UNIVERZITET U BEOGRADU
ELEKTROTEHNIČKI FAKULTET

PRONALAŽENJE SKRIVENOG ZNANJA

Rešenje projektnog zadatka

Mentor:
Dr Dražen Drašković,
vanredni profesor

Kandidat:
Vladimir Janković
2023/3244

Beograd, jun 2024. godine

Sadržaj

Opsi projektnog zadatka

Projektni zadatak se sastoji iz šest celina koje obuhvataju prokupljanje podataka, njihovu obradu i vizuelizaciju i implementaciju algoritama mašinskog učenja. Podaci za obučavanje i testiranje algoritama mašinskog učenja su prikupljeni sa veb stranice Knjižare Vulkan (https://www.knjizare-vulkan.rs/). Odabrana veb stranica poseduje dovoljno veliki broj knjiga (oko 25.000) za potrebe algoritama mašinskog učenja, čime se izbegava pojava *underfitting*-a.

Rešenje projektnog zadatka je realizovano pomoću programskog jezika *Python*, koji pruža dobru podršku za gotovo sve aspekte projektnog zadatka. U nastavku izveštaja su detaljno opisane metode i tehnologije korišćene za rešavanje svake celine projektnog zadatka.

1. Prikupljanje podataka

Problem prikupljanja podataka sa interneta se rešava korišćenjem robota (eng. bots) koji automatizuju parsiranje i preuzimanje veb stranica. Takođe se koriste i roboti koji indeksiraju preuzete veb stranice kako bi se efikasnije dohvatale željene informacije sa stranica. Postoje razne biblioteke za parsiranje i indeksiranje veb stranica kod mnogih programskih jezika. U programskom jeziku *Python* postoje razne biblioteke koje se mogu koristiti za rešavanje problema prikupljanja podataka sa veb stranica. Dve biblioteke koje dobro rade u paru su *BeautifulSoup* i *Requests*. Prva biblioteka je namenjena za dohvatanje sadržaja iz HTML i XML datoteka, dok druga biblioteka služi za slanje HTTP zahteva za dohvatanje veb stranica. Međutim, *Python* biblioteka *Scrapy*, objedinjuje i automatizuje procese parsiranja i indeksiranja, pa je ona korišćena za prikupljanje podataka.

Biblioteka *Scrapy* se koristi za pravljenje veb indeksera pomoću određenih šablona. Instalacija biblioteke se izvršava sledećom naredbom:

- $pip\ install\ scrapy$

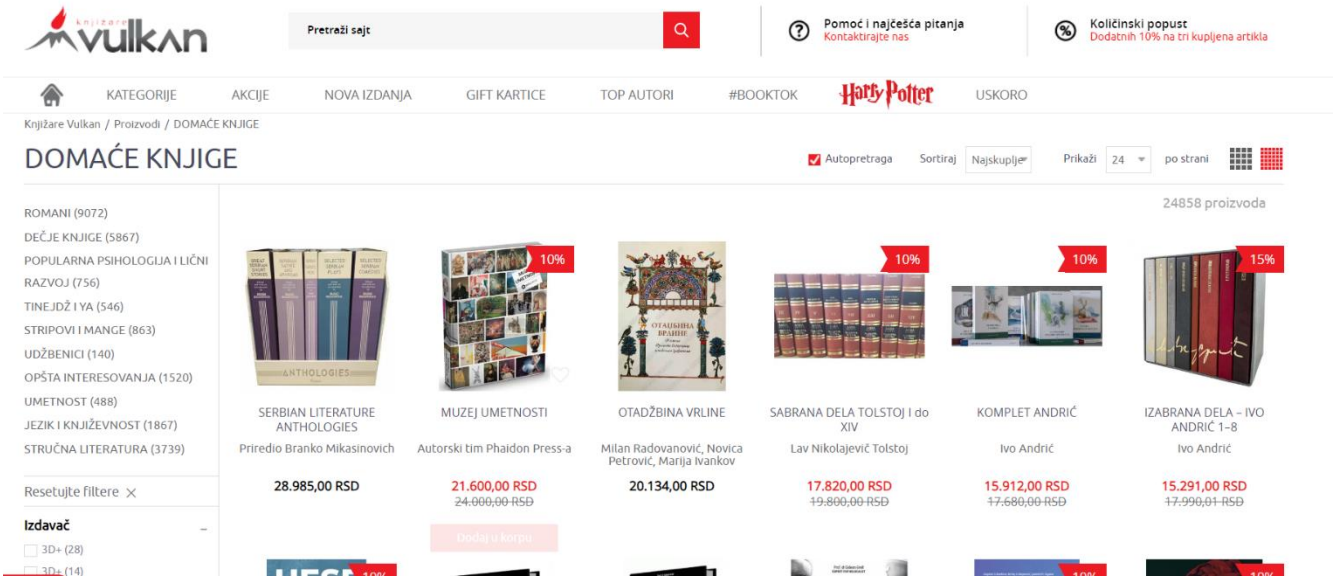
Nakon instalacije biblioteke, pogodno je kreirati projekat koji se generiše po određenom šablonu radi lakše implementacije rešenja zadatka. Sledećom naredbom se kreira nov *Scrapy* projekat:

- $scrapy\ startproject < naziv\_projekta >$

U okviru novokreiranog projekta, postoji nekoliko foldera i datoteka od interesa. Unutar foldera *spiders* se prave i pokreću veb indekseri. Veb indekser se može ručno praviti ili automatski po određenom šablonu izvršavanjem naredbe:

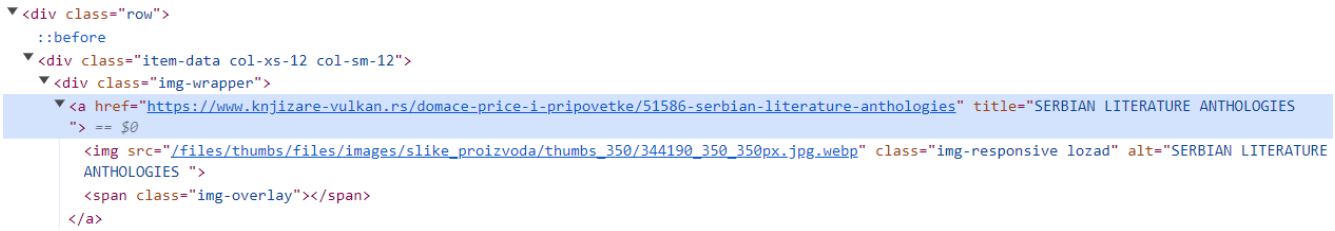
- $scrapy\ genspider < naziv\_indeksera >< domen >$

Naziv_indeksera jedinstveno identifikuje svakog indeksera koji programer napravi. Uloga indeksera, pored dohvatanja veb stranica, je da prati linkove ka drugim stranicama na internetu. Postoji mogućnost da indekser, praćenjem linkova, napusti traženi domen i indeksira neki drugi domen na internetu. Zato je potrebno ograničiti indekser tokom njegovog konstruisanja pomoću argumenta *domen*. Kreirani indekser nasleđuje klasu *scrapy.Spider* koja sadrži atribut *start_urls*, niz početnih linkova koje indekser koristi, i metodu *parse* u okviru koje se implementira ponašanje indeksera.



Slika 1. Veb stranica Knjižare Vulkan – pregled svih knjiga

Na slici 1. je prikazan pregled svih knjiga veb stranice Knjižara Vulkan. Analizom HTML koda je utvrđeno da za svaku knjigu postoji nekoliko linkova koji vode do pregleda informacija datog proizvoda. Rešenje projektnog zadatka koristi linkove koji su postavljeni u okviru HTML elementa koji formira sliku za svaki proizvod, što je prikazano na slici 2.



Slika 2. Link koji sadrži veb adresu za dati proizvod – plavi pravougaonik

Prvi zadatak indeksera je da parsira veb stranicu i dohvati veb adrese za svaki proizvod na datoj stranici. Za svaku veb adresu prikupljenu, indekser šalje zahtev za njihovo preuzimanje i izvršava parsiranje tih veb stranica pomoću metode *parse_book*. Drugi zadatak indeksera je da pređe na narednu veb stranicu kataloga svih proizvoda kako bi dohvatao veb adrese ostalih knjiga. Analizom HTML koda je utvrđeno da desna strelica paginacije sadrži *Javascript* kod za prelazan na narednu stranicu, a ne veb adresu, što je prikazano na slici 3.

Slika 3. Link koji sadrži javascript kod za prelazak na narednu veb stranicu

Daljom analizom veb adresa sajta se može utvrditi da sve relevantne veb adrese kataloga proizvoda imaju isti oblik, gde se samo broj na kraju veb adrese razlikuje:

- https://www.knjizare-vulkan.rs/domace-knjige/page-1
- https://www.knjizare-vulkan.rs/domace-knjige/page-56
- https://www.knjizare-vulkan.rs/domace-knjige/page-734
- ...

Rešenje ovog problema podrazumeva izvlačenje broja iz *Javascript* funkcije sa slike 3. i konkatenacija na kraju niske oblika „https://www.knjizare-vulkan.rs/domace-knjige/page-“. Indekser koristi ovu novu veb adresu kako bi otišao na narednu veb stranicu i ponovo pokrenuo funkciju *parse* za parsiranje naredne stranice. Poslednja veb stranica u katalogu ne sadrži desnu strelicu u paginaciji, što označava kraj kataloga i nakon indeksiranja knjiga sa poslednje veb stranice, indekser prekida rad. Implementacija veb indeksera je prikazana na slikama 4. i 5.

```python
import scrapy
import re

from bookspider.items import BookItem

class MyspiderSpider(scrapy.Spider):
    name = "myspider"
    allowed_domains = ["knjizare-vulkan.rs"]
    start_urls = ["https://www.knjizare-vulkan.rs/domace-knjige"]

    custom_settings = {
        'FEEDS': {
            'knjige.json': {'format': 'json'},
        }
    }

    def parse(self, response):
        book_links = response.css('div.product-listing-items div.text-wrapper div.title a::attr(href)').getall()
        for link in book_links:
            yield response.follow(url=link, callback=self.parse_book)

        next_page = response.css('li.next.first-last a::attr(href)').get()
        if next_page is not None:
            next_page = next_page.split('(')[1].split(')')[0]
            next_page_url = 'https://www.knjizare-vulkan.rs/domace-knjige/page-' + next_page
            yield response.follow(url=next_page_url, callback=self.parse)

```

Slika 4. Izvornog koda veb indeksera MyspiderSpider – deo 1.

```
28
29      def parse_book(self, response):
30
31          text = response.css('#tab_product_description::text').getall()
32          text = [re.sub("\r\n", "", t).strip() for t in text]
33          text = [t for t in text if t != ""]
34          text = " ".join(text)
35
36          table_rows = response.css('table.table tbody tr *::text').getall()
37          table_rows = [re.sub("\r\n", "", t).strip() for t in table_rows]
38          table_rows = [t for t in table_rows if t != ""]
39          table_dict = {}
40          for k in range(0, len(table_rows), 2):
41              table_dict[table_rows[k]] = table_rows[k+1]
42
43          book_item = BookItem()
44
45          book_item['naziv'] = response.css('div.title h1 span::text').get()
46          book_item['autor'] = table_dict.get('Autor')
47          book_item['kategorija'] = table_dict.get('Kategorija')
48          book_item['izdavac'] = table_dict.get('Izdavač')
49          book_item['godina_izdavanja'] = table_dict.get('Godina')
50          book_item['broj_strana'] = table_dict.get('Strana')
51          book_item['tip_poveza'] = table_dict.get('Povez')
52          book_item['format'] = table_dict.get('Format')
53          book_item['opis'] = text
54          book_item['cena'] = response.css('.product-price-without-discount-value::text').get()
55
56          yield book_item
```

Slika 5. Izvornog koda veb indeksera MyspiderSpider – deo 2.

Treći zadatak indeksera je da parsira preuzetu veb stranicu knjige i dohvati sve relevantne informacije vezane za knjigu: naziv, autora/autore, kategoriju, izdavača, godinu izdavanja, broj strana, tip poveza, format, opis i cenu. Funkcije *parse* i *parse_book* u svom potpisu poseduju parametar *response* koji sadrži odgovor na zahtev za preuzimanje veb stranice. Parametar *response* predstavlja objekat klase *Response* iz biblioteke *Scrapy*, i poseduje „selektore“, u vidu metoda *css* i *xpath*, kojima dohvata željeni sadržaj iz HTML dokumenta. U metodi *parse*, selektorima se dohvataju potrebne veb adrese za navigaciju, a u metodi *parse_book* se selektorima dohvataju informacije o knigama. Implementacija rešenja koristi modul *re*, koji služi za rad sa regularnim izrazima, kako bi se odrežene informacije lakše formatirale. Sve relevantne informacije vezane za knjigu se upisuju u polja objekta *book_item*, klase *BookItem*, koji se dalje šalje na obradu. U slučaju da neka informacija vezana za knjigu se ne nalazi na veb stranici, obezbeđuje se da vrednost tog polja objekta *book_item* bude vrednosti *None*.

Pri kreiranju *Scrapy* projekta, generisana je datoteka *items.py* u kojoj se definišu klase objekata u kojima se smeštaju parsirani podaci. Objekti takvih klasa, nakon parsiranja, prolaze kroz *pipeline Scrapy* ahritekture za dalju obradu. Klasa *BookItem* je definisana u datoteci *items.py* u čijim poljima se čuvaju potrebne informacije o knjigama. Na slici 6. je prikazana implementacija klase *BookItem.*

```python
class BookItem(scrapy.Item):
    naziv = scrapy.Field()
    autor = scrapy.Field()
    kategorija = scrapy.Field()
    izdavac = scrapy.Field()
    godina_izdavanja = scrapy.Field()
    broj_strana = scrapy.Field()
    tip_poveza = scrapy.Field()
    format = scrapy.Field()
    opis = scrapy.Field()
    cena = scrapy.Field()
```

Slika 6. Definicija klase BookItem

Pored datoteke *items.py*, generisana je i datoteka *pipelines.py* u kojoj se definišu dalje obrade sa objektima za smeštanje podataka. *Pipeline*-ovi se implementiraju kao klase koje sadrže metodu *process_item*, u okviru koje se dalje obrađuju podaci. Klasa *BookspiderPipeline* vrši obradu podataka pretvaranjem niske za cenu, broj strana i godinu izdavanja u celobrojnu vrednost. Metoda *process_item* treba da vrati objekat nakon obrade kako bi se poslao na dalju obradu u okviru *pipeline*-a. Na slici 7. je prikazana implementacija klase *BookspiderPipeline.*

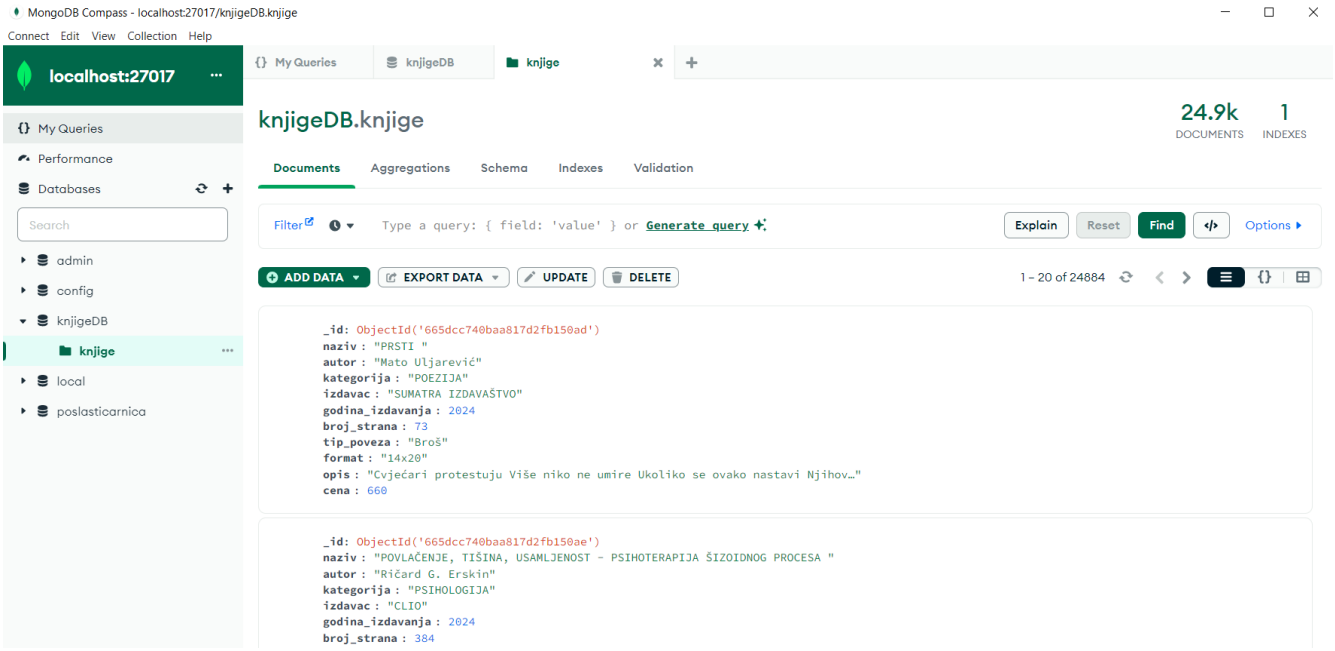
```python
from itemadapter import ItemAdapter

class BookspiderPipeline:

    def process_item(self, item, spider):
        adapter = ItemAdapter(item)
        if adapter['cena'] is not None:
            adapter['cena'] = int(adapter['cena'].split(',')[0].replace('.', ''))
        if adapter['broj_strana'] is not None:
            adapter['broj_strana'] = int(adapter['broj_strana'])
        if adapter['godina_izdavanja'] is not None:
            adapter['godina_izdavanja'] = int(adapter['godina_izdavanja'])
        return item
```

