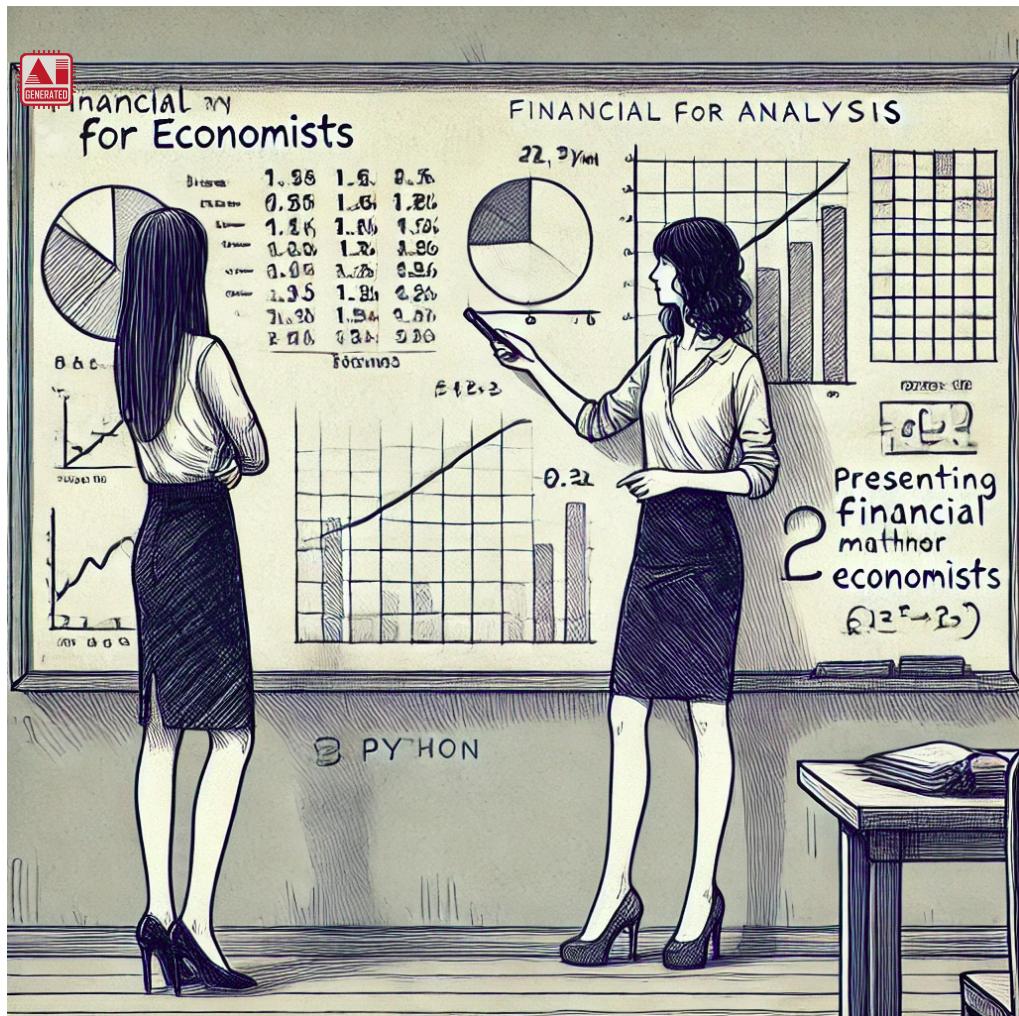
- Provjeriti drugi ARIMA model, npr. ARIMA(2,1,1) ili SARIMA za potencijalne sezonske efekte.
- Dodati zavisne varijable, npr. kamate, zaposlenost, novčana masa, jer CPI ne zavisi samo od prošlih vrijednosti.

6.4.3.3. Složeni ekonometrijski modeli

Složeni ekonometrijski modeli se razmatraju u slučajevima, kada je potrebno sagledati istovremeni uticaj više međusobno povezanih varijabli, što nadilazi mogućnosti pojednostavljenih pristupa. Primjera radi, ARIMA modelima mogu se dodati sezonske komponente (SARIMA), dok se kod vremenskih serija u kombinaciji sa regresionim tehnikama susrećemo sa pristupima kao što su ARDL (Autoregressive Distributed Lag) ili ECM (Error Correction Model).

Istovremeno, primjena savremenih tehnika poput neuronskih mreža i mašinskog učenja pruža mogućnost otkrivanja nelinearnih obrazaca i dubljih veza. U ekonomskoj praksi, ovi modeli često pronalaze svoju ulogu u analizama, koje uključuju dohodak, kamatne stope, uvoz, izvoz, javne rashode i druge makroekonomske varijable istovremeno, pri čemu je nužna obimnija dijagnostika i robusnija baza podataka. Međutim, razrada ovih naprednih postupaka prevazilazi okvire osnovnog udžbenika, te se detaljniji pristup obično izlaže u specijalizovanim studijama i na višim nivoima akademskih programa.

7. FINANSIJSKA MATEMATIKA



"Compound interest is the eighth wonder of the world. He who understands it, earns it; he who doesn't, pays it."

"Kamata na kamatu je osmo svjetsko čudo. Ko je razumije – zarađuje, ko je ne razumije – plaća je."

— Albert Einstein²⁴

²⁴ E=mc²

Finansijska matematika predstavlja nezaobilazan stub ekonomskih analiza i donošenja racionalnih odluka u poslovanju. Ona se bavi proračunima vrijednosti investicija, kredita i novčanih tokova u različitim vremenskim periodima, što čini osnovu za planiranje, budžetiranje i strategijsko odlučivanje. Razumijevanje koncepta kamatnih stopa, akumulacije i diskontovanja novčanih tokova, amortizacije kredita i faktora povrata ključno je za ekonomске stručnjake i menadžere, koji žele da sagledaju ukupne finansijske posljedice svojih poslovnih poduhvata.

Upravo tu se Python, kao jednostavan i izuzetno fleksibilan programski jezik, pokazuje kao moćan saveznik. Zahvaljujući bibliotekama poput NumPy, Pandas i SciPy, ekonomski i finansijska analiza podataka može se značajno ubrzati i pojednostaviti. Bilo da se radi o računima prostog ili složenog interesa, izradi finansijskih tablica ili proračuna amortizacionih planova, Python nudi širok spektar funkcija za efikasan rad. Dodatno, korišćenjem matplotlib biblioteke, moguće je jednostavno vizualizovati trendove u rastu investicije, prikazati razliku između nominalne i efektivne kamatne stope ili ilustrovati promjene dugovanja tokom vremena.

Kroz ovo poglavlje, produbićemo znanje o sljedećim ključnim temama finansijske matematike:

- Procentni račun – kako razumjeti njegovu ulogu u ekonomiji.
- Prost interesni račun – formula za kratkoročni proračun kamate i ilustracija kroz praktične primjere kredita i depozita.
- Izračunavanje interesa – razlika između nominalne i efektivne kamatne stope, obračun periodičnog interesa i akumulacija prostim interesom.
- Obračun potrošačkog kredita – kalkulacija rata kredita kod fiksnih i promjenljivih kamatnih stopa, uz primjenu softverskih simulacija.
- Složeni interesni račun – formula za kapitalizaciju kamate, proračun buduće vrijednosti investicije i praktični primjeri štednje i ulaganja.
- Faktor akumulacije (Prve finansijske tablice) – kreiranje tablica za prikaz rasta investicija i periodičnih uplata.
- Diskontni faktor (Druge finansijske tablice) – računanje sadašnje vrijednosti budućih novčanih tokova i upotreba diskontnih stopa.
- Faktor dodatnih uloga (Treće finansijske tablice) – kontinuirane ili periodične uplate i njihov uticaj na vrijednost investicije.
- Faktor aktualizacije (Četvrte finansijske tablice) – analiza sadašnje vrijednosti novčanih tokova i primjena u investicionim odlukama.
- Faktor povrata ili amortizacija kredita (Pete finansijske tablice) – planiranje otplate kredita i kreiranje amortizacionog plana u praktičnim primjerima.

Primjeri u Pythonu pratiće svaku od ovih cjelina, omogućavajući ne samo teorijsko razumijevanje, već i konkretnu primjenu u okruženjima kao što su Jupyter Notebook.

7.1. Procentni račun

U ovom poglavlju se proučava procentni račun kao osnovni alat finansijske i ekonomske matematike. Procenti su neizostavni u svakodnevnim obračunima — od izračunavanja poreza i kamatnih stopa, preko analiza rasta i pada vrijednosti, do proračuna popusta u trgovinama. U nastavku će biti objašnjene teorijske osnove procentnog računa, prikazane tipične primjene, kao i demonstrirane funkcionalnosti u Pythonu, koje olakšavaju konverzije i obračune povezane s procentima.

Definicija procenta

Procenat, koji potiče od latinskog izraza *per centum* (što znači "na stotinu"), predstavlja dio cjeline izražen u stotinama. Označava simbolom %, a izraz " $x\%$ " znači " x od svakih 100 jedinica".

Matematički, ako vrijednost x želimo da izrazimo u procentima u odnosu na neku osnovu B , to radimo po formuli:

$$\text{Procenat} = \frac{x}{B} \times 100\%$$

Gledano obratno, ako imamo procenat $p\%$ i želimo da dobijemo realnu vrijednost dijela x , računamo:

$$x = \frac{p}{100} \times B$$

U ekonomskoj i finansijskoj praksi, kamatne stope, porezi, popusti i slične stope često se posmatraju i u procentnom i u decimalnom obliku:

- Decimalni oblik: vrijednost je predstavljena kao razlomak u odnosu na 1 (npr. $0.05 = 5\%$).
- Procentni oblik: vrijednost je predstavljena u odnosu na 100 (npr. 5%).

Konverzija se vrši na sljedeći način:

- Iz decimalnog u procentni oblik: pomnožimo decimalnu vrijednost sa 100 i dodamo znak %.
- Iz procentnog u decimalni oblik: podijelimo vrijednost u procentima sa 100 i uklonimo znak %.

Primjeri:

- 0.08 (decimalno) = 8% (procentno)
- 15% (procentno) = 0.15 (decimalno)

Računavanje povećanja i smanjenja vrijednosti u procentima

Povećanje

Ako se neka vrijednost V_0 uveća za $p\%$, nova vrijednost V_1 biće:

$$V_1 = V_0 + (p\% \times V_0) = V_0 \times (1 + \frac{p}{100})$$

Primjer: Cijena proizvoda iznosi 100 KM. Ukoliko se poveća za 15%, nova cijena je: $100 \times (1+0.15) = 115$ KM.

Smanjenje

Ako se vrijednost V_0 umanji za $p\%$, nova vrijednost V_1 dobija se:

$$V_1 = V_0 - (p\% \times V_0) = V_0 \times (1 - \frac{p}{100})$$

Primjer: Cijena proizvoda je 100 KM i ukoliko se smanji za 10%. Nova cijena proizvoda će biti: $100 \times (1-0.10) = 90$ KM.

Višestruka promjena vrijednosti

Ponekad se vrijednost mijenja više puta zaredom u određenim procentualnim iznosima. Na primjer, povećanje od 10%, a zatim dalji pad od 5%. Tada se svaka promjena obračunava u odnosu na novu vrijednost, pa formula postaje:

$$V_1 = V_0 \times (1 + \frac{p_1}{100}) \quad (\text{prvo povećanje})$$

$$V_2 = V_1 \times (1 - \frac{p_2}{100}) \quad (\text{zatim smanjenje})$$

Procenti se nalaze u osnovi mnogih ekonomskih i finansijskih proračuna, a najčešće primjene su:

- 1) Porezi:
 - PDV (porez na dodatu vrijednost) izražava se u procentualnom iznosu (npr. 21% ili 17%).
 - Obračun zarada i doprinosa.
- 2) Kamate:
 - Kamatne stope (godišnje, mjesecne, nominalne, efektivne).
 - Računanje povećanja glavnica na računu pri štednji ili visine rate kredita.
- 3) Popusti:
 - Uobičajeno se iskazuju u procentima (npr. „akcija -20%“).
 - Različite vrste popusta u trgovinama bazirani na bonitetima i sl.

U svakom od ovih slučajeva, suština je pravilno razumjeti šta tih „ $x\%$ “ znači u kontekstu – da li se primjenjuje na osnovnu cijenu, promjenjivu vrijednost, kakvo je vrijeme obračuna itd.

U nastavku su pokazani jednostavnii primjeri upotrebe matematičkih operacija sa procentnima.

```
# Konverzija između decimalnog i procentnog oblika
decimal_value = 0.08
percent_value = decimal_value * 100
print(f"{decimal_value} u procentima je {percent_value}%")

percent_value = 15
decimal_value = percent_value / 100
print(f"{percent_value}% u decimalnom obliku je {decimal_value}")

# Računanje povećanja vrijednosti
base_value = 100
percent_increase = 15
new_value = base_value * (1 + percent_increase / 100)
print(f"Nova vrijednost nakon povećanja od {percent_increase}% je {new_value:.2f} KM")

# Računanje smanjenja vrijednosti
percent_decrease = 10
new_value = base_value * (1 - percent_decrease / 100)
print(f"Nova vrijednost nakon smanjenja od {percent_decrease}% je {new_value:.2f} KM")

# Višestruke promjene vrijednosti
changes = [10, -5] # Prvo povećanje od 10%, zatim smanjenje od 5%
current_value = base_value
for change in changes:
    current_value = current_value * (1 + change / 100)
print(f"Konačna vrijednost nakon višestrukih promjena {current_value:.2f} KM")
```

```
0.08 u procentima je 8.0%
15% u decimalnom obliku je 0.15
Nova vrijednost nakon povećanja od 15% je 115.00 KM
Nova vrijednost nakon smanjenja od 10% je 90.00 KM
Konačna vrijednost nakon višestrukih promjena 104.50 KM
```

7.2. Prost interesni račun

Prost interesni račun odnosi se na obračun kamate na fiksnu glavnicu, koja ostaje nepromijenjena tokom cijelog perioda. Ovaj model je čest kod kratkoročnih kredita i depozita, gdje se sredstva koriste na kraći rok, a efekti kapitalizacije (karakteristični za složeni interes) su zanemarivi ili nisu predviđeni ugovorom.

Ključni elementi su:

- Glavnica (K) – Početni iznos novca.
- Interes (I) – Kamata koju zaradimo ili platimo.
- Kamatna stopa (r) – Stopa interesa izražena u decimalnom obliku.
- Vrijeme (t) – Period za koji se računa interes, obično u godinama.

Osnovna formula za prosti interes je:

$$I = K \times r \times t$$

Ova formula ne dijeli sa 100 jer se kamatna stopa r već koristi u decimalnom obliku (npr. 5% = 0.05).

Osim ove formule za računanje prostog interesa, u literaturi se često koristi i proporcija, koja opisuje odnose između ovih elemenata. U toj proporciji kamatna stopa je izražena u procentima, a ne kao decimalni broj, zbog čega se u ovom obliku formule za računanje interesa pojavljuje dijeljenje sa 100.

$$K : I = 100 : r \times t$$

Iz ove proporcije možemo izvesti formulu za interes:

$$I = \frac{K \times r \times t}{100}$$

Primjer upotrebe osnovne formule bez dijeljenja sa 100

Glavnica:	K=1.000 KM
Kamatna stopa:	r=0.05 (5% u decimalnom obliku),
Vrijeme:	t=2 godine
Interes:	I=1000×0.05×2=100 KM

Primjer upotrebe formule izvedene iz osnovne proporcije sa dijeljenjem sa 100

Glavnica:	K=1.000 KM,
Kamatna stopa:	r=5%,
Vrijeme:	t=2 godine,
Interes:	I=(1000×0.05×2)/100=100 KM

U Pythonu ne postoji poseban modul za obračun prostog interesa jer se radi o jednostavnim matematičkim operacijama, koje se lako mogu implementirati pomoću osnovnih aritmetičkih izraza. Prost interesni račun zahtijeva samo osnovne operacije poput množenja i dijeljenja, koje su već ugrađene u Python, pa nije potrebna posebna biblioteka za ovu svrhu.

Ako imate potrebu za čestim proračunima ili radite na većem projektu u kojem, osim interesa, obračunavate i druge parametre prostog interesnog računa, možete kreirati vlastite funkcije, koje će automatizovati ovaj proces. U nastavku je prikazano kako možete napraviti potrebne funkcije i primijeniti ih u obračunima.

