Predviđanje odustajanja studenata od studija

Ercegović, Tea

SVEUČILIŠTE U SPLITU
PRIRODOSLOVNO MATEMATIČKI FAKULTET

DIPLOMSKI RAD

Predviđanje odustajanja studenata od studija

Tea Ercegović

Split, rujan 2024.

Temeljna dokumentacijska kartica

Diplomski rad

Sveučilište u Splitu

Prirodoslovno-matematički fakultet

Odjel za informatiku

Ruđera Boškovića 33, 21000 Split, Hrvatska

Predviđanje odustajanja studenata od studija

Tea Ercegović

SAŽETAK

Rad se bavi predviđanjem odustajanja studenata od studija pomoću različitih modela strojnog učenja kao što su logistička regresija, stabla odluke, random forest, SVM i gradient boosting. U analizu su uključeni podaci poput godine upisa, godine rođenja, spola, kategorija srednje škole, prosjeka ocjena u srednjoj školi, rezultata na državnoj maturi te uspjeha u pojedinim kolegijima tijekom studija.

Ključne riječi: predviđanje odustajanja studenata, obrazovni podaci, random forest, gradient boosting, akademski uspjeh.

Rad je pohranjen u knjižnici Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Rad sadrži: 84 stranica, 14 grafičkih prikaza, 32 tablice i 30 literaturnih navoda.

Izvornik je na hrvatskom jeziku.

Mentor: **Prof.dr.sc. Ani Grubišić**, redoviti profesor Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Ocjenjivači: **Prof.dr.sc. Ani Grubišić,** redoviti profesor Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Prof.dr.sc. Branko Žitko, redoviti profesor Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Dipl.ing. Ines Šarić-Grgić, asistent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Rad prihvaćen: Rujan, 2024

Basic documentation card

Thesis

University of Split
Faculty of Science
Department of Computer Science
Ruđera Boškovića 33, 21000 Split, Croatia

Student dropout prediction

Tea Ercegović

ABSTRACT

This paper is concentrated on the prediction of students dropping out of studies using different machine learning models such as logistic regression, decision trees, random forest, SVM and gradient boosting. The analysis includes data such as year of enrollment, year of birth, gender, high school category, grade point average in high school, results at the state matriculation exam, and success in individual courses during studies.

Key Words: predicting student dropout, educational data, random forest, gradient boosting, academic success.

Thesis deposited in library of Faculty of science, University of Split.

Thesis consist of: 84 pages, 14 figures, 32 tables and 30 references

Original language: Croatian

Supervisor: **Ani Grubišić, Prof.Ph.D.** Full Professor of Faculty of Science, University of Split

Reviewers: **Ani Grubišić, Prof.Ph.D.** Full Professor of Faculty of Science, University of Split,
Branko Žitko, Prof.Ph.D. Full Professor of Faculty of Science, University of Split,
Ines Šarić-Grgić, M.Eng. Instructor of Faculty of Science, University of Split

Thesis accepted: September, 2024

Sadržaj

Uvod 1

1. Rudarenje podataka u obrazovanju 2

1.1. Odustajanje od školovanja 4

1.2. Povezani radovi 5

2. Metodologija 7

2.1. O podacima 7

2.2. Pretprocesiranje podataka 10

2.3. Modeli strojnog učenja 14

2.4. Metrike evaluacije 19

2.5. Korišteni alati 20

3. Analiza podataka 22

3.1. Dob 22

3.2. Broj upisanih 22

3.3. Spol 23

3.4. Završena srednja škola 24

3.5. Lokacija 26

3.6. Prosjek ocjena 26

3.7. Završetak studija 27

3.8. Odnos prosjeka i završetka studija 29

3.9. Odnos spola i završetka studija 30

3.10. Odnos spola i završene srednje škole 32

3.11. Predmeti 33

4. Rezultati modela predviđanja 36

4.1. Odabir značajki 36

4.1.1. Prvi skup značajki 36

4.1.2. Drugi skup značajki 37
4.2. Rezultati dobiveni nad prvim skupom značajki 39
4.2.1. Podjela po spolu 41
4.2.2. Podjela po kategoriji srednje škole 42
4.2.3. Podjela po lokaciji srednje škole 43
4.2.4. Dodavanje novih značajki 44
4.3. Rezultati dobiveni nad drugim skupom značajki 44
4.3.1. Podjela po spolu 46
4.3.2. Podjela po kategoriji srednje škole 47
4.3.3. Podjela po lokaciji srednje škole 47
4.3.4. Dodavanje novih značajki 48
5. Rasprava 50
Zaključak 55
Literatura 56
Sažetak 58
Summary 59
Skraćenice 60
Privitak 61

Uvod

Odustajanje studenata od školovanja predstavlja jedan od najvećih izazova s kojim se suočavaju institucije visokog obrazovanja diljem svijeta. Problem odustajanja ima široke društvene, ekonomske i individualne posljedice. Na razini društva, visoke stope odustajanja smanjuju potencijal za razvoj kvalificirane radne snage, dok s financijske strane, odustajanje predstavlja gubitak ulaganja kako za same studente, tako i za institucije i vlade koje subvencioniraju obrazovanje. Na osobnoj razini, odustajanje od školovanja može dovesti do osjećaja neuspjeha, smanjenja prilika za karijeru, kao i problema s financijskom stabilnošću zbog nepovratnih troškova školovanja.

Razlozi za odustajanje od školovanja su mnogobrojni i raznoliki. Mogu uključivati akademske poteškoće, osobne ili obiteljske probleme, nedostatak motivacije, financijske prepreke ili neusklađenost očekivanja učenika s onim što pruža školska ustanova. Osim toga, čimbenici kao što su prethodno obrazovanje, socijalni i ekonomski status, te podrška obitelji i prijatelja također igraju važnu ulogu u odluci o nastavku ili prekidu školovanja.

U ovom radu fokus će biti na primjeni algoritama strojnog učenja za predikciju odustajanja studenata. Analizirani su skupovi podataka koji uključuju akademske informacije studenata, kao i njihove rezultate na ispitima i prethodna obrazovna postignuća. Modeli strojnog učenja, poput logističke regresije (engl. *logistic regression*), stabala odluke (engl. *decision tree*), slučajne šume (engl. *random forest*), strojeva potpornih vektora (engl. *support vector machine*) i povećanja gradijenta (engl. *gradient boosting*), korišteni su kako bi se izradili prediktivni modeli koji mogu identificirati studente s većim rizikom od odustajanja.

1. Rudarenje podataka u obrazovanju

Rudarenje podataka (engl. *Data mining*) je proces izvlačenja informacija iz velikih skupova podataka. Kako količina podataka generiranih u različitim sektorima nastavlja eksponencijalno rasti, rudarenje podataka postalo je bitan alat za otkrivanje obrazaca, trendova i odnosa unutar podataka. To je proces otkrivanja zanimljivih obrazaca i znanja iz velikih skupova podataka. Uključuje korištenje statističkih tehnika, algoritama strojnog učenja i alata za upravljanje bazom podataka za analizu velikih skupova podataka i izvlačenje smislenih uvida. Primarni cilj je transformirati sirove podatke u korisne informacije koje mogu podržati procese donošenja odluka [1].

Proces rudarenja podataka općenito slijedi ove korake [1]:

1. Čišćenje podataka (engl. *Data cleaning*): uklanjanje šuma i nedosljednih podataka.
2. Integracija podataka (engl. *Data integration*): moguća kombinacija više izvora podataka.
3. Odabir podataka (engl. *Data selection*): dohvaćanje bitnih podataka iz baza podataka.
4. Transformacija podataka (engl. *Data transformation*): transformacija i sređivanje podataka u obrasce prikladno za rudarenje izvođenjem sumarnih ili agregacijskih operacija.
5. Rudarenje podataka (engl. *Data mining*): proces u kojem se primjenjuju inteligentne metode za izdvajanje obrazaca podataka.
6. Evaluacija obrazaca (engl. *Pattern evaluation*): prepoznavanje zanimljivih obrazaca koji predstavljaju znanje na temelju mjera zanimljivosti.
7. Prezentacija znanja (engl. *Knowledge presentation*): vizualiziranje i primjena tehnika reprezentacija znanja koje se koriste za predstavljanje rudarenog znanja korisnicima.

Koraci od 1 do 4 su različiti oblici pretprocesiranja podataka (engl. *Data preprocessing*), gdje se podaci pripremaju za rudarenje. Korak rudarenja podataka može komunicirati s korisnikom ili bazom znanja, korisniku se prezentiraju zanimljivi uzorci i mogu se

pohraniti kao novo znanje u baza znanja. Rudarenje podataka ima širok raspon primjena, uključujući znanstvena istraživanja, zdravstvo, obrazovanje, bankarsku industriju, telekomunikacije itd. Osim toga, može se primijeniti na različite oblike podataka kao što su prostorni podaci, tokovi podataka, tekstualni podaci i grafikoni ili umreženi podaci [1].

Rudarenje podataka u obrazovanju (engl. *Educational Data Mining*) područje je istraživanja koje koristi rudarenje podataka, strojno učenje i psihološke metode kako bi razumjelo kako učenici uče. Istraživači formuliraju hipoteze i provode studije kako bi predvidjeli nove ishode i generirali novo znanje, čime se poboljšavaju obrazovni sustavi [2].

Primjena rudarenja podataka u obrazovanju raste kako podaci iz raznih simulacija, igrica i inteligentnih tutorskih sustava postaju dostupniji. Bavi se brojnim zadacima i izazovima učenja, uključujući evaluaciju materijala za učenje, otkrivanje atipičnih ponašanja učenika, procjenu uspješnosti učenja učenika, predviđanje studentskih ocjena i praćenje završetka online tečaja, između ostalog [3].

Istraživači u obrazovnom rudarenju podataka istražuju širok raspon tema, uključujući individualno učenje putem obrazovnog softvera, računalno podržano suradničko učenje, računalno prilagođeno testiranje i čimbenike koji pridonose neuspjehu učenika ili napuštanju studija. Jedan od glavnih fokusa u ovom području je poboljšanje studentskih modela, koji predstavljaju različite aspekte studentskog profila, kao što su njihovo znanje, motivacija, metakognicija i stavovi. Modeliranjem ovih individualnih razlika, obrazovni softver može bolje prilagoditi svoje odgovore, značajno poboljšavajući ishode učenja učenika [4].

Metode rudarenja podataka u obrazovanju su omogućile istraživačima da analiziraju veći broj značajki studenata u stvarnom vremenu, pružajući dublji uvid u ponašanja studenata nego prije. Istraživači su proširili analiziranje studenata izvan obrazovnog softvera kako bi identificirali prediktore studentskog neuspjeha ili napuštanja kolegija na studiju ili ukupnog zadržavanja u visokom obrazovanju. Druga značajna primjena rudarenja podataka u obrazovanju je u poboljšanju modela struktura područnog znanja. Kombinirajući psihometrijsko modeliranje s algoritmima strojnog učenja, istraživači su razvili automatizirane metode za otkrivanje točnih modela domene izravno iz podataka. Obrazovno rudarenje podataka se također primjenjuje za proučavanje učinkovitosti različitih oblika pedagoške podrške, kako unutar softvera za učenje tako i u drugim kontekstima kao što je suradničko učenje. Istraživači nastoje utvrditi koje su vrste podrške

najučinkovitije za različite skupine učenika ili u različitim situacijama učenja. Koristi se za prikupljanje empirijskih dokaza za pročišćavanje i proširenje obrazovnih teorija, s ciljem produbljivanja razumijevanja čimbenika koji utječu na učenje [4].

1.1. Odustajanje od školovanja

Odustajanje od školovanja značajan je problem u obrazovnim institucijama diljem svijeta, što dovodi do nepovoljnih ishoda za pojedince i društvo. Razumijevanje i predviđanje odustajanja učenika postalo je središnja točka istraživanja, jer omogućuje institucijama da provedu pravovremene intervencije i primijene sustave podrške koji mogu pomoći zadržati učenike i poboljšati njihov akademski uspjeh [5].

Odustajanje studenata odnosi se na prijevremeni prekid studija od strane studenta prije završetka kolegija ili programa. Stope prekida školovanja mogu uvelike varirati ovisno o razini obrazovanja, vrsti ustanove i demografskim čimbenicima. Posljedice odustajanja od školovanja su teške, uključujući nižu životnu zaradu, smanjene mogućnosti zapošljavanja i šire društvene troškove. Rano prepoznavanje rizičnih učenika na njihovom akademskom putu ključno je za sprječavanje napuštanja škole i poticanje cjeloživotnog i poticajnijeg obrazovnog okruženja [5].

Brojni čimbenici utječu na vjerojatnost odustajanja učenika, a mogu se široko kategorizirati u akademske, demografske, socijalne i psihološke čimbenike [5]:

- Akademski čimbenici: Loš akademski uspjeh, niske ocjene i izostanci jaki su prediktori napuštanja škole. Studenti koji se akademski muče često se osjećaju nepovezano sa svojim studijem, što dovodi do povećanog rizika od ispadanja.
- Demografski čimbenici: Dob, spol, socio-ekonomski status i obiteljsko porijeklo mogu utjecati na stopu napuštanja škole. Na primjer, učenici iz obitelji s nižim prihodima mogu se suočiti s financijskim pritiscima koji povećavaju njihovu vjerojatnost preranog napuštanja škole.
- Društveni čimbenici: Socijalna integracija, odnosi s vršnjacima i sustavi podrške igraju značajnu ulogu u zadržavanju učenika. Veća je vjerojatnost da će se ispisati učenici koji se osjećaju izolirano ili bez podrške.

- Psihološki čimbenici: Problemi s mentalnim zdravljem, nedostatak motivacije i niska samoučinkovitost mogu pridonijeti odluci učenika da napusti školu. Psihološka podrška i usluge savjetovanja bitne su u rješavanju ovih problema.

Odustajanje od školovanja predstavlja globalni problem koji se odnosi na učenike koji napuštaju obrazovni sustav prije završetka formalnog školovanja. Iako je to široko proučavano u kontekstu osnovnog i srednjoškolskog obrazovanja, slični izazovi postoje i na visokoškolskim institucijama, gdje se studenti suočavaju s pritiscima koji ih mogu dovesti do odustajanja od studija. Odustajanje od studija obuhvaća složene uzroke, uključujući akademske, socijalne i financijske faktore, ali i osobne okolnosti poput motivacije i mentalnog zdravlja. S obzirom na to da visoko obrazovanje igra ključnu ulogu u oblikovanju profesionalnih mogućnosti i budućeg života pojedinca, istraživanje odustajanja na razini studija postalo je ključna tema, jer pomaže u razumijevanju kako podržati studente i smanjiti rizik od odustajanja od studija.

1.2. Povezani radovi

Odustajanje studenata iz visokog obrazovanja je pitanje koje predstavlja značajne izazove za sveučilišta diljem svijeta. Sposobnost točnog predviđanja, koji studenti su u opasnosti od napuštanja, može pomoći institucijama da provedu pravovremene intervencije, čime se poboljšavaju stope zadržavanja učenika i ukupni akademski uspjeh. Sve veći broj istraživanja uključuje različite pristupe predviđanju odustajanje od škole.

Delogu i suradnici su istraživali čimbenike koji utječu na stopu odustajanja studenata u talijanskim visokoškolskim ustanovama. Istraživali su učinkovitost metoda strojnog učenja u predviđanju napuštanja studija i otkrili su da su te tehnike vrlo učinkovite, s potencijalom da služe kao sustavi ranog upozorenja. Rezultati pokazuju da algoritmi strojnog učenja, posebice slučajne šume (engl. *random forest*) i algoritam za povećanje gradijenta (engl. *gradient boosting*), pokazuju snažnu prediktivnu točnost, ističući njihovu vrijednost kao ranih pokazatelja rizika od odustajanja. Analiza važnosti obilježja (engl. *feature importance*) otkriva da akademski uspjeh učenika tijekom prve godine igra ključnu ulogu u predviđanju odustajanja. Osim toga, dodatno potvrđuju utjecaj faktora kao što su prihod obitelji, ocjene u srednjoj školi i vrsta srednje škole koju pohađaju [6].

Wagner i drugi su ispitivali pet algoritama i dva različita skupa značajki izvedenih isključivo iz podataka o akademskom uspjehu kako bi predvidjeli odustajanja studenata na

njemačkom sveučilištu. Logistička regresija (engl. *logistic regression*) je dala najbolje rezultate. Ispitivali su je li model logističke regresije nepristran prema studentima i studenticama, kao što je za različite studijske programe. Međutim, otkrili su da to nije uvijek slučaj. To bi moglo biti zato što se stope odustajanja od škole razlikuju među različitim skupinama učenika. Kažu da ovo ograničenje treba uzeti u obzir pri primjeni ovih modela u praksi [7].

U studiji slučaja na Sveučilištu Roma Tre, administrativni podaci od približno 6000 studenata upisanih od 2009. godine na Odjel za obrazovanje na Sveučilištu Roma Tre korišteni su za treniranje konvolucijske neuronske mreže (engl. *Convolutional neural networks*, skraćeno CNN). Trenirana mreža je generirala prediktivni model kako bi odredila postoji li vjerojatnost da će student odustati. Osim toga, usporedili su performanse modela dubokog učenja s onima Bayesovih mreža (engl. *Bayesian networks,* skraćeno BN) [8].

Song i suradnici uočili su da su stope prekida školovanja ravnomjerno raspoređene u svim akademskim godinama u skupu podataka korejskog sveučilišta. Istraživali su tablice univerzalnih značajki prikladne za predviđanje odustajanja od studija među studentima kroz sve akademske godine. Napravili su nekoliko tablica značajki i usporedili izvedbu šest modela strojnog učenja pomoću tih tablica. Rezultati su pokazali da tablica značajki temeljena na srednjim vrijednostima nudi bolju generalizaciju, a model koji koristi tehniku povećanja gradijenta (engl. *gradient boosting*) nadmašuje ostale. Naglasili su ključnu ulogu povijesnih podataka učenika u točnom predviđanju odustajanja od škole [9].

U istraživanju iz 2021. Nunez-Naranjo i suradnici su koristili tehnike rudarenja podataka kako bi se identificirali obrasci i grupirali studenti, pružajući uvid u temeljne čimbenike koji doprinose odustajanju od studija. Algoritam k-srednjih vrijednosti (engl. *K-Means algorithm*) korišten je za klasifikaciju i identifikaciju obrazaca uspješnosti, dok je model stroja potpornih vektora (engl. *support vector machine*, skraćeno SVM) primijenjen za predviđanje ishoda za nove učenike [10].

Rezultati Pereza i suradnika su pokazali da čak i jednostavni algoritmi mogu pouzdano identificirati prediktore odustajanja od studija. Usporedili su izvedbu modela stabala odlučivanja (engl. *decision tree*), logističke regresije (engl. *logistic regression*), *Naive Bayes* i slučajne šume (engl. *random forest*) kako bi odredili najučinkovitiji pristup [11].

2. Metodologija

U kontekstu rješavanja problema klasifikacije, cilj je predvidjeti hoće li student odustati od svog studija ili ne, na temelju niza različitih značajki. Iskorištavanjem ovih značajki, cilj je izgraditi model koji može klasificirati studente u dvije kategorije: one koji će vjerojatno ostati na studiju i one koji su pod rizikom od ispisa sa studija. Ovaj proces uključuje primjenu statističkih tehnika i tehnika strojnog učenja za prepoznavanje obrazaca i odnosa unutar podataka, što u konačnici omogućuje donošenje informiranih predviđanja o odustajanju studenata. Cilj je pružiti djelotvorne uvide koji mogu pomoći obrazovnim ustanovama u provedbi ciljanih intervencija i strategija podrške za poboljšanje stopa zadržavanja učenika i poboljšanje ukupnih obrazovnih ishoda.

Koraci koji su bili potrebni za to su: prikupljanje podataka o studentima i njihovom studiranju, čišćenje i uređivanje podataka da su podaci u obliku koji je prikladan za daljnji rad, analiza podataka i primjena odabranih modela strojnog učenja za predikciju.