Slika 7. Definicija klase BookspiderPipeline

Nakon obrade podataka, potrebno je sačuvati sve podatke u okviru neke datoteke ili baze podataka. Veb indekser poseduje polje *custom_settings* (slika 4.) kojom se nadjačavaju podešavanja definisana u okviru njegovor rečnika. Naime, nadjačava se funkcionalnost smeštanja podataka, tako da se podaci upisuju u okviru datoteke *knjige.json*. Drugi način čuvanja podataka je pomoću baze podataka. Objekat podatka na izlazu iz *pipeline*-a je oblika *Python* rečnika pa je pogodno koristiti neku *NoSQL* bazu. Rešenje zadatka koristi *MongoDB* bazu za smeštanje podataka. Kreiranje i smeštanje podataka u bazu je implementirano kao deo *pipeline*-a u klasi *DatabasePipeline*. Inicijalizacija klase podrazumeva stvaranje veze sa *MongoDB* serverom, kreiranje baze i kolekcije za smeštanje podataka. Metoda *process_item*, ubacuje tekući podatak u bazu, a kad indekser završi sa radom, prekida vezu sa serverom (slika 8.).

```python
from pymongo import MongoClient

class DatabasePipeline:

    def __init__(self) -> None:
        self.connection = MongoClient("mongodb://localhost:27017")
        self.db = self.connection["knjigeDB"]
        if "knjige" in self.db.list_collection_names():
            self.db.drop_collection("knjige")
        self.coll = self.db['knjige']

    def process_item(self, item, spider):
        self.coll.insert_one(dict(item))
        return item

    def close_spider(self, spider):
        self.connection.close()
```

Slika 8. Definicija klase DatabasePipeline



Slika 9. MongoDB Compass – pregled baze podataka

Na slici 9. je prikazan program *MongoDB Compass* koji predstvlja grafički korisnički interfejs za olakšan rad sa *MongoDB* bazom. Može se uočiti da je veb indekser popunio bazu sa skoro 25.000 aktuelnih zapisa o knjigama. Nakon pokretanja veb indeksera instrukcijom:

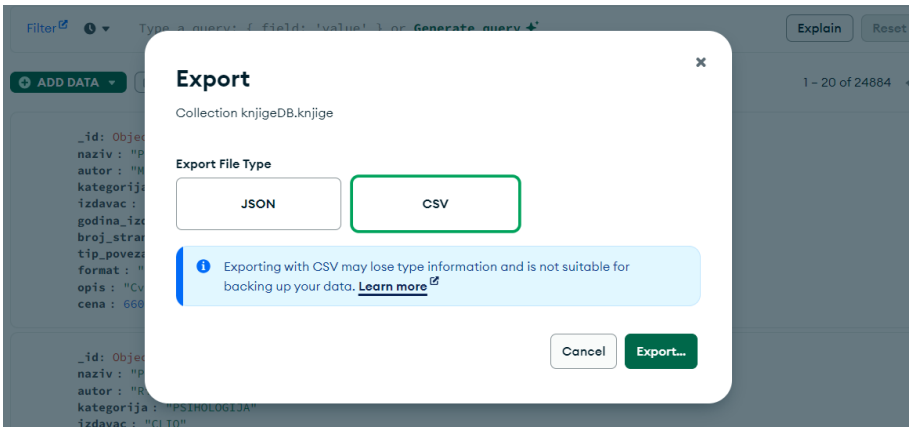
- *scrapy crawl myspider*

Veb indekseru je trebalo oko 2 sata da indeksira, parsira i smesti svih 25.000 zapisa u bazu podataka.

2. Analiza podataka

Nakon prikupljanja i smeštanja podataka u bazu, naredni korak je analiza podataka. Tokom prikupljanja podataka, dešava se da veb stranica ne sadrži određene informacije o knjizi koje su nam potrebne, pa je bitno da se podaci detaljno analiziraju i po potrebi uklone oni kojima fali veći broj polja.

Analiza podataka je obrađenja pomoću Python biblioteke *pandas*, koja sadrži brojne metode za analizu i filtriranje podataka. *Pandas* bilioteka radi sa tabelarnim podacima klase *pandas.DataFrame*, a podaci se učitavaju iz datoteka određenog formata (*CSV*, *JSON*, Excel, ...). Grafički korisnički interfejs MongoDB Compass, pruža podršku za čuvanje celokupne baze podataka u *JSON* ili *CSV* datoteku. Rešenje projektnog zadatka se fokusira na rad sa *CSV* datotekama.



Slika 10. MongoDB Compass – odabir formata za čuvanje baze podataka

Novonastala *CSV* datoteka se može učitati u program pomoću metode *pandas.read_csv()*. Pre analize podataka, potrebno je filtrirati bazu, što podrazumeva uklanjanje podataka kojima fale određena polja, dopunjavanje određenih nedostajućih polja, ispravljanje i formatiranje polja podataka, ... Nakon filtriranja baze, vrši se analiza podataka kroz određeni broj upita i čuvanje dobijenih rezultata. Glavni kod za filtriranje i analiziranje baze je implementiran u okviru *main* funkcije, iz koje je pozivaju pomoćne funkcije za filtriranje i analizu, što je prikazano na slici 11.

```
574  def main():
575      data = pd.read_csv("./knjigeDB.knjige.csv")
576      print(data, end='\n\n\n')
577      print(data.info(), end='\n\n\n')
578      print(data.describe(), end='\n\n\n')
579      data = filter_description(data)
580      data = filter_name(data)
581      data = filter_authors(data)
582      data = filter_pages(data)
583      data = filter_year(data)
584      data = filter_category(data)
585      data = filter_bind(data)
586      data = filter_format(data)
587      data = filter_publishers(data)
588      log_data(data)
589      data.drop(columns='povrsina').to_csv("preprocesirane_knjige.csv", index=False)
```

Slika 11. Glavna funkcija *main* u datoteci *data_analysis.py*

Nakon učitavanja *CSV* datoteke u promenljivu *data*, potrebno je proveriti koji podaci nedostaju u tabelarnom formatu baze podataka. Nedostajući podaci se u *pandas* biblioteci označavaju kao *None* ili *NaN*, a pozivom metoda *pandas.DataFrame.info()* i *pandas.DataFrame.describe()*, dobijaju se određene informacije za sve kolone/odlike podataka. Ispis ovih informacija u konzoli je prikazan na slici 12.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 24884 entries, 0 to 24883
Data columns (total 11 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   _id               24884 non-null  object
 1   naziv             24884 non-null  object
 2   autor             24571 non-null  object
 3   kategorija        24847 non-null  object
 4   izdavac           24541 non-null  object
 5   godina_izdavanja  23692 non-null  float64
 6   broj_strana       23498 non-null  float64
 7   tip_poveza        23395 non-null  object
 8   format            23454 non-null  object
 9   opis              23319 non-null  object
 10  cena              24884 non-null  int64
dtypes: float64(2), int64(1), object(8)
memory usage: 2.1+ MB
None
```

	godina_izdavanja	broj_strana	cena
count	23692.000000	23498.000000	24884.000000
mean	2018.763591	230.439059	1160.709171
std	325.199245	186.944469	1003.310541
min	15.000000	0.000000	49.000000
25%	2014.000000	112.000000	699.000000
50%	2019.000000	202.000000	979.000000
75%	2021.000000	307.000000	1299.000000
max	21016.000000	6687.000000	28985.000000

Slika 12. Ispis metoda *info* (levo) i *describe* (desno)

Analizom ispisa sa slike 12. se može primetiti da postoji veliki broj nedostajućih podataka u tabeli, kao i da postoje greške kod numeričkih podataka u kolonama (najmanja vrednost za godinu izdavanja je 15, a najmanji broj strana knjige je 0). Zaključuje se da je neophodno izvršiti filtriranje svake kolone/odlike u tabeli, što je realizovano pomoćnim funkcijama za svaku kolonu posebno. U nastavku je po stavkama detaljno opisan postupak filtriranja svake kolone.

2.1. Opis knjige

Kolona „opis“ sadrži 1565 polja koja su *None* (24884 – 23319 = 1565). S obzirom na to da kolona „opis“ nema veliku ulogu u trenuranju modela za mašinsko učenje, moguće je dopuniti nedostajuća polja rečenicom „Knjiga nema opis...“, što je implementirano u funkciji *filter_description*, na slici 13.

```
33  def filter_description(data: pd.DataFrame):
34      print("U slucaju da knjiga nema opis, u kolonu 'opis' se upisuje 'Knjiga nema opis...'\n")
35      data['opis'] = data['opis'].fillna("Knjiga nema opis...")
36      return data
```

Slika 13. Definicija funkcije za obradu kolone „opis“

2.2. Naziv knjige

Kolona „naziv“ ne sadrži polja koja su *None*. Funkcije *filter_name* samo pretvara slova u nazivu knjige u mala slova, kako bi se olakšao rad sa kolonom „naziv“. Pretvatanje slova u mala slova se radi i kod ostalih kolona koji su tipa niske. Implementacija funkcije je prikazana na slici 14.

```
39  def filter_name(data: pd.DataFrame):
40      data['naziv'] = data['naziv'].apply(str.lower)
41      return data
```

Slika 14. Definicija funkcije za obradu kolone „naziv“

2.3. Autor knjige

Kolona „autor“ sadrži 313 polja koja su *None* (24884 – 24571 = 313). S obzirom da postoji dosta knjiga kojima nije poznat autor, funkcija *filter_authors* je implementirana tako da se uklanjaju podaci kojima nedostaje vrednost polja kolone „autor“. U slučaju da je broj polja koji su *None* manji (nekoliko desetina), potencijalno rešenje bi bilo dopunjavanje nedostajućih podataka manuelno (pretraga interneta za autorom određene knjige). Za polja u koloni „autor“ koja nisu *None*, vrši se formatiranje vrednosti i popravka neispravnih vrednosti, tako da su nazivi autora odvojeni zarezima ako ima više od jednog. Na slici 15. je prikazana implementacija funkcije *filter_authors*.

```
271 def filter_authors(data: pd.DataFrame):
272     print("Ako nije poznat autor knjige, uklanja se")
273     last_count = data['_id'].count()
274     data = data.drop(index=data.loc[data['autor'].isnull(), 'autor'].index)
275     data = data.reset_index(drop=True)
276     next_count = data['_id'].count()
277     print(f"Uklonjeno je {last_count - next_count} redova u tabeli BEZ autora", end='\n\n')
278 
279     data['autor'] = data['autor'].apply(str.lower)
280 
281     data.loc[data['autor'] == 'nolwena monnier. eve grosset', 'autor'] = 'nolwena monnier, eve grosset'
282     data.loc[data['autor'] == 'izbor, prevod i predgovor tatjana simonović', 'autor'] = 'tatjana simonović'
283     data.loc[data['autor'] == 'izbor i prevod nikola b. cvetković', 'autor'] = 'nikola b. cvetković'
284     data.loc[data['autor'] == 'učim i igram se', 'autor'] = 'učim - igram se'
285     data.loc[data['autor'] == 'bojanka i vežbanka', 'autor'] = 'bojanka - vežbanka'
286     data.loc[data['autor'] == 'skupio i preveo albin vilhar', 'autor'] = 'albin vilhar'
287     data.loc[data['autor'] == 'redaktor i urednik protođakon radomir rakić', 'autor'] = 'radomir rakić'
288     data.loc[data['autor'] == 'priredio i preveo želidrag nikčević', 'autor'] = 'želidrag nikčević'
289     data.loc[data['autor'] == 'priredio i predgovor napisao\\nboško mijatović', 'autor'] = 'boško mijatović'
290     data.loc[data['autor'] == 'popuni i pokloni', 'autor'] = 'popuni - pokloni'
291     data.loc[data['autor'] == 'priredila i prevela bojana kovačević petrović', 'autor'] = 'bojana kovačević petrović'
292     data.loc[data['autor'] == 'sastavi modeli i igraj se', 'autor'] = 'sastavi modeli - igraj se'
293     data.loc[data['autor'] == 'scenario ž. b. đian i olivije legran; crtež i kolo', 'autor'] = 'ž. b. đian, olivije legran'
294     data.loc[data['autor'] == 'gordana živković glavni i odgovorni urednik', 'autor'] = 'gordana živković'
295 
296     data['autor'] = data['autor'].apply(lambda x: re.sub(r',.*:', ',', x))
297     data['autor'] = data['autor'].apply(lambda x: re.sub(r'^.*:', '', x))
298     data['autor'] = data['autor'].apply(lambda x: re.sub(r'[&;]', ',', x))
299     data['autor'] = data['autor'].apply(lambda x: re.sub(r' i ', ',', x))
300 
301     return data
```

Slika 15. Definicija funkcije za obradu kolone „autor“

2.4. Broj strana knjige

Kolona „broj_strana“ sadrži 1386 polja koja su *None* (24884 – 23498 = 1386). S obzirom da postoji dosta knjiga kojima nije poznat broj strana, funkcija *filter_pages* je implementirana tako da se uklanjaju podaci kojima nedostaje vrednost polja kolone „broj_strana“. U slučaju da je broj polja koji su *None* manji (nekoliko desetina), potencijalno rešenje bi bilo dopunjavanje nedostajućih podataka manuelno (pretraga interneta za tačnim brojem strana određene knjige). U koloni „broj_strana“ koja nisu *None*, javljaju se moguće greške u podacima, gde je najmanji broj strana 0 a najveći preko 6000. Da bi se uklonile takve greške uzima se pretpostavka da knjiga može imati najmanje 6 strana, a najviše 1400. Svaka vrednost van ovog opsega se smatra greškom. Detaljnijom analizom baze je utvrđeno da podaci sa brojem strana ispod 6, mogu predstavljati neme karte, brošure turističke ili karte sveta, što ne predstavlja knjigu. Na slici 16. je prikazana implementacija funkcije *filter_pages*.

```
44  def filter_pages(data: pd.DataFrame):
45      print("\nU slucaju da knjiga nema broj strana, uklanja se")
46      last_count = data['_id'].count()
47      data = data.drop(index=data.loc[data['broj_strana'].isnull(), 'broj_strana'].index)
48      data = data.reset_index(drop=True)
49      next_count = data['_id'].count()
50      print(f"Uklonjeno je {last_count - next_count} redova u tabeli BEZ broja strana", end='\n\n')
51
52      print("Moguce da su veliki broj strana netacni podaci pa ako knjiga ima preko 1400 strana, uklanja se'")
53      last_count = data['_id'].count()
54      data = data.drop(index=data.loc[data['broj_strana'] > 1400, 'broj_strana'].index)
55      data = data.drop(index=data.loc[data['broj_strana'] < 6, 'broj_strana'].index)
56      data = data.reset_index(drop=True)
57      next_count = data['_id'].count()
58      print(f"Uklonjeno je {last_count - next_count} redova u tabeli BEZ broja strana", end='\n\n')
59
60      data['broj_strana'] = data['broj_strana'].apply(int)
61
62      return data
```

Slika 16. Definicija funkcije za obradu kolone „broj_strana“

2.5. Godina izdavanja knjige

Kolona „godina_izdavanja“ sadrži 1192 polja koja su *None* (24884 – 23692 = 1192). Analizom vrednosti polja u koloni „godina_izdavanja“, utvrđeni su sledeći problemi. Postoje vrednosti koje predstavljaju skraćeni oblik godine izdavanja (npr. 15 umesto 2015), kao i vrednosti koje nemaju smisla pa je tu bilo potrebno manuelno ispravljanje takvih podataka. Takođe postoje podaci kojima su zamenjene vrednosti u kolonama „godina_izdavanja“ i „broj_strana“. Sledeće je potrebno obraditi podatke kojima nedostaje vrednost u polju. Takvi podaci se mogu ukloniti iz baze, a mogu se i dopuniti polja određenim vrednostima. Detaljnom analizom baze je utvrđeno da se broj knjiga povećava svake naredne godine, pa da se nebi narušila ova pravilnost, implementirano je da se prazna polja popunjavaju vrednostima koje se menjaju ciklično (1. podatak: 2023, 2. podatak: 2022, ... , 17. podatak: 2007, 18. podatak: 2023, ...) kako bi se sačuvao što veći broj podataka. Na kraju je uzeta pretpostavka da se razmatraju podaci koji čija godina izdavanja pripada opsegu od 1960. go 2024. godine, dok se ostali podaci uklanjaju. Implementacija funkcije *fiter_year* je prikazana na slici 17.