1) Funkcija za računanje interesa (simple_interest)

```
def simple_interest(principal, rate, time):
    """
    Računa prosti interes.

    :param principal: Glavnica (K)
    :param rate: Kamatna stopa (r) u procentima
    :param time: Vrijeme (t) u godinama
    :return: Interes (I)
    """

    return (principal * rate * time) / 100
```

Primjer:

```
K = 1000 # Glavnica
p = 5     # Kamatna stopa (5%)
t = 2     # Vrijeme u godinama

I = simple_interest(K, p, t)
print(f"Interes: {I:.2f} KM") # Interes: 100.00 KM
```

Interes: 100.00 KM

2) Funkcija za računanje glavnice (simple_interest_principal)

```
def simple_interest_principal(interest, rate, time):
    """
    Računa glavnicu na osnovu interesa.

    :param interest: Interes (I)
    :param rate: Kamatna stopa (p) u procentima
    :param time: Vrijeme (t) u godinama
    :return: Glavnica (K)
    """

    return (interest * 100) / (rate * time)
```

Primjer:

```
I = 100    # Interes
p = 5      # Kamatna stopa (5%)
t = 2      # Vrijeme u godinama

K = simple_interest_principal(I, p, t)
print(f"Glavnica: {K:.2f}€") # Glavnica: 1000.00 KM
```

Glavnica: 1000.00 KM

3) Funkcija za računanje kamatne stope (simple_interest_rate)

```
def simple_interest_rate(interest, principal, time):
    """
    Računa kamatnu stopu na osnovu interesa.

    :param interest: Interes (I)
    :param principal: Glavnica (K)
    :param time: Vrijeme (t) u godinama
    :return: Kamatna stopa (p) u procentima
    """
    return (interest * 100) / (principal * time)
```

Primjer:

```
I = 100    # Interes
K = 1000   # Glavnica
t = 2      # Vrijeme u godinama

p = simple_interest_rate(I, K, t)
print(f"Kamatna stopa: {p:.2f}%" ) # Kamatna stopa: 5.00%
```

Kamatna stopa: 5.00%

4) Funkcija za računanje vremena (simple_interest_time)

```
def simple_interest_time(interest, principal, rate):
    """
    Računa vrijeme na osnovu interesa.

    :param interest: Interes (I)
    :param principal: Glavnica (K)
    :param rate: Kamatna stopa (p) u procentima
    :return: Vrijeme (t) u godinama
    """
    return (interest * 100) / (principal * rate)
```

Primjer:

```
I = 100    # Interes
K = 1000   # Glavnica
p = 5      # Kamatna stopa (5%)

t = simple_interest_time(I, K, p)
print(f"Vrijeme: {t:.2f} godine") # Vreme: 2.00 godine
```

Vrijeme: 2.00 godine

Obrada većih skupova podataka u Pandas-u

Ako imate veći skup podataka (npr. više depozita ili kredita), možete koristiti pandas za efikasnu analizu. Evo kako možete primijeniti iste funkcije u pandas-u.

```
import pandas as pd

# Opis funkcija
def simple_interest(principal, rate, time):
    return (principal * rate * time) / 100

# Primjer u pandas-u
data = {
    'Glavnica': [1000, 2000, 3000],
    'Kamatna stopa': [5, 4, 3],
    'Vrijeme': [2, 3, 5]
}

df = pd.DataFrame(data)
df['Interes'] = df.apply(lambda row: simple_interest(row['Glavnica'], row['Kamatna stopa'], row['Vrijeme']), axis=1)
print(df)
```

	Glavnica	Kamatna stopa	Vrijeme	Interes
0	1000	5	2	100.0
1	2000	4	3	240.0
2	3000	3	5	450.0

Podrazumjeva se da ste definisali funkciju simple_interest. U primjeru je funkcija definisana skraćeno u odnosu na prethodnu definiciju, odnosno bez dodatnih komentara koji opisuje parametre.

7.3. Složeni interesni račun

Složeni ili komponovani interes označava situaciju u kojoj se ostvarena kamata u prethodnom obračunskom periodu dodaje glavnici i potom zajedno sa njom zaračunava u narednim periodima. Ovaj efekat često se naziva „kamata na kamatu“, a njegova je važnost naročito izražena kod dugoročnih štednji i investicija. Nasuprot tome, prosti interes podrazumijeva da se kamata ne sabira sa glavnica radi daljeg obračuna, što rezultira značajno manjom akumulacijom tokom dužeg vremenskog intervala.

Formula složenog interesa

Ukoliko se glavnom iznosu P u svakoj jedinici vremena (npr. godini) dodaje obračunata kamata, ukupna vrijednost nakon n perioda uz godišnju kamatnu stopu r (u decimalnom obliku) računa se:

$$V = P \cdot (1 + r)^n$$

Ovdje je $(1+r)$ faktor rasta po periodu, a eksponent n označava broj perioda kapitalizacije. U slučaju mjesecne kapitalizacije, ako je godišnja stopa r , mjesecna stopa postaje $r/12$, a ukupan broj perioda $12 \times n$ za n godina:

$$V = P \cdot \left(1 + \frac{r}{12}\right)^{12n}$$

Računavanje buduće vrijednosti investicije

Složeni interes je naročito koristan kada želimo procijeniti koliko će vrijediti određena investicija nakon nekoliko godina. Ako recimo ulažemo 1.000€ na 10 godina uz godišnju stopu od 5%, formula sugerire da će se početni iznos od 1.000€ uvećati na:

$$1,000 \times (1 + 0.05)^{10} = 1,000 \times 1.62889 \approx 1,628.89$$

Da smo koristili samo prosti interes, vrijednost bi bila:

$$1,000 \times (1 + 0.05 \times 10) = 1,500$$

Već na ovom primjeru postaje jasno koliko je efekat „kamate na kamatu“ značajan na duži rok.

Gdje se koristi složeni intereni račun:

- 1) **Štednja u banci:** Ako se polaže novac na štedni račun s godišnjom kamatnom stopom i bankarskom politikom kapitalizacije na kraju svake godine, iznos na računu će poslije svakog obračuna postati nova osnova za narednu godinu.

- 2) **Investicije u fondove:** Kod dugoročnih ulaganja u investicione ili penzijske fondove, primjenjuje se efekat kontinualne ili periodične kapitalizacije, gdje se dobici reinvestiraju. Nakon više godina, novčana vrijednost ulaganja može biti znatno veća nego što proističe iz običnog sabiranja kamate.
- 3) **Kapitalizacija na više perioda godišnje:** Ako je kapitalizacija mjesecna, kvartalna ili polugodišnja, rezultujuća efektivna stopa postaje viša od nominalne godišnje stope, što je dodatni podsticaj kontinuiranom rastu investicije.

Slijedi kratki Python primjer koji prikazuje kako se može izračunati i grafički ilustrovati rast investicije od 1.000€ kroz 20 godina, pri godišnjoj kamatnoj stopi od 5%, za dvije varijante: prosti i složeni interes.

```
import matplotlib.pyplot as plt

def vrijednost_prostog_interesa(pocetni_iznos, godisnja_stopa, godina):
    return pocetni_iznos * (1 + godisnja_stopa * godina)

def vrijednost_slozenog_interesa(pocetni_iznos, godisnja_stopa, godina):
    return pocetni_iznos * ((1 + godisnja_stopa) ** godina)

P = 1000
r = 0.05 # 5% godišnje
max_godina = 20

godine = list(range(max_godina + 1))
vrijednosti_prosti = []
vrijednosti_slozeni = []

for g in godine:
    vp = vrijednost_prostog_interesa(P, r, g)
    vs = vrijednost_slozenog_interesa(P, r, g)
    vrijednosti_prosti.append(vp)
    vrijednosti_slozeni.append(vs)

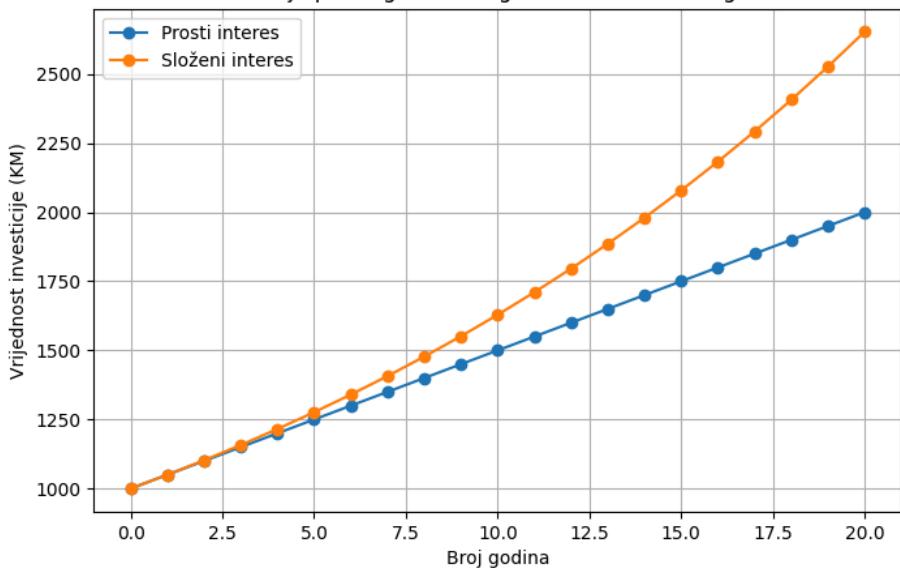
# Tabelarni prikaz
print(f"{'Godina':>6} | {'Prosti (KM)':>12} | {'Složeni (KM)':>12}")
print("-" * 36)
for g, vp, vs in zip(godine, vrijednosti_prosti, vrijednosti_slozeni):
    print(f"{g:6} | {vp:12.2f} | {vs:12.2f}")

# Grafička ilustracija
plt.figure(figsize=(8, 5))
plt.plot(godine, vrijednosti_prosti, label="Prosti interes", marker='o')
plt.plot(godine, vrijednosti_slozeni, label="Složeni interes", marker='o')
plt.title("Poređenje prostog i složenog interesa tokom 20 godina")
plt.xlabel("Broj godina")
plt.ylabel("Vrijednost investicije (KM)")
plt.legend()
plt.grid(True)
plt.show()
```

Godina	Prosti (KM)	Složeni (KM)
0	1000.00	1000.00
1	1050.00	1050.00

2	1100.00	1102.50
3	1150.00	1157.63
4	1200.00	1215.51
5	1250.00	1276.28
6	1300.00	1340.10
7	1350.00	1407.10
8	1400.00	1477.46
9	1450.00	1551.33
10	1500.00	1628.89
11	1550.00	1710.34
12	1600.00	1795.86
13	1650.00	1885.65
14	1700.00	1979.93
15	1750.00	2078.93
16	1800.00	2182.87
17	1850.00	2292.02
18	1900.00	2406.62
19	1950.00	2526.95
20	2000.00	2653.30

Poređenje prostog i složenog interesa tokom 20 godina



Razlika između prostog i složenog interesnog računa

Prosti interes je linearna funkcija, gdje se kamata obračunava samo na početni iznos (principal) i ne mijenja se tokom vremena. Rast je konstantan i predvidljiv.

Sa druge strane, složeni interes je eksponencijalna funkcija, gdje se kamata obračunava na početni iznos plus prethodno zarađene kamate, što dovodi do efekta kapitalizacije. Rast postaje sve brži kako vrijeme prolazi.

Kako bi pokazali razliku između prostog i složenog interesnog računa, u nastavku je prikazana Python skripta sa grafičkim i tabelarnim prikazom rezultata.

```

import matplotlib.pyplot as plt
import numpy as np

# Definišemo funkcije
def prosti_interes(pocetni_iznos, godisnja_stopa, vrijeme):
    return pocetni_iznos * (1 + godisnja_stopa * vrijeme)

def slozeni_interes(pocetni_iznos, godisnja_stopa, vrijeme):
    return pocetni_iznos * ((1 + godisnja_stopa) ** vrijeme)

pocetni_iznos = 100 # Početni iznos (npr. 100 KM)
godisnja_stopa = 0.05 # Godišnja stopa (npr. 5%)

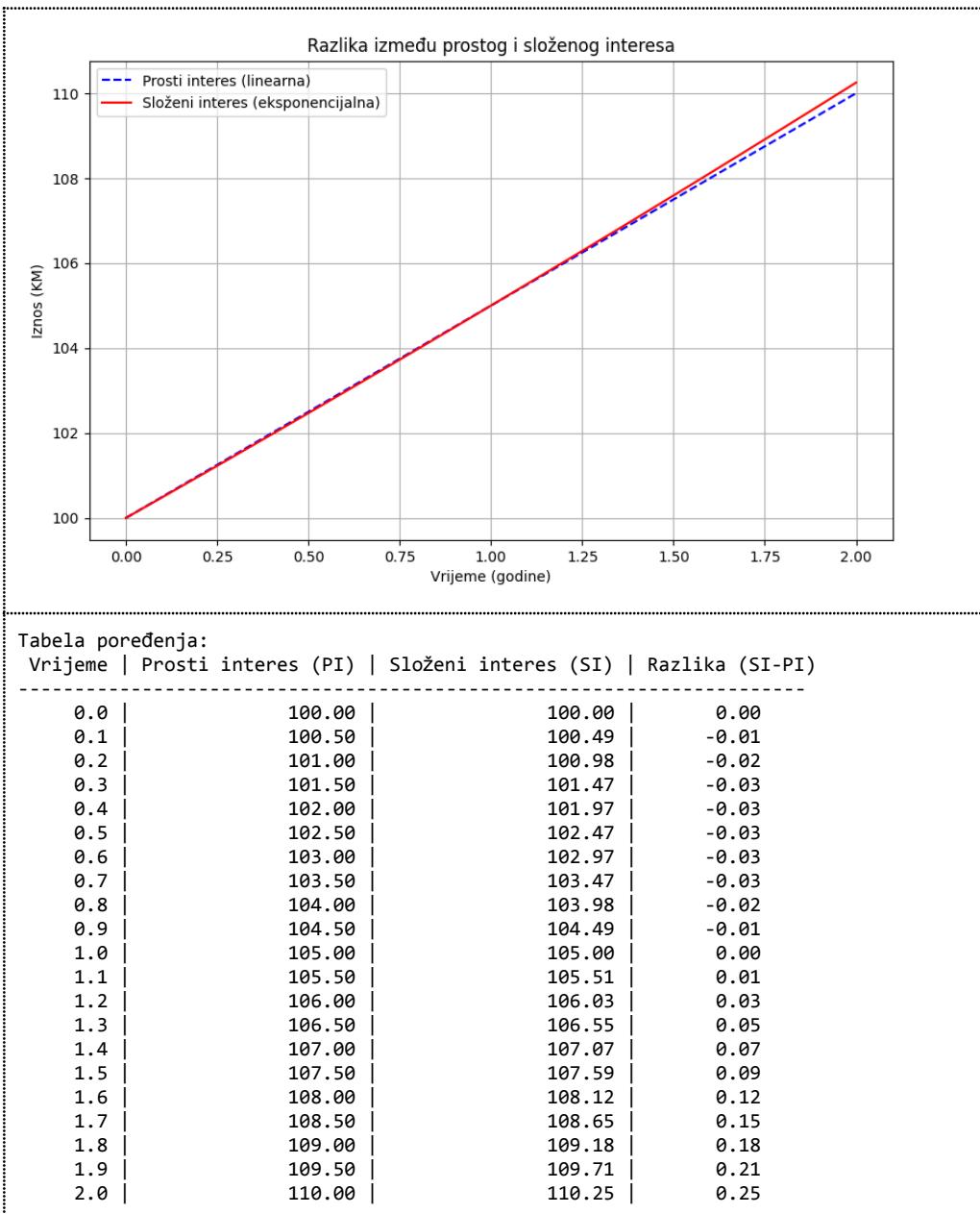
# Kreiramo vremenski niz godina od 0 do 2 sa korakom 0.1
vrijeme = np.arange(0, 2.1, 0.1) # 2.1 da uključimo tačno 2.0

# Računamo obe vrednosti za svaku interval vremena
prosti_vrednosti = [prostti_interes(pocetni_iznos, godisnja_stopa, t) for t in vrijeme]
slozeni_vrednosti = [slozeni_interes(pocetni_iznos, godisnja_stopa, t) for t in vrijeme]

# Crtanje grafika
plt.figure(figsize=(10, 6))
plt.plot(vrijeme, prosti_vrednosti, label='Prosti interes (linearna)', color='blue', linestyle='--')
plt.plot(vrijeme, slozeni_vrednosti, label='Složeni interes (eksponencijalna)', color='red')
plt.xlabel('Vrijeme (godine)')
plt.ylabel('Iznos (KM)')
plt.title('Razlika između prostog i složenog interesa')
plt.legend()
plt.grid(True)
plt.show()

# Kreiramo tabelu sa razlikom za sve tačke u rasteru 0.1
print("Tabela poređenja:")
print(f"{'Vrijeme':>8} | {'Prosti interes (PI)':>19} | {'Složeni interes (SI)':>20} | {'Razlika (SI-PI)':>10}")
print("-" * 70)
for t in np.arange(0, 2.1, 0.1):
    pi = prosti_interes(pocetni_iznos, godisnja_stopa, t)
    si = slozeni_interes(pocetni_iznos, godisnja_stopa, t)
    razlika = si - pi
    print(f"{t:>8.1f} | {pi:>19.2f} | {si:>20.2f} | {razlika:>10.2f}")

```



Objašnjenje:

U početku obračuna, složeni interes raste sporije jer se efekat kapitalizacije još nije značajno ispoljio. Kamata "na kamatu" ne pravi veliku razliku u početnom periodu, a posebno u prvoj godinini gdje je niži od prostog interesa. Prost interes daje veće vrijednosti tokom ovog početnog perioda jer se povećava ravnomjerno.

Nakon određene tačke ($t=1$ godina), složeni interes počinje brže rasti, jer efekat kapitalizacije postaje izraženiji.

Iz ovoga može se zaključiti sljedeće:

- Za kratkoročne investicije ili kredite, prosti interes je jednostavniji i često prikladniji jer ne postoji značajan efekat kapitalizacije.
- Za dugoročne investicije, složeni interes postaje znatno isplativiji jer omogućava rast kapitala zahvaljujući kamati na kamatu.

7.4. Nominalna i efektivna kamatna stopa

U ovom poglavlju proširićemo koncept obračuna kamate, fokusirajući se na razlike između nominalne i efektivne kamatne stope, periodičnu kapitalizaciju (mjesečnu, kvartalnu, godišnju) i dodatne aspekte obračuna proste kamate kroz različite intervale.