2.1. O podacima

U ovom radu korišteni podaci su podaci o studentima Prirodoslovno-matematičkog fakulteta u Splitu, studija Informatike. Podaci su preuzeti iz ISVU sustava (Informacijski sustav visokih učilišta) od strane ovlaštene osobe s fakulteta, te su anonimizirani. Podaci se odnose na studente koji su upisani u prvu godinu preddiplomskog studija Informatike od akademske godine 2012./2013. do akademske godine 2022./2023

Podaci su prikazani u obliku *'.csv'* tablica te postoje četiri tablice. Prva tablica naziva **Opći podaci i matura** se sastoji od općih podataka studenata i podataka državne mature. Ona ima 400 redova i 18 stupaca. Tablica se sastoji od stupaca koji pohranjuju određene podatke, to su:

- Godina upisa na studij
- JMBAG (Jedinstveni Matični Broj Akademskog Građana)
- Godina rođenja
- Spol studenta

- Završena srednja škola
- Prosjek ocjena iz srednje škole
- Bodovi srednje škole
- Bodovi na državnoj maturi
- Bodovi za dodatna postignuća (natjecanja)
- Ukupan broj bodova za upis na studij
- Postotak iz mature engleskoj jezika (A razina)
- Postotak iz mature engleskoj jezika (B razina)
- Postotak iz mature hrvatskog jezika
- Postotak iz mature matematike (A razina)
- Postotak iz mature matematike (B razina)
- Postotak iz mature informatike
- Status studenta (diplomirao, ispisan, aktivan)
- Datum diplomiranja/ispisa

U drugoj tablici **O predmetima** su pohranjeni podaci o svakom predmetu koji je svaki student upisao na studiju od akademske godine 2012./2013. do akademske godine 2023./2024., stupci naziva:

- JMBAG (Jedinstveni Matični Broj Akademskog Građana)
- Status upisanog predmeta
- Oslobođen polaganja
- Obavezan ili izborni predmet
- Način upisa predmeta
- Kratica predmeta
- Naziv predmeta
- Šifra predmeta
- Akademska godina

- Nastavna godina
- Semestar slušanja predmeta

Treća tablica **Studij** se sastoji o podacima o studiranju studenta od akademske godine 2012./2013. do akademske godine 2023./2024., sa stupcima:

- JMBAG (Jedinstveni Matični Broj Akademskog Građana)
- Akademska godina zadnjeg upisa
- Paralelni studij
- Indikator upisa zadnje upisane godine
- Nastavna godina zadnjeg upisa
- Aritmetički prosjek ocjena na studiju
- Težinski prosjek ocjena na studiju
- Broj položenih ispisa na studiju
- Broj položenih ispita koji ulaze u prosjek ocjena na studiju
- Zbroj pozitivnih ocjena na studiju
- Zbroj ECTS bodova na studiju
- Broj predmeta na studiju
- Zbroj ocjena x ECTS bodovi
- Zbroj ocjena x broj položenih ispita
- Osvojeno ECTS bodova koji ulaze u stjecanje kvalifikacije

Četvrta tablica **Završetak studija** predstavlja podatke o studentima koji su završili studij, stupci:

- Datum završetka studija
- JMBAG (Jedinstveni Matični Broj Akademskog Građana)
- Akademska godina završetka studija
- Akademska godina zadnjeg upisa
- Aritmetički prosjek ocjena na studiju

- Težinski prosjek ocjena na studiju
- Ukupan broj ECTS bodova
- Broj položenih ispita
- Broj predmeta
- Koliko je student studirao godina

Poveznica svih tablica je JMBAG (Jedinstveni Matični Broj Akademskog Građana), kojeg svaki student ima različit.

Za jedan dio studenata nema podataka o završenoj srednjoj školi jer su to studenti koji su završili srednju školu izvan Republike Hrvatske.

U tablici **Završetak studija** postoje podaci i o studentima koji su upisani prije 2012. godine, ali za te studente ne postoje podaci o polaganju državne mature.

2.2. Pretprocesiranje podataka

Prije primjene određenih modela, bilo je potrebno pretprocesirati podatke. U ovom koraku, su se čistili i uređivali podaci u tablicama. Za sve tablice bilo je potrebno:

- Promijeniti tip podatka: iz tekstualnih u brojčane ili decimalne vrijednosti
- Vrijednosti datuma postaviti u određen format (YYYY-MM-DD)
- Preimenovati stupce podataka u tablicama
- Provjeriti postoje li duplicirani podaci i prazni podaci: nisu postojali duplicirani podaci, a prazni podaci u brojčanim tipovima podataka su zamijenjeni s nulom.

Za tablice **Studij** i **O predmetima** se nije radilo dodatno uređivanje te su spremljene nakon spomenutog uređenja.

Nadalje, za tablice **Opći podaci i matura** i **Završetak studija** su se radili dodatni i različiti koraci zbog različitih podataka koje pohranjuju.

Za tablicu **Opći podaci i matura** bilo je potrebno dodati:

- upisna dob studenta
- *dummy* varijabla za spol

- varijabla za kategoriju srednje škole
- varijabla za mjesto srednje škole
- varijabla za kategoriju prosjeka ocjena u srednjoj školi: 2, 3, 4, 5
- razdvojiti studente koji još aktivno studiraju od ostalih
- spremanje uređenih podataka

Među podacima postoje *Nan* (engl. *not a number*) vrijednosti za varijable koje pohranjuju podatke o završenoj srednjoj školi i u podacima o datumu završetka studija (ispis ili diploma). Ti podaci nedostaju. Podaci o nazivu završene srednje škole nedostaju za studente koji su tu školu završili izvan Republike Hrvatske. Podaci o datumu završetka studija nedostaju za studente koji su još u aktivnom studiranju.

Upisna dob svakog studenta je izračunata na način da se od godine upisa na studij oduzme godina rođenja. Te vrijednosti su pohranjene u novu varijablu.

upisna dob = upisana godina - godina rođenja

Stvorena je varijabla koja uzima vrijednosti jedan ako je student ženskog spola *'Ž'* ili nulu ako je ostalo, *'M'* vrijednost.

Ovisno o vrsti obrazovnog programa, u Republici Hrvatskoj postoje tri vrste srednjih škola [12]:

- gimnazije (engl. *gymnasiums*): opće, jezične, klasične, prirodoslovno-matematičke, prirodoslovne
- strukovne škole (engl. *vocational schools*)
- umjetničke škole (engl. *art schools*): glazbene, plesne, likovne

Ručno su gledane koje sve srednje škole postoje među podacima, škole su zapisane tako da se sastoje od naziva škole i grada u kojem se nalaze. Uzete su glavne riječi koje opisuju svaku od tih škola ili skupinu škola (imaju isto ime, ali su u različitom gradu). Dodijeljena im kategorija po tim riječima. U tablici 1. su prikazane škole koje su među podacima sa svojom kategorijom:

Tablica 1. Popis kategorija za pojedine škole

Naziv škole	**Kategorija**
Graditeljsko-geodetska tehnička	strukovna
Turistička i ugostiteljska	strukovna
Gimnazija	gimnazija
Tehnička	strukovna
Srednja strukovna	strukovna
Ekonomsko-birotehnička i trgovačka	strukovna
Turističko-ugostiteljska	strukovna
Elektrotehnička	strukovna
Graditeljska obrtnička i grafička	strukovna
gimnazija	gimnazija
Medicinska	strukovna
Ekonomsko-birotehnička	strukovna
Pomorska	strukovna
Trgovačka	strukovna
Ekonomska	strukovna
Medicinska i kemijska	strukovna
Poljoprivredna, prehrambena i veterinarska	strukovna
GIMNAZIJSKI KOLEGIJ	gimnazija
Tehnička i industrijska	strukovna
Glazbena	umjetnička
Prometno-tehnička	strukovna
Srednja zubotehnička	strukovna
Zdravstvena	strukovna
Škola likovnih umjetnosti	umjetnička
Škola primjenjene umjetnosti i dizajna	umjetnička

Ako neka od ovih riječi spada u naziv škole onda se dodjeljuje određena kategorija u novu varijablu u tablici za pohranjivanje kategorije srednje škole.

Također, među podacima su postojali studenti koji su pohađali škole naziva Prirodoslovna škola i Srednja škola. Gledajući samo ime nije bilo moguće odrediti u koju kategoriju spadaju i koji program je student pohađao jer u tim školama postoje gimnazijski programi i strukovni programi. Zbog tog razloga postoji dodatna tablica s podacima za te studente i koji su program pohađali i završili u srednjoj školi, gimnazijski ili strukovni. Tablica se sastoji od:

- JMBAG
- kategorija srednje škole

Tablici **Opći podaci i matura** su se dodale vrijednosti iz dodatne tablice za definiranje kategorije srednje škole za određene studente.

Stvorene su tri varijable:

- `gymnasiums_dummy`
- `vocational_dummy`
- `art_dummy`,

koje poprimaju vrijednost jedan za red u kojem je škola određene kategorije (u `gymnasiums_dummy` spadaju gimnazije, `vocational_dummy` strukovne, a `art_dummy` umjetničke).

Za lokaciju srednje škole izdvojene su vrijednosti pronađene u varijabli koji pohranjuje završenu srednju školu te su u novu varijablu uzeti samo gradovi koji se spominju u takvim redovima. Za podatke bez srednjih škola je stavljena vrijednost *'izvan RH'*. Svi gradovi u kojima su studenti pohađali srednje škole su Split, Dubrovnik, Sinj, Imotski, Metković, Šibenik, Vela Luka, Ogulin, Slavonski Brod, Makarska, Drniš, Zadar, Vrgorac, Korčula, Pula, Ploče, Novska, Trogir, Omiš, Pazin, Benkovac, Županja, Vis, Kaštel Štafilić, Zagreb, Blato, Požega, Sisak, Hvar.

Također je stvorena varijabla koja uzima vrijednosti jedan ako je student pohađao srednju izvan RH ili nulu ako je ostalo, neki od pronađenih gradova.

Kategoriziranje prosjeka ocjena iz srednje škole je definirano po rasponima. Svaki prosjek ocjena svakog studenta je prikazan u decimalnom obliku i spada pod jednu od kategorija. Rasponi i kategorije su definirane:

- Ako je ocjena manja od 2.5, dodjeljuje se kategorija ocjene 2.
- Ako je ocjena između 2.5 (uključivo) i 3.5 (isključivo), dodjeljuje se kategorija ocjene 3.
- Ako je ocjena između 3.5 (uključivo) i 4.5 (isključivo), dodjeljuje se kategoriju ocjene 4.
- Ako je ocjena 4.5 ili viša, dodjeljuje se kategoriju ocjene 5.
- Ako ocjena ne odgovara nijednom od ovih raspona (iako bi uvijek trebala odgovarati jednom od njih), vraća *'Nevažeće'* (engl. *invalid*).

Za status odustajanja, stvorena je varijabla koja uzima vrijednosti jedan ako se student ispisao s fakulteta ili nulu ako je ostalo, aktivan u studiranju i dalje ili dobio diplomu.

Iz prve tablice spremljeni su podaci kao nove *csv* tablice:

- Podaci o studentima koji još aktivno studiraju u akademskoj godini 2023./2024.
- Podaci o studentima koji ne studiraju više (ili su se ispisali ili su dobili diplomu, završili studij) – glavni podaci.

U tablici **Završetak studija** bilo je potrebno izračunati godinu upisa na studij. Kako u toj tablici postoje podaci za datum završetka studija i vrijednosti koliko je student studirao godina, godina upisa se dobila na način da se od datuma završetka studija oduzeo broj godina studiranja. Broj godina studiranja koji je zapisan u decimalnom obliku je prije pretvoren u broj mjeseci.

Iz te tablice su se podaci odvojili u dvije odvojene *csv* tablice:

- Podaci o završetku studija za studente s maturom.
- Podaci o završetku studija za studente bez maturom.

Nakon uređivanja slijedilo je spajanje podataka, tablice **Studij** s tablicom **Opći podaci i matura**. Nastale su dvije tablice podataka:

- Podaci o studentima (opći podaci i podaci s državne mature) koji ne studiraju više (ili su se ispisali ili su dobili diplomu, završili) i podaci o studiranju – glavni podaci
- Podaci o studentima (opći podaci i podaci s državne mature) koji još aktivno studiraju u akademskoj godini 2023./2024 i njihovi podaci o studiranju

Tablice podataka koje su se koristile u daljnjem radu su:

- Podaci o studentima (opći podaci i podaci s državne mature) koji ne studiraju više (ili su se ispisali ili su dobili diplomu, završili) i podaci o studiranju
- Podaci o svakom predmetu koji je svaki student upisao na studiju

2.3. Modeli strojnog učenja

Što se tiče algoritama koji se koriste za izgradnju modela klasifikacije, neki od standardnih algoritama koji se često navode u literaturi za klasifikaciju odustajanja studenata od studija su:

- Logistička regresija (engl. *logistic regression*)
- Stablo odluke (engl. *decision tree*)
- Slučajne šume (engl. *random forest*)

- Stroj potpornih vektora (engl. *support vector machine,* skraćeno SVM)
- Povećanje gradijenta (engl. *gradient boosting*)

Logistička regresija se koristi za probleme binarne klasifikacije, gdje je cilj predvidjeti vjerojatnost kategoričke ovisne varijable (moguća su dva ishoda). Za razliku od linearne regresije, koja modelira linearni odnos između nezavisnih varijabli i kontinuiranog ishoda, logistička regresija koristi se za predviđanje vjerojatnosti da određeni ulaz pripada jednoj od dvije klase. Model se temelji na logističkoj funkciji (poznatoj i kao sigmoidna funkcija), koja pretvara linearni izlaz u vrijednosti između 0 i 1, interpretirane kao vjerojatnosti. Logistička regresija uzima jednu ili više nezavisnih varijabli (koje mogu biti numeričke ili kategoričke) i modelira njihovu povezanost s binarnom ovisnom varijablom. Koristi u zadacima klasifikacije zbog svoje jednostavnosti i interpretabilnosti. Pruža vjerojatnosti za svaku klasu, što može biti korisno za donošenje odluka ili razumijevanje vjerojatnosti različitih ishoda [13].

Kod ispod prikazuje definiranje i pozivanje logističke regresije te treniranje nad skupom podataka koristeći Python:

```
from sklearn.linear_model import LogisticRegression
# split into train and test sets
# initialize the Logistic Regression Model
log_reg = LogisticRegression()
# train the Model
log_reg.fit(X_train, y_train)
```

Stablo odluke popularan je algoritam strojnog učenja koji se koristi za zadatke klasifikacije i regresije. Modelira odluke i njihove moguće posljedice u strukturi poput stabla, što ga čini intuitivnim i lakim za razumijevanje. Struktura slična stablu govori da svaki čvor (engl. *node*) predstavlja odluku temeljenu na vrijednosti neke značajke (ulaznog podatka), dok svaka grana (engl. *branch*) koja izlazi iz tog čvora predstavlja jedan mogući ishod te odluke. Konačni izlazi modela – listovi stabla – predstavljaju klasifikaciju (u slučaju klasifikacijskog zadatka) ili predviđenu vrijednost (u slučaju regresijskog zadatka). Stablo odluke funkcionira na način da proces izgradnje stabla započinje odabirom značajke (varijable) koja najbolje razdvaja podatke u skupu podataka. Na primjer, prosjek ocjena. Nakon odabira značajke, stablo se grana na temelju različitih vrijednosti ili raspona te značajke. Svaka grana predstavlja jednu mogućnost koju ta značajka može poprimiti (na primjer, prosjek ocjena < 3.5). Podaci se zatim dijele u podskupove na temelju grananja, a

algoritam nastavlja odabirati nove značajke unutar svakog podskupa, granajući se dok ne dođe do trenutka kada daljnje razdiobe nisu moguće ili nema potrebe. Kad se dođe do kraja stabla, lisni čvor (engl. *leaf node*), tamo se nalazi oznaka klase (u klasifikaciji) ili predviđena vrijednost (u regresiji). U klasifikacijskom zadatku, lisni čvor predstavlja konačnu odluku o klasi kojoj pripadaju podaci, a u regresiji lisni čvor daje konačnu numeričku vrijednost [14].

Kod ispod prikazuje definiranje i pozivanje stabla odluke te treniranje nad skupom podataka koristeći Python. Parametar `random_state` u klasifikatoru `DecisionTreeClassifier` služi za kontrolu načina na koji se slučajni brojevi generiraju unutar algoritma stabla odluke. Odabirom fiksne vrijednosti (na primjer, 42), osigurava se da, prilikom svakog pokretanja algoritma, se koristi ista sekvenca slučajnih brojeva. To omogućuje konzistentne rezultate, što je važno za usporedbu modela ili prilikom ponovnog testiranja [15].

```
from sklearn.tree import DecisionTreeClassifier
# split into train and test sets
# initialize the Decision Tree Classifier Model
decision_tree = DecisionTreeClassifier(random_state=42)
# train the Model
decision_tree.fit(X_train, y_train)
```

Slučajna šuma je skupna metoda učenja koja se koristi za zadatke klasifikacije i regresije. Temeljena na principu skupnog učenja (engl. *ensemble learning*). Funkcionira tako da gradi više stabala odluke tijekom procesa treniranja, a zatim kombinira njihove rezultate kako bi poboljšala točnost predikcija i smanjila rizik od pretreniranja (engl. *overfitting*). Umjesto da se oslanja na jedno stablo odluke, koristi cijelu "šumu" stabala, čime se smanjuje varijabilnost i povećava stabilnost modela. Izvorni set podataka slučajno podijeli u više podskupova. Na svakom od tih podskupova trenira se jedno stablo odluke. Osim slučajne podjele podataka, slučajna šuma također uvodi slučajnost pri odabiru značajki. Svako stablo trenira na različitim uzorcima značajki, čime se smanjuje međusobna povezanost stabala. To pomaže u smanjenju varijance i čini model otpornijim na pretreniranje. U klasifikaciji, svako stablo daje svoj glas (engl. *vote*), odnosno predikciju klase, a konačna klasa je ona koja dobije najviše glasova (engl. *majority voting*). U regresiji, se predviđanja iz svih stabala prosječe kako bi se dobila konačna predikcija. Primarna ideja koja stoji iza je iskoristiti moć višestrukih stabala odlučivanja kako bi se napravila pouzdanija i točnija predviđanja [16].

Kod ispod prikazuje definiranje i pozivanje slučajne šume te treniranje nad skupom podataka koristeći Python. Parametar `n_estimators` definira broj stabala odluke koja će biti izgrađena unutar slučajne šume. Parametar `max_depth` postavlja maksimalnu dubinu svakog stabla odluke. Ako je dubina ograničena, sprječava se da stablo postane prekomplicirano, što može smanjiti rizik od pretreniranja. Parametar `random_state` kontrolira slučajnost u modelu [17].

```
from sklearn.ensemble import RandomForestClassifier
# split into train and test sets
# initialize the Random Forest Classifier Model
rf = RandomForestClassifier(n_estimators=200, max_depth=10,
random_state=42)
# train the Model
rf.fit(X_train, y_train)
```

Stroj potpornih vektora (SVM) je model strojnog učenja koji se primarno koristi za zadatke klasifikacije, iako se također može prilagoditi za regresiju. Glavna ideja iza SVM-a je pronaći optimalnu hiperravninu (engl. *hyperplane*) koja razdvaja podatke u različite klase uz maksimalnu marginu, čime se postiže što bolje razdvajanje podataka u prostoru značajki. U SVM-u, hiperravnina je površina koja dijeli podatkovne točke dviju različitih klasa. U dvodimenzionalnom prostoru, ta hiperravnina je linija, dok je u trodimenzionalnom prostoru to ravnina. U višedimenzionalnom prostoru, hiperravnina postaje (n-1)-dimenzionalni potprostor. SVM traži onu hiperravninu koja maksimizira marginu. Margina je udaljenost između hiperravnine i najbližih točaka iz svake klase. Veća margina znači bolje razdvajanje klasa i povećanu točnost modela na novim podacima. Točke koje se nalaze najbliže hiperravnini nazivaju se potporni vektori (engl. *support vectors*). Potporni vektori određuju marginu između klasa – udaljenost između tih točaka i hiperravnine je upravo ono što model nastoji maksimizirati. SVM koristi tzv. funkciju gubitka koja penalizira krivo klasificirane točke. Ključni parametar je C koji određuje balans između maksimalne margine i broja grešaka. Manja vrijednost C dopušta modelu da ima širu marginu uz toleriranje većeg broja grešaka, dok veća vrijednost C vodi do uske margine i manjeg broja pogrešaka, što može dovesti do pretreniranja. SVM koristi kernel trik (engl. *kernel trick*) koji omogućuje SVM-u da se bavi složenim problemima gdje su klase nelinearno podijeljene. Umjesto da eksplicitno transformira podatke u viši dimenzionalni prostor, kernel funkcionira tako da računa udaljenosti između podatkovnih

točaka u višedimenzionalnom prostoru, čime izbjegava potrebu za izravnim radom s velikim brojem značajki [18].