```python
def filter_year(data: pd.DataFrame):
    print("Ispravljaju se vrednosti za godinu izdavanja knjige...")
    fix_years = {
        15: 2015, 17: 2017, 19:2019, 20: 2020, 100: 2020, 178: 1997, 200: 2010, 201: 2011, 202: 2019,
        204: 2015, 206: 2006, 207: 2007, 208: 2008, 209: 2009, 214: 2014, 215: 2015, 219: 2019, 220: 2020,
        240: 2019, 280: 2008, 288: 2010, 344: 2006, 2088: 2008, 2105: 2015, 2115: 2015, 2507: 2007, 2996: 1996,
        20011: 2011, 20085: 2008, 20108: 2019, 20158: 2018, 20210: 2021, 21016: 2016
    }
    to_swap = [382, 114, 192, 430]
    for idx in data.index:
        v = data.loc[idx, 'godina_izdavanja']
        if not isnan(v):
            v = int(v)
            if v in to_swap:
                data.loc[idx, 'godina_izdavanja'], data.loc[idx, 'broj_strana'] = data.loc[idx, 'broj_strana'], data.loc[idx, 'godina_izdavanja']
            elif v in fix_years:
                data.loc[idx, 'godina_izdavanja'] = fix_years[v]

    print("Godina izdavanja se ciklicno popunjava kod knjiga kojim fali.")
    year_fill = [x for x in range(2023, 2006, -1)]
    for idx1, idx2 in enumerate(data.loc[data['godina_izdavanja'].isnull(), 'godina_izdavanja'].index):
        data.loc[idx2, 'godina_izdavanja'] = year_fill[idx1 % len(year_fill)]

    print("Zbog mogucih gresaka u podacima za godinu izdavanja, uzimaju se u obzir samo godine od 1960. do 2024.")
    last_count = data['_id'].count()
    data = data.drop(index=data.loc[data['godina_izdavanja'] < 1960, 'godina_izdavanja'].index)
    data = data.drop(index=data.loc[data['godina_izdavanja'] > 2024, 'godina_izdavanja'].index)
    data = data.reset_index(drop=True)
    next_count = data['_id'].count()
    print(f"Uklonjeno je {last_count - next_count} redova u tabeli", end='\n\n')

    data['godina_izdavanja'] = data['godina_izdavanja'].apply(int)

    return data
```

Slika 17. Definicija funkcije za obradu kolone „broj_strana“

2.6. Kategorija knjige

Kolona „kategorija“ sadrži 37 polja koja su *None* (24884 – 24847 = 37). Pretpostavka je da kategoriju knjige određuje sajt knjižare, jer postoji mogućnost da drugi sajtovi postave drugačiju kategoriju za istu knjigu. Dodatno, postoji mali broj knjiga bez kategorije, pa je opravdano ukloniti takve podatke iz baze. Daljom analizom baze i sajta knjižare, utvrđeno je da postoji veliki broj kategorija koje su ustvari podkategorije i koje sadrže mali broj podataka. Implementacija funkcije *filter_category*, pored uklanjanja nedostajućih podataka, obrađuje i grupaciju podklasa u veću klasu, kao što je na sajtu knjižare obeleženo. Time se smanjuje broj jedinstvenih kategorija i povećava broj podataka u glavnim kategorijama, što olakšava rad tokom treniranja modela mašinskog učenja. Takođe, vrši se uklanjanje podataka sa kategorijama koje ne predstavljaju knjigu (npr. mape i karte). Delovi implementacije funkcije *filter_category* su prikazani na slikama 18. i 19.

```python
def filter_category(data: pd.DataFrame):

    print("\nKategoriju knjige odredjuje knjizara, pa ako takvi podaci nedostaju, bice uklonjeni")
    last_count = data['_id'].count()
    data = data.drop(index=data.loc[data['kategorija'].isnull(), 'kategorija'].index)
    data = data.reset_index(drop=True)
    next_count = data['_id'].count()
    print(f"Uklonjeno je {last_count - next_count} redova u tabeli BEZ kategorije", end='\n\n')

    print("\nKategorija 'MAPE I KARTE' ne predstavljaju knjige pa se zato uklanjaju svi redovi sa ovom kategorijom...")
    last_count = data['_id'].count()
    data = data.drop(index=data.loc[data['kategorija'] == 'MAPE I KARTE', 'kategorija'].index)
    data = data.reset_index(drop=True)
    next_count = data['_id'].count()
    print(f"Uklonjeno je {last_count - next_count} redova u tabeli sa kategorijom 'MAPE I KARTE'", end='\n\n')
```

Slika 18. Deo definicije funkcije za obradu kolone „kategorija“ – 1

```
    print("Podkategorije za medicinske knjige imaju malo knjiga pa se mogu grupisati u vecu kategoriju 'MEDICINA'")
    subcategory = ['AKUŠERSTVO I GINEKOLOGIJA', 'ALERGOLOGIJA I IMUNOLOGIJA', 'ANATOMIJA I FIZIOLOGIJA',
                   'DERMATOLOGIJA I VENEROLOGIJA', 'GASTROENTEROLOGIJA', 'GENETIKA', 'ISHRANA I DIJETE STRUČNA',
                   'KARDIOLOGIJA', 'MEDICINA (OPŠTA)', 'NARKOMANIJA I ALKOHOLIZAM', 'NEUROLOGIJA', 'ONKOLOGIJA',
                   'PEDIJATRIJA', 'PSIHIJATRIJA', 'RADIOLOGIJA I NUKLEARNA MEDICINA', 'STOMATOLOGIJA',
                   'UROLOGIJA I NEFROLOGIJA']
    data.loc[data['kategorija'].isin(subcategory), 'kategorija'] = "MEDICINA"

    print("Podkategorije za strane jezike imaju malo knjiga pa se mogu grupisati u vecu kategoriju 'STRANI JEZICI'")
    subcategory = ['ARAPSKI JEZIK', 'DANSKI JEZIK', 'ENGLESKI JEZIK', 'ENGLESKI JEZIK - PRIRUČNICI I KURSEVI',
                   'ENGLESKI JEZIK/GRAMATIKA', 'FRANCUSKI JEZIK', 'GRČKI JEZIK', 'HEBREJSKI JEZIK', 'HOLANDSKI JEZIK',
                   'ITALIJANSKI JEZIK', 'JAPANSKI JEZIK', 'KINESKI JEZIK', 'LATINSKI', 'NEMAČKI JEZIK', 'NORVEŠKI JEZIK',
                   'NORVEŠKI JEZIK', 'OSTALI JEZICI', 'POLJSKI JEZIK', 'PORTUGALSKI JEZIK', 'REČNICI ENGLESKOG JEZIKA',
                   'RUSKI JEZIK', 'TURSKI JEZIK', 'ČEŠKI JEZIK', 'ŠPANSKI JEZIK', 'ŠVEDSKI JEZIK']
    data.loc[data['kategorija'].isin(subcategory), 'kategorija'] = 'STRANI JEZICI'
```

Slika 19. Deo definicije funkcije za obradu kolone „kategorija“ – 2

2.7. Tip poveza knjige

Kolona „tip_poveza“ sadrži 1489 polja koja su *None* (24884 – 23395 = 1489). Analizom kolone je utvrđeno da postoji 16222 knjiga sa broš povezom i 5986 knjiga sa tvrdim povezom. Nedostajuće vrednosti za polja u koloni „tip_poveza“ se dopunjuju vrednošću „Tvrd“, kako bi se povećao broj knjiga sa tvrdim povezom i „malo“ približio broju knjiga sa broš povezom. Implementacija funkcije *filter_bind* je prikazana na slici 20.

```
def filter_bind(data: pd.DataFrame):
    print("\nBroj razlicitih tipova poveza:")
    print(data.loc[data['tip_poveza'].notnull(), 'tip_poveza'].value_counts())
    print("\nPostoji 3x vise poveza tipa 'Broš' nego 'Tvrd' a postoji i jedan 'Tvrd povez'.")
    print("'Tvrd povez' se pretvara u 'Tvrd'")
    data.loc[data['tip_poveza'] == 'Tvrd povez', 'tip_poveza'] = "Tvrd"
    print(data.loc[data['tip_poveza'].notnull(), 'tip_poveza'].value_counts())
    print("\nPosto ima vise Bros od Tvrdog poveza, knjige za koje nije pronadjen povez ce biti Tvrd\n",
          "kako bi se broj tvrdih malo priblizio broju Bros poveza")
    data.loc[data['tip_poveza'].isnull(), 'tip_poveza'] = 'Tvrd'
    print(data.loc[data['tip_poveza'].notnull(), 'tip_poveza'].value_counts(), end='\n\n')

    return data
```

Slika 20. Definicija funkcije za obradu kolone „tip_poveza“

2.8. Format knjige

Kolona „format“ sadrži 1430 polja koja su *None* (24884 – 23454 = 1430). Funkcija *filter_format* je implementirana tako da ukloni podatke sa nedostajućim poljem za format. Analizom kolone „format“, utvrđeno je da postoji ogroman broj različitih formata koji se često bliski jedni drugima (npr. 14.5x21.0, 14x20, 14.3x21.5, ...). Za rešavanje ovog problema, implementirano je grupisanje formata u najsličniji standardni format. Standardni formati predstavljaju formate za štampanje na papiru od A6 do A0, od B6 do B0 i od C6 do C0 veličine. Dodatno, uzeti su u obzir i kvadratni formati knjiga od 6x6 do 40x40, kako bi se sačuvao već postojeći format knjiga, koje uglavnom predstavljaju knjige za malu decu (usrasto od 0 do 3 godine). Postupak grupisanja je sličan algoritmu najbližeg komšije.

Prvo se formatiraju i ispravljaju greške kod vrednosti formata, zatim se koristi Euklidska razdaljina za određivanje najsličnijeg standardnog formata. Najsličniji standardni format se upisuje u polje kolone „format“, umesto originalnog formata. Za svaki format je izračunata površina jedne strane knjige, što će biti od koristi tokom analize u sortiranju veličina formata. Delovi implementacije funkcije *filter_ format* su prikazani na slikama 21. i 22.

```
408  def filter_format(data: pd.DataFrame):
409      print("\nKnjige koje nemaju format kao podatak ce biti obrisane")
410      last_count = data['_id'].count()
411      data = data.drop(index=data.loc[data['format'].isnull(), 'format'].index)
412      data = data.reset_index(drop=True)
413      next_count = data['_id'].count()
414      print(f"Uklonjeno je {last_count - next_count} redova u tabeli BEZ formata", end='\n\n')
415
416      data['povrsina'] = 0.0
417
418      formats = [f'{n}x{n}' for n in range(6, 41)]
419      formats += ['10.5x14.8','14.8x21.0','21.0x29.7','29.7x42.0','42.0x59.4',
420                  '12.5x17.6','17.6x25.0','25.0x35.3','35.3x50.0','50.0x70.7',
421                  '11.4x16.2','16.2x22.9','22.9x32.4','32.4x45.8','45.8x64.8']
422      f = open(folder + 'formating_errors.txt', 'w')
423      print("Knjige u bazi imaju ogroman broj formata, jer postoje slicni formati sa malim odstupanjem\n",
424            "Formati se filtriraju tako sto se grupisu u jedan od standardnih formata: ",
425            "A6-A2, B6-B2, C6-C2 \nili kvadratni format NxN (N=6, 7, 8, ..., 40)", end='\n\n')
426      print("Filtriranje i grupisanje formata knjiga")
```

Slika 21. Deo definicije funkcije za obradu kolone „format“ – 1

```
450              min_val = inf
451              closest_format = ""
452              for fmt in formats:
453                  tmp = [float(x) for x in fmt.split('x')]
454                  dist = ((tmp[0] - s[0])**2 + (tmp[1] - s[1])**2)**0.5
455                  if dist < min_val:
456                      min_val = dist
457                      closest_format = fmt
458
459              f.write('{:.2f} -> {}x{} -> {}\n'.format(min_val, s[0], s[1], closest_format))
460              data.loc[id, 'format'] = closest_format
461              tmp = closest_format.split('x')
462              data.loc[id, 'povrsina'] = float(tmp[0]) * float(tmp[1])
463              # data.loc[id, 'format'] = str(s[0])+'x'+str(s[1])
464              # data.loc[id, 'povrsina'] = s[0] * s[1]
465
466      f.close()
467      print(f"Odradjeno {data.last_valid_index()}/{data.last_valid_index()}")
468      print("\nFiltritanje i grupisanje formata zavrseno. Izlazne informacije sacuvane u formating_errors.txt i format_count.txt")
469
470      return data
```

Slika 22. Deo definicije funkcije za obradu kolone „format“ – 2

2.9. Izdavač knjige

Kolona „izdavac“ sadrži 343 polja koja su *None* (24884 – 24541 = 343). Radi očuvanja što većeg broja podataka, vrši se popunjavanje praznih polja vredošću najfrekventnijeg izdavača. Dodatno se vrši obrada vrednosti polja kako bi se rešile neispravne vrednosti izdavača. Delovi implementacije funkcije *filter_publishers* su prikazani na slikama 23. i 24.

```
304 def filter_publishers(data: pd.DataFrame):
305     print("Ako nije poznat izdavac knjige, popunjava se najfrekventnijim izdavacem.")
306     data.loc[data['izdavac'].isnull(), 'izdavac'] = data['izdavac'].value_counts().index[0]
307 
308     data = data.drop(index=data.loc[data['izdavac'] == 'MAGIC MAP D.O.O.', "izdavac"].index).reset_index(drop=True)
309 
310     data.loc[data['izdavac'] == "ALGORITAM MEDIA", 'izdavac'] = 'ALGORITAM'
311     data.loc[data['izdavac'] == "BEGEN BOOKS", 'izdavac'] = 'BEGEN COMERC'
312     data.loc[data['izdavac'] == "BEL MEDIA", 'izdavac'] = 'BELMEDIA'
313     data.loc[data['izdavac'] == "BIBLIONER/PLATO", 'izdavac'] = 'PLATO'
314     data.loc[data['izdavac'] == "BOOK", 'izdavac'] = 'BOOK MARSO'
```

Slika 23. Deo definicije funkcije za obradu kolone „izdavac“ – 1

```
398     data.loc[data['izdavac'] == 'VULKAN IZDAVAŠTVO - MONO I MANJANA', 'izdavac'] = 'VULKAN IZDAVAŠTVO'
399     data.loc[data['izdavac'] == 'VULKAN ZNANJE', 'izdavac'] = 'VULKAN IZDAVAŠTVO'
400     data.loc[data['izdavac'] == 'ZEBRA', 'izdavac'] = 'ZEBRA PUBLISHING'
401     data.loc[data['izdavac'] == 'ZMAJ  SEZAM BUK', 'izdavac'] = 'ZMAJ'
402 
403     data['izdavac'] = data['izdavac'].apply(str.lower)
404 
405     return data
```

Slika 24. Deo definicije funkcije za obradu kolone „izdavac“ – 2

Nakon što su sve kolone filtrirane i određeni broj podataka uklonjen, može se početi sa analizom preostalih podataka u bazi. U okviru pomoćne funkcije *log_data*, su implementirani upiti po potrebama projektnog zadatka. Rezultati svih upita potrebnih za zadatak su zapisani u izlaznoj datoteci *analysis_output.txt*. Delovi implementacije funkcije *log_data* su prikazani na slikama 25. i 26.