Nominalna kamatna stopa (engl. *Nominal Interest Rate*) je stopa koja se formira za određeni period (obično godinu dana), ali se unutar te godine može obračunavati u kraćim intervalima: mjesečno, kvartalno ili polugodišnje. Drugim riječima, nominalna stopa je ona koja se „deklarativno“ navodi u ugovoru ili reklami, ne uzimajući uvijek u obzir stvarni efekat kapitalizacije u toku godine.

Efektivna kamatna stopa (engl. *Effective Interest Rate* ili *Effective Annual Rate, EAR*) predstavlja stvarnu stopu rasta glavnice u toku jedne godine, uzimajući u obzir i broj kapitalizacija u tom periodu. Zbog toga se efektivna stopa često smatra najtačnijim pokazateljem realne cijene kredita ili realnog prilosa na štednju.

Matematički, ako je:

- i_{nom} – nominalna godišnja stopa,
- m – broj obračunskih (kapitalizacionih) perioda godišnje,
- i_{eff} – efektivna godišnja stopa,

tada vrijedi:

$$i_{\text{eff}} = \left(1 + \frac{i_{\text{nom}}}{m}\right)^m - 1$$

Računanje efektivne kamatne stope je prosta matematička operacija, pa ćemo kao i kod računanja prostog interesnog računa kreirati sopstvene funkcije. U sljedećem primjeru je definisana funkcija `effective_rate()` i prikazana je upotreba funkcije u računanju efektivne kamate.

```
def effective_rate(nominal_rate, m):
    """
    Izračunava efektivnu godišnju stopu na osnovu
    nominalne stope i broja kapitalizacija 'm' godišnje.
    """
    return (1 + nominal_rate / m) ** m - 1

# Primjer upoređivanja nominalne i efektivne stope
nominal_rates = [0.03, 0.06, 0.1] # 3%, 6%, 10%
periods = [1, 4, 12] # godišnje, kvartalno, mjesečno

for nr in nominal_rates:
    print(f"Nominalna stopa: {nr*100:.2f}%")
    for m in periods:
        eff = effective_rate(nr, m) * 100
        print(f" Kapitalizacija {m} put/a godišnje -> efektivna stopa: {eff:.2f}%")
```

```

print("-" * 40)

Nominalna stopa: 3.00%
Kapitalizacija 1 put/a godišnje -> efektivna stopa: 3.00%
Kapitalizacija 4 put/a godišnje -> efektivna stopa: 3.03%
Kapitalizacija 12 put/a godišnje -> efektivna stopa: 3.04%
-----
Nominalna stopa: 6.00%
Kapitalizacija 1 put/a godišnje -> efektivna stopa: 6.00%
Kapitalizacija 4 put/a godišnje -> efektivna stopa: 6.14%
Kapitalizacija 12 put/a godišnje -> efektivna stopa: 6.17%
-----
Nominalna stopa: 10.00%
Kapitalizacija 1 put/a godišnje -> efektivna stopa: 10.00%
Kapitalizacija 4 put/a godišnje -> efektivna stopa: 10.38%
Kapitalizacija 12 put/a godišnje -> efektivna stopa: 10.47%
-----
```

Periodični interes: mjesечni, kvartalni i godišnji obračun

Periodični interes se odnosi na način na koji se kamata dodaje glavnici u određenim intervalima tokom godine, što može biti mjesечно, kvartalno ili godišnje. Ovo ima direktni uticaj na efektivnu kamatnu stopu i ukupnu akumuliranu vrijednost.

Mjesечni obračun kamate podrazumijeva obračun kamate svakog mjeseca, pri čemu se za **prosti interesni račun** godišnja nominalna stopa dijeli na 12 jednakih dijelova. Kamata se obračunava na istu (nepromijenjenu) glavnici svakog mjeseca, ali se ne dodaje samoj glavnici (jer govorimo o prostom, a ne o složenom interesu). Za ukupan broj mjeseci t_m (npr. 6 mjeseci) kamata se računa:

$$I = P \times \left(\frac{i_{\text{nom}}}{12} \right) \times t_m$$

gdje je:

- P – početna glavnica,
- t_m – broj mjeseci,
- i_{nom} – godišnja nominalna stopa.

Analogno tome, **kvartalni obračun kamate** dijeli godinu na četiri tromjesečja. Kamata se kod prostog interesa i dalje obračunava na nepromijenjenu glavnici, ali se računa svakih **3 mjeseca**, odnosno po tromjesečju. Ako ulažete sredstva ili otplaćujete kredit kroz t_{kv} kvartala, kamata je:

$$I = P \times \left(\frac{i_{\text{nom}}}{4} \right) \times t_{kv}$$

Ovakav vid obračuna koristi se kod kraćih investicija ili pozajmica, gdje se ugovori definišu po kvartalnim periodima, a pri tom se kamata ne dodaje glavnici (nema „kamate na kamatu“).

U slučaju **godišnjeg obračuna kamate**, obračun se vrši samo jednom godišnje, pri čemu je efikasno $i_{\text{god}}=i_{\text{nom}}$, jer nema višestrukih pod-perioda unutar godine. Za period od t godina (može biti i djelimično, npr. 1.5 godina), kod prostog obračuna:

$$I = P \times i_{\text{nom}} \times t$$

U praksi to znači da se jednom godišnje registruje iznos kamate i isplaćuje (ili zaračunava), ali ne utiče na visinu glavnice za naredni obračun, ako se govori o prostom interesu.

U nastavku je primjer u Python-u, koji definiše funkcije za prost obračun kamate za svaki tip mjesecni, kvartalni ili godišnji. Kao i u prethodnim primjerima kada smo računali efektivnu i nominalnu kamatnu stopu, definišu se sopstvene funkcije. Ovi primjeri funkcija mogu se korsiti u složenim analizama.

```
def prost_interes_mjesecno(glavnica, godisnja_stopa, broj_mjeseci):
    mj_stopa = godisnja_stopa / 12
    interes = glavnica * mj_stopa * broj_mjeseci
    return interes

def prost_interes_kvartalno(glavnica, godisnja_stopa, broj_kvartala):
    kv_stopa = godisnja_stopa / 4
    interes = glavnica * kv_stopa * broj_kvartala
    return interes

def prost_interes_godisnje(glavnica, godisnja_stopa, broj_godina):
    interes = glavnica * godisnja_stopa * broj_godina
    return interes

glavnica = 2000.0
nominalna_stopa = 0.05

kamata_mjeseci_7 = prost_interes_mjesecno(glavnica, nominalna_stopa, 7)
print(f"Mjesečni obračun (7 mjeseci), prost interes: {kamata_mjeseci_7:.2f}")

kamata_kvartali_3 = prost_interes_kvartalno(glavnica, nominalna_stopa, 3)
print(f"Kvartalni obračun (3 kvartala), prost interes: {kamata_kvartali_3:.2f}")

kamata_godine_15 = prost_interes_godisnje(glavnica, nominalna_stopa, 1.5)
print(f"Godišnji obračun (1.5 godina), prost interes: {kamata_godine_15:.2f}")
```

```
Mjesečni obračun (7 mjeseci), prost interes: 58.33
Kvartalni obračun (3 kvartala), prost interes: 75.00
Godišnji obračun (1.5 godina), prost interes: 150.00
```

7.5. Obračun potrošačkog kredita

Potrošački kredit obično služi za finansiranje svakodnevnih ili neodložnih troškova (kupovina namještaja, elektronike, putovanje ili slično), pri čemu banka daje pozajmicu, koja se zatim otplaćuje u jednakim ili promjenljivim ratama tokom dogovorenog perioda. U nastavku su opisane osnovne karakteristike potrošačkih kredita, razlika između fiksnih i promjenljivih kamatnih stopa, te kratak prikaz kako se takvi krediti mogu obračunavati i simulirati uz pomoć programskog jezika Python.

Struktura potrošačkog kredita i osnovni pojmovi

Potrošački krediti se najčešće odobravaju na period od nekoliko mjeseci do nekoliko godina. Otplaćuje se u ratama (mjesečnim ili kvartalnim), koje pored glavnice (iskorišćen iznos) sadrže i kamatu. Banka obično obračunava kamatu na pozajmljeni iznos, a način obračuna zavisi od ugovorene kamatne stope (fiksne ili promjenljive) i definisanog plana otplate. Prilikom ovih proračuna se koriste sljedeći termini:

- 1) **Glavnica (P)** – početni iznos kredita.
- 2) **Kamatna stopa (i)** – može biti fiksna (ne mijenja se tokom trajanja kredita) ili promjenljiva (npr. vezana za referentnu stopu kao što je EURIBOR²⁵, uz dodatni „maržni“ dio).
- 3) **Rok otplate (n)** – najčešće iskazan u mjesecima.
- 4) **Rata (R)** – mjesečni iznos koji dužnik plaća banci, pokrivajući dio glavnice i dio kamate.

Fiksna kamatna stopa

Kod fiksne kamatne stope, banka i klijent se dogovaraju o godišnjoj stopi, koja ostaje nepromijenjena do kraja otplate. U praksi to znači da se mjesečna rata ne mijenja tokom cijelog perioda, što olakšava planiranje i stabilnost uplata za korisnika kredita.

Ako posmatramo model **anuitetne otplate** (jednake mjesečne rate), formula za mjesečnu ratu (R) može se napisati (u najjednostavnijoj varijanti) kao:

$$R = P \cdot \frac{i}{1-(1+i)^{-n}}$$

gdje su:

- P – glavnica (ukupan iznos kredita),
- i – mjesečna kamatna stopa (ako je godišnja nominalna stopa r, tada je $i=r/12$),
- n – broj mjeseci otplate.

²⁵ EURIBOR je referentna kamatna stopa koju koriste evropske banke kada pozajmljuju sredstva jedna drugoj na međubankarskom tržištu u evrima. On predstavlja prosječnu kamatnu stopu koju velike banke u Eurozoni nude za kratkoročne depozite (obično od 1 nedjelje do 12 mjeseci)

Primjer iz realnog života: Ako se pozajmi 10.000€ na 5 godina (60 mjeseci) uz godišnju fiksnu nominalnu stopu od 6% (tj. 0.06 godišnje), tada je mjesecna stopa $i=0.06/12=0.005$. Za 60 rata, formula daje preciznu visinu jednakne mjesecne rate.

Promjenljiva (varijabilna) kamatna stopa

Kod kredita s promjenljivom kamatnom stopom, godišnja stopa se periodično može mijenjati (npr. svaka 3, 6 ili 12 mjeseci), zavisno od nekog eksternog referentnog indeksa ili propisanih pravila u ugovoru. U praksi, korisnik može zateći različitu visinu rate tokom otplate.

Primjer iz realnog života: Kredit od 10.000€ na 5 godina, čija je osnovna kamatna stopa vezana za EURIBOR + 2% marže. Ako EURIBOR poraste, rata će porasti. Ako se EURIBOR spusti, rata će se smanjiti (ili trajanje kredita, zavisno kako je ugovorom definisano). Banka obično svaka 3 ili 6 mjeseci revidira stopu na osnovu aktuelnog EURIBOR-a.

Amortizacioni plan sa jednakim ratama

Amortizacioni plan prikazuje detaljan raspored otplate mjesecnih anuiteta ili rata, pri čemu se jasno vidi koliki se dio svake rate odnosi na kamatu, a koliki na glavnici. Kod fiksne kamatne stope i anuitetne otplate, matematički je relativno lako izračunati:

- 1) **Kamatni dio u rati** – iznos dugovanja \times mjesecna kamatna stopa (i).
- 2) **Dio glavnice u rati** – razlika između ukupne rate i kamatnog dijela.
- 3) **Novi dug** – stari dug minus dio glavnice plaćen tekućom ratom.

Promjenljiva stopa otežava plan, jer se mora prilagoditi nova mjesecna kamatna stopa, uvijek kada dođe do promjene u referentnom indeksu. Tada se ponovo izračunava rata ili se mijenjaju odnosi glavnice i kamate u narednim ratama.

U praksi, banke koriste posebne finansijske softvere za generisanje amortizacionih planova i proračune kamatnih troškova. Sa programskim jezikom Pythona, moguće je relativno brzo izračunati visinu mjesecne rate, kreirati tabelu otplate ili simulirati efekte promjene kamatne stope. U nastavku je primjer računanja anuitetnog obračun kredita s fiksnom stopom.

```
def mjesecna_rata_anuitet(glavnica, godisnja_stopa, broj_mjeseci):
    # Mjesecna stopa (i)
    i = godisnja_stopa / 12
    # Formula anuiteta
    rata = glavnica * (i / (1 - (1 + i)**(-broj_mjeseci)))
    return rata

def generisi_amortizacioni_plan(glavnica, godisnja_stopa, broj_mjeseci):
    i = godisnja_stopa / 12
    rata = mjesecna_rata_anuitet(glavnica, godisnja_stopa, broj_mjeseci)
    saldo = glavnica
```

```

plan = []
for _ in range(broj_mjeseci):
    # dio kamate za tekući mjesec
    kamata = saldo * i
    # dio glavnice (razlika između ukupne rate i kamate)
    otplata_glavnice = rata - kamata
    # novi saldo nakon otplate dijela glavnice
    saldo -= otplata_glavnice

    plan.append({
        "rata": rata,
        "kamata": kamata,
        "glavnica_placena": otplata_glavnice,
        "preostali_dug": saldo
    })
return plan

# Glavni dio skripte
glavnica_kredita = 10000.0
godisnja_nominalna_stopa = 0.06
mjeseci_otplate = 60 # npr. 5 godina

# Generišemo plan otplate
plan_otplate = generisi_amortizacioni_plan(glavnica_kredita,
godisnja_nominalna_stopa, mjeseci_otplate)

# Formatiran ispis u obliku tabele
print(f"{'Mjesec':>6} | {'Rata':>10} | {'Kamata':>10} | {'Glavnica':>10} |"
{'Preostali dug':>14}")
print("-" * 60)

for idx, mjesec in enumerate(plan_otplate, start=1):
    print(f"{idx:>6d} | {mjesec['rata']:10.2f} | {mjesec['kamata']:10.2f} |"
{mjesec['glavnica_placena']:10.2f} | {mjesec['preostali_dug']:14.2f}")

```

Mjesec	Rata	Kamata	Glavnica	Preostali dug
1	193.33	50.00	143.33	9856.67
2	193.33	49.28	144.04	9712.63
3	193.33	48.56	144.76	9567.86
4	193.33	47.84	145.49	9422.37
5	193.33	47.11	146.22	9276.16
6	193.33	46.38	146.95	9129.21
7	193.33	45.65	147.68	8981.53
...				
56	193.33	4.76	188.57	763.74
57	193.33	3.82	189.51	574.23
58	193.33	2.87	190.46	383.78
59	193.33	1.92	191.41	192.37
60	193.33	0.96	192.37	-0.00

Funkcija `mjeseca_rata_anuitet` prvo računa mjesecnu kamatnu stopu tako što godišnju stopu (`godisnja_stopa`) podijeli sa 12 i dobijenu vrijednost čuva u varijabli `i`. Zatim koristi formulu za anuitetni obračun mjesecne rate, koja daje iznos jednake mjesecne rate `rata`.

Funkcija `generisi_amortizacioni_plan` također računa mjesecnu stopu (`i`) dijeljenjem godišnje stope sa 12, a zatim poziva `mjeseca_rata_anuitet` kako bi unaprijed odredila konstantnu mjesecnu ratu `rata`. U petlji prolazi kroz svaki mjesec otplate, računa dio kamate tako što trenutni saldo pomnoži s `i`, oduzimanjem tog iznosa od `rata` određuje se koliko odlazi na otplatu glavnice. Novi saldo računa umanjivanjem starog salda za otplaćeni dio glavnice. Na kraju svakog kruga, generiše se rječnik s ključevima „rata“, „kamata“, „glavnica_placena“ i „preostali_dug“, i taj rječnik se dodaje u listu `plan`, koja se vraća nakon što petlja obradi sve mjesece.

Amortizacioni plan sa jednakim otplatama glavnice

Kod otplate kredita sa jednakim otplatama glavnice, mjesecni iznos koji korisnik vraća se sastoji od:

- Jednakog dijela glavnice koji se vraća svaki mjesec.
- Kamatnog dijela koji se obračunava na preostali dug i smanjuje se tokom vremena.

Za razliku od anuitetnog modela (gdje su rate jednake), ovdje su mjesecne rate varijabilne, najveće su na početku i smanjuju se kako dug opada, jer kamate postepeno postaju sve manje.

Struktura otplate se sastoji od mjesecnog dijela glavnice (G_m) i kamatnog dijela (K_m). Formule za računanje ukupne mjesecne rate i svakog dijela posebno su:

1) Dio glavnica po rati (G_m) :

$$G_m = \frac{P}{n}$$

gdje je (P) ukupna glavnica kredita, i (n) ukupan broj mjeseci otplate kredita.

2) Kamatni dio (K_m) koji se obračunava za svaki mjesec na preostali dug:

$$K_m = D_m \times i$$

gdje je (D_m) preostali dug u mjesecu (m), dok je (i) — mjesecna kamatna stopa.

3) Mjesecna rata (R_m) :

$$R_m = G + K_m$$

Kako se dug smanjuje, kamatni dio se smanjuje, a time i ukupna rata.

Prednosti ovog načina obračuna mjesecne rate se ogledaju u manjim ukupnim troškovima kamate nego kod anuitetnog modela. Dug se brže smanjuje zbog konstantnog vraćanja glavnice. Sa druge strane nedostatak predstavlja veće početne

rate, koje mogu biti finansijsko opterećenje. U praksi korisnici kredita žele fiksne mjesecne obaveze, kako bi pojednostavili svoje planiranje.