Kod ispod prikazuje definiranje i pozivanje SVM-a te treniranje nad skupom podataka koristeći Python. Izvršava se podešavanje hiperparametara (engl. *hyperparameter tuning*) za model pomoću GridSearchCV što automatizira proces pronalaženja najbolje kombinacije hiperparametara. Parametri su definirani u varijabli `param_grid`, gdje su definirane vrijednosti za testiranje parametara C, gamma i kernel. C kao parametar regularizacije se testira za vrijednosti 0.1, 1, 10. Manja vrijednost C čini model regulariziranijim (jednostavnijim s većom marginom, ali mogućom višom pogrešnom klasifikacijom). Veće vrijednosti C učinit će da model bolje odgovara podacima (manja margina, ali manje pogrešno klasificiranih točaka). Kernel koeficijent gamma se testira za vrijednosti *scale* i *auto*. Definira koliko daleko seže utjecaj jednog primjera treninga. Kernel se testira za *linear* i *rbf* (engl. *radial basis function*) [19].

```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
# split into train and test sets
param_grid = {'C': [0.1, 1, 10],
              'gamma': [‘scale’, ‘auto’],
              'kernel': ['linear', 'rbf']}

# initialize GridSearchCV
grid = GridSearchCV(SVC(), param_grid, refit=True, cv=3,
n_jobs=-1)
# train the Model
grid.fit(X_train, y_train)
```

Pojačavanje gradijenta je tehnika strojnog učenja koja se temelji na ideji postupnog gradnje prediktivnog modela kroz iterativno učenje iz grešaka prethodnih modela. Ključna komponenta ovog algoritma je korištenje više slabih učenika (engl. *learners*), slabih klasifikatora ili regresora, najčešće jednostavnih stabala odluke, koji zajedno čine snažan model. Osnovni koncept pojačavanja gradijenta je da se modeli grade sekvencijalno, tako da svaki novi model pokušava ispraviti pogreške koje su napravili njegovi prethodnici. Pri svakom koraku, model procjenjuje rezidualne pogreške (odnosno razlike između stvarnih i predviđenih vrijednosti) te pokušava prilagoditi svoje parametre kako bi smanjio te pogreške. Umjesto da svaki model doprinosi jednako, ugradnjom novih stabala model

naglašava pogreške prethodnih i na taj način usmjerava se na teško predvidljive uzorke [20].

Kod ispod prikazuje definiranje i pozivanje pojačavanja gradijenta te treniranje nad skupom podataka koristeći Python.

```
from sklearn.ensemble import GradientBoostingClassifier
# split into train and test sets
# initialize the Gradient Boosting Classifier
gb_clf = GradientBoostingClassifier(random_state=42)
# train the Model
gb_clf.fit(X_train, y_train)
```

2.4. Metrike evaluacije

Klasifikacijski modeli u strojnom učenju služe za dodjeljivanje ulaznih primjera u odgovarajuće klase. U binarnoj klasifikaciji, model svrstava podatke u dvije klase: pozitivnu i negativnu. Svaki primjer nakon klasifikacije može pripadati jednom od četiri ishoda [21]:

- Istinito pozitivni (engl. *True Positive*, TP): pozitivne instance koje su točno klasificirane kao pozitivne,
- Lažno negativni (engl. *False Negative*, FN): pozitivne instance koje su pogrešno klasificirane kao negativne,
- Istinito negativni (engl. *True Negative*, TN): negativne instance točno klasificirane kao negativne,
- Lažno pozitivni (engl. *False Positive*, FP): negativne instance pogrešno klasificirane kao pozitivne.

Ovi rezultati prikazuju se u matrici konfuzije (engl. *confusion matrix*) koja sažima uspješnost klasifikacijskog modela (Tablica 2) [21].

Tablica 2. Matrica konfuzije s ishodima

	Predviđeno pozitivne	**Predviđeno negativne**
Stvarno pozitivne	Istinito pozitivne (TP)	Lažno negativne (FN)
Stvarno negativne	Lažno pozitivne (FP)	Istinito negativne (TN)

Metrike evaluacije koje evaluiraju modele strojnog učenja u zadatku klasifikacije [22, 23]:

- Točnost (engl. *accuracy*) - mjeri udio točnih predviđanja modela u cijelom skupu podataka. Izračunava se kao omjer stvarno pozitivnih (TP) i stvarno negativnih (TN) uzoraka prema ukupnom broju uzoraka.
- F-rezultat (engl. *F-score*) - sredina preciznosti i prisjećanja. Koristan je kada se traži ravnoteža između visoke preciznosti i visokog prisjećanja, budući da kažnjava ekstremno negativne vrijednosti bilo koje komponente.
- Preciznost (engl. *precision*) - mjeri udio pravih pozitivnih predviđanja među svim pozitivnim predviđanjima napravljenim modelom. Izračunava se kao omjer TP i zbroja TP i lažno pozitivnih rezultata (FP).
- Odziv (engl. *recall*) - mjeri udio istinskih pozitivnih predviđanja među svim stvarnim pozitivnim slučajevima. Izračunava se kao omjer TP i zbroja TP i lažno negativnih rezultata (FN).
- AUC (engl. *Area Under the Curve*) – koristi se za procjenu kako dobro model može razdvojiti različite klase. Odnos između TP i FP.

Točnost mjeri ukupnu točnost predviđanja modela, dok se preciznost i odziv fokusiraju na kvalitetu pozitivnih i negativnih predviđanja. F-rezultat pruža ravnotežu između preciznosti i prisjećanja, što ga čini opsežnijom metrikom za procjenu klasifikacijskih modela [22].

AUC se često koristi jer je neovisan o pragovima klasifikacije, što znači da mjeri ukupnu sposobnost modela za razdvajanje klasa bez obzira na konkretan prag za pozitivnu ili negativnu klasifikaciju. Što je veća AUC vrijednost, to je model bolji u razlikovanju klasa [23].

2.5. Korišteni alati

Visual Studio Code (skraćeno VS Code) je uređivač otvorenog izvornog koda koji je razvio Microsoft. Široko se koristi u razvoju softvera zbog svog bogatog skupa značajki, uključujući ugrađenu podršku za *Git*, alate za uklanjanje pogrešaka i ogromno tržište proširenja koja mogu poboljšati njegovu funkcionalnost. Jedna ključna značajka VS Codea je njegova svestranost u podržavanju širokog raspona programskih jezika i tipova datoteka, uključujući Jupyter bilježnice (engl. *Jupyter Notebook*), datoteke s ekstenzijom *'.ipynb'*.

Ova podrška omogućena je upotrebom ekstenzije *Jupyter* koja korisnicima omogućuje pokretanje *Jupyterovih* bilježnica izravno unutar uređivača. Omogućuje korištenje fleksibilnosti okruženja VS Codea dok se radi s Python kodom, vizualizacijama i sadržajem *markdowna* u jednom sučelju računala [24].

Korištene Python biblioteke i dodaci koje se često koriste unutar *Jupyter* bilježnica, datoteka za rad s podacima, vizualizaciju i slično:

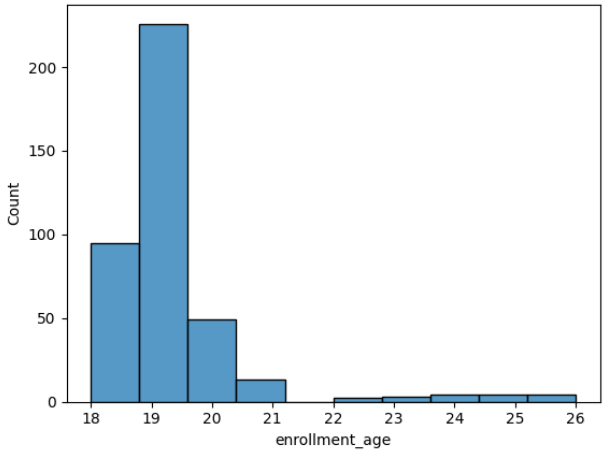
- *Pandas*: Koristi se za manipulaciju i analizu podataka. Pruža podatkovne strukture poput *DataFrames* za učinkovito rukovanje i analizu strukturiranih podataka (tablica s redovima i stupcima) [25].
- *NumPy*: Prvenstveno se koristi za numeričko računanje. Nudi podršku za velike, višedimenzionalne nizove i matrice, zajedno sa širokim rasponom matematičkih funkcija za rad s njima [26].
- *Matplotlib*: Biblioteka za crtanje koja se koristi za stvaranje statičnih, interaktivnih i animiranih vizualizacija u Pythonu. Često se koristi za generiranje grafikona i dijagrama [27].
- *Seaborn*: Pruža sučelje visoke razine za stvaranje estetski ugodnijih i složenijih statističkih grafika, poput toplinskih karti i okvirnih dijagrama [28].
- *datetime*: Koristi za rad s datumima i vremenima u Pythonu, dopuštajući manipulaciju objektima datum/vrijeme, oblikovanje i izračune kao što je pronalaženje razlika između datuma [29].
- *scikit-learn (sklearn):* Python biblioteka otvorenog koda dizajnirana za strojno učenje i analizu podataka. Pruža jednostavne i učinkovite alate za zadatke kao što su klasifikacija, regresija, grupiranje, smanjenje dimenzionalnosti, odabir modela i pretprocesiranje podataka [30].

3. Analiza podataka

Analizirani su podaci studenata koji još aktivno studiraju i studenata koji su završili studij (ispisani ili diplomirali), broj studenata 400. Podaci se sastoje od općih podataka o studentima, podacima s državne mature, podacima o studiranju, od akademske godina 2012./2013. do akademske godine 2022./2023.

3.1. Dob

Analiza studenata po upisnoj dobi, što govori koliko je student imao godina prilikom upisa na studij, je pokazala kako u većini studeni imaju 19 godina prilikom upisa na studij, njih preko 200 se upisuje s tom dobi. Dalje slijede studenti s 18 godina, njih nešto manje od 100. Slika 1. prikazuje distribuciju upisnih dobi studenata u vremenu upisa na studij, raspon godina je od 18 do 26:



Slika 1. Distribucija upisne dobi studenata

3.2. Broj upisanih

Tablica 3. prikazuje broj upisanih studenata po godinama, od godine 2012. do 2022. godine. Najveći broj upisanih studenata bio je u godinama 2018. i 2019., s po 40 studenata. Najmanji broj upisanih bio je 2014. godine, sa samo 30 studenata.

Tablica 3. Broj upisanih studenata po godini upisa

Upisna godina	Broj studenata
2012	35
2013	34
2014	30
2015	35
2016	35
2017	35
2018	40
2019	40
2020	39
2021	39
2022	38

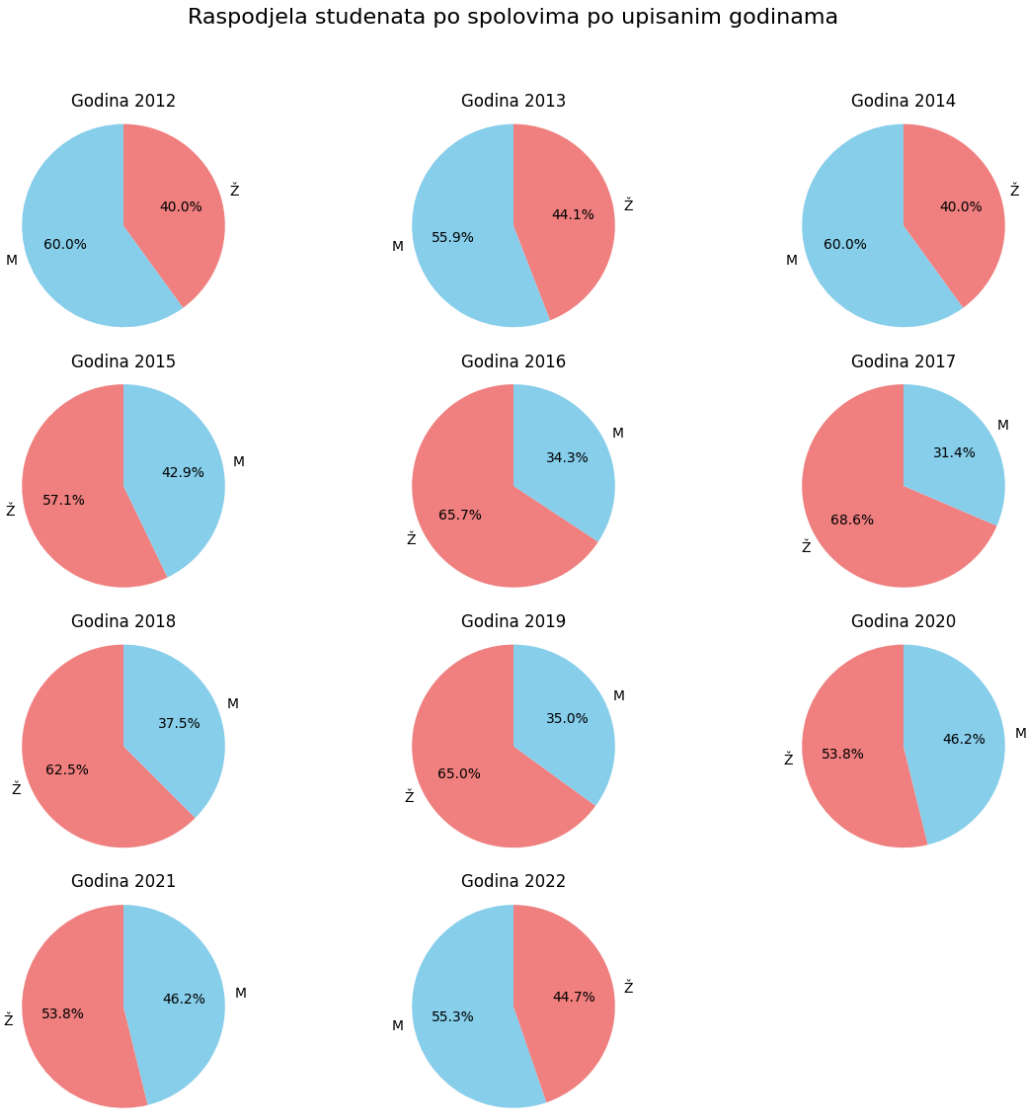
3.3. Spol

Općenita raspodjela studenata u podacima po spolovima je:

- 54.5% žena
- 45.5% muškaraca

Slika 2. prikazuje skup kružnih dijagrama koji pokazuju spolnu distribuciju upisanih studenata po godinama od 2012. do 2022.:

- 2012. – 2014.: većina upisanih studenata bili su muškarci.
- 2015. – 2017.: većina upisanih studenata bile žene.
- 2018. – 2022.: i dalje dominiraju studentice, od 2020. nešto više muškaraca nego godina prije.



Slika 2. Raspodjela studenata po spolovima po upisnim godinama

3.4. Završena srednja škola

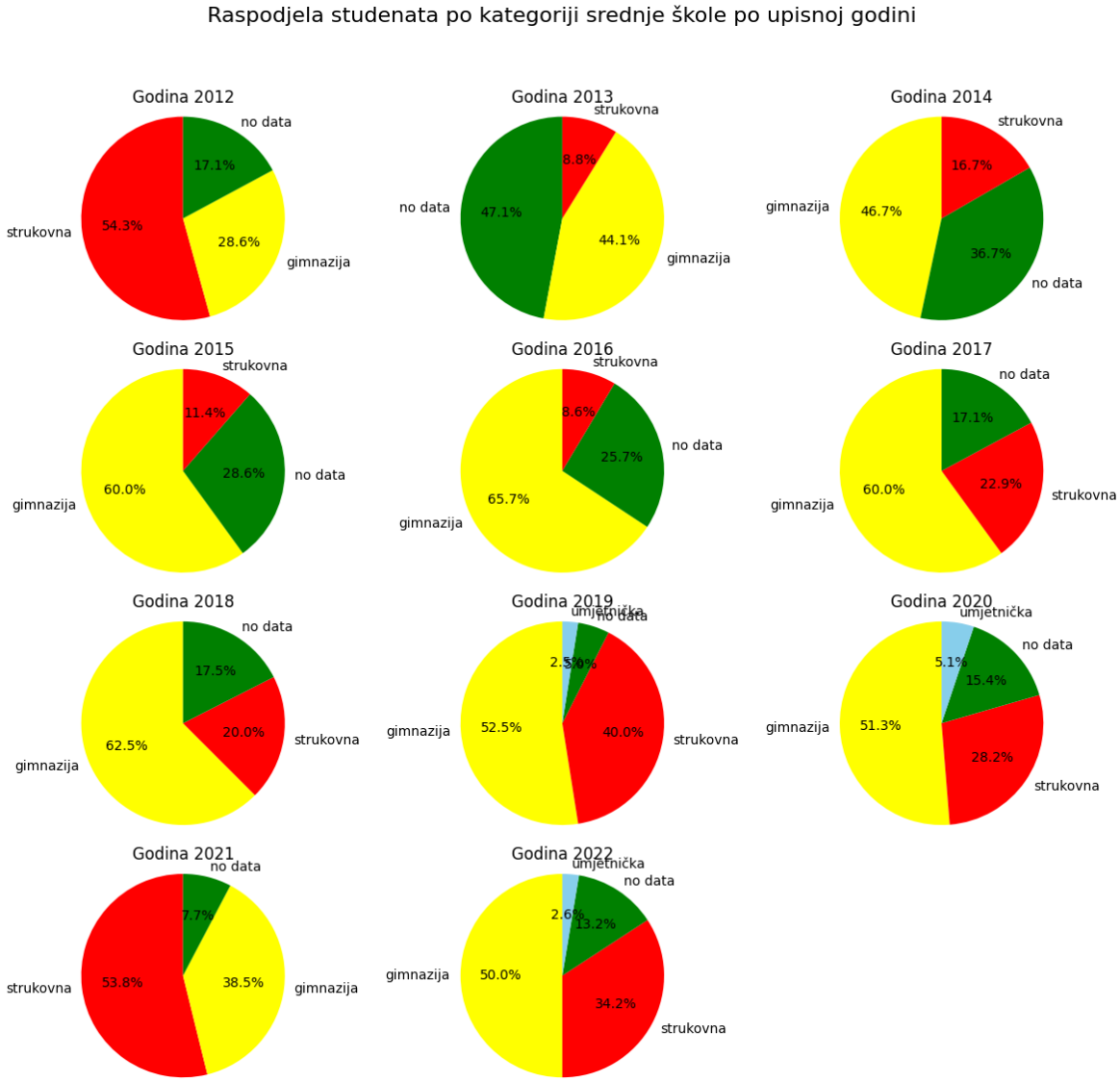
Tablica 4. prikazuje raspodjela studenata po kategorijama srednje škole koje su pohađali:

Tablica 4. Raspodjela studenata po kategorijama srednje škole

Kategorija srednje škole	Broj studenata
Gimnazija	204
Strukovna	111
Umjetnička	4
Bez podatka	81

Najviše studenata dolazi iz gimnazija, zatim iz strukovnih. Kategorija 'Bez podataka' (engl. *no data*) označava studente koji su završili srednju školu izvan RH.

Slika 3. prikazuje skup kružnih dijagrama koji pokazuju distribuciju upisanih studenata po kategoriji završene srednje škole (gimnazija, strukovna, umjetnička) za svaku od godina od 2012. do 2022.:



Slika 3. Raspodjela studenata po kategoriji srednje škole po upisnim godinama

Tijekom godina, studenti koji su završili gimnaziju čine značajan dio upisanih.

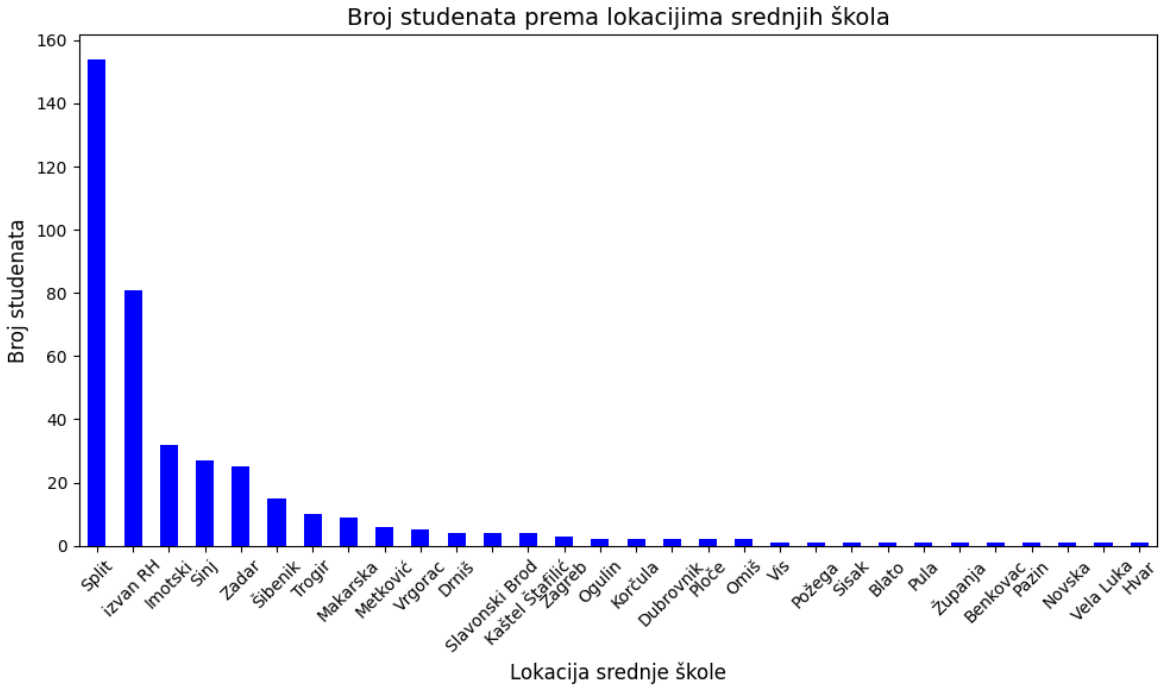
Studenti iz strukovnih škola variraju u udjelu ovisno o godini, s nekim godinama u kojima čine veći dio, poput 2012. (54.3%) i 2021. (53.8%). Međutim, u nekim godinama poput 2016. (8.6%) i 2019. (40%) taj udio je manji.