```
473 def log_data(data: pd.DataFrame):
474 
475     f = open(folder + 'format_count.txt', 'w')
476     pprint(data.loc[:, 'format'].value_counts().to_dict(), stream=f)
477     f.close()
478 
479     f = open(folder + "category_list.txt", "w", encoding='utf-8')
480     d = data.loc[data['kategorija'].notnull(), 'kategorija'].value_counts().to_dict()
481     for key in d:
482         f.write('{: <5}\t{}\n'.format(d[key], key))
483     f.close()
```

Slika 25. Deo definicije funkcije za analizu filtriranih podataka – 1

```
525     f.write("\n#########################################################################\n")
526     f.write("\nBroj knjiga izdato po godinama, poslednjih 7 godina.\n")
527     last_7_years = [2023, 2022, 2021, 2020, 2019, 2018, 2017]
528     d = data.loc[:, 'godina_izdavanja'].value_counts().to_dict()
529     for year in last_7_years:
530         f.write('{}. godina: {} knjiga\n'.format(year, d[year]))
531 
532     f.write("\n#########################################################################\n")
533     f.write('\nRang lista 30 najskupljih knjiga u prodaji:\n')
534     d = data.sort_values(by='cena', ascending=False).head(n=30).reset_index(drop=True).drop(columns=['opis', '_id', 'povrsina'])
535     d = pd.DataFrame(d['cena']).join(d.drop(columns='cena'))
536     d.index += 1
537     f.write(d.to_string())
```

Slika 25. Deo definicije funkcije za analizu filtriranih podataka – 1

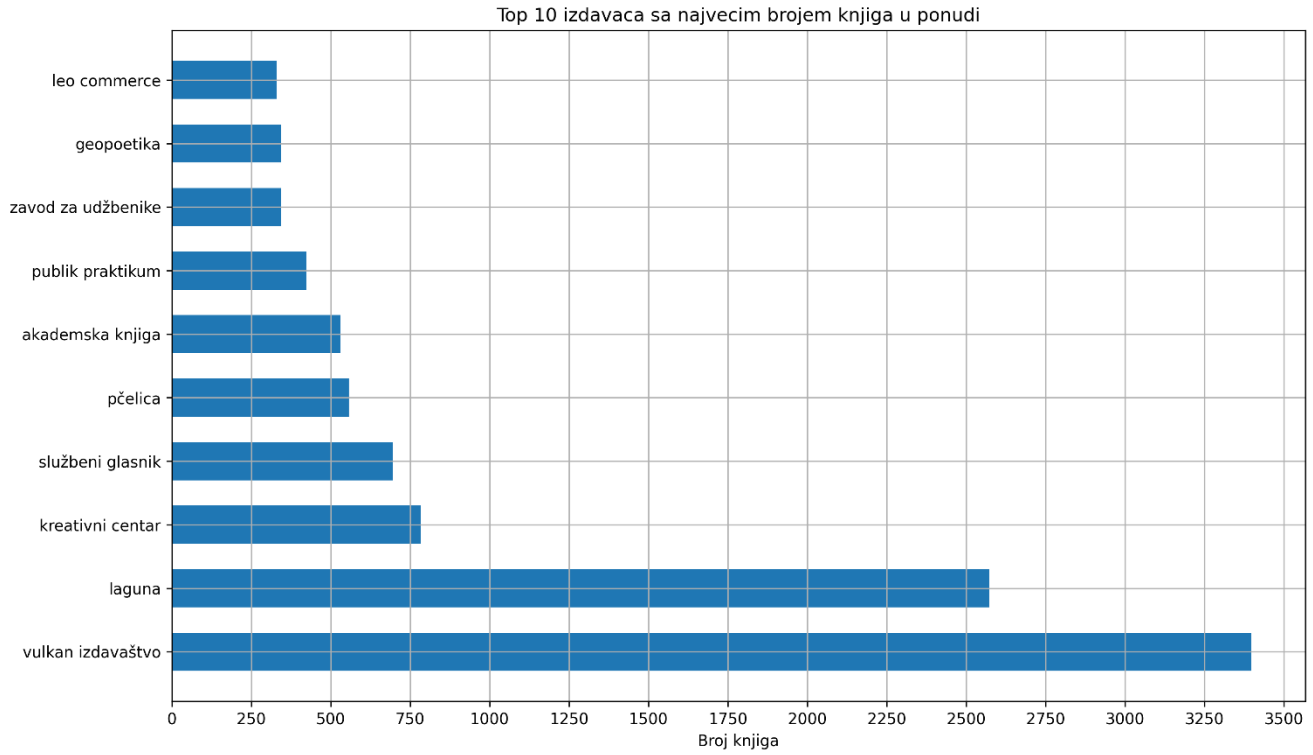
Filtrirani podaci se čuvaju u okviru nove *CSV* datoteke pod nazivom *procesirane_knjige.csv* i u MongoDB bazi, koji će se dalje koristiti u narednim delovima projektnog zadatka.

3. Vizuelizacija podataka

Baza podataka, nastala u prethodnom delu zadatka, se vizuelizuje pomoću Python bibioteke *matplotlib*. U nastavku su prikazane slike implementacije vizuelizacije podataka za svaki zahtev:

3.1. Top 10 izdavača po broju izdatih knjiga u ponudi:

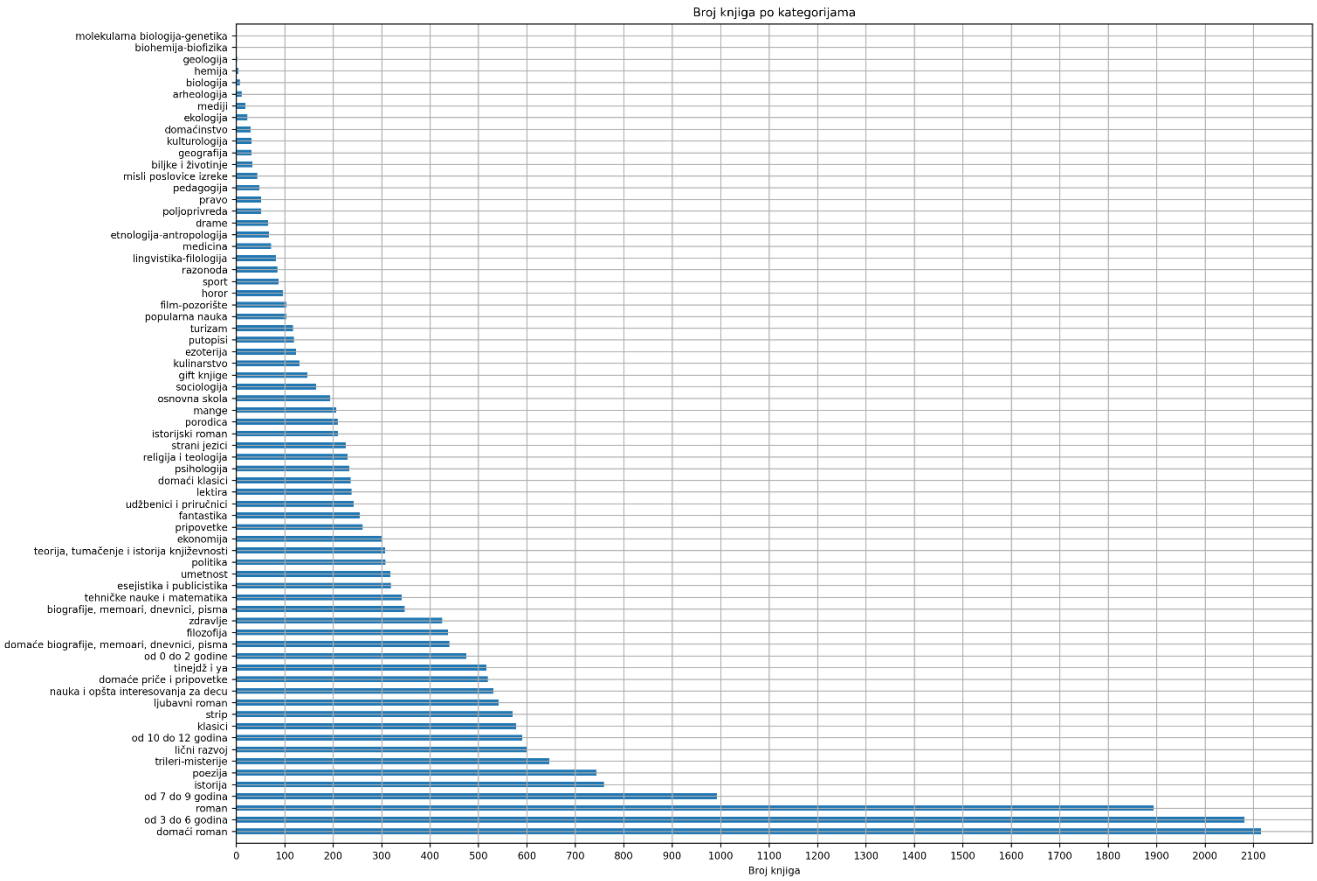
```
33  def main():
34
35      if not os.path.exists('./charts'):
36          os.mkdir('./charts')
37
38      data = pd.read_csv("preprocesirane_knjige.csv")
39
40      top_10_publishers = data.loc[:, 'izdavac'].value_counts().sort_values(ascending=False).head(n=10).to_dict()
41      plt.figure(1, figsize=(12, 7))
42      plt.title("Top 10 izdavaca sa najvecim brojem knjiga u ponudi")
43      plt.barh(top_10_publishers.keys(), top_10_publishers.values(), height=0.6)
44      plt.xlabel("Broj knjiga")
45      plt.xticks(ticks=range(0, max(top_10_publishers.values())+250, 250))
46      plt.grid()
47      plt.tight_layout()
48      plt.savefig("./charts/a.top10Publishers.png", dpi=300)
49      plt.close()
```



Slika 26. Kod i grafikon - 1

3.2. Broj knjiga po kategorijama:

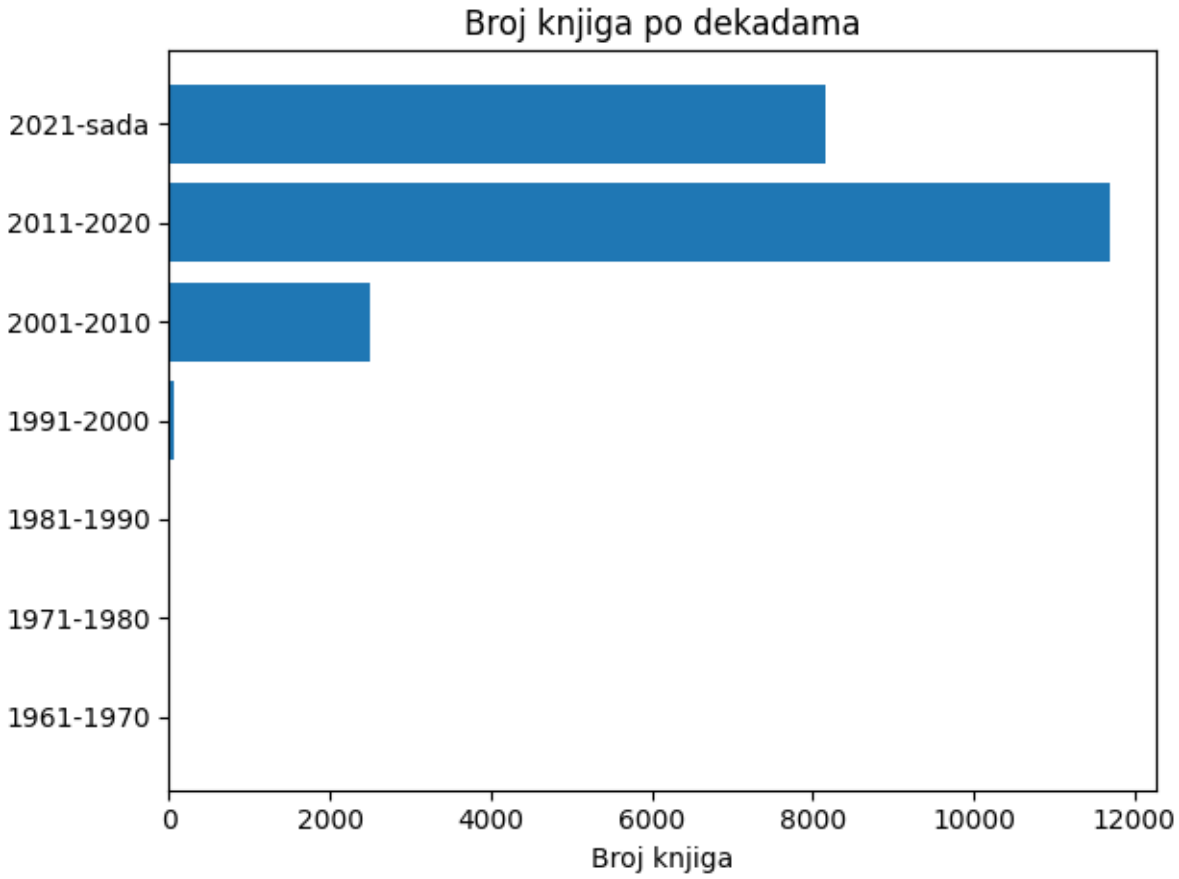
```
54      categories = data['kategorija'].apply(str.lower).value_counts().sort_values(ascending=False).to_dict()
55      plt.rcParams.update({'font.size':8.0})
56      plt.figure(2, figsize=(15, 10))
57      plt.title("Broj knjiga po kategorijama")
58      plt.barh(categories.keys(), categories.values(), height=0.5)
59      plt.grid()
60      plt.ylim(-1, len(categories.keys()))
61      plt.xticks(ticks=range(0, max(categories.values()), 100))
62      plt.xlabel("Broj knjiga")
63      plt.tight_layout()
64      plt.savefig("./charts/b.categories.png", dpi=300)
65      plt.rcParams.update({'font.size':10.0})
66      plt.close()
```



Slika 27. Kod i grafikon - 2

3.3. Broj izdatih knjiga po dekadama:

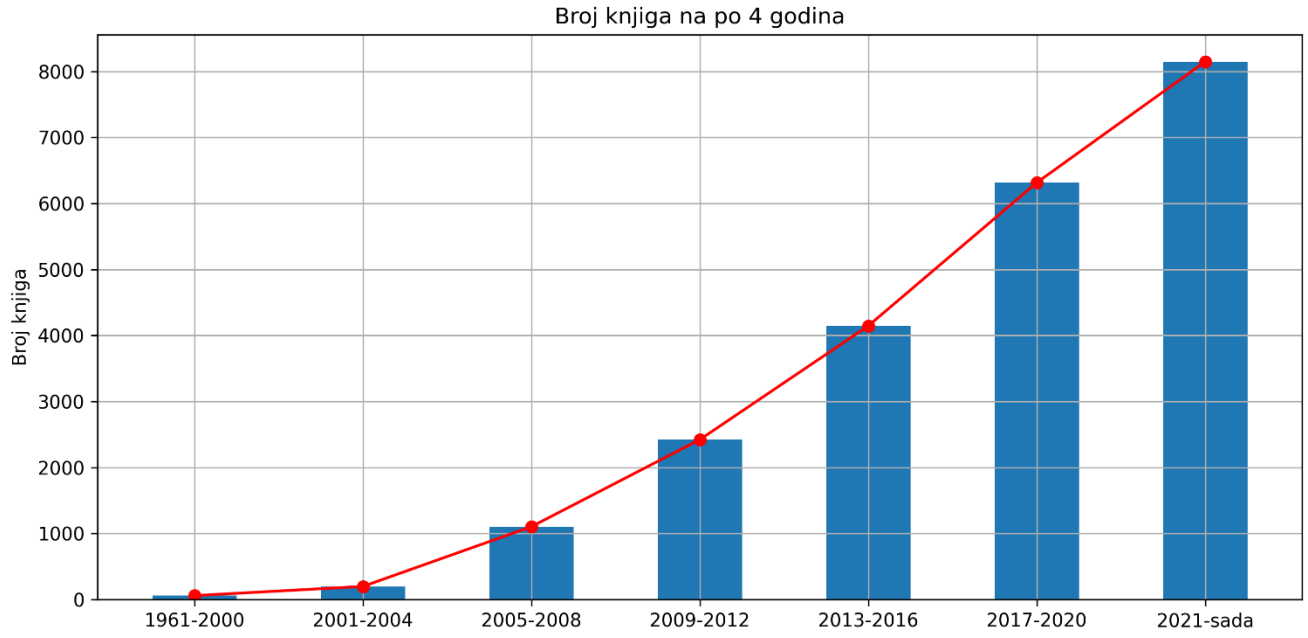
```
71  years = {
72      '1961-1970': data.loc[(data['godina_izdavanja'] > 1960) & (data['godina_izdavanja'] <= 1970), 'godina_izdavanja'].count(),
73      '1971-1980': data.loc[(data['godina_izdavanja'] > 1970) & (data['godina_izdavanja'] <= 1980), 'godina_izdavanja'].count(),
74      '1981-1990': data.loc[(data['godina_izdavanja'] > 1980) & (data['godina_izdavanja'] <= 1990), 'godina_izdavanja'].count(),
75      '1991-2000': data.loc[(data['godina_izdavanja'] > 1990) & (data['godina_izdavanja'] <= 2000), 'godina_izdavanja'].count(),
76      '2001-2010': data.loc[(data['godina_izdavanja'] > 2000) & (data['godina_izdavanja'] <= 2010), 'godina_izdavanja'].count(),
77      '2011-2020': data.loc[(data['godina_izdavanja'] > 2010) & (data['godina_izdavanja'] <= 2020), 'godina_izdavanja'].count(),
78      '2021-sada': data.loc[data['godina_izdavanja'] > 2020, 'godina_izdavanja'].count()
79  }
80
81  plt.figure(3)
82  plt.title("Broj knjiga po dekadama")
83  plt.barh(years.keys(), years.values())
84  plt.xlabel("Broj knjiga")
85  plt.tight_layout()
86  plt.savefig('./charts/c.booksByDecades.png')
87  plt.close()
```



Slika 28. Kod i grafikon - 3

3.4. Broj izdatih knjiga po 4 godina:

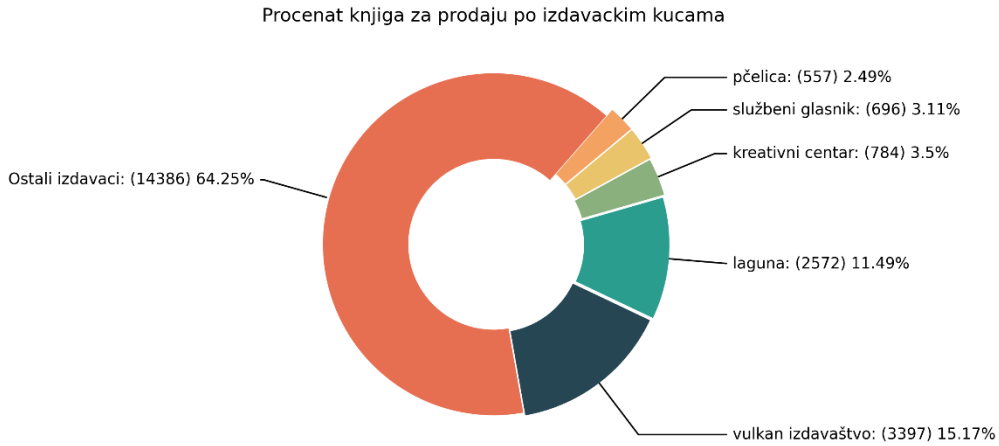
```
    years = {
        '1961-2000': data.loc[data['godina_izdavanja'] <= 2000, 'godina_izdavanja'].count(),
        '2001-2004': data.loc[(data['godina_izdavanja'] > 2000) & (data['godina_izdavanja'] <= 2004), 'godina_izdavanja'].count(),
        '2005-2008': data.loc[(data['godina_izdavanja'] > 2004) & (data['godina_izdavanja'] <= 2008), 'godina_izdavanja'].count(),
        '2009-2012': data.loc[(data['godina_izdavanja'] > 2008) & (data['godina_izdavanja'] <= 2012), 'godina_izdavanja'].count(),
        '2013-2016': data.loc[(data['godina_izdavanja'] > 2012) & (data['godina_izdavanja'] <= 2016), 'godina_izdavanja'].count(),
        '2017-2020': data.loc[(data['godina_izdavanja'] > 2016) & (data['godina_izdavanja'] <= 2020), 'godina_izdavanja'].count(),
        '2021-sada': data.loc[data['godina_izdavanja'] > 2020, 'godina_izdavanja'].count()
    }