U nastavku je primjer u Python-u koji generiše amortizacioni plan sa jednakim otplatama glavnice. Prepostavimo kredit od 10.000 KM, rok otplate 5 godina (60 mjeseci), uz godišnju kamatnu stopu od 6%.

```

def otplata_jednake_glavnice(glavnica, godisnja_stopa, broj_mjeseci):
    # Mjesečna kamatna stopa
    i = godisnja_stopa / 12
    # Jednaki iznos glavnice po rati
    otplata_glavnice = glavnica / broj_mjeseci
    # Početni saldo
    saldo = glavnica

    plan = []
    for mjesec in range(1, broj_mjeseci + 1):
        kamata = saldo * i
        rata = otplata_glavnice + kamata
        saldo -= otplata_glavnice

        plan.append({
            "mjesec": mjesec,
            "rata": rata,
            "kamata": kamata,
            "glavnica_placena": otplata_glavnice,
            "preostali_dug": saldo
        })
    return plan

# Parametri kredita
glavnica_kredita = 10000.0
godisnja_nominalna_stopa = 0.06
mjeseci_otplate = 60

# Generišemo plan otplate
plan_otplate = otplata_jednake_glavnice(glavnica_kredita, godisnja_nominalna_stopa,
mjeseci_otplate)

# Ispis plana otplate
print(f"{'Mjesec':>6} | {'Rata':>10} | {'Kamata':>10} | {'Glavnica':>10} | "
{'Preostali dug':>14}")
print("-" * 60)

for mjesec in plan_otplate:
    print(f"{mjesec['mjesec']:6d} | {mjesec['rata']:10.2f} | "
{mjesec['kamata']:10.2f} | {mjesec['glavnica_placena']:10.2f} |
{mjesec['preostali_dug']:14.2f})
```

Mjesec	Rata	Kamata	Glavnica	Preostali dug
1	216.67	50.00	166.67	9833.33
2	215.83	49.17	166.67	9666.67
3	215.00	48.33	166.67	9500.00
4	214.17	47.50	166.67	9333.33
5	213.33	46.67	166.67	9166.67
6	212.50	45.83	166.67	9000.00
7	211.67	45.00	166.67	8833.33

8	210.83	44.17	166.67	8666.67
9	210.00	43.33	166.67	8500.00
10	209.17	42.50	166.67	8333.33
11	208.33	41.67	166.67	8166.67
...				
45	180.00	13.33	166.67	2500.00
46	179.17	12.50	166.67	2333.33
47	178.33	11.67	166.67	2166.67
48	177.50	10.83	166.67	2000.00
49	176.67	10.00	166.67	1833.33
50	175.83	9.17	166.67	1666.67
51	175.00	8.33	166.67	1500.00
52	174.17	7.50	166.67	1333.33
53	173.33	6.67	166.67	1166.67
54	172.50	5.83	166.67	1000.00
55	171.67	5.00	166.67	833.33
56	170.83	4.17	166.67	666.67
57	170.00	3.33	166.67	500.00
58	169.17	2.50	166.67	333.33
59	168.33	1.67	166.67	166.67
60	167.50	0.83	166.67	0.00

7.6. Faktor akumulacije - Prve finansijske tablice

Faktor akumulacije predstavlja osnovnu veličinu u finansijskoj matematici, koja opisuje koliko će se neki početni iznos uvećati tokom određenog perioda, uz određenu kamatu stopu i prepostavljenu periodičnu kapitalizaciju. Ovaj faktor najčešće se označava kao $(1 + i)^n$, gdje je i kamatna stopa po periodu, a n broj perioda. Ideja je da se svaka kamatna rata dodaje glavnici, čime se ostvaruje efekat složenog (komponovanog) interesa.

Definicija faktora akumulacije

U najjednostavnijoj formi, faktor akumulacije za jedan period sa kamatnom stopom i jeste:

$$(1 + i)$$

Ako se posmatra veći broj perioda (n), a kamata se nakon svakog perioda reinvestira (dodaje glavnici), tada se faktor akumulacije množi sam sa sobom onoliko puta koliko traje ukupan period:

$$(1 + i)^n$$

Na primjer, ukoliko iznos od 1€ raste uz 5% godišnje kamatne stope kroz pet godina, krajnja vrijednost biće:

$$1 \times (1 + 0.05)^5 = 1.27628 \approx 1.28$$

Dakle, faktor akumulacije u ovom slučaju je $(1+0.05)^5 \approx 1.27628$.

Kreiranje prve finansijske tablice za akumulaciju

U praksi, finansijske tablice se često koriste kako bi se za različite kamatne stope i različit broj perioda (godina, mjeseci, kvartala) odmah mogla očitati vrijednost faktora akumulacije. To može biti u obliku:

- 1) Redova koji predstavljaju broj godina (npr. od 1 do 10, ili od 1 do 30).
- 2) Kolona koje predstavljaju različite godišnje kamatne stope (npr. 3%, 4%, 5%, 6%, itd.).

Svakom od ovih brojeva može se pristupiti kao „faktoru akumulacije“ za odgovarajuću kombinaciju kamatne stope i broja godina. Na primjer, faktorska vrijednost 1.2763 znači da će se 1 KM uz 5% godišnju stopu kroz 5 godina uvećati na 1.2763 KM.

Tabela 7.1. Tipičan izgled tablice faktora akumulacije

n	2.0%	2.5%	3.0%	4.0%	5.0%	6.0%	7.0%	8.0%
1	1.0200	1.0250	1.0300	1.0400	1.0500	1.0600	1.0700	1.0800
2	1.0404	1.0506	1.0609	1.0816	1.1025	1.1236	1.1449	1.1664
3	1.0612	1.0769	1.0927	1.1249	1.1576	1.1910	1.2250	1.2597
4	1.0824	1.1038	1.1255	1.1699	1.2155	1.2625	1.3108	1.3605
5	1.1041	1.1314	1.1593	1.2167	1.2763	1.3382	1.4026	1.4693

U nastavku će biti prikazano kako se može u Python-u generisati tablica akumulacionih faktora za više kamatnih stopa i više godina. Koristi se Pandas DataFrame radi preglednog formata.

```
import pandas as pd

def kreiraj_tabelu_akumulacije(godine_max, stope):
    """
    Kreira 'prvu finansijsku tablicu' akumulacije do godine_max (n)
    za listu kamatnih stopa (stopa u decimalnom obliku, npr. 0.03, 0.04...).
    """
    indeks_godina = range(1, godine_max + 1)
    # Struktura će biti: kolone su stope, a redovi su godine
    kolone = {}

    for stopa in stope:
        vrijednosti = []
        for n in indeks_godina:
            faktor = (1 + stopa) ** n
            vrijednosti.append(round(faktor, 4))
        # Ključ kolone biće npr. '3%' za stopu 0.03
        naziv_kolone = f"{round(stopa*100, 1)}%"
        kolone[naziv_kolone] = vrijednosti

    df = pd.DataFrame(kolone, index=indeks_godina)
    df.index.name = "Godina (n)"
    return df

# Primjer upotrebe
stope_za_ispis = [0.02, 0.025, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08]
tabela_akumulacije = kreiraj_tabelu_akumulacije(10, stope_za_ispis)
print(tabela_akumulacije)
```

Godina (n)	2.0%	2.5%	3.0%	4.0%	5.0%	6.0%	7.0%	8.0%
1	1.0200	1.0250	1.0300	1.0400	1.0500	1.0600	1.0700	1.0800
2	1.0404	1.0506	1.0609	1.0816	1.1025	1.1236	1.1449	1.1664
3	1.0612	1.0769	1.0927	1.1249	1.1576	1.1910	1.2250	1.2597
4	1.0824	1.1038	1.1255	1.1699	1.2155	1.2625	1.3108	1.3605
5	1.1041	1.1314	1.1593	1.2167	1.2763	1.3382	1.4026	1.4693
6	1.1262	1.1597	1.1941	1.2653	1.3401	1.4185	1.5007	1.5869
7	1.1487	1.1887	1.2299	1.3159	1.4071	1.5036	1.6058	1.7138

8	1.1717	1.2184	1.2668	1.3686	1.4775	1.5938	1.7182	1.8509
9	1.1951	1.2489	1.3048	1.4233	1.5513	1.6895	1.8385	1.9990
10	1.2190	1.2801	1.3439	1.4802	1.6289	1.7908	1.9672	2.1589

Promjenom vrijenosti u nizu `stope_za_ispis` moguće je mijenjati kamatne stope za koje se generiše Prva finansijska tablica. Opseg godina za koje se generišu ove tablice definisan je kao prvi parametar u pozivu funkcije `kreiraj_tabelu_akumulacije()`.

Umjesto da tražimo izraze $(1+i)^n$ svaki put kada želimo da proračunamo buduću vrijednost neke investicije, finansijske tablice se mogu unaprijed izraditi i predstavljaju koristan orijentir za planiranje investicija, kredita ili štednje.

Praktična upotreba

U nastavku je primjer u kome posmatramo scenarij gdje se 1.000 KM uplaćuje na kraju svake godine tokom 10 godina uz godišnju kamatnu stopu od 5% i u kome tražimo iznos na kraju desete godine. Potrebno je izračunati budući vrijednost, koja se dobije nakon perioda investiranja. Ovaj primjer ćemo riješiti na tri načina:

- 1) Direktno primjenom formula za računanje buduće vrijednosti.

$$FV = D \times \frac{(1+i)^n - 1}{i}$$

gdje je:

- D – iznos godišnje uplate (1.000 KM),
- i – godišnja kamatna stopa (0.05),
- n – broj godina (10).

- 2) Iterativnim proračunom, gdje se svake godine prvo obračuna kamata na postojeći saldo, a zatim na kraju godine sumiraju prethodni saldo sa obračunatim kamatama. Dobijena suma predstavlja novi saldo za sljedeću godinu.
- 3) Upotrebom gotove funkcije `npf.fv` iz modula `numpy_financial`. Ova funkcija ima sintaksu

```
fv = npf.fv(stopa, nper, pmt, pv)
```

gdje je:

- `rate` – Kamatna stopa po periodu.

- `nper` – Ukupan broj perioda.
- `pmt` – Plaćanje po periodu (za redovne uplate ili isplate).
- `pv` – Sadašnja vrijednost (početna investicija, često negativna jer predstavlja odliv novca).
- `when` – Kada se plaćanja vrše ('`begin`' ili '`end`' perioda), podrazumjevano je '`end`'.

```

import numpy_financial as npf

def FV(uplata, godisnja_stopa, broj_godina):
    """
    Direktna formula za buduću vrijednost (FV) obične anuitetne serije.
    FV = D * [(1 + i)^n - 1] / i
    """
    return uplata * ((1 + godisnja_stopa) ** broj_godina - 1) / godisnja_stopa

def FV_iterativno(uplata, godisnja_stopa, broj_godina):
    """
    Iterativni obračun: svake godine se
    1) prvo obračuna kamata na postojeći saldo
    2) doda uplata na kraju godine
    Vraća krajnji iznos poslije n godina.
    """
    saldo = 0.0
    istorija = [] # Za tabelu

    for godina in range(1, broj_godina + 1):
        kamata = saldo * godisnja_stopa # Kamata na postojeći saldo
        saldo += kamata # Novo stanje poslije kamate
        saldo += uplata # Uplata na kraju godine
        istorija.append({
            "godina": godina,
            "kamata": kamata,
            "uplata": uplata,
            "stanje_poslije_uplate": saldo
        })
    return saldo, istorija

# Parametri
uplata_godisnja = 1000.0
stopa = 0.05
n = 10

# 1) Direktna formula
fv_formulom = FV(uplata_godisnja, stopa, n)

# 2) Iterativni pristup
fv_iterativno, istorija = FV_iterativno(uplata_godisnja, stopa, n)

# 3) Korištenje numpy_financial.fv
# Napomena: pmt je negativan jer predstavlja uplatu, odnosno finansijski odliv
fv_npf = npf.fv(stopa, n, -uplata_godisnja, 0, when='end')

# Tabelarni ispis rezultata iterativnog pristupa
print(f"{'Godina':>6} | {'Kamata (KM)':>12} | {'Uplata (KM)':>12} | {'Stanje poslije update':>22}")
print("-" * 62)
for zapis in istorija:
    print(zapis)

```

```

print(f"{zapis['godina']:6} | "
      f"{zapis['kamata']:12.2f} | "
      f"{zapis['uplata']:12.2f} | "
      f"{zapis['stanje_poslije_uplate']:22.2f}")
print("-" * 62)

# Rezultati
print(f"Rezultat (direktna formula) : {fv_formulom:.2f} KM")
print(f"Rezultat (iterativno)      : {fv_iterativno:.2f} KM")
print(f"Rezultat (numpy_financial) : {fv_npf:.2f} KM")

```

Godina	Kamata (KM)	Uplata (KM)	Stanje poslije uplate
1	0.00	1000.00	1000.00
2	50.00	1000.00	2050.00
3	102.50	1000.00	3152.50
4	157.62	1000.00	4310.12
5	215.51	1000.00	5525.63
6	276.28	1000.00	6801.91
7	340.10	1000.00	8142.01
8	407.10	1000.00	9549.11
9	477.46	1000.00	11026.56
10	551.33	1000.00	12577.89

```

Rezultat (direktna formula) : 12,577.89 KM
Rezultat (iterativno)      : 12,577.89 KM
Rezultat (numpy_financial) : 12,577.89 KM

```

7.7. Diskontni faktor - Druge finansijske tablice

Kada govorimo o sadašnjoj vrijednosti novčanih tokova, ključno je shvatiti da 1 KM danas nije isti kao 1 KM sutra ili za godinu dana. Vrijednost novca opada s vremenom, bilo zbog inflacije, troškova propuštenih prilika ili drugih tržišnih okolnosti. Upravo zbog toga se uvodi diskontni faktor – veličina kojom se budući novčani iznos množi da bismo dobili njegovu ekvivalentnu vrijednost u sadašnjem trenutku (tzv. *present value*).

Diskontni faktor za jednu obračunsku godinu s godišnjom kamatnom stopom r (u decimalnom obliku) obično se označava kao:

$$DF = \frac{1}{(1+r)}$$

Ako je u pitanju period od n godina s godišnjom stopom r , diskontni faktor postaje:

$$DF = \frac{1}{(1+r)^n}$$

U suština ako vaša projekcija kaže da će nakon n godina dobiti određeni iznos novca, množenjem tog iznosa sa DF dobijate njegovu današnju vrijednost. Ta logika je osnova u finansijskom odlučivanju, jer omogućava menadžerima, investitorima ili pojedincima da procijene koliko je zapravo vrijedno imati 1.000 KM u budućnosti naspram 1.000 KM danas.

Kreiranje druge finansijske tablice

Nakon što smo u prvoj finansijskoj tablici sistematično predstavili faktore akumulacije $(1+i)^n$, u drugoj tablici fokus je na diskontnim faktorima (DF). Pomoću tih tablica moguće je brzo procijeniti sadašnju vrijednost novčanog iznosa, koja će biti primljena u budućnosti.

Tabela 7.2. Tipičan izgleda tablice diskontnih faktora

n	2.0%	2.5%	3.0%	4.0%	5.0%	6.0%	7.0%	8.0%
1	0.9804	0.9756	0.9709	0.9615	0.9524	0.9434	0.9346	0.9259
2	0.9612	0.9518	0.9426	0.9246	0.9070	0.8900	0.8734	0.8573
3	0.9423	0.9286	0.9151	0.8890	0.8638	0.8396	0.8163	0.7938
4	0.9238	0.9060	0.8885	0.8548	0.8227	0.7921	0.7629	0.7350
5	0.9057	0.8839	0.8626	0.8219	0.7835	0.7473	0.7130	0.6806

U tabeli, svaka ćelija predstavlja DF za odgovarajuću kamatnu stopu (3%, 4%, 5%, 6%, ...) i broj godina (1, 2, 3, 4, 5, ...). Praktična korist je u brzom očitavanju diskontnog faktora umjesto ponovnog računanja.