U određenim godinama postoji značajan udio studenata kod kojih nedostaje informacija o kategoriji srednje škole, kao što je 2013. (47.1%) i 2017. (60%).

Studenti iz umjetničkih škola prisutni su u manjem broju samo u kasnijim godinama poput 2019. (2.5%), 2020. (5.1%), i 2022. (2.6%).

3.5. Lokacija

Graf na slici 4. prikazuje broj studenata prema lokacijama srednjih škola koje su završili. Na x-osi su navedene lokacije srednjih škola, dok y-os prikazuje broj studenata iz svake lokacije:



Slika 4. Broj studenata prema lokacijama srednje škole

Najveći broj studenata dolazi iz Splita, više od 140 studenata. Split je daleko najdominantniji u odnosu na sve ostale lokacije. Izvan RH (studenti koji su pohađali srednje škole izvan Hrvatske) nalazi se na drugom mjestu s nešto više od 80 studenata. Manji gradovi poput Imotskog, Sinja i Zadra također pridonose s manjim brojem studenata (od 20 do 30 studenata).

3.6. Prosjek ocjena

Što se tiče prosjeka ocjena iz srednje škole, među svim studentima za koje postoje podaci, većina njih ima prosjek ocjena 4 (svi u rasponu ocjena 3.5 do 4.5 (neuključeno)). Raspodjela ide ovako:

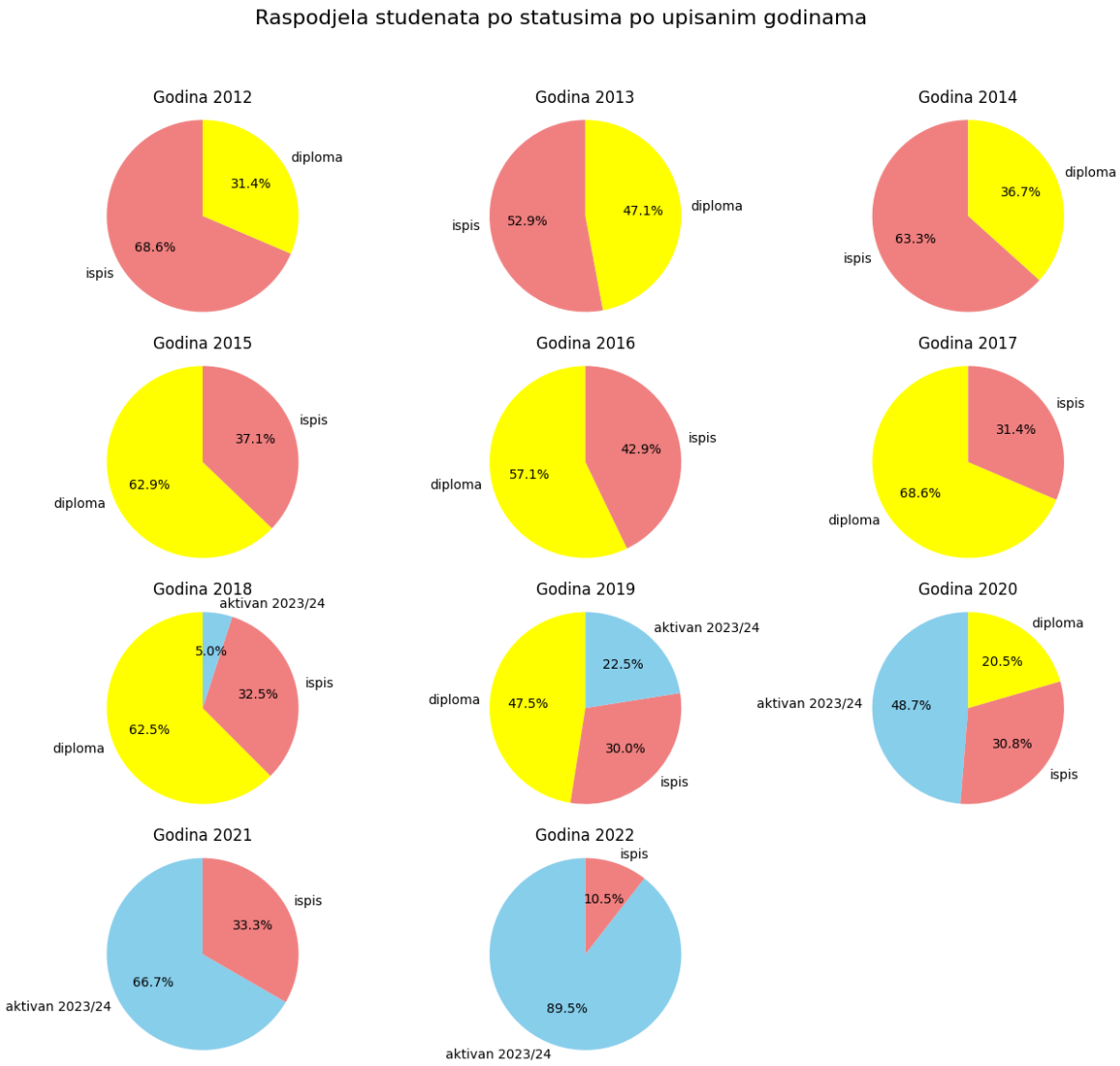
- Ocjena 2: 0.5% studenata
- Ocjena 3: 23.5% studenata
- Ocjena 4: 63.5% studenata
- Ocjena 5: 12.5% studenata

3.7. Završetak studija

Do sada spomenuta analiza je napravljena sa studentima koji su još u aktivnom studiranju i sa studentima koji su završili ili su se ispisali sa studija, njihova podjela među podacima je:

- Završili s diplomom: 156 studenta
- Ispisani sa studija: 154 studenta
- Aktivno studiranje u 2023./2024.: 90 studenata

Slika 5. prikazuje skup kružnih dijagrama koji pokazuju distribuciju studenata po statusu (diploma, ispis, aktivan) za svaku od godina od 2012. do 2022.:



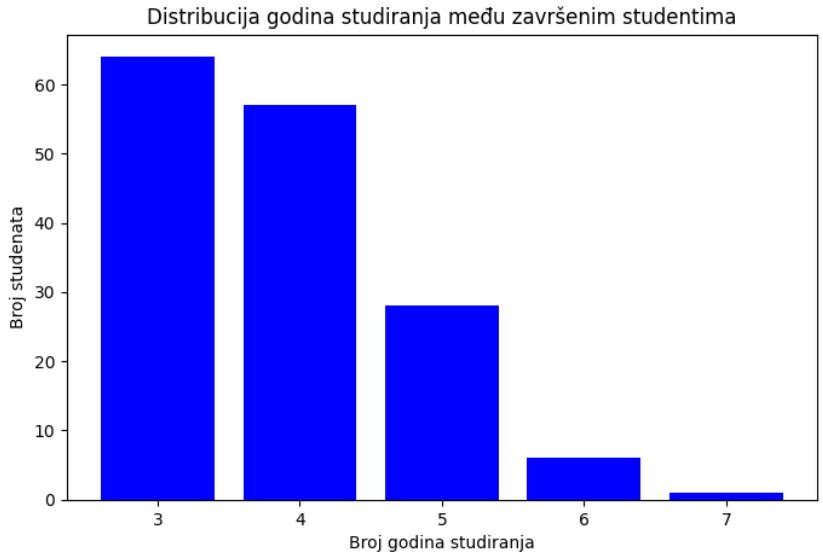
Slika 5. Raspodjela studenata po statusima po upisnim godinama

Ranijih godine od 2012. do 2017. prikazuju studente koji su uglavnom završili studij s diplomom i oni koji su se ispisali sa studija. Nema aktivnih studenata iz tih godina.

U godinama od 2012. do 2014. u većini su se studenti ispisivali nego završavali studij s diplomom.

Nakon 2014. raste broj studenata koji završavaju studij s diplomom.

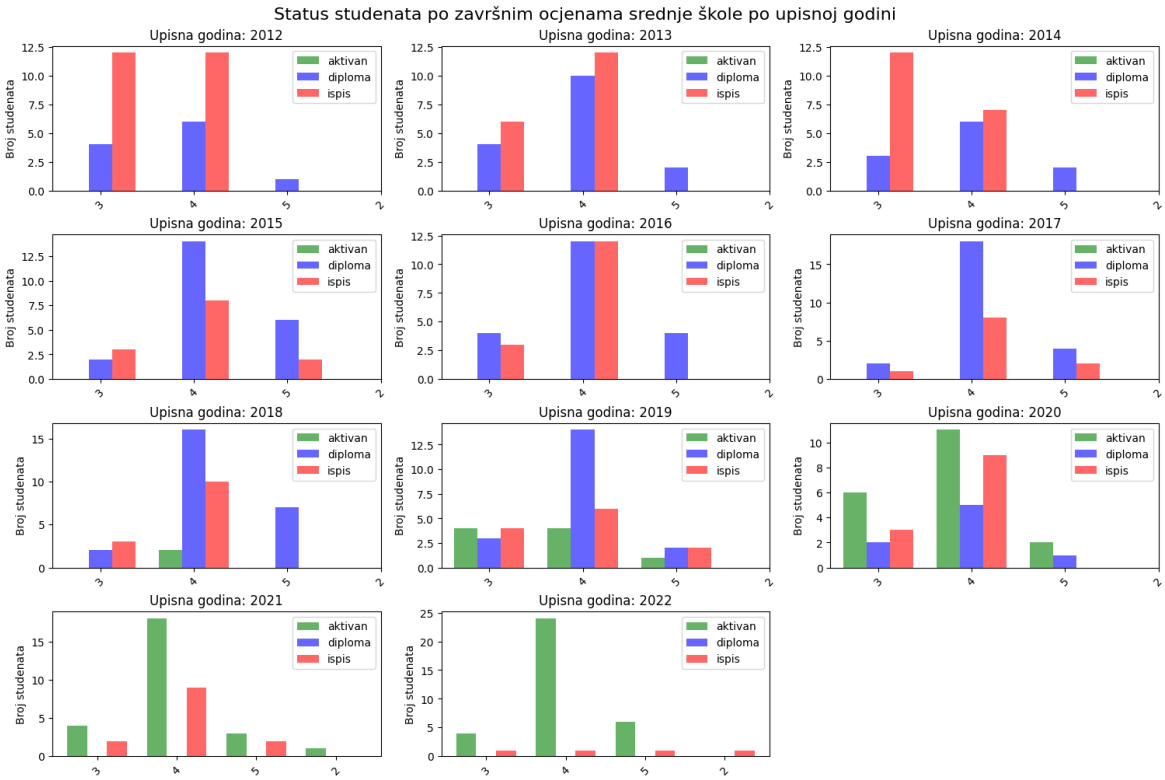
Slika 6. prikazuje distribuciju broja godina studiranja među završenim studentima (broj tih studenata 156). Na x-osi je prikazan broj godina studiranja, dok y-os prikazuje broj studenata. Oko 65 studenata je završilo za 3 godine, dok je nešto manje od 60 studenata završilo za 4 godine. Manji broj studenata je završio za 5 godina (oko 30 studenata). Vrlo mali broj studenata je studirao 6 godina (oko 5 studenata), dok je samo jedan ili dva studenta studiralo 7 godina.



Slika 6. Distribucija studenata po godinama studiranja

3.8. Odnos prosjeka i završetka studija

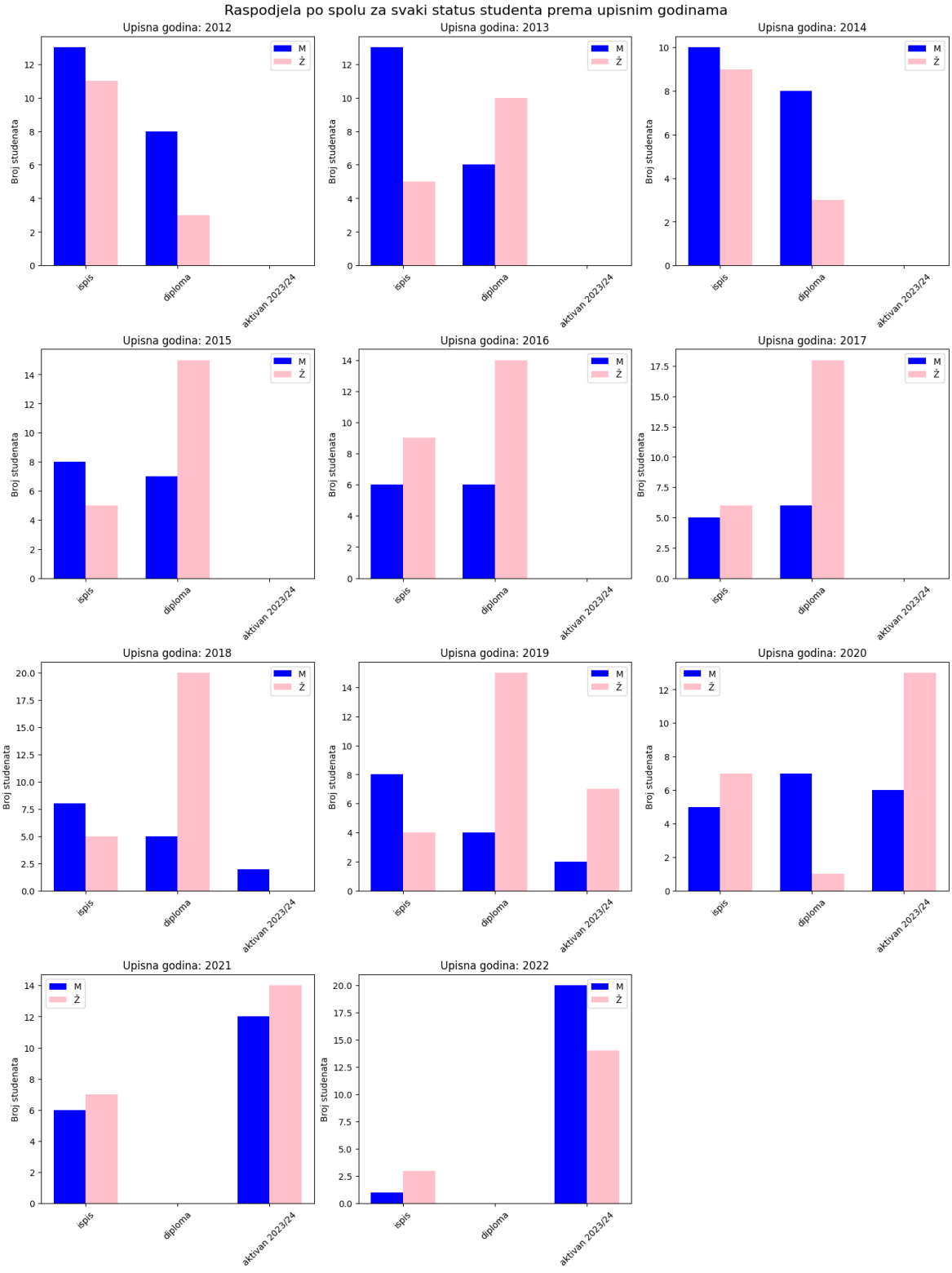
Grafovi na slici 7. prikazuju status studenata na temelju njihovih završnih ocjena iz srednje škole (od 2 do 5) po upisnim godinama od 2012. do 2022. godine. Svaki graf unutar ove slike predstavlja jednu upisnu godinu, a statusi studenata su podijeljeni u tri kategorije aktivan, diploma i ispis. X-os prikazuje završne ocjene iz srednje škole (od 2 do 5), a y-os prikazuje broj studenata za određenu ocjenu. Studenti s nižim završnim ocjenama (ocjene 2 i 3) češće su ispisani dok studenti s višim ocjenama (4 i 5) češće diplomiraju ili ostaju aktivni.



Slika 7. Statusi studenata po završnim ocjenama srednje škole po upisnim godinama

3.9. Odnos spola i završetka studija

Slika 8. prikazuje grafove raspodjele studenata po spolu za različite statuse studenta po upisnim godinama od 2012. do 2022. Od 2012. do 2014. je bio najveći broj ispisa studenata, nešto više muškaraca nego žena. Većina diplomiranih studenata su žene u skoro svim prikazanim upisnim godinama.



Slika 8. Raspodjela po spolovima za svaki status po upisnim godinama

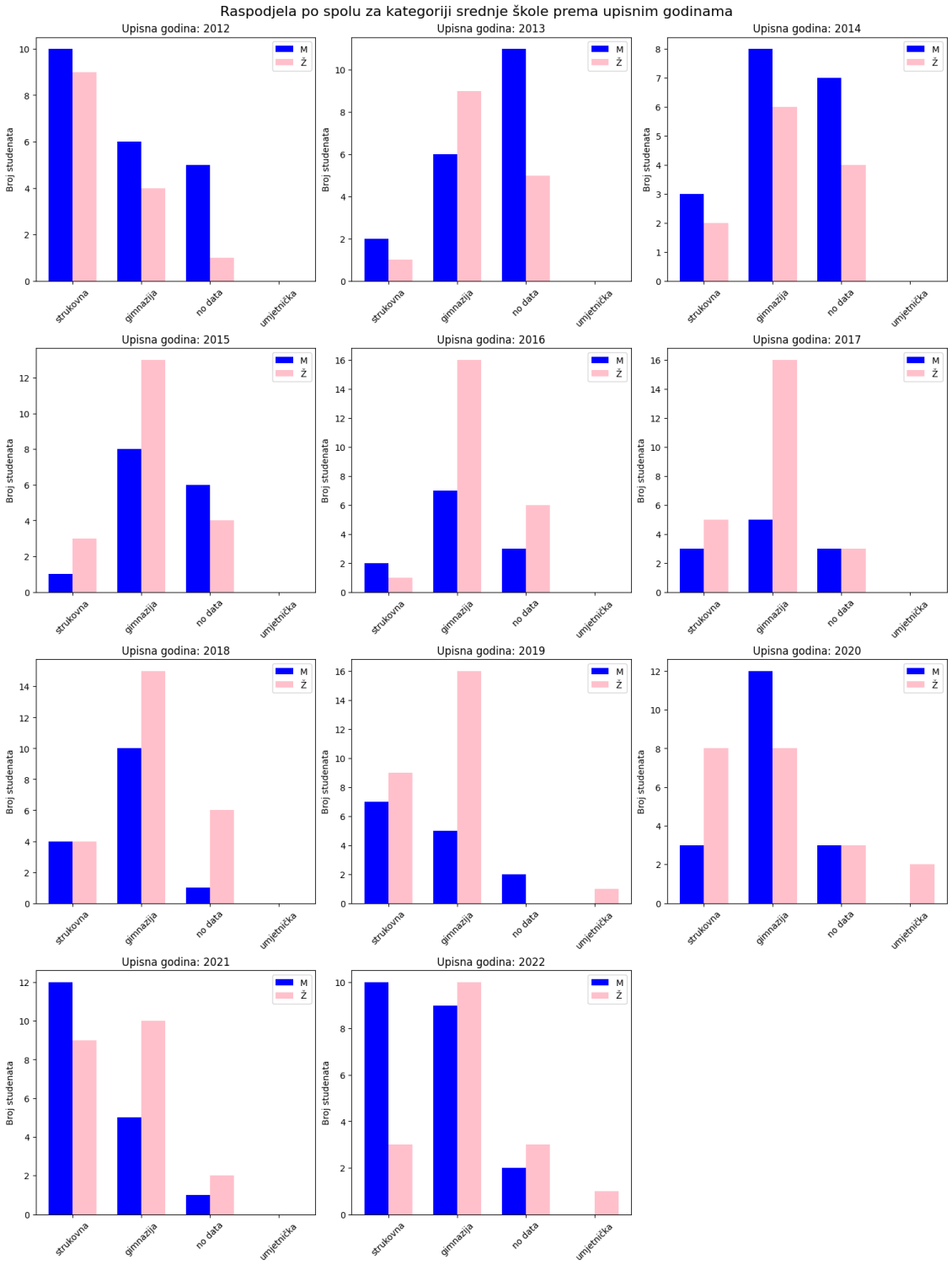
Slika 9. prikazuje raspodjelu spolova po godinama studiranja. Na x-osi su prikazane godine studiranja, a na y-osi broj studenata. Za 3 godine studiranja broj muških i ženskih studenata je prilično ujednačen. Malo više ima ženskih studenata (oko 32), dok muškaraca ima oko 30. Značajno više žena završava studij za 4 godine (oko 40), dok je broj muškaraca znatno

manji (oko 20). Za 5 godina studiranja, postoji razlika, žene dominiraju, sa oko 25 studentica, dok muškaraca ima nešto manje od 15. Vrlo mali broj studenata završava studij nakon 6 godina. Muškarci i žene su ovdje podjednako zastupljeni, ali broj je vrlo mali (oko 2-3 studenta).