    plt.figure(4, figsize=(10, 5))
    plt.title("Broj knjiga na po 4 godina")
    plt.bar(years.keys(), years.values(), width=0.5)
    plt.plot(range(len(years)), years.values(), 'ro', ls='-')
    plt.grid()
    plt.ylabel("Broj knjiga")
    plt.tight_layout()
    plt.savefig('./charts/c.booksBy4Years.png', dpi=300)
    plt.close()
```



Slika 29. Kod i grafikon - 4

3.5. Broj i procenat knjiga za top 5 izdavačkih kuća po prodaji:

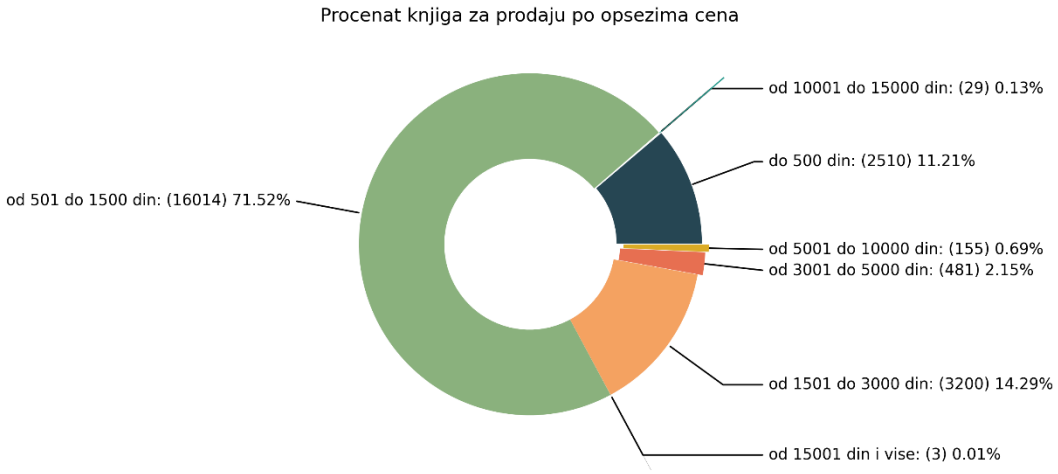
```
121    top_publishers = data.loc[:, 'izdavac'].value_counts().sort_values(ascending=False).to_dict()
122    keys = list(top_publishers.keys())
123    tmp = {}
124    for k in range(5):
125        tmp[keys[k]] = top_publishers[keys[k]]
126    tmp['Ostali izdavaci'] = 0
127    for k in range(5, len(top_publishers)):
128        tmp['Ostali izdavaci'] += top_publishers[keys[k]]
129
130    colors = ["#264653", "#2a9d8f", "#8ab17d", "#e9c46a", "#f4a261", "#e76f51"]
131
132    pretty_pie_chart(dict_data=tmp, colors=colors, explode=[0.02, 0.03, 0.04, 0.05, 0.05, 0],
133                     startangle=-80, title="Procenat knjiga za prodaju po izdavackim kucama",
134                     save_path="./charts/d.publishersPieChart.png")
```



Slika 30. Kod i grafikon - 5

3.6. Broj i procenat svih knjiga po opsezima cena I:

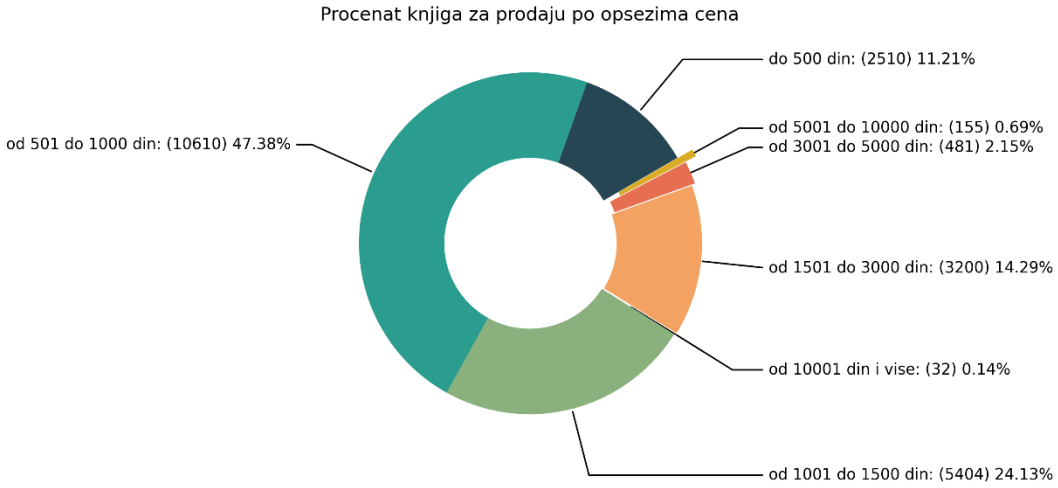
```
    prices = {
        'do 500 din': data.loc[data['cena'] <= 500, 'cena'].count(),
        'od 10001 do 15000 din': data.loc[(data['cena'] > 10000) & (data['cena'] <= 15000), 'cena'].count(),
        'od 501 do 1500 din': data.loc[(data['cena'] > 500) & (data['cena'] <= 1500), 'cena'].count(),
        'od 15001 din i vise': data.loc[data['cena'] > 15000, 'cena'].count(),
        'od 1501 do 3000 din': data.loc[(data['cena'] > 1500) & (data['cena'] <= 3000), 'cena'].count(),
        'od 3001 do 5000 din': data.loc[(data['cena'] > 3000) & (data['cena'] <= 5000), 'cena'].count(),
        'od 5001 do 10000 din': data.loc[(data['cena'] > 5000) & (data['cena'] <= 10000), 'cena'].count(),
    }
    colors = ["#264653", "#2a9d8f", "#8ab17d", "#000000", "#f4a261", "#e76f51", "#daab25"]

    pretty_pie_chart(dict_data=prices, colors=colors, explode=[0.01, 0.5, 0, 0.5, 0, 0.03, 0.05],
                     startangle=0, title="Procenat knjiga za prodaju po opsezima cena",
                     save_path="./charts/e.pricesPieChart.png")
```



Slika 31. Kod i grafikon - 6

3.7. Broj i procenat svih knjiga po opsezima cena II:

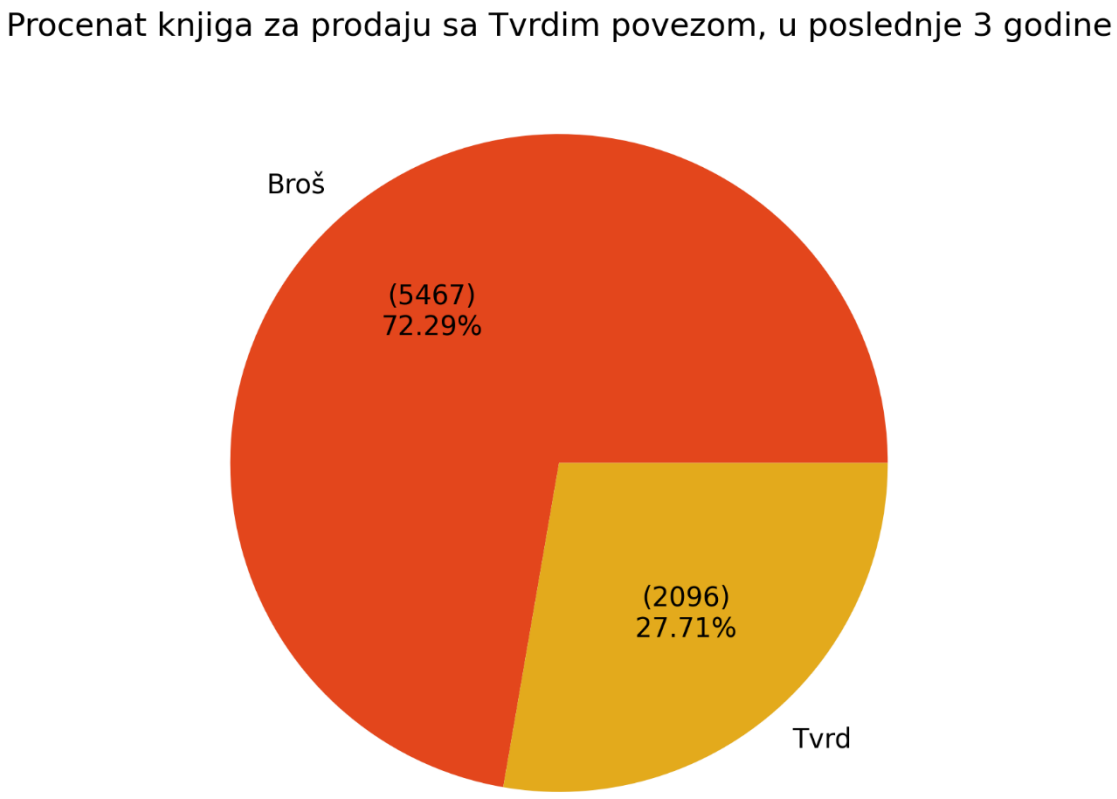
```
156     prices = {
157         'do 500 din': data.loc[data['cena'] <= 500, 'cena'].count(),
158         'od 501 do 1000 din': data.loc[(data['cena'] > 500) & (data['cena'] <= 1000), 'cena'].count(),
159         'od 1001 do 1500 din': data.loc[(data['cena'] > 1000) & (data['cena'] <= 1500), 'cena'].count(),
160         'od 10001 din i vise': data.loc[data['cena'] > 10000, 'cena'].count(),
161         'od 1501 do 3000 din': data.loc[(data['cena'] > 1500) & (data['cena'] <= 3000), 'cena'].count(),
162         'od 3001 do 5000 din': data.loc[(data['cena'] > 3000) & (data['cena'] <= 5000), 'cena'].count(),
163         'od 5001 do 10000 din': data.loc[(data['cena'] > 5000) & (data['cena'] <= 10000), 'cena'].count(),
164     }
165     colors = ["#264653", "#2a9d8f", "#8ab17d", "#000000", "#f4a261", "#e76f51", "#daab25"]
166 
167     pretty_pie_chart(dict_data=prices, colors=colors, explode=[0, 0, 0, 0.2, 0.01, 0.03, 0.1],
168                      startangle=30, title="Procenat knjiga za prodaju po opsezima cena",
169                      save_path="./charts/e.prices2PieChart.png")
```



Slika 32. Kod i grafikon - 7

3.8. Broj i procentualni odnos tipa poveza:

```
174     def func(pct, allvals):
175         absolute = int(np.round(pct * allvals / 100.0))
176         return f"({absolute})\n{pct:.2f}%"
177 
178     last_3_years = data.loc[data['godina_izdavanja'].isin([2023, 2022, 2021]), 'tip_poveza'].value_counts().to_dict()
179     total = sum(last_3_years.values())
180     plt.figure(6)
181     plt.title("Procenat knjiga za prodaju sa Tvrdim povezom, u poslednje 3 godine")
182     plt.pie(last_3_years.values(),labels=last_3_years.keys(), colors=["#E3461C", "#E3AA1C"],
183             autopct=lambda pct: func(pct, total))
184     plt.tight_layout()
185     plt.savefig("./charts/f.bindPieChart.png", dpi=300)
186     plt.close()
```



Slika 33. Kod i grafikon - 8

Za potrebe vizuelizacije kružnih grafikona (eng. *Pie chart*), implementirana je pomoćna funkcija *pretty_pie_chart*, prikazana na slici 34, koja iscrtava kružni grafikon u obliku prstena, sa određenim bojama, koeficijentima pomeraja delova, rotacionim uglom i dodatnim parametrima koje korisnik šalje funkciji. Povoljnost korišćenja ove funkcije je prikaz strelica koje povezuju deo grafikona sa njegovom labelom. Određene grupacije (npr. Broj knjiga čija je cena preko 10000 dinara) sadrži izuzetno malo podataka, što se neprimetno na grafikonu. Funkcija iscrtava strelice kako bi pojačala vidljivost takvih grupacija na grafikonu (slike 31. i 32.).

```python
def pretty_pie_chart(dict_data: dict, colors: list, explode: list, startangle: int, title: str, save_path: str):

    base_d = sum(list(dict_data.values()))
    final_data = {k+f': ({m})':m/base_d*100 for k,m in dict_data.items()}

    fig, ax = plt.subplots(figsize=(12, 5), subplot_kw=dict(aspect="equal"))
    recipe = list(final_data.keys())
    dict_data = list(final_data.values())
    perc = [str(round(e, 2)) + '%' for e in dict_data]
    plt.title(title)
    wedges, texts = ax.pie(dict_data, wedgeprops=dict(width=0.5),explode=explode, colors=colors, startangle=startangle)
    kw = dict(arrowprops=dict(arrowstyle="-"), zorder=0, va="center")

    for i, p in enumerate(wedges):
        ang = (p.theta2 - p.theta1)/2. + p.theta1
        y = np.sin(np.deg2rad(ang))
        x = np.cos(np.deg2rad(ang))
        horizontalalignment = {-1: "right", 1: "left"}[int(np.sign(x))]
        kw["arrowprops"].update({"connectionstyle": f"angle,angleA=0,angleB={ang}"})
        ax.annotate(recipe[i] + ' ' + perc[i], xy=(x, y), xytext=(1.4*np.sign(x), 1.4*y),
                    horizontalalignment=horizontalalignment, **kw)

    plt.savefig(save_path, dpi=300)
    plt.close()
```

Slika 34. Definicija funkcije *pretty_pie_chart*

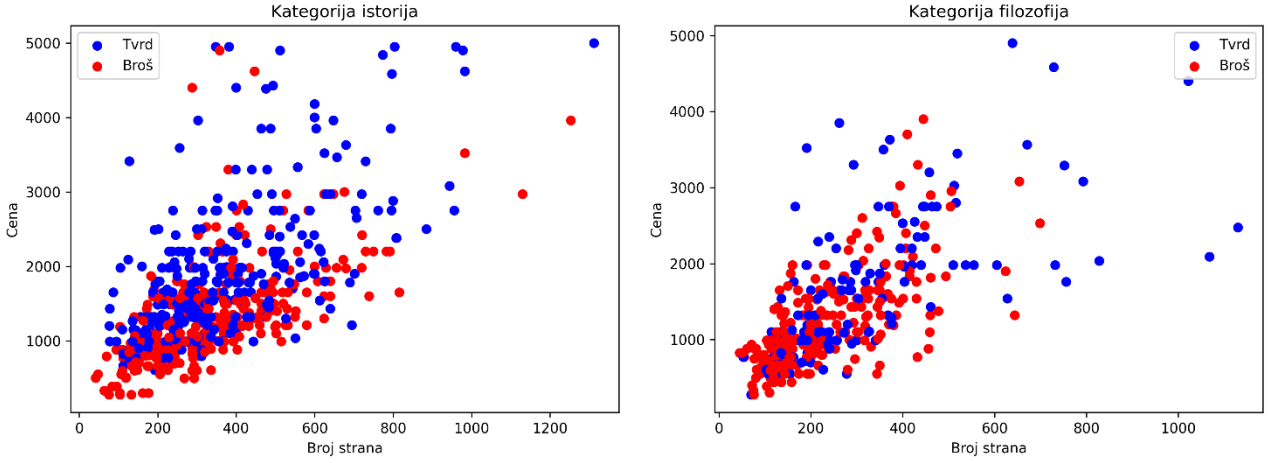
4. Implementacija regresije

Rešenje ove celine podrazumeva implementaciju Linearne Regresije, modela mašinskog učenja koji predviđa kontinualnu vrednost izlaza podatkaka. Odlike koje se mogu razmatrati za obučavanje modela su autor, izdavač, kategorija, broj strana, tip poveza, format i godina izdavanja. Pre nego što se implementiraju i obučavaju modeli za predviđanje, potrebno je proveriti da li baza podataka sadrži neželjene podatke ili *outlier*-e koji narušavaju pravilnost koja doprinosi u predviđanju. U otkrivanju *outlier*-a korisno je vizuelizovati podatke i zavisnost određenih atributa.

Deo koda na slici 35. konstruiše grafičke prikaze odnosa između broja strana i cene knjige za svaku kategoriju, kao i indikaciju tipa poveza. S obzirom da su grafici zavisnosti svake odlike i cene knjige vrlo gusto popunjen zbog broja podataka, pomaže u analizi ako se odradi grupacija podataka po nekoj odlici. Kategorija knjige se smatra bitnom odlikom za obučavanje modela pa je odrađena grupacija podataka po kategoriji. Za svaku kategoriju se formira grafik zavisnosti broja strana od cene, a tip poveza određuje boju podatka na grafiku. Na slici 36. su prikazani grafici zavrsnosti za kategorije „istorija“ i „filozofija“.