U nastavku će biti prikazana Python-u skripta, koja generiše tablicu diskontnih faktora za više kamatnih stopa i više godina. Logika je slična tablici akumulacije, samo se sada računa inverzna vrijednost.

```
import pandas as pd

def kreiraj_tabelu_diskontnih_faktora(godine_max, stope):
    """
    Kreira 'drugu finansijsku tablicu' - tabelu diskontnih faktora
    do godine_max (n) za listu kamatnih stopa (u decimalnom obliku, npr. 0.03,
    0.05).
    """
    indeks_godina = range(1, godine_max + 1)
    kolone = {}

    for stopa in stope:
        vrijednosti = []
        for n in indeks_godina:
            # Diskontni faktor = 1 / (1 + stopa)^n
            diskont = 1 / ((1 + stopa) ** n)
            vrijednosti.append(round(diskont, 4))

        # Naziv kolone, npr. '3%' za 0.03
        naziv_kolone = f"{round(stopa*100,1)}%"
        kolone[naziv_kolone] = vrijednosti

    df = pd.DataFrame(kolone, index=indeks_godina)
    df.index.name = "Godina (n)"
    return df

# Primjer upotrebe
stope_za_ispis = [0.02, 0.025, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08]
tabela_diskonta = kreiraj_tabelu_diskontnih_faktora(10, stope_za_ispis)
print(tabela_diskonta)
```

	2.0%	2.5%	3.0%	4.0%	5.0%	6.0%	7.0%	8.0%
Godina (n)	0.9804	0.9756	0.9709	0.9615	0.9524	0.9434	0.9346	0.9259
1	0.9612	0.9518	0.9426	0.9246	0.9070	0.8900	0.8734	0.8573
2	0.9423	0.9286	0.9151	0.8890	0.8638	0.8396	0.8163	0.7938
3	0.9238	0.9060	0.8885	0.8548	0.8227	0.7921	0.7629	0.7350
4	0.9057	0.8839	0.8626	0.8219	0.7835	0.7473	0.7130	0.6806
5	0.8880	0.8623	0.8375	0.7903	0.7462	0.7050	0.6663	0.6302
6	0.8706	0.8413	0.8131	0.7599	0.7107	0.6651	0.6227	0.5835
7	0.8535	0.8207	0.7894	0.7307	0.6768	0.6274	0.5820	0.5403
8	0.8368	0.8007	0.7664	0.7026	0.6446	0.5919	0.5439	0.5002
9	0.8203	0.7812	0.7441	0.6756	0.6139	0.5584	0.5083	0.4632
10								

Upotreba diskontnog faktora za izračunavanje sadašnje vrijednosti

Ako očekujete da primite budući iznos FV nakon n godina, pri godišnjoj stopi r , njegova sadašnja vrijednost (PV) glasi:

$$PV = \frac{FV}{(1+r)^n} = FV \times \left(\frac{1}{(1+r)^n} \right)$$

Primjer iz ekonomije:

Kompanija treba da naplati iznos od 20.000 KM za 3 godine, a trenutna tržišna kamatna stopa je 6%. Da bi izračunali koliko to vrijedi danas, računamo:

$$PV = 20,000 \times \frac{1}{(1+0.06)^3} \approx 20,000 \times 0.8396 = 16,792 \text{ KM}$$

Time se uočava da iznos od 20.000 KM za 3 godine, pri stopi od 6%, zapravo danas vrijedi oko 16.792 KM.

U nastavku će biti prikazano poređenje buduće i sadašnje vrijednosti novčanog toka za različite kamatne stope. Posmatra se očekivana dobit od 10.000 KM, koju će primiti za 10 godina, pri čemu će se izračunati sadašnja vrijednost uz godišnje kamatne stope od 3%, 5% i 8%. Rezultati poređenja su prikazani u tabelarnom i grafičkom obliku, uz primjenu programskog jezika Python, čime se omogućava jasan i precizan prikaz finansijskih proračuna.

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

# Funkcija za izračunavanje sadašnje vrijednosti
def sadasnja_vrijednost(buduca_vrijednost, stopa, vreme):
    faktor_diskont = 1 / ((1 + stopa) ** vreme)
    return buduca_vrijednost * faktor_diskont

# Parametri
buduca_vrijednost = 10000 # Buduća vrijednost
stope = [0.03, 0.05, 0.08] # Različite kamatne stope (3%, 5%, 8%)
pocetna_godina = 2025
krajnja_godina = 2035
periodi = range(0, krajnja_godina - pocetna_godina + 1) # Od 0 do 10 godina
# Kreiranje tabele (DataFrame)
tabela = pd.DataFrame(index=[pocetna_godina + p for p in periodi], columns=stope)

# Popunjavanje tabele sadašnjim vrijednostima
for s in stope:
    for p in periodi:
        if p == max(periodi):
            # U krajnjoj godini vrijednost je jednaka budućoj vrijednosti
            tabela.loc[pocetna_godina + p, s] = buduca_vrijednost
        else:
            tabela.loc[pocetna_godina + p, s] =
round(sadasnja_vrijednost(buduca_vrijednost, s, max(periodi) - p), 2)

# Ispis tabele
```

```

print("Tabela sadašnjih vrijednosti budućeg iznosa (od 2025. do 2035.):")
print(tabela.round(2))

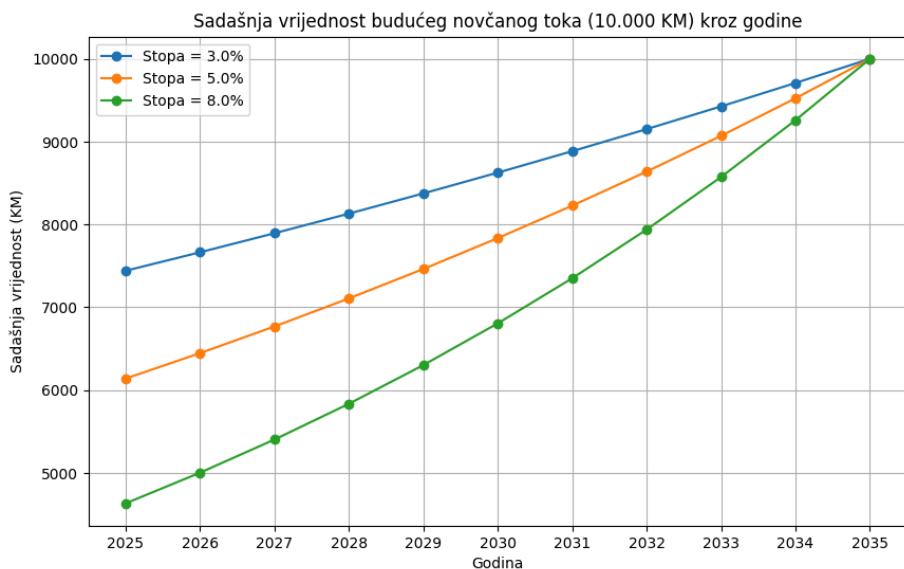
# Crtanje grafika
plt.figure(figsize=(10, 6))
for s in stope:
    plt.plot(tabela.index, tabela[s], label=f'Stopa = {s*100}%', marker='o')

plt.title('Sadašnja vrijednost budućeg novčanog toka (10.000 KM) kroz godine')
plt.xlabel('Godina')
plt.ylabel('Sadašnja vrijednost (KM)')
plt.legend()
plt.grid(True)
plt.xticks(tabela.index) # Postavljanje svih godina na x-osi
plt.show()

```

Tabela sadašnjih vrijednosti budućeg iznosa (od 2025. do 2035.):

	0.03	0.05	0.08
2025	7440.94	6139.13	4631.93
2026	7664.17	6446.09	5002.49
2027	7894.09	6768.39	5402.69
2028	8130.92	7106.81	5834.9
2029	8374.84	7462.15	6301.7
2030	8626.09	7835.26	6805.83
2031	8884.87	8227.02	7350.3
2032	9151.42	8638.38	7938.32
2033	9425.96	9070.29	8573.39
2034	9708.74	9523.81	9259.26
2035	10000	10000	10000



Na grafikonu x-osa prikazuje vremenski period od 2025. do 2035. godine, čime se jasno ilustruje kako se očekivani iznos od 10.000 KM (koji će biti primljen 2035. godine) diskontuje unazad ka 2025. godini. Na ovaj način vizuelno je prikazan procesa diskontovanja.

Poređenje sadašnje i buduće vrijednosti novčanog toka

Pored prethodno prikazanih metoda, gdje su funkcije za izračunavanje diskontnog faktora i sadašnje vrijednosti buduće isplate bile ručno definisane, moguće je koristiti i gotove funkcije iz specijalizovanih Python modula. Jedan od tih modula je [numpy-financial](#), koji pruža skup funkcija za finansijske proračune. Funkcija `pv` (*present value*) iz ovog modula omogućava jednostavno izračunavanje sadašnje vrijednosti budućih novčanih tokova, bez potrebe za ručnim implementiranjem formula.

Sintaksa funkcije je:

```
npf.pv(rate, nper, pmt, fv=0, when='end')
```

gdje je:

- `rate` – Periodična kamatna stopa (npr. godišnja stopa od 5% se unosi kao 0.05).
- `nper` – Ukupan broj perioda (npr. broj godina).
- `pmt` – Plaćanje po periodu (za jednokratnu isplatu se koristi 0).
- `fv` – Buduća vrijednost koju diskontujete (npr. iznos koji očekujete nakon 2 godine).
- `when` – Kada se plaćanja vrše — `'end'` (na kraju perioda, a to je najčešće na kraju mjeseca) ili `'begin'` (na početku perioda). Standardno je `'end'`.

Zamislimo da firma ima dvije ponude za naplatu potraživanja:

- Ponuda A: Odmah isplata u iznosu od 46.000 KM.
- Ponuda B: Isplata u iznosu od 50.000 KM za dvije godine.

Firma želi da utvrdi sadašnju vrijednost budućih 50.000 KM kako bi je uporedila sa ponudom od 46.000 KM koja se može uzeti odmah. Prepostavimo da je tržišna kamatna stopa (ili oportunitetni trošak kapitala) 5% godišnje.

```
import numpy_financial as npf

# Pretpostavljeni parametri
odmah = 46000.0          # Ponuda A (isplata odmah)
buduca_isplata = 50000.0 # Ponuda B (isplata za 2 godine)
godisnja_stopa = 0.05    # 5% godišnje
godine = 2                 # Broj godina do isplate

# Izračunavanje sadašnje vrijednosti koristeći npf.pv
pv_buduca_isplata = abs(npf.pv(rate=godisnja_stopa, nper=godine, pmt=0,
                                  fv=-buduca_isplata))

# Ispis rezultata
print(f"Sadašnja vrijednost buduće isplate od {buduca_isplata:.2f} KM za {godine} godine iznosi {pv_buduca_isplata:.2f} KM.")
```

```
# Poređenje sa trenutnom isplatom
if pv_buduca_isplata > odmah:
    print("Bolje je čekati 2 godine, jer je diskontovana vrijednost buduće isplate
veća od trenutne isplate.")
else:
    print("Bolje je uzeti novac odmah, jer je trenutna ponuda veća od diskontovane
buduće vrijednosti.")
```

Sadašnja vrijednost buduće isplate od 50000.00 KM za 2 godine iznosi 45351.47 KM.
Bolje je uzeti novac odmah, jer je trenutna ponuda veća od diskontovane buduće vrijednosti.

7.8. Faktor dodatnih uloga - Treće finansijske tablice

Faktor dodatnih uloga (**faktor za anuitetu seriju uplata**) opisuje koliki će iznos dostići višestruke ili kontinuirane uplate, kada se one periodično ponavljaju uz definisanu kamatnu stopu, sve do kraja odabranog vremenskog perioda. Za razliku od faktora akumulacije, koji prikazuje rast jednog početnog uloga i diskontnog faktora, koji služi za konverziju budućeg iznosa u današnju vrijednost, faktor dodatnih uloga se primjenjuje u slučajevima redovnih i višestrukih uplata – poput mjesecnih, kvartalnih ili godišnjih investicija ili štednje. Na taj način, možemo precizno izračunati ukupnu akumuliranu sumu svih tih uplata po isteku definisanog perioda. Upravo iz tog razloga, ovaj faktor je neizostavan alat prilikom analiza kontinuiranih investicija, štednih planova i svih oblika anuitetnih uplata.

Koristi se za:

- Štednju: Ako redovno uplaćujete određeni iznos na račun za štednju, faktor dodatnih uloga pokazuje koliko će vaša štednja vrijediti nakon određenog perioda.
- Investicije: Ako redovno ulažete u fondove ili druge instrumente, faktor dodatnih uloga pomaže u procjeni buduće vrijednosti vaših ulaganja.
- Anuiteti: Redovne uplate koje se vrše u jednakim intervalima (npr. mjesечно ili godišnje) mogu se analizirati koristeći ovaj faktor.

Kada se redovno (periodično) uplaćuje određeni iznos – recimo na kraju svake godine – po kamatnoj stopi i i kroz n godina, buduća vrijednost svih tih uplata računa se formulom za običnu anuitetu seriju:

$$FV_{\text{uplate}} = D \times \frac{(1+i)^n - 1}{i}$$

gdje je:

- D – iznos svake periodične uplate,
- i – kamatna stopa po periodu (npr. godišnja),
- n – broj perioda (npr. broj godina).

Dijagram tokova novca kod takve anuitetne serije pokazuje da prva uplata ima $(n-1)$ period za rast, posljednja uplata praktično ostaje u fondu samo trenutak (ukoliko se uplaćuje na **kraju** perioda).

Napomena: Ukoliko se uplate vrše **na početku** svakog perioda, formula za buduću vrijednost je nešto drugačija, ali suština ostaje slična.

$$FV_{\text{uplate}} = D \times \frac{(1+i)^n - 1}{i} \times (1 + i)$$

Kreiranje treće finansijske tablice

Treća finansijska tablica prikazuje faktor dodatnih uloga, tj. vrijednosti $\frac{(1+i)^n - 1}{i}$ za različite kamatne stope i brojne periode (npr. 1, 2, ..., 30 godina). U praksi, svaka celija te tablice govori koliko će se akumulirati, ako svake godine (na kraju perioda) uplaćujete 1 jedinicu novca, pri dатoj kamatnoj stopi i u određenom broju godina.

Dakle, ako u tablici pronađete vrijednost 5.525 pri stopi od 5% za 5 godina, to znači da ćete, ulažući po 1 jedinicu godišnje, na kraju pete godine imati ukupan iznos od 5.525 jedinica. Ako uplaćujete, recimo, 100 jedinica svake godine, taj iznos jednostavno množite sa 100. Na taj način dobijate brz i pregledan način da procijenite efekat redovnih uplata tokom dužeg vremena.

Tabela 7.3. Primjer tablice dodatnih uloga

n	2%	2.5%	3%	4%	5%	6%	7%	8%
1	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
2	2.0200	2.0250	2.0300	2.0400	2.0500	2.0600	2.0700	2.0800
3	3.0604	3.0756	3.0909	3.1216	3.1525	3.1836	3.2156	3.2464
4	4.1216	4.1525	4.1836	4.2465	4.3101	4.3746	4.4399	4.5061
5	5.2040	5.2563	5.3091	5.4163	5.5256	5.6371	5.7507	5.8666

Napomena: Vrijednosti su zaokružene na 4 decimale, pa mogu minimalno odstupati zbog zaokruživanja.

U nastavku je prikazana Python-u skripta, koja generiše tablicu dodatnih uloga, koja prati prethodnu tabelu.

```
import pandas as pd

def kreiraj_tabelu_dodatnih_uloga(godine_max, stopa):
    """
    Kreira 'treću finansijsku tablicu' - tabelu faktora dodatnih uloga
    do godine_max (n) za listu kamatnih stopa (decimalnom obliku, npr. 0.03).
    Formula: ( (1 + i)^n - 1 ) / i
    """
    indeks_godina = range(1, godine_max + 1)
    kolone = {}

    for stopa in stopa:
        vrijednosti = []
        for n in indeks_godina:
            faktor_anuiteta = ((1 + stopa) ** n - 1) / stopa
            vrijednosti.append(round(faktor_anuiteta, 4))
        kolone[stopa] = vrijednosti

    return pd.DataFrame(kolone)
```

```

# Naziv kolone (npr. "3%" za 0.03)
naziv_kolone = f"{stopa*100:.1f}%"
kolone[naziv_kolone] = vrijednosti

df = pd.DataFrame(kolone, index=indeks_godina)
df.index.name = "Godina (n)"
return df

# Glavni dio skripte

# Kamatne stope: 2%, 2.5%, 3%, 4%, 5%, 6%, 7%, 8%
stope_za_ispis = [0.02, 0.025, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08]
# Broj godina = 5
tabela_dodatnih_uloga = kreiraj_tabelu_dodatnih_uloga(5, stope_za_ispis)

print("Tabela faktora dodatnih uloga (anuitetne serije) za 5 godina:")
print(tabela_dodatnih_uloga)

```

	2.0%	2.5%	3.0%	4.0%	5.0%	6.0%	7.0%	8.0%
Godina (n)	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
1	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
2	2.0200	2.0250	2.0300	2.0400	2.0500	2.0600	2.0700	2.0800
3	3.0604	3.0756	3.0909	3.1216	3.1525	3.1836	3.2149	3.2464
4	4.1216	4.1525	4.1836	4.2465	4.3101	4.3746	4.4399	4.5061
5	5.2040	5.2563	5.3091	5.4163	5.5256	5.6371	5.7507	5.8666

Praktična upotreba faktora dodatnih uloga

U sljedećem primjeru biće kreirana funkcija `fv_custom`, koja računa ukupnu akumuliranu sumu za zadati broj perioda, kamatnu stopu i godišnju (ili periodičnu) uplatu. Funkcija omogućava izbor između uplata na početku ili na kraju svakog perioda. Pretpostavka u klasičnom proračunu je da se uplate vrše na kraju svakog perioda (obična anuitetna serija), dok se u slučaju uplata na početku perioda koristi modifikovani obračun poznat kao anuitetna serija s uplatom na početku, gdje svaka uplata odmah počinje da ostvaruje kamatu.

Pored ručne implementacije formule za proračun buduće vrijednosti anuitetnih serija, prikazana je i upotreba ugrađene funkcije `fv` iz modula `numpy-financial`, koja je ranije opisana u okviru izrade prve finansijske tablice. Ova funkcija omogućava jednostavan i tačan proračun buduće vrijednosti bez potrebe za dodatnim formulama.