Slika 9. Raspodjelu spolova po godinama studiranja

3.10. Odnos spola i završene srednje škole

Grafovi na slici 10. prikazuju podjelu na spolove po kategorijama srednje škole koje su studenti završili po upisnim godinama. Iz strukovnih škola veći je broj studenata, muškog spola nego studentica. U skoro svim godinama, broj žena koje dolaze iz gimnazija je veći u odnosu na muškarce. Iz umjetničkih škola dolazi najmanji broj studenata, vrlo malo, u 2019. godini primjećen je mali broj muških studenata iz umjetničkih škola, što je izuzetak u odnosu na ostale godine. Za kategoriju bez podataka o srednjoj školu vidi se veći broj muških studenata u ranijim godinama, dok je u novijim godinama veći broj žena.

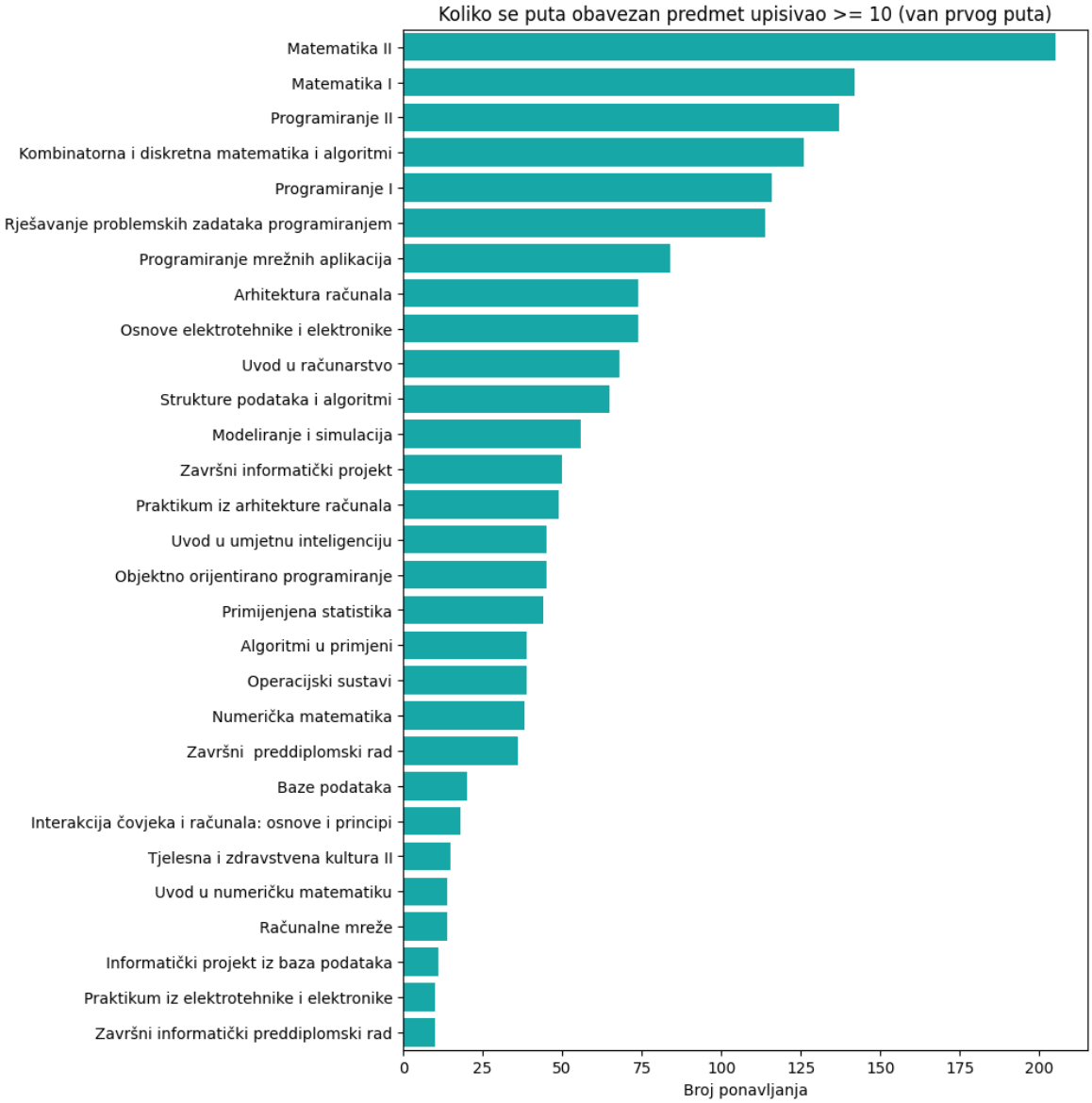


Slika 10. Raspodjela po spolovima po kategoriji srednje škole prema upisnim godinama

3.11.Predmeti

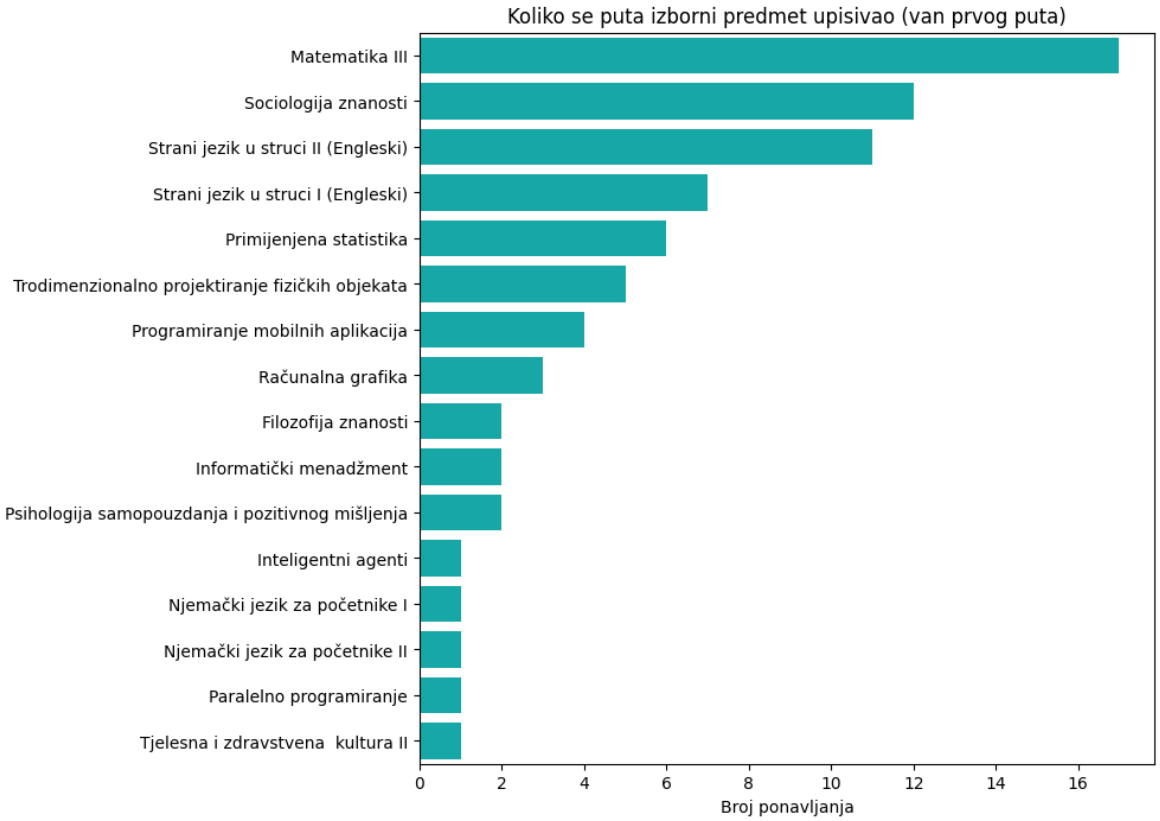
Slika 11. prikazuje koliko se puta određeni obavezni premet upisivao, ne uključujući prvi put kada su studenti upisivali. Prikazani su oni koji su upisani više puta od stane deset i

više studenata. Predmeti su poredani prema broju ponovljenih upisa, a predmeti s najviše ponovljenih upisa su na vrhu. Najviše ponavljani predmeti su Matematika II, Matematika I, Programiranje II, Kombinatorna i diskretna matematika i algoritmi. Predmeti poput Arhitektura računala, Programiranje mrežnih aplikacija, Uvod u računarstvo i Osnove elektrotehnike i elektronike također imaju značajan broj ponavljanja, ali nešto manje u usporedbi s Matematikom i Programiranjem.



Slika 11. Broj ponavljanja obaveznih predmeta

Graf na slici 12. prikazuje koliko se puta određeni izborni predmet ponovno upisivao, ne uključujući prvi upis predmeta. Predmeti su poredani prema broju ponovljenih upisa, a predmeti s najviše ponovljenih upisa su na vrhu. Najviše ponavljani predmeti su Matematika III, Sociologija znanosti i Strani jezik u struci (Engleski).



Slika 12. Broj ponavljanja izbornih predmeta

4. Rezultati modela predviđanja

U kontekstu predviđanja odustajanja studenata, proces uključuje korištenje nekoliko metrika evaluacije, koje služe kao temelj za razumijevanje koliko dobro model može klasificirati ili predvidjeti odustajanje u odnosu na one koji ne odustaju od studiranja. Mjerni podaci kao što su točnost, preciznost, odziv, F-rezultat i AUC obično se koriste za procjenu prediktivne moći različitih modela. Analiza podataka igra ključnu ulogu u pretvaranju sirovih podataka u smislene uvide.

4.1. Odabir značajki

Prilikom izgradnje modela za predikciju odustajanja studenata, odabir značajki (engl. *features*) je ključan korak u procesu. Varijabla `dropout_dummy` predstavlja ciljnu varijablu i korištena je u klasifikaciji, s binarnim ishodom 1 ili 0. Korištena su dva načina odabira.

4.1.1. Prvi skup značajki

Značajke koje su korištene ovom skupu su odabrane nakon analize relevantnih istraživanja na temu predikcije odustajanja od školovanja, kako je opisano u poglavlju *Povezani radovi*. U tim istraživanjima, korištene značajke su detaljno proučene kako bi se identificirale one koje najviše doprinose točnosti predikcijskih modela. Pri odabiru značajki za ovaj skup, uzete su u obzir upravo one varijable za koje su bile dostupne među podacima (podaci opisani u dijelu *O podacima*), dok značajke koje su spomenute u literaturi, ali nisu bile dostupne u podacima, nisu bile uključene. Ovaj pristup omogućio je da se model temelji na provjerenim i relevantnim značajkama, prilagođenima dostupnim resursima, uz održavanje kvalitete predikcija.

Te varijable su:

- `enrollment_year` – godina upisa
- `birth_year` – godina rođenja
- `gender_dummy` – kategorija za spol

- `gymnasiums_dummy` – kategorija za gimnaziju
- `vocational_dummy` – kategorija za strukovnu
- `art_dummy` – kategorija za umjetničku
- `enrollment_age` – upisna dob studenta
- `high_school_average_grade` – prosjek ocjena u srednjoj školi
- `outside_cro_high_school_dummy` – kategorija je li srednja škola izvan RH

4.1.2. Drugi skup značajki

Značajke korištene u drugom skupu su odabrane analizom njihove povezanosti s ishodom odustajanja studenata od studija, što je prikazano kroz matricu korelacije (engl. *correlation matrix*). U ovoj analizi posebna pažnja posvećena je korelacijama svake pojedine značajke s varijablom `dropout_dummy`, koja predstavlja binarni ishod odustajanja (1 – ispisani, 0 – dobivanje diplome). Promatranjem korelacija moglo se zaključiti koje su značajke najrelevantnije za predikciju ishodnih stanja. Značajke koje su pokazale veći stupanj povezanosti s varijablom ispisivanja zadržane su u modelu, dok su one s niskom ili nepostojećom korelacijom isključene kako bi se osigurala bolja učinkovitost modela i smanjio utjecaj potencijalno nebitnih varijabli. Ovaj pristup omogućio je da se iz dostupnih podataka izdvoje one varijable koje najbolje doprinose predikciji ishodnih stanja, čime je poboljšana interpretacija rezultata i točnost predikcija.

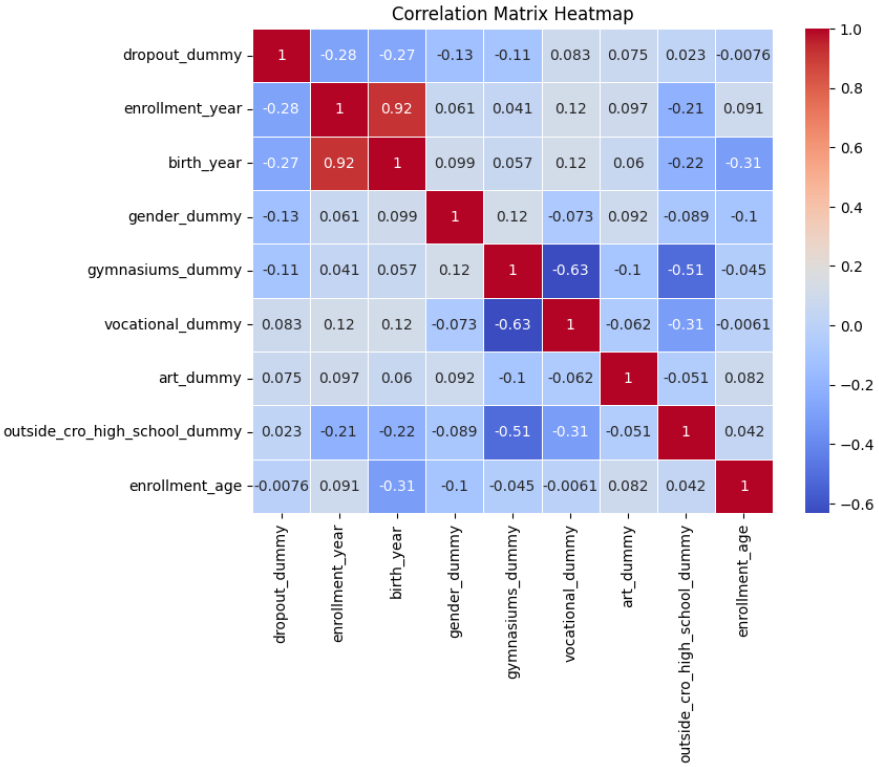
Te varijable su:

- `enrollment_year` – godina upisa
- `birth_year` – godina rođenja
- `gender_dummy` – kategorija za spol
- `gymnasiums_dummy` – kategorija za gimnaziju
- `high_school_average_grade` – prosjek ocjena u srednjoj školi
- `state_matriculation_exam_points` – bodovi s državne mature
- `total_points_for_study` – ukupan broj bodova za upis na studij

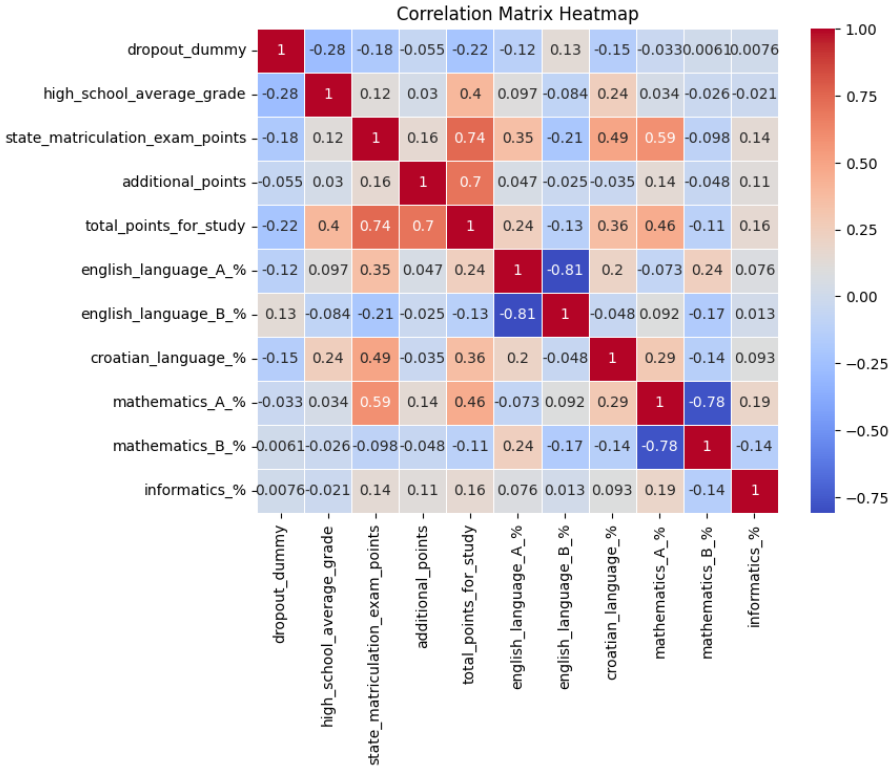
- `english_language_A_%` - postotak iz mature engleskog jezika, A razina
- `english_language_B_%` - postotak iz mature engleskog jezika, B razina
- `croatian_language_%` - postotak iz mature hrvatskog jezika

Slike 13. i 14. prikazuju matrice korelacije koje mjere povezanosti između različitih značajki u skupu podataka. Ključne korelacije sa varijablom `dropout_dummy` su:

- Negativna korelacija s varijablom upisne godine (-0.28) i godinom rođenja (-0.27)
- Negativna korelaciju s prosječnom ocjenom u srednjoj školi (-0.28). Ovo znači da studenti s višim prosjekom u srednjoj školi imaju manju vjerojatnost ispisa.
- Slaba negativna korelaciju s bodovima na državnoj maturi (-0.18), što također može značiti da studenti s boljim rezultatima na maturi imaju manju vjerojatnost ispisa.
- Pozitivna korelaciju s ukupnim bodovima za upis na studij (0.22), što bi značilo da veći broj ukupnih bodova može biti povezan s većom vjerojatnošću ispisa.



Slika 13. Matrica korelacije između općih podataka studenata i ciljane varijable



Slika 14. Matrica korelacije između podataka s državne mature i ciljane varijable

Podaci su podijeljeni na skupove za treniranje i skupove za testiranje u omjeru 75% za treniranje modela i 25% za testiranje, kako bi se procijenila učinkovitost modela na neviđenim podacima. Ovakav pristup osigurao je da se model trenira na najinformativnijim varijablama i testira njegova sposobnost predikcije na temelju relevantnih značajki.

4.2. Rezultati dobiveni nad prvim skupom značajki

Za prvi skup značajki rezultati metrika za modele su prikazani u tablici 5.