```
100     d = sorted(data['kategorija'].unique())
101     for cat in d:
102
103         tmp = data.loc[data['kategorija'] == cat, ['broj_strana', 'tip_poveza', 'cena']]
104
105         y = tmp['cena'].to_numpy()
106         x = tmp['broj_strana'].to_numpy().reshape(-1, 1)
107         c = []
108         for t in tmp['tip_poveza']:
109             c.append('b' if t == 'Tvrd' else 'r')
110         plt.scatter(x, y, c=c)
111         plt.scatter([], [], c='b', label='Tvrd')
112         plt.scatter([], [], c='r', label='Broš')
113         plt.xlabel("Broj strana")
114         plt.ylabel("Cena")
115         plt.title(f"Kategorija {cat}")
116         plt.legend()
117         plt.tight_layout()
118         plt.savefig(cat_path + f'PagesVsPrice/{cat}.png', dpi=300)
119         plt.close()
120         print(f'\n\n{cat}')
121         print(tmp.corr(numeric_only=True))
```

Slika 35. Vizuelizacija odnosa između cene, broja strana i tipa poveza, za svaku kategoriju



Slika 36. Vizuelizacija odnosa između cene, broja strana i tipa poveza, za dve kategorije

Analizom grafika zavisnosti uočava se postojanje *outlier*-a koje je potrebno ukloniti. Na slici 37. je prikazan deo koda koji uklanja *outlier*-e oslanjajući se na nalaze sa dobijenih grafikona. Međutim, neželjeni podaci se mogu naći i na globalnom nivou, pa je zbog njih implementirano filtriranje podataka po određenim kriterijumima. Bitno je da što veći broj podataka ostane u bazi, pa se na osnovu vizuelizacije podataka iz prethodne celine i dobijenih grafikona, pažljivo odrede granice filtriranja, što je prikazano na slici 38. Iako filtriranje na „globalnom“ nivou uklanja *outlier*-e, dodatno filtriranje po kategorijama dovodi do preciznijeg i specifičnijeg uklanjanja *outlier*-a koji su ostali.

```
38  ########################################################################
39  # REMOVE SPECIFIC
40  # data = data.drop(index=data.loc[(data['izdavac'] == 'studio bečkerek'), 'izdavac'].index).reset_index(drop=True)
41  # data = data.drop(index=data.loc[(data['izdavac'] == 'happy print'), 'izdavac'].index).reset_index(drop=True)
42  data = data.drop(index=data.loc[data['naziv'].str.contains('komplet'), 'naziv'].index).reset_index(drop=True)
43  data = data.drop(index=data.loc[data['kategorija'] == 'gift knjige', 'naziv'].index).reset_index(drop=True)
44  data = data.drop(index=data.loc[data['kategorija'] == 'od 0 do 2 godine', 'naziv'].index).reset_index(drop=True)
45  data = data.drop(index=data.loc[(data['kategorija'] == 'udžbenici i priručnici') & (data['broj_strana'] > 300), 'naziv'].index).reset_index(drop=True)
46  data = data.drop(index=data.loc[(data['kategorija'] == 'strip') & (data['tip_poveza'] == 'Broš'), 'naziv'].index).reset_index(drop=True)
47  data = data.drop(index=data.loc[(data['kategorija'] == 'od 10 do 12 godina') & (data['tip_poveza'] == 'Tvrd'), 'naziv'].index).reset_index(drop=True)
48  data = data.drop(index=data.loc[(data['kategorija'] == 'strani jezici') & (data['cena'] > 2000) & (data['broj_strana'] < 400), 'naziv'].index).reset_index(drop=True)
49  data = data.drop(index=data.loc[(data['kategorija'] == 'sociologija') & (data['cena'] > 1500) & (data['broj_strana'] < 150), 'naziv'].index).reset_index(drop=True)
50  data = data.drop(index=data.loc[(data['kategorija'] == 'sociologija') & (data['cena'] < 3000) & (data['broj_strana'] > 600), 'naziv'].index).reset_index(drop=True)
51  data = data.drop(index=data.loc[(data['kategorija'] == 'razonoda') & (data['cena'] > 1500) & (data['broj_strana'] < 300), 'naziv'].index).reset_index(drop=True)
52  data = data.drop(index=data.loc[(data['kategorija'] == 'putopisi') & (data['cena'] > 1900) & (data['broj_strana'] < 400), 'naziv'].index).reset_index(drop=True)
53  data = data.drop(index=data.loc[(data['kategorija'] == 'psihologija') & (data['cena'] > 3000) & (data['broj_strana'] < 300), 'naziv'].index).reset_index(drop=True)
54  data = data.drop(index=data.loc[(data['kategorija'] == 'pripovetke') & (data['cena'] > 3000), 'naziv'].index).reset_index(drop=True)
55  data = data.drop(index=data.loc[(data['kategorija'] == 'porodica') & (data['cena'] > 3000) & (data['broj_strana'] < 100), 'naziv'].index).reset_index(drop=True)
56  data = data.drop(index=data.loc[(data['kategorija'] == 'esejistika i publicistika') & (data['cena'] > 3000), 'naziv'].index).reset_index(drop=True)
57  data = data.drop(index=data.loc[(data['kategorija'] == 'domaći klasici') & (data['cena'] > 4000), 'naziv'].index).reset_index(drop=True)
58  data = data.drop(index=data.loc[(data['kategorija'] == 'domaće priče i pripovetke') & (data['cena'] > 4000), 'naziv'].index).reset_index(drop=True)
```

Slika 37. Uklanjanje *outlier*-a u kategorijama

```
62  # FILTER OUTLIERS
63  max_price = 5000
64  max_surface = 900
65  min_publish = 20
66  min_year = 2005
67
68  data = data.drop(index=data.loc[(data['cena'] > max_price), 'cena'].index).reset_index(drop=True)
69
70  data['povrsina'] = data['format'].apply(surface)
71  data = data.drop(index=data.loc[(data['povrsina'] >= max_surface), 'povrsina'].index).reset_index(drop=True)
72
73  data = data.drop(index=data.loc[(data['godina_izdavanja'] < min_year), 'godina_izdavanja'].index).reset_index(drop=True)
74
75  d = data['izdavac'].value_counts()
76  remove = [idx for idx in d.index if d[idx] < min_publish]
77  data = data.drop(index=data.loc[data['izdavac'].isin(remove), 'izdavac'].index).reset_index(drop=True)
```

Slika 38. Uklanjanje *outlier*-a na nivou baze

Nakon što je uklonjena većina *outlier*-a, jer nije moguće sve ukloniti, potrebno je pripremiti podatke za mašinsko učenje. To podrazumeva zadržavajne relevantnih odlika i njihovu transformaciju po potrebi. Za potrebe narednih celina, čuva se tabela podataka sa uklonjenim odlikama koje ne utiču na obučavanje modela u fajlu „filtrirane_knjige.csv“ (slika 39.).

```
125  data = data.drop(columns=['_id', 'opis', 'naziv'])
126  print(data.head())
127  print(data.describe())
128  data.to_csv('./filtrirane_knjige.csv', index=False)
129  original_data = data.copy(deep=True)
```

Slika 39. Uklanjanje neznačajnih odlika

Kontinualni numerički podaci se mogu skalirati kako bi olakšali radu algoritma mašinskog učenja. Postoje razni načini za skaliranje podataka, od jednostavnog deljenja do kompleksnog skaliranja na osnovu određenih vrednosti.

Za odliku „broj_strana“ koja ima opseg vrednosti od 6 do 1400, odrađeno je jednostavno deljenje sa 100 (slika 40.).

```
188     data['broj_strana'] = data['broj_strana'] / 100
```

Slika 40. Skaliranje odlike „broj_strana“

Za odliku „godina_izdavanja“ koja ima opseg vrednosti od 2005 do 2024, uzeto je u razmatranje samo poslednje dve cifre, s obzirom da su prve dve uvek iste, što je odrađeno operacijom modul sa 100 (slika 41.)

```
199     data['godina_izdavanja'] = data['godina_izdavanja'] % 100
```

Slika 41. Skaliranje odlike „godina_izdavanja“

Za odliku „format“ čija je niska oblika AxB, gde su A i B brojevi koje predstavljaju visinu i širinu strane knjige, izračunata je površina strane, koja se skalira metodom *MinMax*. Algoritam radi po formuli:

$$x = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Na ovaj način se vrši skaliranje površine u opsegu od 0 do 1 (slika 42.).

```
202     min_surface = min(data['povrsina'].to_numpy())
203     max_surface = max(data['povrsina'].to_numpy())
204     data['povrsina'] = (data['povrsina'] - min_surface) / (max_surface - min_surface)
```

Slika 42. Skaliranje odlike „format“

Za odliku „tip_poveza“ čije su moguće vrednsoti „Tvrd“ i „Broš“, može se izvršiti labeliranje vrednosti. S obzirom da odlika ima samo 2 moguće vrednosti, odrađeno je binarno mapiranje gde se vrednost „Tvrd“ mapira u 1 a vrednost „Broš“ u 0 (slika 43.).

```
220     data['tip_poveza'] = data['tip_poveza'].apply(lambda x: 1 if x == "Tvrd" else 0)
```

Slika 43. Transformacija odlike „tip_poveza“

Za odliku „autor“ koja je tipa niske, potrebno je pretvoriti je u numeričku vrednost, zarmotren je princip računanja broja autora knjige. Autori treba da budu zarezom odvojeni jedni od drugih, pa se računa broj autora odvojenih zarezom. Postoji i vrednost „grupa autora“, koja predstavlja veći broj autora knjige, pa je usvojeno da se za nju računa 5 kao broj autora (slika 44.)

```
261     def author_number(authors: str):
262         return len(authors.split(','))
263
264     data['broj_autora'] = 0
265     data.loc[data['autor'] != 'grupa autora', 'broj_autora'] = data.loc[data['autor'] != 'grupa autora', 'autor'].apply(author_number)
266     data.loc[data['autor'] == 'grupa autora', 'broj_autora'] = 5
```

Slika 44. Transformacija odlike „autor“

Odlike „kategorija“ i „izdavac“ su kategoričke prirode, pa je potrebno zameniti ih numeričkim vrednostima. Jedan način je da se kategorije labeliraju numerički ali postoji mogućnost da model uči po nekoj logici numeracije kategorija koja ne postoji (npr. Smatra da su numeracije rangiranja kategorija). Drugi način je da se za svaku kategoričku vrednost kreira nova kolona sa indikatorom 1 ako podatak pripada toj kategoriji, odnosno 0 ako ne pripada. Međutim, problem ovakvog pristupa je formiranje velikog broja kolona u tabeli, što usporava obučavanje modela. Usvojeno je korišćenje metode *TargetEncoding*, koja u funkciji pretvaranja koristi izlazni atribut. Ovim metodom se za svaku kategoriju i izdavača pridružuje prosečna vrednost cene za datu kategoričku vrednost. Pretpostavka je da izdavači izdaju knjige sa sličnim cenama i da knjige određene kategorije imaju relativno slične cene. Problem ovog pristupa je preterano prilagođavanje modela izlazu, što ne sme da se radi. Korišćenjem unakrsne validacije i težinskog korigovanja, se smanjuje mogućnost preteranog prilagođavanja, što je implementirano u funkciji *KfoldBayesianTargetEncoding*, prikazanoj na slici 45.

```
208  def KFoldBayesianTargetEncoding(data: pd.DataFrame, column: str, target: str, k: int, alpha: int):
209      categories = sorted(data[column].unique().tolist())
210      global_mean = data[target].mean()
211
212      encoded = np.zeros(data.shape[0])
213      data = data.sample(frac=1, random_state=7).reset_index(drop=True)
214
215      fold_size = ceil(len(data) / k)
216      for t in range(k):
217          train_idx = list(range(0, t * fold_size)) + list(range(fold_size + t * fold_size, len(data)))
218          test_idx = list(range(0 + t * fold_size, fold_size + t * fold_size if t < 9 else len(data)))
219
220          encode_dict = {c: global_mean for c in categories}
221
222          means = data.loc[train_idx, [column, target]].groupby(by=column).mean()
223          counts = data.loc[train_idx, [column, target]].groupby(by=column).count()
224
225          smoothed_means = (counts * means + alpha * global_mean) / (counts + alpha)
226
227          for c in smoothed_means.index:
228              encode_dict[c] = smoothed_means[target][c]
229
230          encoded[test_idx] = data.loc[test_idx, column].map(encode_dict)
231
232      encode_dict = data.join(pd.DataFrame(data=encoded, columns=['encoded']))
233      encode_dict = encode_dict.loc[:, [column, 'encoded']].groupby(by=column).mean()['encoded'].to_dict()
234
235      data = data.drop(columns=column)
236      data[column] = encoded
237
238      return data, encode_dict
```

Slika 45. Definicija funkcije *KFoldBayesianTargetEncoding*

U funkciji se početni skup podataka deli na *k* delova, gde se u svakom delu određeni skup podataka koristi za računanje prosečne vrednosti izlaza za ostale podatke. Promenljiva *encoded* čuva vrednosti tokom obrade, kako se ne bi menjale vrednosti iz originalnog skupa tokom ovog procesa. Računanje proseka izlaza za kategoriju sa malim brojem podataka može izazvati probleme, pa se zato koristi težinsko ispravljanje, a parametar *alpha* označava meru ispravljanja, tj. uticaja globalnog proseka. Efekat težinskog ispravljanja je prikazan na slici 46. Nakon enkodovanja kategoričkih vrednosti, formira se rečnik *encode_dict* kojim se mogu enkodovati novi podaci.

Slika 46. Primer efekte težinskog korigovanja

Nakon enkodovanja kategoričkih odlika „kategorija“ i „izdavac“, odrađeno je i skaliranje vrednosti deljenjem sa 100 (slike 47. i 48.). Na taj način su numeričke vrednosti svih odlika u sličnim redovima veličina, što olakšava radu algoritama mašinskog učenja (slika 49.). Usvojeno je da se za potrebe obučavanja modela koriste odlike „broj_strana“, „godina_izdavanja“, „povrsina“, „tip_poveza“, „kategorija“ i „izdavac“.

```
228    data, cat_enc_dict= KFoldBayesianTargetEncoding(data, 'kategorija', 'cena', k=10, alpha=2)
229    data['kategorija'] = data['kategorija'] / 100
```

Slika 47. Transformacija odlike „kategorija“

```
249    data, pub_enc_dict = KFoldBayesianTargetEncoding(data, 'izdavac', 'cena', k=10, alpha=5)
250    data['izdavac'] = data['izdavac'] / 100
```

Slika 48. Transformacija odlike „izdavac“

	cena	broj_strana	godina_izdavanja	povrsina	tip_poveza	kategorija	izdavac
0	1320	3.50	14	0.459377	0	9.903978	14.335161
1	999	2.04	14	0.301527	0	10.724098	10.094884
2	891	0.26	23	0.190593	1	8.009003	8.642704
3	570	2.34	13	0.190593	0	8.069810	9.622601
4	1390	0.56	17	0.828296	1	11.513364	8.153432
...	...	...	...	...	...	...	...

Slika 49. Izgled odlika podataka nakon skaliranja/transformacija

U okviru datoteka *machine_learning.py* se nalazi implementacija modela linearne regresije u vidu klase *LinearRegressionGradienDescent*. S obzirom da je korišćeno više odlika u obučavanju, implementirana je višestruka linearna regresija. Konstruktor klase prima jedan opcioni parametar *alpha* koji određuje jačinu L2 regularizacije linearne regresije. U okviru metode *fit* (slika 50.) se vrši obučavanje modela, gde se modelu daju odlike i izlazni atribut, brzine učenja za svaku odliku i broj iteracija obučavanja. Hipoteza linearne regresije je suma proizvoda težinskoh parametara i odlika. U svakoj iteraciji obučavanja se vrši ažuriranje težinskih parametara modela računanjem parcijalnog izvoda funkcije greške za svaki težinski parametar. Promenljiva *l2* predstavlja niz kojim se množe težinski parametri ako je uključena L2 regularizacija. Bitno je da se težinski parametar w0 ne regularizuje što obezbeđuje linija 30.