```

import numpy_financial as npf

def fv_custom(uplata, godisnja_stopa, broj_godina, kada='kraj'):
    """
    Računa buduću vrijednost serije uplata koristeći ručnu formulu.
    Formula:
    - FV = D * ((1 + i)^n - 1) / i [uplata na kraju perioda]
    - FV = D * ((1 + i)^n - 1) / i * (1 + i) [uplata na početku perioda]
    Parametri:
    - uplata: iznos koji se uplaćuje po periodu
    - godisnja_stopa: kamatna stopa
    - broj_godina: broj perioda
    """

```

```

- kada: 'kraj' ili 'pocetak' (uplata na kraju ili početku perioda)
"""

i = godisnja_stopa
fv_manual = uplata * (((1 + i) ** broj_godina - 1) / i)
if kada == 'pocetak':
    fv_manual *= (1 + i)
return fv_manual

# Parametri
uplata = 1000.0      # Godišnja uplata
stopa = 0.05          # 5% godišnja kamatna stopa
godina = 10           # Period od 10 godina

# 1 Ručni proračuni
fv_manual_kraj = fv_custom(uplata, stopa, godina, kada='kraj')
fv_manual_pocetak = fv_custom(uplata, stopa, godina, kada='pocetak')

# 2 Korišćenje ugrađene funkcije npf.fv izvan funkcije
fv_npf_kraj = npf.fv(rate=stopa, nper=godina, pmt=-uplata, pv=0, when='end')
fv_npf_pocetak = npf.fv(rate=stopa, nper=godina, pmt=-uplata, pv=0, when='begin')

# 3 Ispis rezultata
print(f"Ukupna akumulirana vrijednost nakon {godina} godina (uplata na kraju perioda):")
print(f" - Ručno izračunato: {fv_manual_kraj:.2f} KM")
print(f" - npf.fv rezultat: {fv_npf_kraj:.2f} KM\n")
print(f"Ukupna akumulirana vrijednost nakon {godina} godina (uplata na početku perioda):")
print(f" - Ručno izračunato: {fv_manual_pocetak:.2f} KM")
print(f" - npf.fv rezultat: {fv_npf_pocetak:.2f} KM")

```

Ukupna akumulirana vrijednost nakon 10 godina (uplata na kraju perioda):

- Ručno izračunato: 12577.89 KM
- npf.fv rezultat: 12577.89 KM

Ukupna akumulirana vrijednost nakon 10 godina (uplata na početku perioda):

- Ručno izračunato: 13206.79 KM
- npf.fv rezultat: 13206.79 KM

7.9. Faktor aktualizacije - Četvrte finansijske tablice

Kada govorimo o vrednovanju različitih novčanih tokova u vremenu, potrebno je sagledati njihovu sadašnju vrijednost (eng. *Present Value* – PV). Čak i ako se radi o prihodima ili rashodima koji će se realizovati nakon više godina, u praksi je važno znati kolika im je današnja ekvivalentna vrijednost. Upravo zato se koristi proces **aktualizacije**, odnosno diskontovanja, pri čemu se budući iznosi „vraćaju“ u sadašnji trenutak.

U slučaju jednokratnog priliva ili odliva u godini n , ova vrijednost dobija se jednostavnim faktorom

$$\frac{1}{(1+i)^n}$$

gdje je i diskontna (kamatna) stopa, a n broj godina.

Ako, međutim, imamo više novčanih tokova (npr. seriju uplata ili isplata raspoređenih u različitim godinama), ukupna sadašnja vrijednost računa se kao zbir pojedinačnih diskontovanih iznosa:

$$PV = \sum_{t=1}^n \frac{CF_t}{(1+i)^t}$$

gdje je CF_t novčani tok koji se dešava na kraju godine t . Na taj način možemo istovremeno uključiti prilive i odlive (pozitivne i negativne vrijednosti), te dobiti sveukupnu sadašnju vrijednost čitavog investicionog ili finansijskog projekta.

Proces aktualizacije je suprotan akumulaciji. Faktor akumulacije pokazuje kako početni iznos raste kroz n perioda, dok faktor aktualizacije pokazuje koliko budući iznos vrijedi danas. Upravo na ovome počiva većina daljih finansijskih analiza, uključujući proračune **Net Present Value (NPV)** i **Internal Rate of Return (IRR)**, jer se svi budući tokovi konvertuju na zajedničku, današnju osnovu.

Kreiranje četvrte finansijske tablice

Četvrta finansijska tablica obično prikazuje sadašnje vrijednosti novčanih tokova, bilo da se radi o pojedinačnim iznosima ili kumulativnim vrijednostima za čitav niz uplata ili isplata. Za svaku kombinaciju godišnje kamatne stope i broja godina navede se faktori, koji olakšavaju obračun ukupne sadašnje vrijednosti serije novčanih tokova. Opis tabele:

- Redovi odgovaraju uzastopnim godinama (1, 2, 3, ...).
- Kolone predstavljaju različite godišnje kamatne stope (3%, 4%, 5%, 6% i sl.).

- Ćelije mogu prikazivati kumulativne aktualizovane faktore ili koeficijente za diskontovanje serije uplata ili isplata, što omogućava brzo izračunavanje sadašnje vrijednosti za odabranu kombinaciju stope i godina.

Tabela 7.4. Primjer tablice aktualizacije.

n	2.0%	2.5%	3.0%	4.0%	5.0%	6.0%	7.0%	8.0%
1	0.9804	0.9756	0.9709	0.9615	0.9524	0.9434	0.9346	0.9259
2	1.9416	1.9274	1.9135	1.8861	1.8594	1.8334	1.8080	1.7833
3	2.8839	2.8560	2.8286	2.7751	2.7232	2.6730	2.6243	2.5771
4	3.8077	3.7620	3.7171	3.6299	3.5460	3.4651	3.3872	3.3121
5	4.7135	4.6458	4.5797	4.4518	4.3295	4.2124	4.1002	3.9927

Ako pogledate ćeliju za n=5 i i=5% (4.3295), to znači da pet isplata od po 1 KM, na kraju svake od 5 godina, danas zajedno vrijede oko 4.3295 KM ako je diskontna stopa 5%. Ako konkretno svake godine isplaćujete X KM, množite vrijednost iz tabele s X da biste dobili ukupnu sadašnju vrijednost tih uplata.

U nastavku je prikazana Python-u skripta, koja generiše tablicu kumulativnih faktora aktualizacije, koja prate prethodnu tabelu.

```
import pandas as pd

def kreiraj_tabelu_aktualizacije(godine_max, stope):
    """
    Kreira 'četvrtu finansijsku tablicu' - kumulativnu vrijednost
    sum_{k=1}^n [1 / (1+stope)^k] za svaki n u rasponu 1..godine_max
    i za svaku stopu iz liste stope (decimalno, npr. 0.03 = 3%).
    """
    indeks_godina = range(1, godine_max + 1)
    kolone = {}

    for stopa in stope:
        vrijednosti = []
        for n in indeks_godina:
            kumulativni_diskont = 0.0
            for k in range(1, n + 1):
                kumulativni_diskont += 1 / ((1 + stopa) ** k)

            vrijednosti.append(round(kumulativni_diskont, 4))

        naziv_kolone = f"{stopa*100:.1f}%"
        kolone[naziv_kolone] = vrijednosti

    df = pd.DataFrame(kolone, index=indeks_godina)
    df.index.name = "Godina (n)"
    return df
```

```

# Kamatne stope: 2%, 2.5%, 3%, 4%, 5%, 6%, 7%, 8%
stope_za_ispis = [0.02, 0.025, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08]
# Broj godina = 5
tabela_aktualizacije = kreiraj_tabelu_aktualizacije(5, stope_za_ispis)

print("Četvrta finansijska tablica - kumulativni faktori aktualizacije za 5 godina:")
print(tabela_aktualizacije)

```

Četvrta finansijska tablica - kumulativni faktori aktualizacije za 5 godina:								
	2.0%	2.5%	3.0%	4.0%	5.0%	6.0%	7.0%	8.0%
Godina (n)								
1	0.9804	0.9756	0.9709	0.9615	0.9524	0.9434	0.9346	0.9259
2	1.9416	1.9274	1.9135	1.8861	1.8594	1.8334	1.8080	1.7833
3	2.8839	2.8560	2.8286	2.7751	2.7232	2.6730	2.6243	2.5771
4	3.8077	3.7620	3.7171	3.6299	3.5460	3.4651	3.3872	3.3121
5	4.7135	4.6458	4.5797	4.4518	4.3295	4.2124	4.1002	3.9927

Primjer analize sadašnje vrijednosti novčanih tokova

Kompanija planira projekat u koji odmah (godina 0) ulaže 50.000 KM. Potom, u narednim godinama predviđeni su sljedeći prilivi:

- Godina 1: 20.000 KM
- Godina 2: 30.000 KM
- Godina 3: 25.000 KM

Želimo izračunati neto sadašnju vrijednost (NPV) čitavog projekta pri **5%** godišnjoj stopi. Ta stopa odražava alternativni trošak kapitala (tzv. oportunitetni trošak).

NPV se računa tako što se svaki budući priliv diskontuje na današnju vrijednost i zatim sabere, a početna investicija (50.000 KM) unese kao negativan iznos (odliv) na vrijeme 0. Formula:

$$NPV = -50,000 + \frac{20,000}{(1+0.05)^1} + \frac{30,000}{(1+0.05)^2} + \frac{25,000}{(1+0.05)^3}$$

Ako izračunata vrijednost ispadne **pozitivna**, to znači da je projekat isplativ (zarada nadmašuje troškove). Ako je **negativna**, projekat ne pokriva očekivanu stopu od 5%, pa se ne preporučuje.

U praksi, neto sadašnja vrijednost (**NPV – Net Present Value**) predstavlja zbir svih priliva i odliva, svedenih na današnju vrijednost. Interna stopa rentabilnosti (**IRR – Internal Rate of Return**) je ona kamatna stopa pri kojoj je NPV jednak nuli. Ukoliko IRR premašuje traženu stopu (npr. 5%), projekat se smatra ekonomski opravdanim.

Posmatrajmo ovaj primjer uraden u Python-u koristeći standardne matematičke operacije.

```

cash_flows = [-50000, 20000, 30000, 25000] # Godina 0, 1, 2, 3
stopa = 0.05

npv_value = 0.0 # inicijalizacija početne vrijednosti
for t, cf in enumerate(cash_flows):
    # t kreće od 0, 1, 2...
    npv_value += cf / ((1 + stopa) ** t)

print(f"Neto sadašnja vrijednost (ručni obračun): {npv_value:.2f}")

```

Neto sadašnja vrijednost (ručni obračun): 17854.44

Pored pisanja sopstvene funkcije za proračune koristeći proste matematičke operacije, u Pythonu možemo koristiti i modul `numpy_financial`, koji nudi funkcije `npf.npv(rate, values)` i `npf.irr(values)` za računanje neto sadašnje vrijednosti (NPV) i interne stope rentabilnosti (IRR).

Sintaksa funkcija.

- `npf.npv(rate, values)` - Funkcija vraća neto sadašnju vrijednost (NPV) svih navedenih novčanih tokova.
- `npf.irr(values)` - Funkcija vraća internu stopu rentabilnosti (IRR) kao decimalnu vrijednost, tj. stopu pri kojoj je NPV jednak nuli.

Gdje je:

- **rate**: Diskontna stopa po periodu (u decimalnom obliku, npr. 0.05 za 5%).
- **values**: Lista ili niz novčanih tokova, gdje se očekuje da prvi element odgovara prvoj periodi.

U nastavku je rješenje prethodnog primjera upotrebom funkcija `npy` i `irr`. Za pravilan rad koda, neophodno je u okviru vašeg Python okruženja instalirati modul `numpy_financial` sa sljedećom komandom: `pip install numpy_financial`

```

import numpy_financial as npf

# Novčani tokovi, godina 0 do 3:
# -50000 sada (investicija), +20000 u godini 1, +30000 u godini 2, +25000 u godini 3
cash_flows = [-50000, 20000, 30000, 25000]
required_rate = 0.05 # Tražena stopa (5%)

# NPV
npv_val = npf.npv(required_rate, cash_flows)
print(f"Neto sadašnja vrijednost (NPV) na 5% = {npv_val:.2f} KM")

# IRR
irr_val = npf.irr(cash_flows)
print(f"Interna stopa rentabilnosti (IRR) = {irr_val*100:.2f}%")

# Zaključak

```

```
if npv_val > 0:  
    print("Projekat je isplativ prema kriterijumu NPV (pozitivna vrijednost).")  
else:  
    print("Projekat nije isplativ prema NPV (negativna vrijednost).")  
  
if irr_val > required_rate:  
    print("Projekat je isplativ i prema IRR kriterijumu (IRR > tražene stope).")  
else:  
    print("Projekat nije isplativ prema IRR kriterijumu (IRR < tražene stope).")
```

Neto sadašnja vrijednost (NPV) na 5% = 17854.44 KM

Interna stopa rentabilnosti (IRR) = 22.40%

Projekat je isplativ prema kriterijumu NPV (pozitivna vrijednost).

Projekat je isplativ i prema IRR kriterijumu (IRR > tražene stope).

7.10. Faktor povrata - Pete finansijske tablice

Faktor povrata predstavlja koeficijent, koji omogućava jednostavan proračun fiksne rate za otplatu kredita. Ovaj faktor se koristi kako bi se izračunalo koliko iznosi stalna rata, kojom će se dug otplatiti u jednakim iznosima tokom određenog vremenskog perioda, uz unaprijed poznatu kamatnu stopu. Koeficijent se predstavlja sljedećom formulom:

$$\frac{i}{1-(1+i)^{-n}}$$

Kreiranje pete finansijske tablice

Peta finansijska tablica prikazuje faktore povraćaja (amortizacione faktore), koji omogućavaju jednostavan proračun fiksnih anuitetnih rata za otplatu kredita. Ova tablica služi za određivanje visine jednakih uplata potrebnih za potpunu otplatu glavnice i kamata tokom određenog broja godina, uz unaprijed poznatu kamatnu stopu.

Za svaku kombinaciju godišnje kamatne stope i broja godina otplate navode se faktori, koji olakšavaju izračunavanje fiksne rate.

Tabela 7.5. Primjer tablice faktora povraćaja

n	2.0%	2.5%	3.0%	4.0%	5.0%	6.0%	7.0%	8.0%
1	1.0204	1.0256	1.0309	1.0417	1.0526	1.0638	1.0753	1.0870
2	0.5159	0.5198	0.5226	0.5284	0.5378	0.5470	0.5562	0.5654
3	0.3468	0.3496	0.3515	0.3570	0.3672	0.3769	0.3864	0.3956
4	0.2635	0.2657	0.2674	0.2723	0.2820	0.2915	0.3007	0.3096
5	0.2160	0.2179	0.2194	0.2236	0.2330	0.2427	0.2518	0.2608

Struktura pете finansijske tablice:

- Redovi predstavljaju uzastopne godine otplate (1, 2, 3, ...),
- Kolone prikazuju različite godišnje kamatne stope (npr. 2%, 3%, 4%, 5%, itd.),
- Čelije sadrže faktore povraćaja za određenu kombinaciju stope i broja godina.

Primjena ove tablice omogućava korisniku da brzo izračuna visinu fiksne godišnje, mjesecne ili kvartalne rate kredita, jednostavnim množenjem faktora povraćaja sa iznosom kredita. Na ovaj način, peta finansijska tablica olakšava kreiranje amortizacionog plana, prikazujući raspodjelu svake uplate između kamate i otplate glavnice, kao i praćenje preostalog duga tokom vremena.

U nastavku se nalazi Python skripta, koja generiše petu finansijsku tablicu sa faktorima povrata za zadate kamatne stope i periode otplate. Kamatne stope i broj perioda mogu se lako prilagoditi promjenom vrijednosti u skripti. Ako korisnik želi prikaz za druge kamatne stope koje trenutno nisu uključene, dovoljno je da se dodaju nove vrijednosti u niz kamatnih stopa.

Takođe, moguće je promijeniti broj godina ili mjeseci za koje se generiše tabela. Važno je obratiti pažnju na usklađenost kamatnih stopa i perioda. Ako se koriste godišnje kamatne stope, period otplate treba biti izražen u godinama. Ako se tabela želi generisati za mjesecne periode, godišnju kamatnu stopu treba podijeliti sa 12, dok se za kvartalne periode kamatna stopa dijeli sa 4.

Na ovaj način, skripta omogućava fleksibilno generisanje amortizacionih tablica u skladu sa različitim uslovima otplate kredita.