Tablica 5. Rezultati metrika za modele s prvim skupom značajki

Model	***Accuracy***	***Precision***	***Recall***	***F-score***	***AUC***
Logistic regression	0.63	0.68	0.60	0.63	0.63
Decision tree	0.60	0.65	0.57	0.61	0.61
Random forest	0.67	0.70	0.67	0.68	0.67
SVM	0.63	0.69	0.57	0.62	0.63
Gradient boosting	0.63	0.68	0.60	0.63	0.63

Logistička regresija

Model je ispravno klasificirao 63% primjera. Kada model predvidi određenu klasu, 68% tih predikcija je točno. Od svih stvarnih pozitivnih primjera, model je uspio prepoznati 60%. F-rezultat pokazuje balans između ta preciznosti i odziva. Model ima slabu

sposobnost razlikovanja između klasa, ali je bolji od slučajnog pogađanja. Matrica konfuzije u tablici 6.:

Tablica 6. Matrica konfuzije za logističku regresiju s prvim skupom značajki

		Stvarna vrijednost	
		0	1
Predviđena vrijednost	0	TP = **24**	FP = **12**
	1	FN = **17**	TN = **25**

Stablo odluke

Točnost modela je nešto niža, s 60% točno klasificiranih primjera. Kada model predvidi pozitivnu klasu, 65% tih predikcija je ispravno. Model prepoznaje 57% stvarnih pozitivnih primjera. F-rezultat ukazuje na nešto slabiji balans između preciznosti i odziva. Model ima slabu sposobnost razlikovanja između klasa, ali je bolji od slučajnog pogađanja. Matrica konfuzije u tablici 7.:

Tablica 7. Matrica konfuzije za stablo odluke s prvim skupom značajki

		Stvarna vrijednost	
		0	1
Predviđena vrijednost	0	TP = **23**	FP = **13**
	1	FN = **18**	TN = **24**

Slučajna šuma

Model ima najvišu točnost (67%). Kada model predvidi određenu klasu, 70% tih predikcija je točno. Model prepoznaje 67% stvarnih pozitivnih primjera. F-rezultat pokazuje dobar balans između preciznosti i odziva. Model ima slabu sposobnost razlikovanja između klasa, ali je bolji od slučajnog pogađanja. Matrica konfuzije u tablici 8.:

Tablica 8. Matrica konfuzije za slučajnu šumu s prvim skupom značajki

		Stvarna vrijednost	
		0	1
Predviđena vrijednost	0	TP = **24**	FP = **12**
	1	FN = **14**	TN = **28**

SVM

Točnost modela je 63%. Preciznost iznosi 69%. Model prepoznaje 57% stvarnih pozitivnih primjera. Sličan F-rezultat kao kod logističke regresije. Model ima slabu sposobnost

razlikovanja između klasa, ali je bolji od slučajnog pogađanja. Matrica konfuzije u tablici 9.:

Tablica 9. Matrica konfuzije za SVM s prvim skupom značajki

		Stvarna vrijednost	
		0	1
Predviđena vrijednost	0	TP = **25**	FP = **11**
	1	FN = **18**	TN = **24**

Gradient boosting

Točnost je jednaka kao kod logističke regresije (63%). Preciznost je također jednaka kao kod logističke regresije (68%). Odziv je jednak (60%). Sličan F-rezultat kao kod logističke regresije i SVM-a. Također slabija sposobnost razlikovanja između klasa, ali je bolji od slučajnog pogađanja. Matrica konfuzije u tablici 10:

Tablica 10. Matrica konfuzije za gradient boosting s prvim skupom značajki

		Stvarna vrijednost	
		0	1
Predviđena vrijednost	0	TP = **24**	FP = **12**
	1	FN = **17**	TN = **25**

4.2.1. Podjela po spolu

Za predikciju podataka muškog spola u tablici 11 logistička regresija pokazuje najbolje rezultate s točnošću od 63%, preciznošću od 75%, odziv od 72% i F-rezultat od 73%, dok je AUC skromnih 0.56. Ovaj model najtočnije predviđa i balansira između prepoznavanja pozitivnih primjera i smanjenja lažnih pozitivnih.

Za ženski spol u tablici 12, modeli pokazuju značajno slabije rezultate nego kod muškaraca. SVM pokazuje najbolje performanse s točnošću od 67%, ali izuzetno visoka preciznost od 100% sugerira da predviđa vrlo malo pozitivnih primjera, dok je odziv 45% relativno nizak. Ovaj rezultat ukazuje na to da model radi izuzetno precizno, ali propušta mnoge pozitivne primjere.

Tablica 11. Rezultati metrika za modele s prvim skupom značajki (spol muški)

Model	***Accuracy***	***Precision***	***Recall***	***F-score***	***AUC***
Logistic regression	0.63	0.75	0.72	0.73	0.56
Decision tree	0.43	0.63	0.48	0.55	0.39
Random forest	0.54	0.71	0.60	0.65	0.50
SVM	0.57	0.69	0.72	0.71	0.46
Gradient boosting	0.57	0.73	0.64	0.68	0.52

Tablica 12. Rezultati metrika za modele s prvim skupom značajki (spol ženski)

Model	***Accuracy***	***Precision***	***Recall***	***F-score***	***AUC***
Logistic regression	0.53	0.50	0.25	0.33	0.52
Decision tree	0.56	0.53	0.40	0.46	0.55
Random forest	0.56	0.56	0.25	0.34	0.54
SVM	0.67	1.00	0.30	0.46	0.65
Gradient boosting	0.67	0.75	0.45	0.56	0.66

4.2.2. Podjela po kategoriji srednje škole

Na podacima za kategoriju *umjetnička* se nije radila predikcija jer postoji mali broj podataka (3 studenta s podacima o završenoj umjetničkoj srednjoj školi). Tako su predikcije modela napravljene na podacima po kategoriji gimnazije i strukovne škole.

Tablice 13. i 14. prikazuju metrike za modele prema kategoriji srednje škole (gimnazija i strukovna škola).

Tablica 13. Rezultati metrika za modele s prvim skupom značajki (kategorija gimnazija)

Model	***Accuracy***	***Precision***	***Recall***	***F-score***	***AUC***
Logistic regression	0.68	0.64	0.53	0.58	0.66
Decision tree	0.65	0.64	0.41	0.50	0.62
Random forest	0.62	0.56	0.53	0.55	0.61
SVM	0.65	0.60	0.53	0.56	0.63
Gradient boosting	0.62	0.55	0.65	0.59	0.63

Tablica 14. Rezultati metrika za modele s prvim skupom značajki (kategorija strukovna)

Model	***Accuracy***	***Precision***	***Recall***	***F-score***	***AUC***
Logistic regression	0.70	0.75	0.86	0.80	0.60
Decision tree	0.70	0.79	0.79	0.79	0.64
Random forest	0.75	0.80	0.86	0.83	0.68
SVM	0.75	0.80	0.86	0.83	0.68
Gradient boosting	0.75	0.80	0.86	0.83	0.68

Kategorija gimnazije: modeli imaju lošije rezultate u ovoj kategoriji. Logistička regresija je najprecizniji model, dok *gradient boosting* ima bolji odziv, što sugerira da različiti modeli prepoznaju razne aspekte podataka.

Kategorija strukovne škole: modeli znatno bolji, s ukupno višim točnostima, F-rezultatima i preciznošću. Strukovne škole imaju dosljedno bolje metrike za sve modele. *Gradient boosting* i SVM ističu se kao najbolji modeli za ovu kategoriju.

4.2.3. Podjela po lokaciji srednje škole

Tablice 15, 16 i 17 prikazuju rezultate prediktivnih modela podijeljenih prema različitim lokacijama (Split, izvan Splita, izvan RH).

Tablica 15. Rezultati metrika za modele s prvim skupom značajki (škole Split)

Model	***Accuracy***	***Precision***	***Recall***	***F-score***	***AUC***
Logistic regression	0.59	0.69	0.56	0.62	0.60
Decision tree	0.70	0.79	0.69	0.73	0.71
Random forest	0.67	0.73	0.69	0.71	0.66
SVM	0.56	0.64	0.56	0.60	0.55
Gradient boosting	0.59	0.69	0.56	0.62	0.60

Tablica 16. Rezultati metrika za modele s prvim skupom značajki (škole van Splita)

Model	***Accuracy***	***Precision***	***Recall***	***F-score***	***AUC***
Logistic regression	0.71	0.75	0.67	0.71	0.71
Decision tree	0.62	0.65	0.61	0.63	0.62
Random forest	0.53	0.55	0.61	0.58	0.52
SVM	0.53	0.55	0.61	0.58	0.52
Gradient boosting	0.50	0.53	0.56	0.54	0.50

Tablica 17. Rezultati metrika za modele s prvim skupom značajki (škole van RH)

Model	***Accuracy***	***Precision***	***Recall***	***F-score***	***AUC***
Logistic regression	0.50	0.33	0.29	0.31	0.46
Decision tree	0.44	0.36	0.57	0.44	0.47
Random forest	0.44	0.36	0.57	0.44	0.47
SVM	0.50	0.40	0.57	0.47	0.51
Gradient boosting	0.50	0.40	0.57	0.47	0.51

Za srednje škole u Splitu, stablo odluke je najbolji model s najvišim vrijednostima metrika.

Za srednje škole izvan Splita, logistička regresija je najbolji model s najvišim vrijednostima metrika.

Za srednje škole izvan RH, performanse modela značajno lošije u usporedbi s prethodnim skupinama. Nijedan model nema točnost veću od 0.50.

4.2.4. Dodavanje novih značajki

Uključivanje broja ponavljanja upisa iz predmeta koji su se analizom pokazali kao najviše ponavljani među studentima (Matematika II, Matematika I, Programiranje II).

Tablica 18. pokazuje rezultate metrika modela nad cijelim skupom podataka. *Gradient boosting* i *random forest* su najuspješniji model u ovoj usporedbi, s visokim točnostima i F-rezultatima, kao i visokim AUC vrijednostima.

Tablica 18. Rezultati metrika za modele s prvim skupom značajki (uz dodatne značajke)

Model	***Accuracy***	***Precision***	***Recall***	***F-score***	***AUC***
Logistic regression	0.69	0.74	0.67	0.70	0.69
Decision tree	0.65	0.69	0.64	0.67	0.65
Random forest	0.71	0.74	0.69	0.72	0.71
SVM	0.68	0.74	0.62	0.68	0.68
Gradient boosting	0.73	0.80	0.67	0.73	0.74

4.3. Rezultati dobiveni nad drugim skupom značajki

Za drugi skup značajki rezultati metrika za modele:

Tablica 19. Rezultati metrika za modele sa drugim skupom značajki

Model	***Accuracy***	***Precision***	***Recall***	***F-score***	***AUC***
Logistic regression	0.63	0.66	0.64	0.65	0.63
Decision tree	0.69	0.72	0.69	0.71	0.69
Random forest	0.71	0.74	0.69	0.72	0.71
SVM	0.69	0.74	0.67	0.70	0.69
Gradient boosting	0.71	0.72	0.74	0.73	0.70

Tablica 19. prikazuje rezultate evaluacije različitih modela strojnog učenja na temelju pet metrika.

Logistička regresija

Točnost modela je niža od ostalih (63%). Kada model predvidi određenu klasu, 66% tih predikcija je točno. Od svih stvarnih pozitivnih primjera, model je uspio prepoznati 64%. Kombinacija preciznosti i odziva pokazuje slabiju izvedbu u odnosu na druge modele. Model ima slabu sposobnost razlikovanja između klasa, ali je bolji od slučajnog pogađanja. Matrica konfuzije u tablici 20.:

Tablica 20. Matrica konfuzije za logističku regresiju sa drugim skupom značajki

		Stvarna vrijednost	
		0	1
Predviđena vrijednost	0	TP = **22**	FP = **14**
	1	FN = **15**	TN = **27**

Stablo odluke

Točnost modela je 69%. Kada model predvidi pozitivnu klasu, 72% tih predikcija je ispravno. Model prepoznaje 69% stvarnih pozitivnih primjera. F-rezultat pokazuje dobar balans između preciznosti i odziva. Model ima slabu sposobnost razlikovanja između klasa, ali je bolji od slučajnog pogađanja. Matrica konfuzije u tablici 21.:

Tablica 21. Matrica konfuzije za stablo odluke sa drugim skupom značajki

		Stvarna vrijednost	
		0	1
Predviđena vrijednost	0	TP = **25**	FP = **11**
	1	FN = **13**	TN = **29**

Slučajna šuma

Model ima visoku točnost, 71% točno klasificiranih primjera. Kada model predvidi određenu klasu, 74% tih predikcija je točno. Model prepoznaje 69% stvarnih pozitivnih primjera. F-rezultat pokazuje dobar balans između preciznosti i odziva. Model ima umjerenu sposobnost razlikovanja klasa (AUC od 0.71). Matrica konfuzije u tablici 22.:

Tablica 22. Matrica konfuzije za slučajnu šumu sa drugim skupom značajki

		Stvarna vrijednost	
		0	1
Predviđena vrijednost	0	TP = **26**	FP = **10**
	1	FN = **13**	TN = **29**

SVM

Točnost od 69%. Preciznost od 74% ukazuje na dobru sposobnost modela da točno predvidi pozitivne primjere. Ovaj model prepoznaje 67% stvarnih pozitivnih primjera. F-rezultat pokazuje dobar balans između preciznosti i odziva. Vrijednost AUC od 0.69 znači da će model u 69% slučajeva pravilno rangirati pozitivan primjer ispred negativnog. Matrica konfuzije u tablici 23.:

Tablica 23. Matrica konfuzije za SVM sa drugim skupom značajki

		Stvarna vrijednost	
		0	1
Predviđena vrijednost	0	TP = **26**	FP = **10**
	1	FN = **14**	TN = **28**

Gradient boosting

Točnost je ista kao kod *random foresta* (71%). Preciznost je 72%. Model prepoznaje 74% stvarnih pozitivnih primjera. F-rezultat pokazuje dobar balans između preciznosti i odziva. Model također ima umjerenu sposobnost razlikovanja klasa. Vrijednost AUC od 0.70 znači da će model u 70% slučajeva pravilno rangirati pozitivan primjer ispred negativnog. Matrica konfuzije u tablici 24.:

Tablica 24. Matrica konfuzije za *gradient boosting* sa drugim skupom značajki

		Stvarna vrijednost	
		0	1
Predviđena vrijednost	0	TP = **24**	FP = **12**
	1	FN = **11**	TN = **31**

4.3.1. Podjela po spolu

U tablicama 25. i 26. su prikazani rezultati različitih modela za dva skupa podataka podijeljena po spolu (muški i ženski).

Tablica 25. Rezultati metrika za modele sa drugim skupom značajki (spol muški)

Model	***Accuracy***	***Precision***	***Recall***	***F-score***	***AUC***
Logistic regression	0.71	0.83	0.76	0.79	0.68
Decision tree	0.60	0.74	0.68	0.71	0.54
Random forest	0.69	0.75	0.84	0.79	0.57
SVM	0.57	0.86	0.48	0.62	0.64
Gradient boosting	0.69	0.77	0.80	0.78	0.60

Tablica 26. Rezultati metrika za modele sa drugim skupom značajki (spol ženski)

Model	***Accuracy***	***Precision***	***Recall***	***F-score***	***AUC***
Logistic regression	0.53	0.50	0.40	0.44	0.53
Decision tree	0.67	0.71	0.50	0.59	0.66
Random forest	0.67	0.67	0.60	0.63	0.67
SVM	0.63	0.67	0.40	0.50	0.61
Gradient boosting	0.70	0.73	0.55	0.63	0.69

Logistička regresija je znatno bolji za muški spol, s mnogo višim rezultatima u svim metrikama, dok je za ženski spol učinak prilično loš. *Gradient boosting* se pokazuje

konzistentno dobrim za oba spola, ali za ženski spol postiže bolje rezultate u gotovo svim metrikama.

4.3.2. Podjela po kategoriji srednje škole

Na podacima za kategoriju *umjetnička* se nije radila predikcija jer postoji mali broj podataka (3 studenta s podacima o završenoj umjetničkoj srednjoj školi). Tako su predikcije modela napravljene na podacima po kategoriji gimnazije i strukovne škole.

Dvije tablice 27. i 28. prikazuju rezultate modela na dva različita skupa podataka – kategorija gimnazija i kategorija strukovna škola.

Tablica 27. Rezultati metrika za modele sa drugim skupom značajki (kategorija gimnazija)

Model	***Accuracy***	***Precision***	***Recall***	***F-score***	***AUC***
Logistic regression	0.60	0.53	0.47	0.50	0.58
Decision tree	0.57	0.50	0.47	0.48	0.56
Random forest	0.72	0.71	0.59	0.65	0.71
SVM	0.53	0.47	0.82	0.60	0.56
Gradient boosting	0.65	0.59	0.59	0.59	0.64

Tablica 28. Rezultati metrika za modele sa drugim skupom značajki (kategorija strukovna)

Model	***Accuracy***	***Precision***	***Recall***	***F-score***	***AUC***
Logistic regression	0.70	0.79	0.79	0.79	0.64
Decision tree	0.55	0.69	0.64	0.67	0.49
Random forest	0.75	0.80	0.86	0.83	0.68
SVM	0.70	0.79	0.79	0.79	0.64
Gradient boosting	0.70	0.83	0.71	0.77	0.69

Random forest dominira u oba skupa podataka, s najboljim rezultatima u točnosti, preciznosti i F-rezultatom. Međutim, za strukovne škole, performanse su još bolje nego za gimnazije, s višom točnošću (0.75 prema 0.72). Logistička regresija pokazuje značajno bolji učinak za strukovne škole, s velikim skokom u preciznosti i F-rezultatom u usporedbi s gimnazijama.

4.3.3. Podjela po lokaciji srednje škole

Na temelju prikazanih rezultata u tablicama 29, 30 i 31, može se vidjeti kako se modeli ponašaju za podatke razdvojene prema lokacijama (srednje škole u Splitu, srednje škole izvan Splita i srednje škole izvan Hrvatske).

Tablica 29. Rezultati metrika za modele sa drugim skupom značajki (škole Split)

Model	*Accuracy*	*Precision*	*Recall*	*F-score*	*AUC*
Logistic regression	0.70	0.79	0.69	0.73	0.71
Decision tree	0.52	0.60	0.56	0.58	0.51
Random forest	0.70	0.79	0.69	0.73	0.71
SVM	0.48	0.57	0.50	0.53	0.48
Gradient boosting	0.67	0.71	0.75	0.73	0.65

Tablica 30. Rezultati metrika za modele sa drugim skupom značajki (škole izvan Splita)

Model	*Accuracy*	*Precision*	*Recall*	*F-score*	*AUC*
Logistic regression	0.71	0.75	0.67	0.71	0.71
Decision tree	0.62	0.67	0.56	0.61	0.62
Random forest	0.68	0.73	0.61	0.67	0.68
SVM	0.62	0.67	0.56	0.61	0.62
Gradient boosting	0.62	0.65	0.61	0.63	0.62

Tablica 31. Rezultati metrika za modele sa drugim skupom značajki (škole van RH)

Model	*Accuracy*	*Precision*	*Recall*	*F-score*	*AUC*
Logistic regression	0.56	0.44	0.57	0.50	0.56
Decision tree	0.78	0.67	0.86	0.75	0.79
Random forest	0.50	0.40	0.57	0.47	0.51
SVM	0.56	0.45	0.71	0.56	0.58
Gradient boosting	0.44	0.36	0.57	0.44	0.47

Lokacija Split: Logistička regresija i *random forest* imaju slične rezultate s točnošću od 0.70. Ova dva modela pokazuju najstabilnije performanse za podatke iz škola u Splitu.

Lokacija izvan Splita: Ovdje logistička regresija ima najveću točnost od 0.71, uz AUC od 0.71, pokazuje da ovaj model dobro funkcionira u predikciji na podacima iz škola izvan Splita. *Random forest* također pokazuje dobre rezultate, s točnošću od 0.68.

Lokacija izvan RH: Stablo odluke ima najbolju točnost od 0.78 i AUC od 0.79.

4.3.4. Dodavanje novih značajki

Uključivanje broja ponavljanja upisa iz predmeta koji su se analizom pokazali kao najviše ponavljani među studentima (Matematika II, Matematika I, Programiranje II).

Tablica 32. pokazuje rezultate metrika modela nad cijelim skupom podataka. Logistička regresija bilježi značajan skok u performansama, s točnosti od 73% i izrazito visokom preciznošću (80%). *Random forest* zadržava snažan balans metrika, s točnošću od 73%, visokom preciznošću (76%) i boljim odzivom (74%) nego prije. To ga čini jednim od najboljih modela. *Gradient boosting* je sada najbolji model u smislu sveukupnih

performansi, s točnošću od 77%, najvišim odzivom (81%), te uravnoteženim F-rezultatom (79%) i AUC-om (77%).

Tablica 32. Rezultati metrika za modele sa drugim skupom značajki (uz dodatne značajke)

Model	***Accuracy***	***Precision***	***Recall***	***F-score***	***AUC***
Logistic regression	0.73	0.80	0.67	0.73	0.74
Decision tree	0.68	0.71	0.69	0.70	0.68
Random forest	0.73	0.76	0.74	0.75	0.73
SVM	0.72	0.81	0.62	0.70	0.73
Gradient boosting	0.77	0.77	0.81	0.79	0.77

5. Rasprava

Broj učenika koji se upisuju u dobi od 20 i više godina značajno opada, s vrlo malim brojem učenika u dobnim skupinama od 22 do 26 godina. Ovaj obrazac sugerira da se većina učenika upisuje odmah nakon srednje škole (oko 18. i 19. godine), što je tipično za preddiplomske programe.

Podaci otkrivaju jasnu promjenu u rodnoj distribuciji tijekom godina. Dok su muškarci bili većina u ranijim godinama (2012. - 2014.), žene su počele dominirati počevši od 2015. i ostale su veća skupina u većini sljedećih godina. Ova promjena može ukazivati na promjene u studentskim preferencijama ili institucionalnim trendovima koji favoriziraju upis žena.