```
8   class LinearRegressionGradientDescent:
9
10      def __init__(self, alpha=0):
11          self.W = None
12          self.alpha = alpha
13
14      def h(self, X: np.ndarray):
15          return X.dot(self.W)
16
17      def fit(self, X: pd.DataFrame, y: pd.DataFrame, learning_rate, iter: int):
18          _X = pd.DataFrame(data=np.ones(shape=(len(X), 1)), columns=['x0'])
19          _X = _X.join(X.reset_index(drop=True))
20
21          self.W = np.ones(shape=(len(_X.columns), 1))
22          learning_rate = np.array(learning_rate)
23
24          _X = _X.to_numpy()
25          _y = y.to_numpy().reshape(-1, 1)
26
27          m = len(_X)
28
29          l2 = 1 - learning_rate * self.alpha / m
30          l2[0] = 1
31
32          for _ in range(iter):
33              dJ = (1 / m) * _X.T.dot(self.h(_X) - _y)
34
35              self.W = self.W * l2 - learning_rate * dJ
36
37          return self
```

Slika 50. Deo 1 definicije klase *LinearRegressionGradienDescent*

```
    def predict(self, X: pd.DataFrame):
        _X = pd.DataFrame(data=np.ones(shape=(len(X), 1)), columns=['x0'])
        _X = _X.join(X.reset_index(drop=True)).to_numpy()

        return self.h(_X).reshape(-1, 1).flatten()

    def error_function(self, X: pd.DataFrame, y: pd.DataFrame):
        m = len(X)
        _X = pd.DataFrame(data=np.ones(shape=(len(X), 1)), columns=['x0'])
        _X = _X.join(X.reset_index(drop=True)).to_numpy()
        _y = y.to_numpy().reshape(-1, 1)

        return (1 / (2 * m)) * (np.sum((self.h(_X) - _y) ** 2) + self.alpha * np.sum(self.W[1:] ** 2))

    def score(self, y_pred: np.ndarray, y_true: np.ndarray):
        # R^2 score
        return 1 - np.sum((y_true - y_pred) ** 2) / np.sum((y_true - np.mean(y_true)) ** 2)
```

Slika 51. Deo 2 definicije klase *LinearRegressionGradienDescent*

Na slici 51. je prikazan ostatak definicije klase *LinearRegressionGradienDescent*. Nakon obučavanja modela, model čuva težinske parametre i koristi ih za predviđanje cene novih knjiga. Predviđanje se vrši pomoću hipoteze modela, koja se poziva u okviru metode *predict*. Funkcija *score* implementira R^2 rezultat, a funkcija *error_function* implementira srednju kvadratnu fnukciju greške i obe se koristi za određivanje performansi regresivnih modela predviđanja. Na slici 52. je prikazao korišćenje klase linearne regresije za predviđanje cene knjiga.

```
y = data['cena'] / 100
x = data.drop(columns=['cena'])

x_train, x_test, y_train, y_test = split_dataset(x, y, train_size=0.75, random_state=123)

learning_rate = [
    [1],        # w0
    [0.001],    # broj_strana
    [0.0001],   # godina_izdavanja
    [0.1],      # povrsina
    [0.01],     # tip_poveza
    [0.001],    # kategorija
    [0.0001]    # izdavac
]
iterations = 5000

linreg_gd = LinearRegressionGradientDescent()
linreg_gd.fit(x_train, y_train, learning_rate=learning_rate, iter=iterations)
linreg_gd_out = linreg_gd.predict(x_test)

print("\n\nLinear Regression with Gradient Descent performance:")
print(f"Mean squared error: {linreg_gd.error_function(x_train, y_train):.3f}")
print(f"R^2 Score: {linreg_gd.score(linreg_gd_out, y_test.to_numpy()):.3f}\n\n")
```

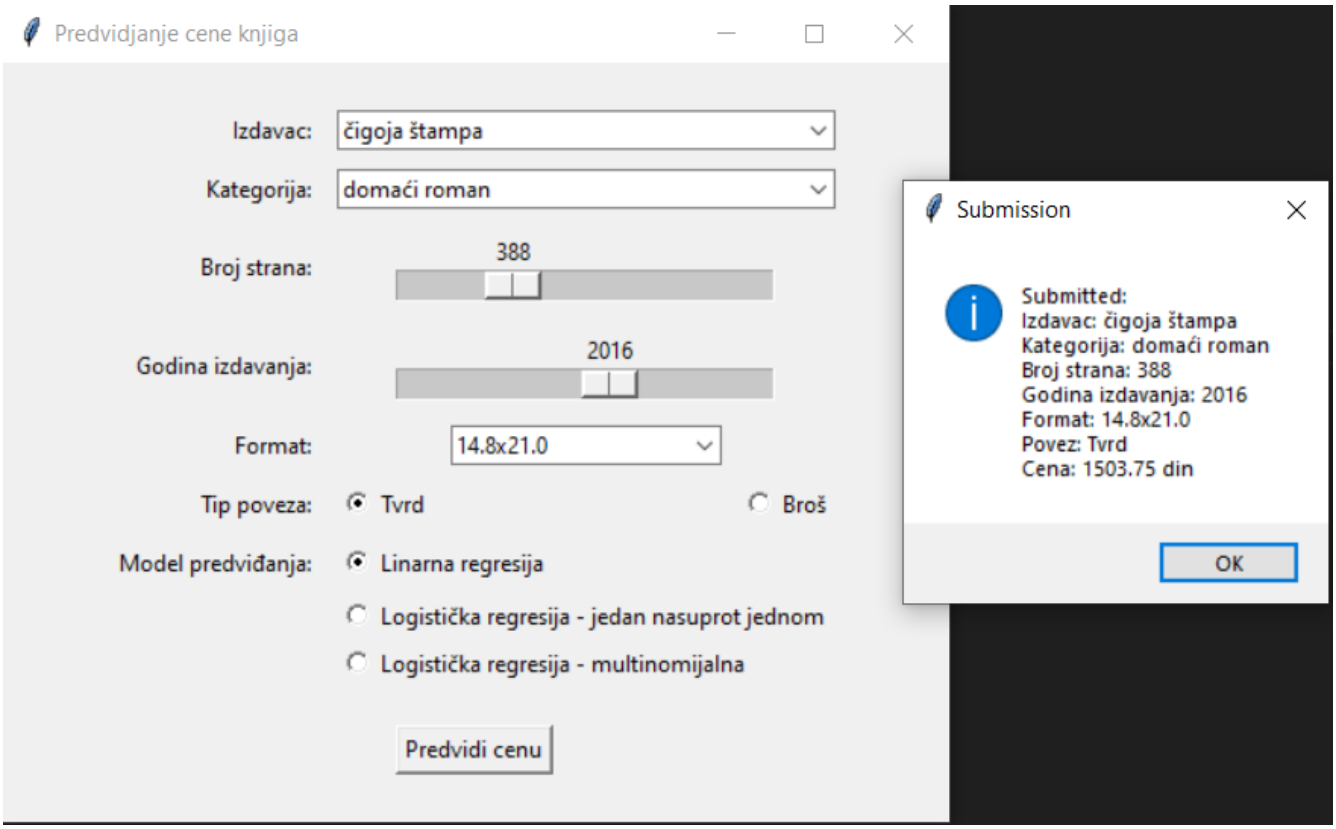
Slika 52. Predviđanje cene knjiga pomoću modela linearne regresije

Na slici 52. je iskorišćena funkcija *split_dataset* kojom se skup podataka deli na skupove za obučavanje i testiranje u određenoj razmeri. Implementacija ove funkcije je prikazana na slici 53. Takođe, izlazna vrednost je slakirana deljnjem sa 100, kako bi bila u istom redu veličie kao i odlike. Za svaku odliku su testirane i određene najbolje brzine učenja koje se nalaze u promenljivoj *learnin_rate*.

```
241  def split_dataset(x: pd.DataFrame, y: pd.DataFrame, train_size: float, random_state: int):
242      d = pd.DataFrame(x).join(y)
243      d = d.sample(frac=1, random_state=random_state).reset_index(drop=True)
244      split_idx = ceil(train_size * len(d))
245      train_idx = list(range(0, split_idx))
246      test_idx = list(range(split_idx, len(d)))
247      train_data = d.iloc[train_idx]
248      test_data = d.iloc[test_idx]
249      x1, x2 = train_data.drop(columns=y.name), test_data.drop(columns=y.name)
250      y1, y2 = train_data[y.name], test_data[y.name]
251      return x1, x2, y1, y2
```

Slika 53. Definicija funkcije *split_dataset*

Realizovana je mala aplikacija korišćenjem *tkinter* biblioteke, kako bi se omogućio korisnički unos vrednosti odlika knjige za predviđanje cene knjige na osnovu nekog modela mašinskog učenja. Implementacija grafičke aplikacije se nalazi u datoteci *gui.py*. Na slici 54. je prikazan rad aplikacije pri unosu proizvoljnih podataka i predviđanje cene knjige za dati unos.



Slika 54. Aplikacija za predviđanje cene knjiga, model linearna regresija

5. Implementacija klasifikacije

Implementacija logističke regresije kao klasifikacioni model za predviđanje, se nalazi u datoteci *machine_learning.py*. Klasa *LogisticRegression* je implementirana na gotovo isti način kao i klasa *LinearRegressionGradientDescent*. Jedine razlike su u metodama za hipotezu modela, funkcija greške i metrika za predviđanje. Hipoteza logističke regresije je sigmoid funkcija oblika:

$$h(x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + \cdots + w_n x_n)}}$$

```
58  class LogisticRegression:
59
60      def __init__(self, alpha=0):
61          self.W = None
62          self.alpha = alpha
63
64      def h(self, X: np.ndarray):
65          return 1 / (1 + np.exp(-X.dot(self.W)))
```

Slika 55. Hipoteza klase *LogisticRregression*

Funkcija greške mora biti konveksna što znači da funkcija gubitka mora biti konveksna. Kvadratna funkcija ne ispunjava ovaj uslov pa se za logističku regresiju koristi logistički gubitak. Metrika za klasifikacione probleme je F-mera, koja se koristi u slučaju nebalansiranih klasa podataka.

```
95      def error_function(self, X: pd.DataFrame, y: pd.DataFrame):
96          m = len(X)
97          _X = pd.DataFrame(data=np.ones(shape=(len(X), 1)), columns=['x0'])
98          _X = _X.join(X.reset_index(drop=True)).to_numpy()
99          _y = y.to_numpy().reshape(-1, 1)
100
101         return ((-1 / m) * (_y.T.dot(np.log(self.h(_X))) + (1 - _y).T.dot(np.log(1 - self.h(_X))))
102                 + (self.alpha / (2 * m)) * np.sum(self.W[1:] ** 2))
103
104     def score(self, y_pred: np.ndarray, y_true: np.ndarray):
105         # F1 score
106         tp = np.sum((y_pred == 1) & (y_true == 1))
107         fp = np.sum((y_pred == 1) & (y_true == 0))
108         fn = np.sum((y_pred == 0) & (y_true == 1))
109
110         p = tp / (tp + fp) if tp + fp > 0 else 0
111         r = tp / (tp + fn) if tp + fn > 0 else 0
112         if p + r == 0:
113             return 0
114
115         return 2 * p * r / (p + r)
```

Slika 56. Funkcija greške i F-mera klase *LogisticRegression*

Za potrebe višeklasne klasifikacije implementirane su dve metode:

- Kombinovanje binarnih logističkih regresija - Jedan nasuprot jednom
- Multinomijalna logistička regresija

Kombinovanje jedan nasuprot jednom podrazumeva kreiranje *n* binarnih klasifikatora logističke regresije za svaki par klasa, gde je $n = \frac{k(k-1)}{2}$. Klasa *LogisticRegressionOneVsOne* implementira kombinovanje pojedinačnih binarnih klasifikatora u metodi *fit*, gde za svaki par klasa pronađe podatke tih klasa, određuje koja klasa je pozitivna, a koja negativna i instancira klasu *LogisticRegression* za predviđanje između datog para klasa. Instanca klase se čuva u nizu *models* ujedno sa informacijom o paru klasa za koje je instancirana klasa logističke regresije.

```
118  class LogisticRegressionOneVsOne:
119
120      def __init__(self, alpha=0):
121          self.models = []
122          self.alpha = alpha
123
124      def fit(self, X: pd.DataFrame, y: pd.DataFrame, learning_rate, iter: int):
125          classes = np.unique(y)
126          for idx, k1 in enumerate(classes):
127              for k2 in classes[idx+1:]:
128                  _X = X.loc[(y == k1) | (y == k2)]
129                  _y = y.loc[(y == k1) | (y == k2)]
130
131                  positive_class_idx = _y == k1
132                  negative_class_idx = _y == k2
133
134                  _y.loc[positive_class_idx] = 1
135                  _y.loc[negative_class_idx] = 0
136
137                  t = time.time()
138                  self.models.append((k1, k2, LogisticRegression(alpha=self.alpha).fit(_X, _y, learning_rate, iter)))
139                  print(f"OneVsOne classifier {k1}_vs_{k2} created, t={time.time() - t:.3f}s")
140
141          return self
```

Slika 57. Deo 1 definicije klase *LogisticRegressionOneVsOne*

Nakon obučavanja, svaki model iz liste *models* predviđa izlaznu klasu podatak za testiranje. Konačno predviđanje klase se bazira na metodu glasanja, gde svaki model glasa da li je podatak u pozitivnoj ili negativnoj klasi, a većinskim odlučivanjem se određuje klasa podatka kojoj pripada. Implementacija postupka većinskog glasanja se nalazi u metodi *predict*. Metrika za višeklasnu klasifikaciju je takođe F-mera, ali malo izmenjena. Međutim, za F-meru na *micro* nivou, ona je jednaka tačnosti modela (*accuracy*) koja predstavlja odnos broj tačnih klasifikovanja od ukupnog broja podataka. F-mera na *macro* nivou gleda prosek F-mera za sve klase. Klasa *LogisticRegressionOneVsOne* implementira F-meru na *micro* nivou u metodi *score* (slika 58.).