```
import pandas as pd

def kreiraj_tabelu_povracaja(godine_max, stope):
    """
    Kreira tabelu faktora povraćaja (amortizacioni faktor) za period od 1 do
    godine_max,
    za listu godišnjih kamatnih stopa (u decimalnom obliku, npr. 0.03 za 3%).
    Formula: Faktor = i / (1 - (1+i)^(-n))
    """
    indeks_godina = range(1, godine_max + 1)
    kolone = {}

    for stopa in stope:
        vrijednosti = []
        for n in indeks_godina:
            faktor = stopa / (1 - (1 + stopa) ** (-n))
            vrijednosti.append(round(faktor, 4))

        naziv_kolone = f"{stopa * 100:.1f}%"
        kolone[naziv_kolone] = vrijednosti

    df = pd.DataFrame(kolone, index=indeks_godina)
    df.index.name = "Godina (n)"
    return df

# Definisane kamatne stope: 2%, 2.5%, 3%, 4%, 5%, 6%, 7%, 8%
stope_za_ispis = [0.02, 0.025, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08]
max_godina = 5 # Period od 1 do 5 godina

tabela_povracaja = kreiraj_tabelu_povracaja(max_godina, stope_za_ispis)

print("Peta finansijska tablica -faktor povraćaja za 5 godina:")
print(tabela_povracaja)

Peta finansijska tablica -faktor povraćaja za 5 godina:
      2.0%   2.5%   3.0%   4.0%   5.0%   6.0%   7.0%   8.0%
Godina (n)
1      1.0200  1.0250  1.0300  1.0400  1.0500  1.0600  1.0700  1.0800
2      0.5150  0.5188  0.5226  0.5302  0.5378  0.5454  0.5531  0.5608
3      0.3468  0.3501  0.3535  0.3603  0.3672  0.3741  0.3811  0.3880
```

4	0.2626	0.2658	0.2690	0.2755	0.2820	0.2886	0.2952	0.3019
5	0.2122	0.2152	0.2184	0.2246	0.2310	0.2374	0.2439	0.2505

Amortizacioni plan

Primjena faktora povrata omogućava kreiranje amortizacionog plana, u kojem je svaka pojedinačna rata podjeljena, na dio koji ide na otplate kamate i dio koji smanjuje glavnici. Na početku otplate kredita veći dio rate odlazi na pokrivanje kamata, dok se vremenom, kako se glavnica smanjuje, udio kamate u rati smanjuje, a udio otplate glavnice raste. Ova ravnoteža između kamate i glavnice u svakoj rati jasno se vidi u amortizacionim tabelama, koje se koriste kako bi se detaljno pratio tok otplate kredita.

1) Formula za fiksnu godišnju ili mjesecnu ratu kod anuitetne otplate je:

$$R = P \times \frac{i}{1-(1+i)^{-n}}$$

gdje je:

- R – jednaka godišnja rata,
- P – početna glavnica (iznos kredita),
- i – godišnja kamatna stopa (izražena u decimalnom obliku),
- n – broj perioda (godina ili mjeseci).

2) Udio otplate kamate u fiksnoj rati se računa po sljedećeoj formuli:

$$K_t = G_{t-1} \times i$$

gdje je:

- K_t — kamata za period t ,
- G_{t-1} — preostala glavnica prije plaćanja rate za period t ,
- i — kamatna stopa.

Na početku otplate kredita, kamata se obračunava na cijelu preostalu glavnicu.

3) Udio otplate glavnice u fiksnoj rati se računa kao razlika između fiksne rate i kamate za taj period:

$$O_t = R - K_t$$

gdje je:

- O_t — iznos koji smanjuje glavnicu u periodu t ,
- R — fiksna rata,
- K_t — kamata za taj period.

4) Preostala glavnica nakon uplate se smanjuje se za iznos otplate glavnice. Postupak se ponavlja do posljednjeg perioda otplate, kada preostali iznos glavnice ne postane nula.

$$G_t = G_{t-1} - O_t$$

Korišćenjem faktora povrata, onak koji uzima kredit može jednostavno da izračuna koliko će plaćati svakog mjeseca, znajući iznos pozajmljenog novca, kamatnu stopu i rok otplate. Ovo olakšava planiranje ličnih ili poslovnih finansija i pruža jasnu sliku o ukupnom trošku kredita kroz cijelokupni period otplate.

U nastvaku je Python skripta, koja generiše amortizacioni plan otplate kredita gdje su sljedeće uslovi:

- Glavnica P=100,000 KM,
- Godišnja kamatna stopa i=5%=0.05,
- Rok otplate n=5 godina.

U funkciji amortizacioni_plan, koja na osnovu ulaznih parametara generiše Pandas DataFrame, ugrađene su prethodne formula za računanje raspodjele fiksne mjesecne rate na udio u kamati i glavnici.

```
import pandas as pd

def amortizacioni_plan(glavnica, godisnja_stopa, broj_godina):
    """
    Računa amortizacioni plan za kredit uz fiksnu godišnju ratu.

    Parametri:
    - glavnica: početna suma kredita (P)
    - godisnja_stopa: godišnja kamatna stopa (i, npr. 0.05 za 5%)
    - broj_godina: broj godina otplate (n)

    Vraća pandas DataFrame sa sledećim kolonama:
    - Period: redni broj godine
    - Rata: fiksna godišnja rata
    - Kamata: deo rate koji pokriva kamatu
    - Otplata glavnice: deo rate koji se koristi za otplatu glavnice
    - Preostali dug: dug nakon izvršene uplate
    """

    # Izračunavanje fiksne rate prema anuitetnoj formuli
    rata = glavnica * (godisnja_stopa / (1 - (1 + godisnja_stopa) ** (-broj_godina)))
    preostali_dug = glavnica
    plan = []

    for godina in range(1, broj_godina + 1):
        kamata = preostali_dug * godisnja_stopa
        otplata_glavnice = rata - kamata
        preostali_dug -= otplata_glavnice

        plan.append({
            "Period": godina,
```

```

        "Rata": round(rata, 2),
        "Kamata": round(kamata, 2),
        "Otplata glavnice": round(otplata_glavnice, 2),
        "Preostali dug": round(preostali_dug, 2)
    })

df = pd.DataFrame(plan)
return df

# Primer upotrebe
# Pretpostavimo da se uzima kredit od 100,000 KM, godišnja stopa je 5%, a period
# otplate je 5 godina.
glavnica = 100000
godisnja_stopa = 0.05
broj_godina = 5

tabela_amortizacije = amortizacioni_plan(glavnica, godisnja_stopa, broj_godina)
print("Amortizacioni plan kredita:")
print(tabela_amortizacije)

```

Amortizacioni plan kredita:

Period	Rata	Kamata	Otplata glavnice	Preostali dug
0	1	23097.48	5000.00	18097.48
1	2	23097.48	4095.13	19002.35
2	3	23097.48	3145.01	19952.47
3	4	23097.48	2147.38	20950.10
4	5	23097.48	1099.88	21997.60

Ukoliko bi željeli da promjenimo uslove za uzimanje kredita, kao i vremenski period za otplatu kredita, možemo da primjenimo identičnu funkciju, ali sa promjenjenim ulaznim parametrima.

Posmatrajmo prethodne uslovi za uzimanje kredita, ali sa fiksnim mjesечnim ratama. Tada je potrebno godišnju kamatu podijeliti sa 12 mjeseci, a godišnji period otplate pomnožiti sa 12 mjeseci. Time bi sve parametre u definisanoj funkciji sveli na isti vremenski okvir. Parametre je moguće diretno proslijediti funkciji u matematičkom obliku (kao 0.05/12)