Može se primijetiti varijabilnost kroz godine, gdje se udio studenata iz različitih kategorija srednjih škola mijenja. Na primjer, nakon 2016. značajno raste udio gimnazijalaca, dok u ranijim godinama (poput 2012. i 2013.) strukovne škole dominiraju. Raspodjela studenata prema tipu srednje škole mijenjala se kroz godine. Gimnazijalci čine najveći dio upisanih u većini godina, dok strukovne škole imaju veći udio u ranijim godinama i ponovno u 2021. i 2022.

Split kao najveći grad u regiji i grad u kojem je fakultet odakle su podaci, prirodno ima najviše učenika koji upisuju fakultet, dok su drugi manji gradovi ili općine znatno slabije zastupljeni. Studenti izvan Hrvatske također čine značajan udio, što može upućivati na internacionalnu komponentu ili dijasporu. Ostale lokacije, posebice manji gradovi, imaju relativno nisku zastupljenost u broju studenata.

Studenti s višim završnim ocjenama (4 i 5) imaju veću vjerojatnost da će diplomirati ili ostati aktivni. Ovo sugerira da studenti koji su bili uspješni u srednjoj školi imaju tendenciju pokazati sličan uspjeh na fakultetu. Posebno se to vidi u godinama kao što su 2018., 2019. i 2020., gdje studenti s ocjenom 4 i 5 imaju veću zastupljenost među diplomiranim i aktivnim studentima. U većini prikazanih godina, studenti sa završnim ocjenama 2 ili 3 češće se ispisuju. Ovaj trend je konstantan u svim upisnim godinama, što sugerira da studenti koji su imali poteškoća u srednjoj školi možda imaju izazove i na fakultetu. Na primjer, u godinama 2013. i 2014., najveći broj ispisanih studenata dolazi iz grupe s ocjenom 3. Studenti s ocjenom 4 često su podjednako raspoređeni između aktivnih,

diplomiranih i ispisanih statusa. Ovo može značiti da ova skupina ima različite pristupe učenju i prilagodbi na visokoškolsko obrazovanje. Studenti iz ranijih upisnih godina, kao što su 2012. i 2013., pokazuju veću tendenciju ispisivanja, posebno oni s nižim ocjenama. S vremenom, čini se da se smanjuje broj ispisanih, posebno među studentima s boljim ocjenama, što može ukazivati na bolju pripremu novih generacija za fakultetske izazove.

Većina diplomiranih studenata su žene u skoro svim prikazanim upisnim godinama, što sugerira da žene češće završavaju studij u odnosu na muškarce. Broj ispisanih studenata varira. Broj aktivnih studenata značajno raste u posljednjim upisnim godinama (2021. i 2022.), posebno kod žena. To ukazuje da su novije generacije još uvijek studiraju. Starije generacije (2012. – 2016.) nemaju aktivnih studenata i veći broj diplomiranih, što je očekivano s obzirom na proteklo vrijeme od upisa. Novije generacije (2017. – 2022.) imaju veći broj aktivnih studenata, a manji broj diplomiranih, jer ti studenti još uvijek studiraju. Općenito, žene češće završavaju studij, dok muškarci češće ispadaju ili ostaju aktivni na duži rok.

Veliki broj studenata završava za 3 ili 4 godine, što može značiti da su studenti uglavnom motivirani i efikasni u ispunjavanju svojih akademskih obaveza. To može ukazivati na dobru organizaciju studija, dobar pristup učenju i radu studenata. Studenti koji studiraju 5 godina ili duže predstavljaju manju grupu. Ovi studenti mogu imati različite razloge za produženje studija kao što su moguće poteškoće s nekim ispitima ili predmetima, paralelno zaposlenje što bi moglo usporiti njihov napredak, također zdravstveni, osobni ili obiteljski razlozi mogu uticati na produženje studija. Ako većina studenata završava studij na vrijeme, to može značiti da su studenti dobro pripremljeni i fakultet pruža prikladne uvjete za uspjeh. Općenito, ovi podaci sugeriraju da većina studenata uspijeva završiti studij u planiranom roku, dok manji broj produžuje studij zbog različitih individualnih ili nekakvih drugih faktora.

Broj studenata koji završavaju studij za 3 godine je sličan za oba spola, što upućuje da su muškarci i žene podjednako uspješni u završavanju studija u standardnom roku. Dominacija žena u kategorijama 4 i 5 godina studiranja upućuje na to da veći broj studentica može produžiti studij izvan standardnog roka od 3 godine. Ovo može biti posljedica različitih faktora, kao što su dodatne obaveze, kao što su prakse, rad tokom studija ili druge aktivnosti/izazovi koje utječu na produženje studija. Većina studenata uspješno završava studij unutar 5 godina, što ukazuje na visoku efikasnost sustava obrazovanja. Muškarci češće završavaju studij u kraćem vremenskom roku (za 3 godine),

dok žene pokazuju veću sklonost prema produžavanju studija (4 ili 5 godina). Ovo može ukazivati na razlike u načinu na koji se oba spola nose s izazovima tokom studija. Muškarci mogu biti usmjereni na što brže završavanje studija. Žene mogu biti podložnije temeljitijem pristupu učenju ili fokusirane na dodatne akademske obaveze (prakse, projekti, istraživanja).

Muški studenti su zastupljeniji u strukovnim školama kroz sve upisne godine, što je očekivano s obzirom na veću orijentaciju muškaraca prema tehničkim i praktičnim obrazovanjima. Žene su manje zastupljene u ovoj kategoriji, ali njihov broj ostaje stabilan kroz sve upisne godine. U skoro svim upisnim godinama, broj žena koje dolaze iz gimnazija je veći u odnosu na muškarce. Ovaj trend je najizraženiji u upisnim godinama od 2015. do 2019. godine. Muški studenti su manje zastupljeni, ali prisutni, i njihov broj varira iz godine u godinu. Može se reći da većina studenata, posebno žena, dolazi iz gimnazija, što ukazuje na široko opće obrazovanje i spremnost za dalje studiranje. Muški studenti su zastupljeniji među onima koji dolaze iz tehničkih i strukovnih škola, što može odražavati njihove interese i tradicionalne obrazovne smjerove. Umjetničke škole su najmanje zastupljene, vjerojatno zbog specifičnih interesa koja nisu usmjerena prema studiju.

Obavezni predmeti povezani s matematikom i programiranjem imaju značajno veći broj ponovljenih upisa. To može sugerirati da studenti te predmete smatraju težima ili izazovnijima za prolazak. Visok broj ponavljanja tih predmeta može ukazivati na to da studenti imaju problema s razumijevanjem gradiva, što može sugerirati potrebu za preispitivanja kurikuluma, uvođenja dodatnih predavanja, kao i na potencijalne prilagodbe u metodama poučavanja. Visok broj ponavljanja može ukazivati na to da studenti nisu adekvatno pripremljeni za težinu gradiva kada dolaze s prethodnih obrazovnih razina. Ako su ispiti previše teški, to može pridonijeti visokom broju ponavljanja. Također, nedostatak dodatnih materijala za učenje, nedovoljno konzultacija s profesorima može utjecati na prolaznost. Predmeti koji se ponavljaju u velikom broju možda zahtijevaju previše vremena i resursa od studenata u odnosu na druge obaveze koje imaju tijekom studija. Studenti bi mogli biti preopterećeni brojem sati potrebnim za pripremu, osobito ako moraju balansirati više teških predmeta istovremeno. Metode predavanja i nastavni planovi za ove predmete možda nisu optimalno prilagođeni studentima. Moguće je da postoje područja gdje studenti redovito zapinju, a koja nisu dovoljno pokrivena na predavanjima ili vježbama. Ako je broj studenata koji moraju ponoviti predmet jako visok, institucije bi

mogle razmisliti o promjeni pristupa učenju, na primjer, uvođenjem više praktičnih primjera ili drugačijim stilom ocjenjivanja.

Što se tiče izbornih predmeta, Matematika III na vrhu s najvišim brojem ponavljanja vjerojatno ukazuje na to da studenti, iako je izborni predmet, imaju poteškoća s naprednijim matematičkim konceptima. Također, matematika kao predmet često izaziva strah ili stres kod studenata. Predmeti poput stranih jezika i sociologije također predstavljaju izazov, možda zbog razlike u vještinama koje su potrebne za tehničke i humanističke discipline.

Za prvi skup značajki *random forest* je najuspješniji model. *Random forest* postiže najbolji ukupni balans između svih metrika. Logistička regresija, SVM i *gradient boosting* imaju vrlo slične rezultate, što ukazuje na to da su ovi modeli jednako učinkoviti u ovom specifičnom slučaju, ali zaostaju za *random forestom*. Rezultati pokazuju da su modeli puno uspješniji u predikciji za studente strukovnih škola nego za gimnazijalce. Za strukovne škole, modeli kao što su *gradient boosting* i SVM pokazuju najbolju ukupnu učinkovitost. Kod gimnazija, modeli su manje precizni, a logistička regresija ima najbolje rezultate u toj kategoriji.

Za drugi skup značajki *random forest* i *gradient boosting* su najbolji modeli u ovoj usporedbi, s točnošću od 71% te dobrim F-rezultatima. *Gradient boosting* ima najviši odziv (74%), što znači da najbolje prepoznaje stvarne pozitivne primjere. Također ima vrlo uravnotežene rezultate između preciznosti i odziva. *Random forest* se ističe po visokoj preciznosti i AUC vrijednosti, pa je dobar izbor kad je važno minimizirati lažno pozitivne rezultate, ali ima nešto niži odziv (69%) u odnosu na *gradient boosting*. Rezultati metrika između spolova se razlikuju zbog različitih karakteristika podataka te mogućih razlika u brojnosti ili kvaliteti podataka za svaki spol. Također, modeli poput logističke regresije ili SVM bolje rade u jednostavnijim slučajevima, dok *random forest* i *gradient boosting* postižu bolje rezultate na složenijim podacima, što objašnjava varijacije u metrikama između muških i ženskih skupina.

Nedostatak podataka o srednjim školama u podacima izvan RH vrlo vjerojatno ima značajan utjecaj na rezultate modela, smanjujući njihovu točnost i sposobnost pravilne klasifikacije.

Što se tiče podmodela, kategorija srednje škole se pojavljuje kao najbolji prediktor odustajanja, s dobrim rezultatima za studente iz strukovnih škola. *Random forest* i *gradient*

boosting su najbolji modeli za ovu vrstu predikcije, pri čemu bi *random forest* bio idealan za minimizaciju lažno pozitivnih rezultata, dok bi *gradient boosting* bio bolji za maksimalizaciju prepoznavanja stvarnih odustajanja.

Također, dodavanje značajki koje uključuju broj upisa na predmete koji se najviše ponavljaju među studentima (Matematika II, Matematika I, Programiranje II), u drugi skup značajki, pokazuje značajna poboljšanja u učinkovitosti modela. Logistička regresija i *random forest* ostvarili su jasna poboljšanja u preciznosti, točnosti i F-rezultatu. *Gradient boosting* postao je najbolji model, s najvišom točnošću i odzivom, što ga čini idealnim za prepoznavanje stvarnih slučajeva odustajanja. Broj upisa na predmete koji se često ponavljaju kao ulazna značajka pokazuje se kao važna i korisna informacija za modele, jer je dovela do značajnih poboljšanja u performansama modela.

Random forest i *gradient boosting* sada se ističu kao najbolji modeli, s uravnoteženim performansama kroz sve metrike. *Random forest* je posebno koristan kada je preciznost ključna (minimiziranje lažnih pozitivnih rezultata), dok je *gradient boosting* bolji kad je ključno uhvatiti što više pozitivnih primjera (odustajanja), što je važno za ranu intervenciju.

Na temelju rezultata, *gradient boosting* je sada najbolji model za predikciju odustajanja studenata jer ima najbolji balans između preciznosti, odziva i F-rezultata. Također, *Random forest* ostaje izvrsna opcija, pogotovo kad je važno imati visoku preciznost. Ako je cilj prepoznati što više studenata koji će vjerojatno odustati (visok odziv), *gradient boosting* je najbolji izbor.

Analizom rezultata modela utvrđeno je da se drugi skup značajki (uz podatke o broj upisa na predmete koji se najviše ponavljaju među studentima) pokazao boljim nego prvi skup značajki. Podaci kao što su bodovi s državne mature, ukupni bodovi za upis i postoci uspješnosti u engleskom i hrvatskom jeziku na maturi, značajno su poboljšali performanse modela, povećavajući preciznost, točnost i F-rezultata u usporedbi s prvim skupom značajki. Ovi rezultati sugeriraju da su detaljniji podaci o uspješnosti studenata na ključnim predmetima te rezultatima mature bolji prediktori akademskog uspjeha i odustajanja od studija u usporedbi s općim demografskim informacijama.

Zaključak

Predikcija odustajanja studenata od studija, kao jedan od ključnih izazova u obrazovnim institucijama, pokazala se kao vrijedan alat za identificiranje rizičnih skupina studenata i razumijevanje obrazovnih obrazaca. Ovaj rad pružio je uvid u različite čimbenike koji utječu na uspjeh ili neuspjeh studenata u visokoškolskoj instituciji. Na temelju analize podataka, uočeno je kako dob prilikom upisa, spol, vrsta srednje škole te ocjene iz srednje škole imaju značajan utjecaj na uspjeh u visokom obrazovanju. Posebno je zanimljivo uočiti da žene češće završavaju studij, dok muškarci pokazuju veći rizik od odustajanja. Osim toga, studenti s višim završnim ocjenama iz srednje škole pokazuju veću vjerojatnost uspjeha na fakultetu.

Što se tiče izvedbe modela, *random forest* i *gradient boosting* su pokazali najvišu razinu uspješnosti u predviđanju, što ukazuje na njihovu sposobnost da se nose sa složenim obrazovnim podacima. Posebno je važno napomenuti da su ovi modeli bili uspješniji u predikciji za studente iz strukovnih škola, dok su modeli imali slabiju preciznost u predikciji za gimnazijalce. To sugerira da različite grupe studenata zahtijevaju različite pristupe analizi i intervenciji.

Međutim, nedostatak podataka izvan Hrvatske, posebno u pogledu srednjih škola, predstavljao je značajan izazov. Ovaj nedostatak mogao je smanjiti točnost modela u tim slučajevima, ukazujući na potrebu za boljim prikupljanjem i organiziranjem podataka kako bi se poboljšala točnost predikcija u budućnosti.

Općenito, rezultati ovog rada mogu poslužiti kao polazište za obrazovne institucije u razvijanju strategija za smanjenje odustajanja studenata, kroz ciljane intervencije, poboljšanja nastavnih planova i individualiziranu podršku rizičnim skupinama.

Literatura

[1] HAN, J., KAMBER, M., & PEI, J. (2012). *Data Mining: Concepts and. Techniques*, Waltham: Morgan Kaufmann Publishers

[2] BIENKOWSKI, M., FENG, M., & MEANS, B. (2012). *Enhancing Teaching and Learning through Educational Data Mining and Learning Analytics: An Issue Brief*. Office of Educational Technology, US Department of Education.

[3] ROMERO, C., ESPEJO, P. G., ZAFRA, A., ROMERO, J. R., & VENTURA, S. (2013). *Web usage mining for predicting final marks of students that use Moodle courses.* Computer Applications in Engineering Education, 21(1), 135-146.

[4] BAKER, R. S., & YACEF, K. (2009). *The state of educational data mining in 2009: A review and future visions.* JEDM - Journal of Educational Data Mining, 1(1), 3-17.

[5] BEAN, J. P., & METZNER, B. S. (1985). *A Conceptual Model of Nontraditional Undergraduate Student Attrition*

[6] DELOGU, MARCO & LAGRAVINESE, RAFFAELE & PAOLINI, DIMITRI & RESCE, GIULIANO, 2024. *Predicting dropout from higher education: Evidence from Italy*, Economic Modelling, Elsevier, vol. 130(C).

[7] WAGNER K., VOLKENING H., BASYIGIT S., MERCERON A., SAUER P. AND PINKWART N. (2023). *Which Approach Best Predicts Dropouts in Higher Education?*

[8] AGRUSTI F., MEZZINI M., BONAVOLONTÀ G., (2020), *Deep learning approach for predicting university dropout: a case study at Roma Tre University*. Journal of E-Learning and Knowledge Society, 16(1), 44-54

[9] SONG, Z.; SUNG, S.-H.; PARK, D.-M.; PARK, B.-K. *All-Year Dropout Prediction Modeling and Analysis for University Students.* Appl. Sci. 2023, 13, 1143.

[10] NUÑEZ-NARANJO, ARACELLY & AYALA CHAUVIN, MANUEL & RIBA SANMARTÍ, GENÍS. (2021). *Prediction of University Dropout Using Machine Learning*.

[11] PÉREZ, BORIS & CASTELLANOS, CAMILO & CORREAL, DARIO. (2018). *Predicting Student Drop-Out Rates Using Data Mining Techniques: A Case Study: First IEEE Colombian Conference*, ColCACI 2018, Medellín, Colombia, May 16-18, 2018, Revised Selected Papers.

[12] https://gov.hr/hr/vrste-srednjih-skola/1028?lang=hr

[13] JAMES, G., WITTEN, D., HASTIE, T., TIBSHIRANI, R., (2014). *An Introduction to Statistical Learning: with Applications in R*, Springer.

[14] CHRISTOPHER M. BISHOP, (2006). *Pattern Recognition and Machine Learning*, Springer.

[15] https://scikit-learn.org/stable/common_pitfalls.html#randomness

[16] HASTIE T., TIBSHIRANI R., FRIEDMAN J., (2009). *The Elements of Statistical Learning*, Springer.

[17] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[18] ABE S., (2010). *Support Vector Machines for Pattern Classification*, Springer.

[19] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC

[20] GUIDO S., MÜLLER A., (2016). *Introduction to Machine Learning with Python*

[21] BARIČEVIĆ M., (2022). *Klasifikacija zvuka primjenom prijenosa znanja*, Sveučilište u Rijeci, diplomski rad

[22] https://klu.ai/glossary/accuracy-precision-recall-f1

[23] FAWCETT, T. (2006). *An introduction to ROC analysis*, Pattern Recognition Letters, 27(8), 861-874

[24] https://code.visualstudio.com/

[25] https://pandas.pydata.org/docs/

[26] https://numpy.org/doc/stable/

[27] https://matplotlib.org/

[28] https://seaborn.pydata.org/

[29] https://docs.python.org/3/library/datetime.html

[30] https://scikit-learn.org/stable/

Sažetak

Ovaj rad bavi se predviđanjem odustajanja studenata od studija pomoću različitih modela strojnog učenja. U analizu su uključeni podaci poput godine upisa, godine rođenja, spola, kategorija srednje škole, prosjeka ocjena u srednjoj školi, rezultata na državnoj maturi te uspjeha u pojedinim kolegijima tijekom studija. Korišteni su modeli kao što su logistička regresija, stabla odluke, *random forest*, SVM i *gradient boosting*. Na temelju dobivenih rezultata, može se zaključiti da su značajke vezane uz akademske performanse, poput broja ponavljanja upisa na određene predmete te uspjesi na državnoj maturi, ključni prediktori odustajanja studenata. Također, modeli kao što su *gradient boosting* i *random forest* pokazali su se kao najučinkovitiji alati za ovu vrstu problema, omogućujući relativno točna predviđanja s dobro uravnoteženim metrikama.

Ključne riječi: predviđanje odustajanja studenata, obrazovni podaci, random forest, gradient boosting, akademski uspjeh

Summary

This paper deals with the prediction of student dropouts using different machine learning models. Data such as year of enrollment, year of birth, gender, category of high school, grade point average in high school, results at the state matriculation exam, and success in individual courses during studies are included in the analysis. Models such as logistic regression, decision trees, random forest, SVM and gradient boosting were used. Based on the obtained results, it can be concluded that features related to academic performance, such as the number of repetitions of enrollment in certain courses and success at the state matriculation exam, are key predictors of student dropout. Also, models such as gradient boosting and random forest have proven to be the most effective tools for this type of problem, allowing relatively accurate predictions with well-balanced metrics.

Keywords: predicting student dropout, educational data, random forest, gradient boosting, academic success

Skraćenice

CNN	*Convolutional neural networks*	Konvolucijske neuronske mreže
BN	*Bayesian networks*	Bayesove mreže
SVM	*Support vector machine*	Stroj potpornih vektora
VS	*Visual Studio*	Visual Studio
TP	*True Positive*	Istinito pozitivni
FN	*False Negative*	Lažno negativni
TN	*True Negative*	Istinito negativni
FP	*False Positive*	Lažno pozitivni

Privitak

U nastavku predstavljeni su kodovi iz tri bilježnice koje su korištene u ovom radu: bilježnica za prethodnu obradu podataka (engl. *preprocessing*), bilježnica za analizu podataka, te bilježnica s implementacijom modela.