```
143     def predict(self, X: pd.DataFrame):
144         n = 1 + int(np.sqrt(1 + 8 * len(self.models))) // 2
145         predictions = np.zeros(shape=(len(X), n))
146 
147         for k1, k2, model in self.models:
148             pred = model.predict(X)
149             for idx, p in enumerate(pred):
150                 if p == 1:
151                     predictions[idx][k1] += 1
152                 else:
153                     predictions[idx][k2] += 1
154 
155         predictions = np.argmax(predictions, axis=1)
156 
157         return predictions
158 
159     def score(self, y_pred: np.ndarray, y_true: np.ndarray):
160         # F1 micro == accuracy
161         return np.mean(y_pred == y_true)
```

Slika 58. Deo 2 definicije klase *LogisticRegressionOneVsOne*

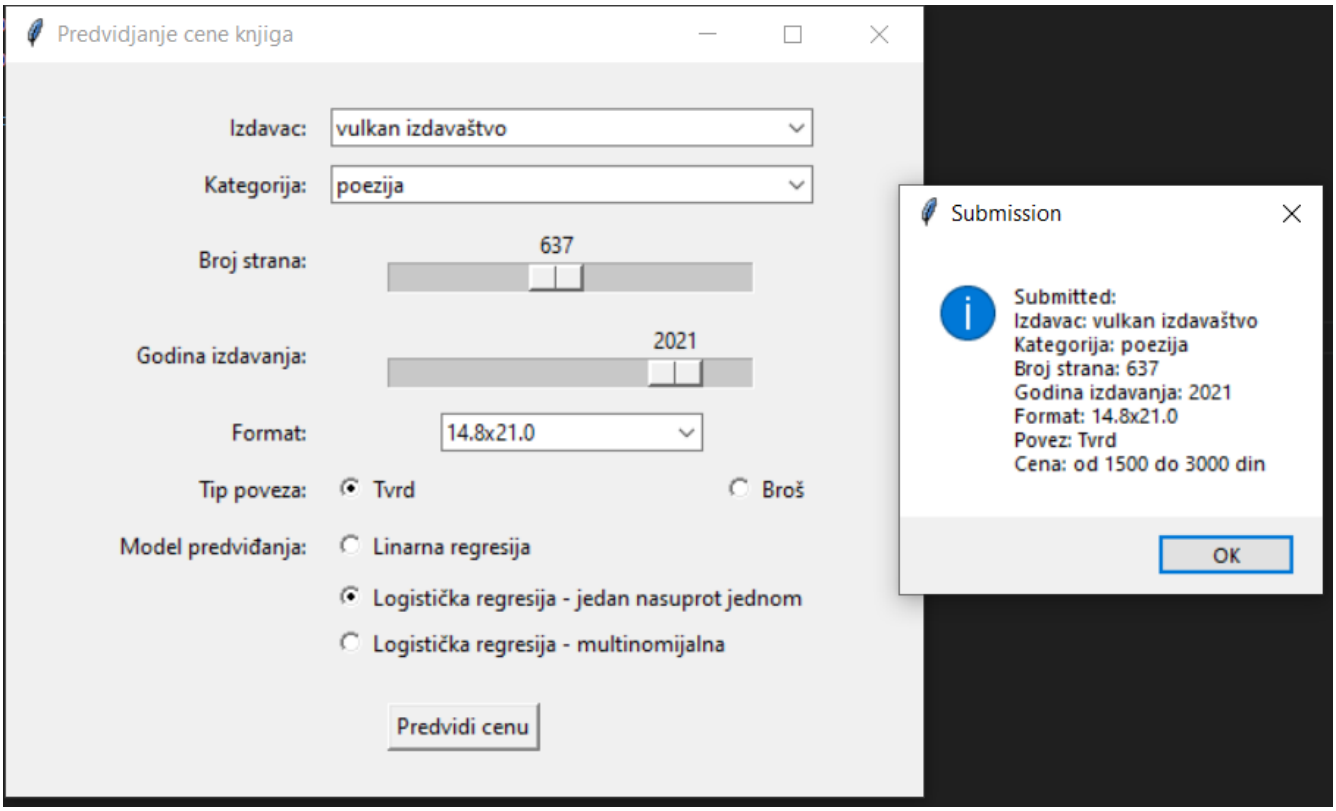
```
327 ranges = [0, 500, 1000, 1500, 3000, 5000]
328 price_cat_dict = {
329     0: "do 500 din",
330     1: "od 500 do 1000 din",
331     2: "od 1000 do 1500 din",
332     3: "od 1500 do 3000 din",
333     4: "od 3000 do 5000 din"
334 }
335 for k in range(len(ranges) - 1):
336     idx = (data['cena'] > ranges[k]) & (data['cena'] <= ranges[k + 1])
337     data.loc[idx, 'cena'] = k
338 
339 print(data)
340 
341 y = data['cena']
342 x = data.drop(columns=['cena'])
343 
344 x_train, x_test, y_train, y_test = split_dataset(x, y, train_size=0.75, random_state=123)
345 
346 learning_rate = [
347     [1],        # w0
348     [0.1],      # broj_strana
349     [0.01],     # godina_izdavanja
350     [0.1],      # povrsina
351     [0.01],     # tip_poveza
352     [0.01],     # kategorija
353     [0.01]      # izdavac
354 ]
355 iterations = 10000
356 
357 logreg_ovo = LogisticRegressionOneVsOne(alpha=0.1)
358 logreg_ovo.fit(x_train, y_train, learning_rate=learning_rate, iter=iterations)
359 logreg_ovo_out = logreg_ovo.predict(x_test)
```

Slika 59. Predviđanje cene knjiga pomoću modela logističke regresije

Izlazne vrednosti podataka su kontinualne, pa je potrebno transformisati ih u kategoričke. Određene klase u koje se podaci smeštaju u zavisnosti od izlazne vrednosti su (slika 59.):

- 0 – do 500 din
- 1 – od 500 do 1000 din
- 2 – od 1000 do 1500 din
- 3 – od 1500 do 3000 din
- 4 – od 3000 do 5000 din

Za svaku odliku su testirane i određene najbolje brzine učenja koje se nalaze u promenljivoj *learnin_rate*. Na slici 60. je prikazan rad aplikacije pri unosu proizvoljnih podataka i predviđanje cene knjige za dati unos.



Slika 60. Aplikacija za predviđanje cene knjiga, model logistička regresija – jedan nasuprot jedan

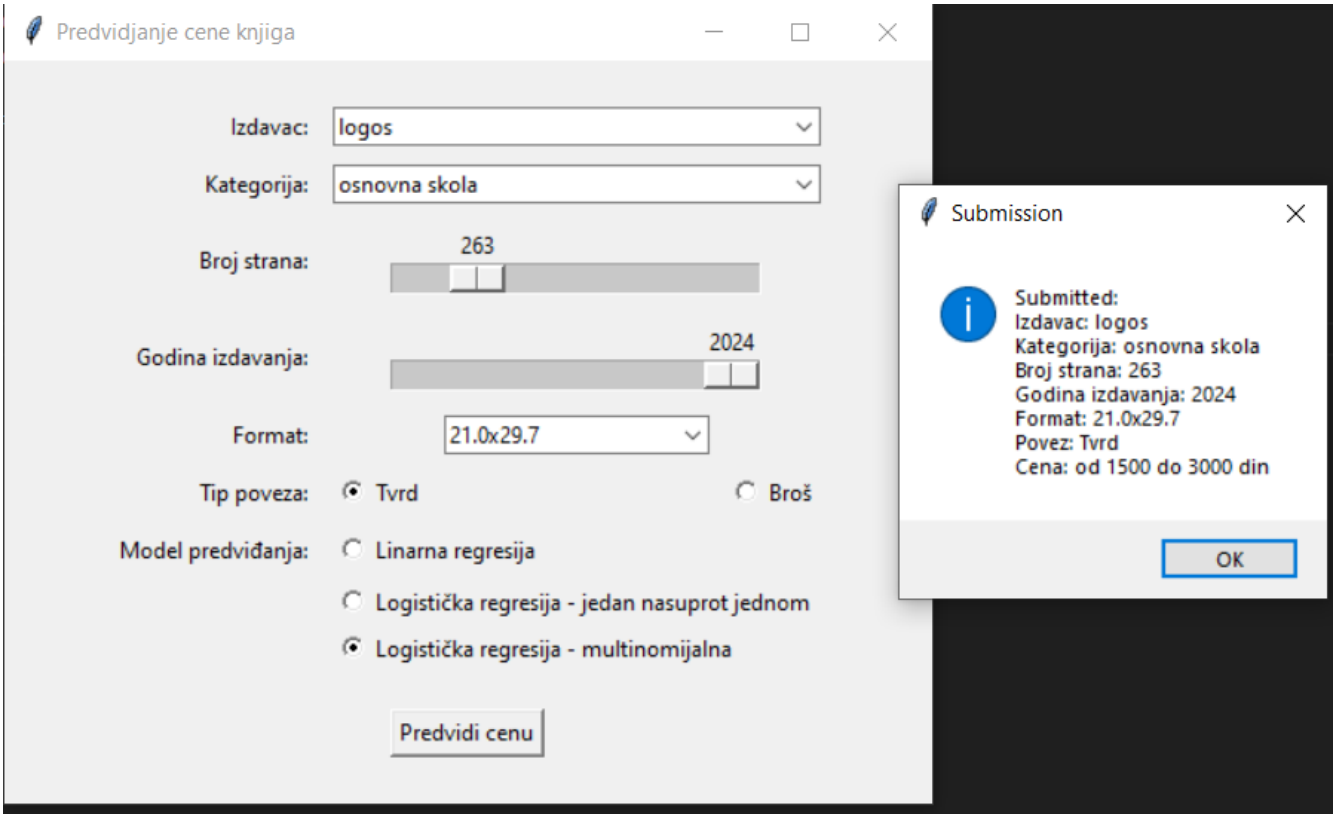
Multinomijalna logistička regresija predstavlja prirodno proširenje na rad sa više klasa. Hipoteza koristi *softmax* regresiju koja izražava verovatnoću pripadnosti podatka nekoj od *k* klasa. Multinomijalna logistička regresija je implmentiana u klasi *LogisticRegressionMultinomial*. Na slici 61. je prikazana implementacije klase.

```python
class LogisticRegressionMultinomial:

    def __init__(self):
        self.W = None

    def h(self, X: np.ndarray):
        exp_h = np.exp(X.dot(self.W))
        exp_sum = np.sum(exp_h, axis=1, keepdims=1)
        return exp_h / exp_sum

    def fit(self, X: pd.DataFrame, y: pd.DataFrame, learning_rate, iter: int):
        _X = pd.DataFrame(data=np.ones(shape=(len(X), 1)), columns=['x0'])
        _X = _X.join(X.reset_index(drop=True))

        classes = np.unique(y)
        self.W = np.ones(shape=(len(_X.columns), len(classes)))

        _X = _X.to_numpy()
        _y = np.eye(len(classes))[y]

        m = len(_X)

        for k in range(iter):
            dJ = (1 / m) * _X.T.dot(self.h(_X) - _y)

            self.W = self.W - learning_rate * dJ

            if k % 1000 == 0:
                print(f'Multinomial fiting, iteration {k}/{iter}')

        return self

    def predict(self, X: pd.DataFrame):
        _X = pd.DataFrame(data=np.ones(shape=(len(X), 1)), columns=['x0'])
        _X = _X.join(X.reset_index(drop=True)).to_numpy()

        return np.argmax(self.h(_X), axis=1)

    def score(self, y_pred: np.ndarray, y_true: np.ndarray):
        # F1 micro == accuracy
        return np.mean(y_pred == y_true)
```

Slika 61. Definicija klase *LogisticRegressionMultinomial*

```
361     learning_rate = [
362         [1],        # w0
363         [0.01],     # broj_strana
364         [0.01],     # godina_izdavanja
365         [0.1],      # povrsina
366         [0.1],      # tip_poveza
367         [0.01],     # kategorija
368         [0.01]      # izdavac
369     ]
370     iterations = 10000
371
372     logreg_multi = LogisticRegressionMultinomial()
373     logreg_multi.fit(x_train, y_train, learning_rate=learning_rate, iter=iterations)
374     logreg_multi_out = logreg_multi.predict(x_test)
375
```

Na slici 62. je prikazana upotreba klase *LogisticRegressionMultinomial* za predviđanje podataka za testiranje. Za svaku odliku su testirane i određene najbolje brzine učenja koje se nalaze u promenljivoj *learnin_rate*. Na slici 63. je prikazan rad aplikacije pri unosu proizvoljnih podataka i predviđanje cene knjige za dati unos.



Slika 63. Aplikacija za predviđanje cene knjiga, model multinomijalna logistička regresija

6. Implementacija klasterizacije

Sačuvani filtrirani podaci iz datoteke „filtrirane_knjige.csv“ se učitavaju u promenljivu *data*. U ovoj celini će se razmatrati numeričke odlike podataka: broj strana, godina izdavanja, površina i cena. Razmatraće se takođe i odlika „tip_poveza“ koja ima samo 2 vrednosti i s obzorom da je ova odlika kategorička, transformisaće se tako za vrednost „Tvrd“ bude 1, a vrednost „Broš“ bude 0. Grafičko prikazivanje klasterizacije je 3-dimenzionalno, što označava korišćene 3 odlika za klasterizaciju. Rešenje celine koristi nekoliko promenljivih koji određuju indekse odlika koje učestvuju u klasterizaciji *f1, f2* i *f3*, promenljiva *k* koja određuje broj klastera u algoritmu i promenljiva *iterations* koja određuje broj iteracija algoritma klasterizacije (slika 64.).

```
27  data = pd.read_csv('./filtrirane_knjige.csv')
28  data = data[['broj_strana', 'godina_izdavanja', 'tip_poveza', 'povrsina', 'cena']]
29  data['tip_poveza'] = data['tip_poveza'].apply(lambda x: 1 if x == "Tvrd" else 0)
30
31  print(data)
32
33  features = {0: 'broj_strana', 1: 'godina_izdavanja', 2: 'tip_poveza', 3: 'povrsina', 4: 'cena'}
34  f1 = 0
35  f2 = 2
36  f3 = 3
37  k = 3
38  iterations = 100
39
40  data = data.to_numpy()
41  x = data[:, [f1, f2, f3]]
42  labels, centroids = kmeans(x, k, iterations)
```

Slika 64. Učitavanje podataka i odabir odlika za klasterizaciju

U funkciji *kmeans* je implementiran algoritam K-srednjih vrednosti koji radi na sledeći način. Inicijalno se nasumčno odaberu K centromera iz skupa podataka. U svakoj iteraciji se računa udaljenost svakog podatka od K centromera. Podatak pripada klasteru njemu najbliže centromere. Nakon određivanja pripadnosti podataka klasterima, računa se srednja vrednost položaja podataka (novi centar klastera), za svaki klaster. U slučaju da su novi centri dovoljno bliski tekućim centromerama, algoritam može ranije da se prekine. Novi centri postaju tekuće centromere i prelazi se u narednu iteraciju algoritma. Nakon poslednje iteracije ili ranog napuštanja, kao povratna vrednost funkcije su oznake klastera kojima podaci pripadaju i koordinate centromera, koji se koriste tokom vizuelizacije klastera.

Udaljenost podatka od centromera se može računati na nekoliko načina. Euklidska razdaljina računa razdaljinu kao koren suma kvadrata razlika koordinara podatka i centromere:

$$dist = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2}$$

Menhetn razdaljina računa razdaljinu kao suma apsolutnih vrednosti razlika koordinata podatka i centromera:

$$dist = |p_1 - q_1| + |p_2 - q_2| + \cdots + |p_n - q_n|$$

Čebiševa razdaljina računa razdaljinu kao maksimum apsolutnih vrednosti razlika koordinata podatka i centromera:

$$dist = \max\{|p_1 - q_1|, |p_2 - q_2|, \ldots, |p_n - q_n|\}$$

Na slici 65. su prikazane definicije funkcije *kmeans* i funkcija za razdaljinu *euclid, manhattan* i *chebyshev*.

```python
def euclid(X: np.ndarray, centroids: np.ndarray):
    return np.sqrt(np.sum((X - centroids[:, np.newaxis]) ** 2, axis=2))

def manhattan(X: np.ndarray, centroids: np.ndarray):
    return np.sum(np.abs(X - centroids[:, np.newaxis]), axis=2)

def chebychev(X: np.ndarray, centroids: np.ndarray):
    return np.max(np.abs(X - centroids[:, np.newaxis]), axis=2)

def kmeans(X: np.ndarray, k: int, iter: int):

    centroid_idx = np.random.choice(len(X), k, replace=False)
    centroids = X[centroid_idx]

    for _ in range(iter):
        labels = np.argmin(euclid(X, centroids), axis=0)
        new_centroids = np.array([X[labels == cluster].mean(axis=0) for cluster in range(k)])
        if np.all(centroids == new_centroids):
            break
        centroids = new_centroids

    return labels, centroids

```

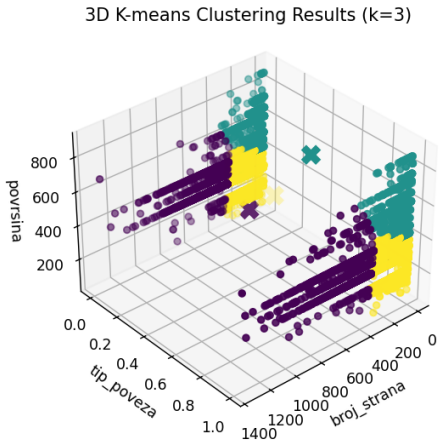
Slika 65. Definicije funkcija za distancu i algoritma K-srednjih vrednosti

Vizuelizacija klasterovanja se postiže pomoću *matplotlib* bilioteke koja podržava 3-dimenzionalnu vizelizaciju podataka. Funkcija *scatter* se koristi za grafičko iscrtavanje podataka, bojeći svaki podatak odgovarajućom bojom koja označava pripadnost određenom klasteru. Dodatno, iscrtavaju se i centromere kako bi se prikazali centri klastera. Na slici 66. se nalazi deo koda za grafički prikaz klasterovanja podataka.

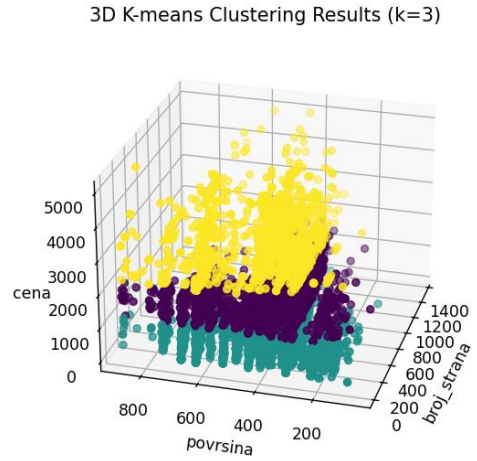
```python
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x[:, 0], x[:, 1], x[:, 2], c=labels)
ax.scatter(centroids[:, 0], centroids[:, 1], centroids[:, 2], c=range(len(centroids)), marker='x', s=100, lw=5)
ax.set_title(f'3D K-means Clustering Results (k={k})')
ax.set_xlabel(features[f1])
ax.set_ylabel(features[f2])
ax.set_zlabel(features[f3])

plt.show()
```

Slika 66. Deo koda za vizuelizaciju klasterovanja podataka

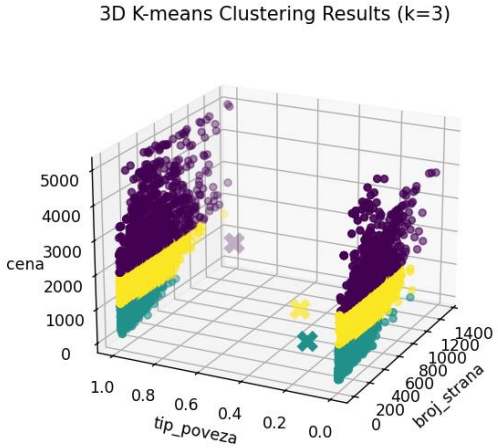
Na narednim slikama (67, 68. i 69.) su prikazani razultati klasterovanja uzimajući u obzir različite kombinacije odlika podataka.



Slika 67. Odnos površine, tipa poveza i broja strana knjige



Slika 68. Odnos površine, broja strana i cene knjige



Slika 69. Odnos tipa poveza, broja strana i cene knjige