```

tabela_amortizacije = amortizacioni_plan(100000, 0.05/12, 60)
print("Amortizacioni plan kredita:")
print(tabela_amortizacije)

```

Amortizacioni plan kredita:

Period	Rata	Kamata	Otplata glavnice	Preostali dug
0	1	1887.12	416.67	1470.46
1	2	1887.12	410.54	1476.58
2	3	1887.12	404.39	1482.74
3	4	1887.12	398.21	1488.91
4	5	1887.12	392.01	1495.12
5	6	1887.12	385.78	1501.35
6	7	1887.12	379.52	1507.60
7	8	1887.12	373.24	1513.88
8	9	1887.12	366.93	1520.19
9	10	1887.12	360.60	1526.53
10	11	1887.12	354.24	1532.89

11	12	1887.12	347.85	1539.27	81944.47
12	13	1887.12	341.44	1545.69	80398.79
13	14	1887.12	334.99	1552.13	78846.66
14	15	1887.12	328.53	1558.60	77288.06
15	16	1887.12	322.03	1565.09	75722.97
16	17	1887.12	315.51	1571.61	74151.36
17	18	1887.12	308.96	1578.16	72573.20
18	19	1887.12	302.39	1584.74	70988.47
19	20	1887.12	295.79	1591.34	69397.13
20	21	1887.12	289.15	1597.97	67799.16
21	22	1887.12	282.50	1604.63	66194.53
...					
48	49	1887.12	91.85	1795.27	20248.63
49	50	1887.12	84.37	1802.75	18445.88
50	51	1887.12	76.86	1810.27	16635.61
51	52	1887.12	69.32	1817.81	14817.81
52	53	1887.12	61.74	1825.38	12992.42
53	54	1887.12	54.14	1832.99	11159.43
54	55	1887.12	46.50	1840.63	9318.81
55	56	1887.12	38.83	1848.29	7470.51
56	57	1887.12	31.13	1856.00	5614.52
57	58	1887.12	23.39	1863.73	3750.79
58	59	1887.12	15.63	1871.50	1879.29
59	60	1887.12	7.83	1879.29	-0.00

8. Prilog A

Osnovne komande za rad sa direktorijumima i fajlovima u Windows i Linux okruženjima

Razumijevanje osnovnih komandi za rad sa direktorijumima i fajlovima u komandnoj liniji ključno je za efikasan rad u programerskom i administrativnom okruženju. Bez obzira da li koristite Windows ili Linux operativni sistem, komandna linija pruža moćan način za upravljanje datotekama, direktorijumima i pokretanje programa.

U ovom prilogu će biti detaljno objašnjene najvažnije komande za rad sa direktorijumima i fajlovima u **Windows Command Prompt-u (CMD)** i **Linux terminalu**. Svaka komanda biće opisana zajedno sa njenom funkcionalnošću, osnovnim parametrima i praktičnim primjerima korištenja.

U nastavku će biti odvojeno prikazane komande za **Windows** i **Linux** okruženja. Komande su razvrstane prema logičkim funkcijama koje obavljaju, kako bi njihova primjena bila preglednija i lakša za razumijevanje. Sljedeće grupe komandi će biti obrađene:

- 1) Navigacija kroz direktorijume:
Komande koje omogućavaju kretanje kroz strukturu direktorijuma na vašem računaru.
- 2) Upravljanje direktorijumima (direktorijumima):
Komande za kreiranje, preimenovanje i brisanje direktorijuma.
- 3) Upravljanje fajlovima:
Komande za kreiranje, kopiranje, premještanje i brisanje fajlova.
- 4) Pregled sadržaja fajlova:
Komande za čitanje i pregled sadržaja tekstualnih fajlova.
- 5) Upravljanje pravima pristupa:
Komande za podešavanje i provjeru prava pristupa fajlovima i direktorijumima.
- 6) Ostale korisne komande:
Različite komande koje mogu olakšati rad, poput provjere sistema informacija ili kreiranja arhiva.

Ovakva struktura omogućava jasno razumijevanje svrhe svake komande, olakšava pronalaženje odgovarajuće funkcije i doprinosi efikasnijem radu u oba operativna sistema.

A.1. Komande u Windows Command Prompt-u (CMD)

A.1.1. Navigacija kroz direktorijume

[cd] Promjena trenutnog direktorijuma

Sintaksa:	<code>cd [putanja]</code>
Opis:	Komanda <code>cd</code> koristi se za promjenu trenutnog direktorijuma u komandnoj liniji. Njena osnovna funkcija je da omogući korisnicima kretanje kroz strukturu direktorijuma u sistemu, kao i pristup resursima u različitim direktorijumima.
Parametri:	<code>putanja</code> : Putanja do direktorijuma u koji želite preći. Može biti relativna ili apsolutna.
Primjeri:	<code>cd Documents</code> Prelazak u direktorijum <code>Documents</code> <code>cd C:\Users\predrag\Desktop</code> Prelazak u direktorijum <code>Desktop</code> <code>cd ..</code> Vraćanje u prethodni direktorijum <code>cd ~</code> Vraćanje u home (Početni) direktorijum

[dir] Prikaz sadržaja direktorijuma

Sintaksa:	<code>dir [parametri]</code>
Opis:	Komanda <code>dir</code> prikazuje listu fajlova i direktorijuma u trenutnom ili navedenom direktorijumu.
Parametri:	<code>/p</code> - Zaustavlja prikaz nakon svake stranice. <code>/w</code> - Prikazuje listu u širokom formatu. <code>/a</code> - Prikazuje fajlove sa određenim atributima (npr., skrivene fajlove).
Primjeri:	<code>dir</code> - Prikaz sadržaja trenutnog direktorijuma <code>dir /p</code> - Prikaz sadržaja sa detaljima <code>dir C:\Users\predrag\Desktop</code> - Prikaz sadržaja Desktop direktorijuma

A.1.2. Upravljanje direktorijumima (direktorijumima)

[md][mkdir] Kreiranje novog direktorijuma

Sintaksa:	<code>md [naziv_direktorijuma]</code>
Opis:	Komanda <code>md</code> (<i>make directory</i>) ili <code>mkdir</code> kreira novi direktorijum sa datim imenom.
Parametri:	<code>naziv_direktorijuma</code> - Ime novog direktorijuma koji se želi kreirati.
Primjeri:	<code>md Projekat</code> - Kreira direktorijum sa nazivom Projekat u trenutnom direktorijumu <code>mkdir Projekat</code> - Kreira direktorijum sa nazivom Projekat <code>md C:\Users\Projekat</code> - Kreira direktorijum sa nazivom Projekat na tačno navedenoj putanji

[rd][rmdir] Brisanje direktorijuma

Sintaksa:	<code>rd [parametri][naziv_direktorijuma]</code>
Opis:	Komanda <code>rm</code> (<i>remove directory</i>) ili <code>rmdir</code> briše direktorijum sa datim imenom.
Parametri:	<code>naziv_direktorijuma</code> - Ime direktorijuma koji želite obrisati. <code>/s</code> - Briše direktorijum i sve poddirektorijume i fajlove u njemu. <code>/q</code> - Tihi način rada bez potvrde brisanja.
Primjeri:	<code>rd Projekat</code> - Briše prazan direktorijum sa nazivom Projekat <code>rd /s /q StariProjekat</code> - Briše direktorijum Projekat sa sadržajem

A.1.3. Upravljanje fajlovima

[copy] Kopiranje fajlova

Sintaksa:	<code>copy [izvor] [odredište]</code>
Opis:	Komanda <code>copy</code> kopira jedan ili više fajlova na novu lokaciju.
Parametri:	<code>izvor</code> - Putanja i naziv fajla koji se želi kopirati. <code>odredište</code> - Putanja i naziv gdje se želi kopirati fajl.
Primjeri:	<code>copy dokument.txt Backup</code> - Kopiranje fajla dokument.txt u direktorijum Backup. <code>copy *.txt C:\Backup</code> - Kopiranje svih tekstualnih fajlova u drugi direktorijum.

[move] Premještanje fajlova i direktorijuma

Sintaksa:	<code>move [izvor] [odredište]</code>
Opis:	Komanda <code>move</code> premješta fajlove ili direktorije na novu lokaciju.
Parametri:	<code>izvor</code> - Putanja i naziv fajla ili direktorijuma koji se premješta. <code>odredište</code> - Putanja i naziv gdje se želi premjestiti fajl ili direktorijum.
Primjeri:	<code>move slika.jpg Slike</code> - Premještanje fajla slika.jpg u direktorijum Slike. <code>move Projekat Arhiva</code> - Premještanje direktorijuma Projekat u direktorijum Arhiva.

[del][erase] Brisanje fajlova

Sintaksa:	<code>del [parametri] [fajlovi]</code>
Opis:	Komanda <code>del</code> ili <code>erase</code> briše jedan ili više fajlova.
Parametri:	<code>/p</code> - Traži potvrdu prije brisanja svakog fajla. <code>/f</code> - Forsira brisanje fajlova. <code>/s</code> - Briše navedene fajlove iz svih poddirektorijuma.
Primjeri:	<code>del stari_dokument.docx</code> - Brisanje fajla stari_dokument.docx <code>del /s *.tmp</code> - Brisanje svih tmp fajlova u trenutnom i poddirektorijumu.

[ren][rename] Preimenovanje fajlova i direktorijuma

Sintaksa:	<code>ren [stara_ime] [nova_ime]</code>
Opis:	Komanda <code>ren</code> ili <code>rename</code> preimenuje fajl ili direktorijuma.
Parametri:	<code>stara_ime</code> - Ime fajla koji se želi preimenovati. <code>nova_ime</code> - Novo ime koje se želi dati fajlu.
Primjeri:	<code>ren dokument.txt dokument_stari.txt</code> - Preimenovanje fajla dokument.txt u dokument_stari.txt <code>ren Projekat Projekat_2023</code> - Preimenovanje direktorijuma Projekat u Projekat_2023

A.1.4. Pregled i čitanje sadržaja fajlova

[type] Prikaz sadržaja tekstualnog fajla

Sintaksa:	<code>type [fajl]</code>
Opis:	Komanda <code>type</code> prikazuje sadržaj tekstualnog fajla u komandnoj liniji.
Parametri:	
Primjeri:	<code>type napomene.txt</code> - Prikaz sadržaja fajla napomene.txt

A.1.5. Dodatne korisne komande

[attrib] Prikaz i promjena atributa fajlova

Sintaksa:	<code>attrib [parametri] [fajlovi]</code>
Opis:	Komanda <code>attrib</code> prikazuje ili mijenja atribute fajlova (skriven, samo za čitanje, sistemski fajl).
Parametri:	<code>+H</code> ili <code>-H</code> - Postavlja ili uklanja atribut skrivenog fajla. <code>+R</code> ili <code>-R</code> - Postavlja ili uklanja atribut samo za čitanje. <code>+S</code> ili <code>-S</code> - Postavlja ili uklanja atribut sistemskog fajla.
Primjeri:	<code>attrib dokument.txt</code> - Prikaz atributa fajla dokument.txt. <code>attrib +H dokument.txt</code> - Postavljanje fajla dokument.txt kao skrivenog.

[help] Prikaz pomoći za komande

Sintaksa:	<code>help [komanda]</code>
Opis:	Komanda <code>help</code> prikazuje listu dostupnih komandi ili detaljnju pomoć za određenu komandu.
Parametri:	
Primjeri:	<code>help</code> - Prikaz svih dostupnih komandi. <code>help copy</code> - Prikaz pomoći za komandu copy.

A.2. Komande u Linux terminalu

A.2.1. Navigacija kroz direktorijume

[pwd] Prikaz trenutnog direktorijuma

Sintaksa:	<code>pwd [options]</code>
Opis:	Komanda <code>pwd</code> se koristi za prikaz trenutnog direktorijuma u terminalu.
Parametri:	<code>-L</code> Štampa logičku putanju do tekućeg direktorijuma. <code>-P</code> Štampa fizičku putanju do tekućeg direktorijuma.
Primjeri:	<code>cd Documents</code> Prelazak u direktorijum <code>Documents</code> . <code>cd ..</code> Vraćanje u prethodni direktorijum. <code>cd ~</code> Vraćanje u home (Početni) direktorijum.

[cd] Promjena trenutnog direktorijuma

Sintaksa:	<code>cd [putanja]</code>
Opis:	Komanda <code>cd</code> (<i>change directory</i>) se koristi za promjenu trenutnog direktorijuma u terminalu.
Parametri:	<code>putanja</code> : Putanja do direktorijuma u koji želite preći. Može biti relativna ili apsolutna.
Primjeri:	<code>cd Documents</code> Prelazak u direktorijum <code>Documents</code> <code>cd ..</code> Vraćanje u prethodni direktorijum. <code>cd ~</code> Vraćanje u home direktorijum.

[ls] Prikaz sadržaja direktorijuma

Sintaksa:	<code>ls [parametri] [putanja]</code>
Opis:	Komanda <code>ls</code> prikazuje listu fajlova i direktorijuma u trenutnom ili navedenom direktoriju.
Parametri:	<code>-l</code> - Detaljan prikaz sa informacijama o fajlovima i direktorijumima. <code>-a</code> - Prikazuje sve fajlove, uključujući skrivene (one koji počinju tačkom). <code>-h</code> - Prikazuje veličinu fajla u čovjeku čitljivom formatu (KB, MB).
Primjeri:	<code>ls</code> - Prikaz sadržaja trenutnog direktorijuma. <code>ls -la</code> - Prikaz sadržaja sa detaljima.

A.2.2. Upravljanje direktorijumima

[mkdir] Kreiranje novog direktorijuma

Sintaksa:	<code>mkdir [parametri] naziv_direktorijuma</code>
Opis:	Komanda <code>mkdir</code> kreira novi direktorijum sa datim imenom.
Parametri:	<code>naziv_direktorijuma</code> - Ime novog direktorijuma koji se želi kreirati. <code>-p</code> - Kreira roditeljske direktorije ako ne postoje.
Primjeri:	<code>mkdir Projekat</code> - Kreira direktorijum sa nazivom Projekat <code>mkdir Dir1 Dir2 Dir3</code> - Kreiranje više direktorijuma odjednom. <code>mkdir -p Projekat/2023/April</code> - Kreiranje hijerarhije direktorijuma.

[rmdir] Brisanje direktorijuma

Sintaksa:	rd [parametri][naziv_direktorijuma]
Opis:	Komanda rmdir briše samo prazne direktorije.
Parametri:	naziv_direktorijuma - Ime direktorijuma koji se želi obrisati. --ignore-fail-on-non-empty - Ignoriše greške ako direktorijum nije prazan.
Primjeri:	rd Projekat - Briše prazan direktorijum sa nazivom Projekat.

A.2.3. Upravljanje fajlovima

[cp] Kopiranje fajlova

Sintaksa:	cp [parametri] izvor odredište
Opis:	Komanda copy kopira jedan ili više fajlova na novu lokaciju.
Parametri:	-r Rekurzivno kopiranje direktorijuma. -i Traži potvrdu prije prepisivanja postojećih fajlova. -v Prikazuje informacije o procesu kopiranja.
Primjeri:	cp dokument.txt Backup/ - Kopiranje fajla dokument.txt u direktorijum Backup cp -r Projekat/ Backup/ -Rekurzivno kopiranje direktorijuma Projekat u Backup

[mv] Premještanje fajlova i direktorijuma

Sintaksa:	mv [parametri] izvor odredište
Opis:	Komanda mv premješta fajlove ili direktorije na novu lokaciju.
Parametri:	-i : Traži potvrdu prije prepisivanja. -v : Prikazuje informacije o procesu premještanja.
Primjeri:	mv slika.jpg Slike/ - Premještanje fajla slika.jpg u direktorijum Slike mv stari_dokument.txt novi_dokument.txt -Preimenovanje fajla stari_dokument.txt u novi_dokument.txt

[rm] Brisanje fajlova

Sintaksa:	rm [parametri] [fajlovi]
Opis:	Komanda del ili erase briše jedan ili više fajlova.
Parametri:	-i - Traži potvrdu prije brisanja svakog fajla. -f - Forsira brisanje fajlova samo bez potvrde. -r - Rekurzivno brisanje direktorijuma i njihovog sadržaja.
Primjeri:	rm stari_dokument.docx - Brisanje fajla stari_dokument.docx rm -r Projekat/ - Rekurzivno brisanje direktorijuma Projekat rm -rf StariProjekat/ - Forsirano brisanje bez potvrde

[touch] Kreiranje praznog fajla ili ažuriranje vremenske oznake

Sintaksa:	<code>touch naziv_fajla</code>
Opis:	Komanda <code>touch</code> kreira novi prazan fajl ili ažurira vremensku oznaku postojećeg fajla.
Parametri:	
Primjeri:	<code>touch novi_fajl.txt</code> - Kreiranje novog fajla novi_fajl.txt

A.2.4. Pregled i čitanje sadržaja fajlova

[cat] Prikaz sadržaja fajla

Sintaksa:	<code>cat [parametri] fajlovi</code>
Opis:	Komanda <code>cat</code> prikazuje sadržaj tekstualnog fajla u komandnoj liniji.
Parametri:	<code>-n</code> : Numeriše linije
Primjeri:	<code>cat napomene.txt</code> - Prikaz sadržaja fajla napomene.txt

[less] Pregled sadržaja fajla stranica po stranica

Sintaksa:	<code>less fajl</code>
Opis:	Komanda <code>less</code> omogućava pregled velikih fajlova stranica po stranica.
Parametri:	
Primjeri:	<code>less dug_fajl.txt</code> - Prikaz sadržaja fajla napomene.txt

A.2.5. Upravljanje pravima pristupa

[chmod] Promjena prava pristupa fajlovima i direktorijumima

Sintaksa:	<code>chmod [parametri] prava fajlovi</code>
Opis:	Komanda <code>chmod</code> mijenja prava pristupa za navedene fajlove ili direktorije.
Parametri:	<code>R</code> : Rekurzivni mod <code>v</code> : Verbose mod, prikaz poruka prava: rwx (Read,Write,Execute) pozicije: Vlasnik, Grupa, Ostali
Primjeri:	<code>chmod u+x skripta.sh</code> - Dodavanje prava izvršavanja za vlasnika. <code>chmod 666 dokument.txt</code> - Postavljanje prava čitanja i pisanja za sve.

[chown] Promjena vlasnika fajlova i direktorijuma

Sintaksa:	<code>chown [parametri] vlasnik[:grupa] fajlovi</code>
Opis:	Komanda <code>chown</code> mijenja vlasnika i/ili grupu fajlova i direktorijuma.
Parametri:	<code>-R</code> : Rekurzivni mod. <code>v</code> : Verbose mod, prikaz poruka.
Primjeri:	<code>chown marko dokument.txt</code>

	- Promjena vlasnika fajla na korisnika marko.
--	---

A.2.6. Dodatne korisne komande

[man] Prikaz priručnika za komande

Sintaksa:	<code>man komanda</code>
Opis:	Komanda <code>man</code> (<i>manual</i>) prikazuje detaljnju dokumentaciju za odredenu komandu.
Parametri:	
Primjeri:	<code>man ls</code> - Prikaz priručnika za komandu ls

[echo] Prikaz tekstualnog niza ili vrijednosti varijable

Sintaksa:	<code>echo [niz]</code>
Opis:	Komanda <code>echo</code> prikazuje navedeni tekst ili vrijednost varijable.
Parametri:	
Primjeri:	<code>echo "Dobrodošli u Linux terminal!"</code> - Prikaz jednostavnog teksta. <code>echo \$HOME</code> - Prikaz vrijednosti varijable.

A.3. Dodatne napomene

Razlike između Windows i Linux komandi:

Iako neke komande imaju isto ime (npr. cd), njihove opcije i ponašanje mogu se razlikovati između Windows i Linux okruženja.

Linux je osjetljiv na velika i mala slova, dok Windows uglavnom nije. Ovo znači da u Linux okruženju Fajl.txt i fajl.txt su različiti fajlovi.

Korištenje sudo u Linuxu:

- `sudo` (*superuser do*) se koristi za izvršavanje komandi s privilegijama administratora. Koristite ga pažljivo, npr: `sudo rm -rf /neki_direktorij/`

Navigacija putem absolutnih i relativnih putanja:

- Apsolutne putanje počinju od korijena direktorijuma (npr., `C:\` na Windows-u ili `/` na Linux-u).
- Relativne putanje su u odnosu na trenutni direktorijum (npr., `Documents\Projekat` ili `../Drugidirektorijum`).

9. LITERATURA

- [1] Beezer, R. A. (2022). A First Course in Linear Algebra (4th ed.), CreateSpace Independent Publishing, Seattle.
- [2] Blum, R., & Bresnahan, C. (2022). Linux Command Line and Shell Scripting Bible (4th ed.), Wiley, Indianapolis.
- [3] Boričić B, Ivović M, Ilić M, Matematika, Ekonomski fakultet, Beograd, 2015
- [4] Boyd, S., & Vandenberghe, L. (2024). Introduction to Applied Linear Algebra (2nd ed.), Cambridge University Press, Cambridge.
- [5] Bruce Johnson, Professional Visual Studio 2013, JohnWiley & Sons, Inc., Indianapolis, Indiana, 2014.
- [6] Downey, A. (2021). Think Stats: Exploratory Data Analysis (2nd ed.), O'Reilly Media, Sebastopol.
- [7] Dr Jozo J. Dujmović, Programske jezici i metode programiranja, Akademska misao, Beograd, 2003.
- [8] Dr Lazar Miličević, Mr Lazar Radovanović, Programiranje (Visual Studio 2013), Ekonomski fakultet, Brčko, 2005.
- [9] Dr Tihomir Latinović, Osnove programiranja (Visual Studio 2013), Biblioteka Informacione tehnologije, Banja Luka, 2007.
- [10] Gajić, D., Petrović, A. (2023). Uvod u Python: Osnove programiranja i praktične primene. Elektrotehnički fakultet, Beograd.
- [11] Hull, J. C. (2022). Fundamentals of Futures and Options Markets (10th ed.), Pearson, New York.
- [12] Kočović, J., Pavlović, M. (2018). Uvod u finansijsku matematiku, Ekonomski fakultet, Beograd,
- [13] Kočović, J., Rakonjac-Antić, T. (2017). Zbirka rešenih zadataka iz finansijske i aktuarske matematike, Ekonomski fakultet, Beograd,
- [14] Lutz, M. (2013). Learning Python (5th ed.), O'Reilly Media, Sebastopol.
- [15] Marković, L., Jovanović, S. (2024). Python za početnike: Vežbe, zadaci i algoritmi. Fakultet organizacionih nauka, Beograd.
- [16] Matthes, E. (2023). Python Crash Course (3rd ed.), No Starch Press, San Francisco.
- [17] McKinney, W. (2022). Python for Data Analysis: Data Wrangling with pandas, NumPy, and Jupyter (3rd ed.), O'Reilly Media, Sebastopol.
- [18] McKinney, W. (2023). Effective Pandas: Patterns for Data Manipulation and Analysis, O'Reilly Media, Sebastopol.
- [19] Michael Halvorson, Developer Step by Step Microsoft Visual Studio 2013 2013, O'Reilly Media, Inc., California, 2013.
- [20] Microsoft Visual Studio 2013 Help, Microsoft, 2013.

- [21] Molin, S. (2022). Hands-On Data Analysis with Pandas (2nd ed.), Packt Publishing, Birmingham.
- [22] Nurkanović M., Kurtanović O., (2013). Matematika za ekonomiste, Tuzla, 2013
- [23] Prem S. Mann (2009): Uvod u statistiku (prevod 6. izdanja John Wiley and Sons Inc.), CID Ekonomski fakultet Beograd
- [24] Raschka, S., & Mirjalili, V. (2023). Python Machine Learning (4th ed.), Packt Publishing, Birmingham.
- [25] Shaw, Z. A. (2022). Learn Python 3 the Hard Way: A Very Simple Introduction to Python Programming (3rd ed.), Addison-Wesley, Boston.
- [26] Shotts, W. (2019). The Linux Command Line: A Complete Introduction (2nd ed.), No Starch Press, San Francisco.
- [27] Srđan Damjanović, Predrag Katanić, Borislav Drakul, Zbirka zadataka iz poslovne informatike, Fakultet spoljne trgovine, Bijeljina, 2008.
- [28] Srđan Damjanović, Predrag Katanić, Programske jezike VEE Pro, Elektrotehnički Fakultet, Istočno Sarajevo, 2011.
- [29] Srđan Damjanović, Predrag Katanić, Programske jezike Visual Basic Zbirka zadataka, Fakultet poslovne ekonomije, Bijeljina, 2014.
- [30] Sweigart, A. (2022). Automate the Boring Stuff with Python (2nd ed.), No Starch Press, San Francisco.
- [31] Toomey D., (2016). Learning Jupyter, Packt Publishing
- [32] Urban, M., Murach, J. (2023). Murach's Python Programming (2nd ed.), Mike Murach & Associates, Fresno.
- [33] VanderPlas, J. (2022). Python Data Science Handbook (2nd ed.), O'Reilly Media, Sebastopol.
- [34] Yves J. Hilpisch, (2018). Python for Finance, 2nd Edition, O'Reilly
- [35] Zima, P., & Brown, R. L. (2010). Mathematics of Finance (2nd ed.), Schaum's Outlines, McGraw-Hill, New York.

Internet linkovi:

- [36] CMD Commands (Microsoft),
<https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/windows-commands>, 20.02.2025.
- [37] Compound Interest Calculator,
<https://www.investor.gov/financial-tools-calculators/calculators/compound-interest-calculator>, 20.02.2025.
- [38] Financial Functions in NumPy,
https://numpy.org/doc/stable/reference/generated/numpy_financial.html, 20.02.2025.
- [39] Google Colab, <https://colab.research.google.com/>, 10.02.2025.

- [40] Input-Output Analysis,
<https://www.bea.gov/resources/methodologies/input-output-methods>,
18.02.2025.
- [41] Introduction to Probability and Statistics,
<https://www.khanacademy.org/math/statistics-probability>, 19.02.2025.
- [42] Investopedia Financial Mathematics,
<https://www.investopedia.com/financial-mathematics-5196889>, 20.02.2025.
- [43] JSON Official Website, <https://www.json.org/json-en.html>, 14.02.2025.
- [44] JupyterLab Documentation, <https://jupyterlab.readthedocs.io/en/stable/>,
12.02.2025.
- [45] Linear Algebra Essentials, <https://mathworld.wolfram.com/LinearAlgebra.html>,
18.02.2025.
- [46] Linux Commands Reference, <https://man7.org/linux/man-pages/>, 20.02.2025.
- [47] Matplotlib Documentation, <https://matplotlib.org/stable/index.html>, 16.02.2025.
- [48] NumPy Documentation, <https://numpy.org/doc/>, 15.02.2025.
- [49] NumPy Linear Algebra Documentation,
<https://numpy.org/doc/stable/reference/routines.linalg.html>, 17.02.2025.
- [50] Pandas Documentation, <https://pandas.pydata.org/docs/>, 17.02.2025.
- [51] Pandas Statistical Functions,
<https://pandas.pydata.org/docs/reference/frame.html#computations-descriptive-stats>, 19.02.2025.
- [52] Project Jupyter, <https://jupyter.org/>, 12.02.2025.
- [53] Python Built-in Types, <https://docs.python.org/3/library/stdtypes.html>,
13.02.2025.
- [54] Python Math Module, <https://docs.python.org/3/library/math.html>, 15.02.2025.
- [55] Python Modules, <https://docs.python.org/3/tutorial/modules.html>, 14.02.2025.
- [56] Python Official Documentation, <https://docs.python.org/3/>, 11.02.2025.
- [57] Python Official Website, <https://www.python.org/>, 10.02.2025.
- [58] Python Statistics Module, <https://docs.python.org/3/library/statistics.html>,
19.02.2025.
- [59] Python Tutorial, <https://docs.python.org/3/tutorial/>, 13.02.2025.
- [60] Python virtual environments (venv), <https://docs.python.org/3/library/venv.html>,
12.02.2025.
- [61] SciPy Documentation, <https://docs.scipy.org/doc/scipy/>, 16.02.2025.
- [62] SciPy Linear Algebra Module,
<https://docs.scipy.org/doc/scipy/reference/linalg.html>, 18.02.2025.
- [63] SciPy Stats Documentation,
<https://docs.scipy.org/doc/scipy/reference/stats.html>, 19.02.2025.
- [64] YAML Official Documentation, <https://yaml.org/spec/>, 14.02.2025.
- [65] Zen of Python (PEP 20), <https://peps.python.org/pep-0020/>, 11.02.2025.

CIP