U nastavku slijedi kod za preprocesiranje podataka.

In [1]:
```
# library

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import datetime as dt
pd.set_option('display.max_columns',None)
pd.set_option('display.max_rows',None)
```

1. Import data - student data (from the academic year 2012/13 to the academic year 2022/23)

- rename columns
- change data types
- check for duplicates and empty data
- add student age of enrollment
- column for gender (male and female)
- high school category
- high school location
- separate students that are still in active studying and students that are finished with studying
- save data

In [ ]:
```
# importing data
data_path = "../data/raw/opci_podatci.csv"
df_student = pd.read_csv(data_path, sep=';')
```

In [ ]:
```
df_student.shape
```

Rename columns

In [4]:
```
df_student=df_student.rename(columns = {'godina upisa':'enrollment_year', 'GODINA ROĐENJA':'birth
                                        'SPOL':'gender','Završena SŠ':'graduated_high_school',
                                        'prosjek SŠ':'high_school_average_grade',
                                        'Bodovi srednje škole':'high_school_points',
                                        'Bodovi na državnoj maturi':'state_matriculation_exam_poi
                                        'Bodovi za dodatna  postignuća (natjecanja)':'additional_
                                        'Ukupno bodova za upis na studij':'total_points_for_study
                                        'Engleski jezik (A) %':'english_language_A_%',
                                        'Engleski jezik (B) %':'english_language_B_%',
                                        'Hrvatski jezik %':'croatian_language_%',
                                        'Matematika (A) %':'mathematics_A_%','Matematika (B) %':'
                                        'Informatika %':'informatics_%',
                                        'STATUS (diplomirao, ispisan, aktivan studij)':'student_s
                                        'Datum diplomiranja/ispisa':'graduation_or_discharge_date
```

Changing data types and filling Nan in numeric columns

In [5]:
```
# define columns and their fill values and target types
columns_info = {
    'enrollment_year': (0, int),
    'JMBAG': (0, int),
    'birth_year': (0, int),
    'high_school_average_grade': (0,float),
    'high_school_points': (0,float),
    'state_matriculation_exam_points': (0,float),
    'additional_points': (0,float),
    'total_points_for_study': (0,float),
    'english_language_A_%': (0,float),
    'english_language_B_%': (0,float),
    'croatian_language_%': (0,float),
    'mathematics_A_%': (0,float),
    'mathematics_B_%': (0,float),
    'informatics_%': (0,float)}
# columns that need comma replacement before conversion to float
comma_replace_columns = [
    'high_school_average_grade',
    'high_school_points',
    'state_matriculation_exam_points',
    'total_points_for_study',
    'english_language_A_%',
    'english_language_B_%',
    'croatian_language_%',
    'mathematics_A_%',
    'mathematics_B_%',
    'informatics_%']
# apply fillna and astype in a loop
for column, (fill_value, dtype) in columns_info.items():
    df_student[column] = df_student[column].fillna(fill_value)
    if column in comma_replace_columns:
        df_student[column] = df_student[column].str.replace(',', '.').astype(float)
        df_student[column] = df_student[column].fillna(fill_value)
    else:
        df_student[column] = df_student[column].fillna(fill_value).astype(dtype)
```

In [ ]:
```
# convert the date column from 'day.month.year' to datetime format
df_student['graduation_or_discharge_date']=pd.to_datetime(df_student['graduation_or_discharge_dat
                                                          format='%d.%m.%Y', errors='coerce')
# format the datetime column to 'YYYY-MM-DD'
df_student['graduation_or_discharge_date']=df_student['graduation_or_discharge_date'].dt.strftime
```

Importing additional table for high school category

In [7]:
```
data_path = "../data/raw/srednje_skole_dodatno.csv"
df_high_school = pd.read_csv(data_path,sep=';')
```

In [8]:
```
# renaming column in df_high_school
df_high_school = df_high_school.rename(columns = {'kategorija':'high_school_category'})
```

Check empty values and duplicated values

In [ ]:
```
df_student.isna().sum()
```

NOTE:

- empty high school data - students outside of Croatia
- empty graduation or discharge date for students that are still studying

In [ ]:
```
# get only duplicated rows
duplicated_rows = df_student[df_student.duplicated()]
# display the duplicated rows
duplicated_rows
```

NOTE: No duplicated rows

Student age of enrollment

In [11]:
```
# enrollment year - birth year = enrollment age
df_student['enrollment_age'] = df_student['enrollment_year'] - df_student['birth_year']
```

male and *female* dummy variable

In [ ]:
```
# create a new dummy variable 'gender_dummy' where:
# 1 = Ž, 0 = M
df_student['gender_dummy'] = df_student['gender'].apply(lambda x: 1 if x == 'Ž' else 0)
df_student[['gender','gender_dummy']].head()
```

High school category

Depending on the type of educational program, there are three types of secondary schools in the Republic of Croatia: source

- gymnasiums (general, language, classical, science-mathematics, science) (hrv. *gimnazija*)
- vocational schools (hrv. *strukovna škola*)
- art schools (music, dance, art) (hrv. *umjetnička škola*)

In [13]:
```
# high school category
school_category = {
    'Graditeljsko-geodetska tehnička' : 'strukovna',
    'Turistička i ugostiteljska' : 'strukovna',
    'Gimnazija' : 'gimnazija',
    'Tehnička' : 'strukovna',
    'Srednja strukovna' : 'strukovna',
    'Ekonomsko-birotehnička i trgovačka' : 'strukovna',
    'Turističko-ugostiteljska' : 'strukovna',
    'Elektrotehnička' : 'strukovna',
    'Graditeljska obrtnička i grafička' : 'strukovna',
    'gimnazija' : 'gimnazija',
    'Medicinska' : 'strukovna',
    'Ekonomsko-birotehnička' : 'strukovna',
    'Pomorska' : 'strukovna',
    'Trgovačka' : 'strukovna',
    'Ekonomska' : 'strukovna',
    'Medicinska i kemijska' : 'strukovna',
    'Poljoprivredna, prehrambena i veterinarska' : 'strukovna',
    'GIMNAZIJSKI KOLEGIJ' : 'gimnazija',
    'Tehnička i industrijska' : 'strukovna',
    'Glazbena' : 'umjetnička',
    'Prometno-tehnička' : 'strukovna',
    'Srednja zubotehnička' : 'strukovna',
    'Zdravstvena' : 'strukovna',
    'Škola likovnih umjetnosti' : 'umjetnička',
    'Škola primjenjene umjetnosti i dizajna' : 'umjetnička'}
# function to map high school names to categories
def get_school_category(high_school):
    if pd.isna(high_school):
        return 'no data'
    for key in school_category.keys():
        if key in high_school:
            return school_category[key]
    return None
# apply the function to create the new column
df_student['high_school_category'] = df_student['graduated_high_school'].apply(get_school_categor
```

Special part for high schools "Srednja škola" and "Prirodoslovna škola" because from just a name of the high school isnt familiar in which category should go.

In [14]:
```
# merge the dataframes
df_student_new = pd.merge(df_student, df_high_school, on='JMBAG', how='left', suffixes=('', '_new
```

In [ ]:
```
# fill in the missing values
df_student_new['high_school_category']=df_student_new['high_school_category'].combine_first(df_st
```

```
# drop the temporary '_new' column
df_student_new = df_student_new.drop(columns=['high_school_category_new'])
```

Making dummy variables for each high school category

- gymnasiums_dummy
- vocational_dummy
- art_dummy

In [16]:
```
# create columns
df_student_new['gymnasiums_dummy'] = (df_student_new['high_school_category'] == 'gimnazija').asty
df_student_new['vocational_dummy'] = (df_student_new['high_school_category'] == 'strukovna').asty
df_student_new['art_dummy'] = (df_student_new['high_school_category'] == 'umjetnička').astype(int
```

High school location

In [ ]:
```
# list of all locations in data
locations = ['Split', 'Dubrovnik', 'Sinj', 'Imotski', 'Metković', 'Šibenik', 'Vela Luka', 'Ogulin
             'Slavonski Brod','Makarska', 'Drniš', 'Zadar', 'Vrgorac', 'Korčula', 'Pula', 'Ploče'
             'Novska', 'Trogir', 'Omiš','Pazin', 'Benkovac', 'Županja', 'Vis', 'Kaštel Štafilić',
             'Zagreb', 'Blato', 'Požega', 'Sisak', 'Hvar']
# function to extract the location
def get_location(high_school):
    if pd.isna(high_school):
        return 'izvan RH'
    high_school = high_school.lower()
    for location in locations:
        location_lower = location.lower()
        if location_lower in high_school:
            return location
    return 'no data'
# apply the function to create the new column
df_student_new['high_school_location'] = df_student_new['graduated_high_school'].apply(get_locati
```

Make dummy variable for student that are from outside of Croatia.

In [18]:
```
df_student_new['outside_cro_high_school_dummy']=(df_student_new['high_school_location']=='izvan R
```

Average grade at high school

In [ ]:
```
# define the grade ranges and corresponding categories
def categorize_grade(grade):
    if grade < 2.5:
        return 2  # range for grade 2
    elif 2.5 <= grade < 3.5:
        return 3  # range for grade 3
    elif 3.5 <= grade < 4.5:
        return 4  # range for grade 4
    elif 4.5 <= grade:
        return 5  # range for grade 5
    else:
        return 'Invalid'  # handle any unexpected values
# apply the categorization function
df_student_new['high_school_grade_category']=df_student_new['high_school_average_grade'].apply(ca
```

Make dummy variable for dropout status

In [ ]:
```
# create a new dummy variable 'dropout_dummy' where:
# 1 = ispis, 0 = diploma ili aktivan
df_student_new['dropout_dummy']=df_student_new['student_status'].apply(lambda x: 1 if x == 'ispis
```

```
df_student_new[['student_status', 'dropout_dummy']].head()
```

Separation of data on active students and graduated/discharged students

In [ ]:
```
df_student_new['student_status'].value_counts()
```

In [ ]:
```
# number of unique students
df_student_new['JMBAG'].nunique()
```

In [ ]:
```
# active students
df_student_active = df_student_new.loc[df_student_new['student_status']=='aktivan 2023/24']
df_student_active.reset_index(drop=True).head(2)
```

In [ ]:
```
# droput or graduated students
df_student_main = df_student_new.loc[df_student_new['student_status']!='aktivan 2023/24']
df_student_main.reset_index(drop=True).head(2)
```

Save as csv

In [25]:
```
df_student_active.to_csv('../data/processed/active_student_data.csv', encoding='utf-8', index=Fal
df_student_main.to_csv('../data/processed/main_student_data.csv', encoding='utf-8', index=False)
```

2. Import data - student study data

- rename columns
- change data type
- check for duplicated, values in columns
- merge data with csv for active and main data

In [ ]:
```
data_path = "../data/raw/studij_dodatni_elementi.csv"
df_study = pd.read_csv(data_path, sep=';')
```

In [ ]:
```
df_study.shape
```

Rename columns

In [28]:
```
df_study = df_study.rename(columns = {'Ak. godina zadnjeg upisa':'academic_year_of_last_enrolleme
                                      'Paralelni studij':'parallel_study',
                                      'Indikator upisa zadnje up. god.':'enrollment_indicator_las
                                      'Nastavna godina zadnjeg upisa':'last_study_year',
                                      'STUDIJ: Prosjek ocjena':'average_grade',
                                      'STUDIJ: Težinski prosjek ocjena':'weighted_grade_average',
                                      'STUDIJ: Broj položenih ispita':'number_of_passed_exams',
                                      'STUDIJ: Broj položenih ispita koji ulaze u prosjek':'numbe
                                      'STUDIJ: Suma pozitivnih ocjena':'positive_grade_sum',
                                      'STUDIJ: Suma ECTS':'ECTS_sum',
                                      'STUDIJ: Broj predmeta':'number_of_courses',
                                      'STUDIJ: Suma Ocjena x ECTS (brojnik)':'grade_sum_ECTS',
                                      'Suma ECTS x Broj položenih ispita (nazivnik)':'ECTS_sum_nu
                                      'Osvojeno ECTS koji ulaze u stjecanje kvalifikacije':'earne
```

Changing data types and filling Nan in numeric columns

In [ ]:
```
# define columns and their fill values and target types
columns_info = {
    'average_grade': (0, float),
    'weighted_grade_average': (0, float),
    'ECTS_sum_number_of_passed_exams': (0, float),
    'earned_ECTS_included_in_qualification': (0,float)}
# apply fillna and astype in a loop
for column, (fill_value, dtype) in columns_info.items():
    df_study[column] = df_study[column].str.replace(',', '.').astype(float)
    df_study[column] = df_study[column].fillna(fill_value)
```

Check for duplicates

In [ ]:
```
# number of unique students
df_study['JMBAG'].nunique()
```

In [ ]:
```
# get only duplicated rows
duplicated_rows = df_study[df_study.duplicated()]
# display the duplicated rows
duplicated_rows
```

NOTE: No duplicated data

Merging data with student data for active and main student data

Active student data

In [32]:
```
# merging df with student data with study data for active and for the rest of the students
df_student_active_study = pd.merge(df_student_active, df_study, on="JMBAG", how="left")
```

In [ ]:
```
df_student_active_study.shape
```

Main student data (dropout and graduated)

In [34]:
```
# merging df with student data with study data for active and for the rest of the students
df_student_main_study = pd.merge(df_student_main, df_study, on="JMBAG", how="left")
```

In [ ]:
```
df_student_main_study.shape
```

Save as csv

In [36]:
```
df_student_active_study.to_csv('../data/interim/active_student_data_w_study.csv', encoding='utf-8
df_student_main_study.to_csv('../data/interim/main_student_data_w_study.csv', encoding='utf-8', i
```

3. Import data - data of finishing the study

- rename columns
- change data types
- check for duplicated
- get enrollment year of each student
- separate data for main and the rest of data

In [ ]:
```
# import data
data_path = "../data/raw/zavrsetak_studija.csv"
df_endstudy = pd.read_csv(data_path, sep=';')
```

In [ ]:
```
df_endstudy.shape
```

Rename columns

In [39]:
```
df_endstudy=df_endstudy.rename(columns={'Datum završetka studija':'study_end_date',
```

```
                                'Ak. god. završetka studija':'study_end_academic_year',
                                'Ak. god. zadnjeg upisa na studiju':'academic_year_of_las
                                'Prosjek ocjena na studiju':'average_grade',
                                'Težinski prosjek ocjena na studiju':'weighted_grade_aver
                                'Ukupno ECTS bodova':'total_ECTS','Položeno ispita':'pass
                                'Broj predmeta':'number_of_courses', 'Student studirao go
```

Changing data types and filling Nan in numeric columns

In [40]:
```
# define columns and their fill values and target types
columns_info = {
    'average_grade': (0, float),
    'weighted_grade_average': (0, float),
    'studying_years': (0, float),}
# apply fillna and astype in a loop
for column, (fill_value, dtype) in columns_info.items():
    df_endstudy[column] = df_endstudy[column].str.replace(',', '.').astype(float)
    df_endstudy[column] = df_endstudy[column].fillna(fill_value)
```

In [ ]:
```
# convert the date column from 'day.month.year' to datetime format
df_endstudy['study_end_date'] = pd.to_datetime(df_endstudy['study_end_date'],
                                               format='%m.%d.%Y', errors='coerce')
# format the datetime column to 'YYYY-MM-DD'
df_endstudy['study_end_date'] = df_endstudy['study_end_date'].dt.strftime('%Y-%m-%d')
```

Check for duplicates

In [ ]:
```
# number of unique students
df_endstudy['JMBAG'].nunique()
```

In [ ]:
```
# get only duplicated rows
duplicated_rows = df_endstudy[df_endstudy.duplicated()]
# display the duplicated rows
duplicated_rows
```

NOTE: No duplicated data

Get enrollment year

In [ ]:
```
# function for converting franction value of year to months
def fractional_to_year_month (years_fractional):
    # extract the integer part as the number of years
    years = int(years_fractional)
    # convert the fractional part to months
    months = int((years_fractional - years) * 12)
    return years, months
# function for getting the enrollment year
def get_begin_year(end_date, duration_years, duration_months):
    # calculate the total duration in months
    total_months = duration_years * 12 + duration_months
    # calculate the enrollment date by subtracting the duration from the finish date
    enrollment_date = end_date - pd.DateOffset(months=total_months)
    # if you only need the year of enrollment
    enrollment_year = enrollment_date.year
    return int(enrollment_year)
# convert 'end_year' to datetime
df_endstudy['study_end_date'] = pd.to_datetime(df_endstudy['study_end_date'])
# apply the function to each row
df_endstudy['enrollment_year'] = df_endstudy.apply(lambda row:get_begin_year(row['study_end_date'
                                                   *fractional_to_year_month(row['studying_years'
                                                    axis=1)
```

Separating on main and the rest of the data (students with matura and students without)

In [45]:
```
df_graduated_matura = df_endstudy.loc[(df_endstudy['JMBAG'].isin(df_student_main['JMBAG']))].rese
```

In [ ]:
```
df_graduated_matura.shape
```

In [47]:
```
df_graduated_no_matura = df_endstudy.loc[(~df_endstudy['JMBAG'].isin(df_student_main['JMBAG']))].
```

Save as csv

In [48]:
```
df_graduated_matura.to_csv('../data/processed/graduated_students_w_matura.csv', encoding='utf-8',
df_graduated_no_matura.to_csv('../data/processed/graduated_students_n_matura.csv', encoding='utf-
```

4. Import data - data on enrolled courses

- rename columns
- change data types

In [ ]:
```
# import data
data_path = "../data/raw/predmeti_upisano_polozeno.csv"
df_courses = pd.read_csv(data_path, sep=';')
```

In [ ]:
```
df_courses.shape
```

Rename columns

In [51]:
```
df_courses = df_courses.rename(columns = {'Status upisanog predmeta':'student_course_status',
                                          'Oslobođen polaganja':'exempted_from_taking_the_course_
                                          'Obavezan/izborni predmet':'mandatory_or_elective_cours
                                          'Način upisa predmeta':'course_enrollment_method',
                                          'Kratica predmeta':'course_abbreviation',
                                          'Naziv predmeta':'course_name', 'Šifra predmeta':'cours
                                          'Akademska godina':'academic_year','Nastavna godina':'s
                                          'Semestar slušanja predmeta':'semester_of_listening_to_
```

Changing data types and filling Nan in numeric columns

In [ ]:
```
# define columns and their fill values and target types
columns_info = {
    'JMBAG': (0, int),
    'course_code': (0, int),
    'semester_of_listening_to_course': (0, int)}
# apply fillna and astype in a loop
for column, (fill_value, dtype) in columns_info.items():
    df_courses[column] = df_courses[column].fillna(fill_value).astype(dtype)
```

Check for duplicates

In [ ]:
```
# number of unique students
df_courses['JMBAG'].nunique()
```

In [ ]:
```
# get only duplicated rows
duplicated_rows = df_courses[df_courses.duplicated()]
# display the duplicated rows
duplicated_rows
```

NOTE: No duplicated rows

Save as csv

In [55]:
```
df_courses.to_csv('../data/processed/student_courses_preprocessed.csv', encoding='utf-8', index=F
```

U nastavku slijedi kod za analizu podataka.

In [1]:
```
# library
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import datetime as dt
pd.set_option('display.max_columns',None)
pd.set_option('display.max_rows',None)
```

Import data

- main data: dropout students and graduated
- active student data

In [ ]:
```
data_path = "../data/interim/main_student_data_w_study.csv"
df_main_student_study = pd.read_csv(data_path)
df_main_student_study.shape
```

In [ ]:
```
data_path = "../data/interim/active_student_data_w_study.csv"
df_active_student_study = pd.read_csv(data_path)
df_active_student_study.shape
```

In [ ]:
```
# concat both df for analysis per years
df_all_enrolled = pd.concat([df_main_student_study, df_active_student_study],
                            axis=0, ignore_index=True)
df_all_enrolled.shape
```

Analysis by enrolled years

In [ ]:
```
df_all_enrolled[['high_school_average_grade','high_school_points', 'state_matriculation_exam_poin
                 'additional_points', 'total_points_for_study', 'english_language_A_%',
                'english_language_B_%', 'croatian_language_%', 'mathematics_A_%','mathematics_B_%
                'informatics_%']].describe()
```

Distribution of enrollment age of students

In [ ]:
```
sns.histplot(df_all_enrolled['enrollment_age'])
```

Matrix of correlations

In [ ]:
```
corr_matrix = df_all_enrolled[['dropout_dummy','enrollment_year','birth_year',
                               'gender_dummy', 'gymnasiums_dummy', 'vocational_dummy','art_dummy',
                               'outside_cro_high_school_dummy', 'enrollment_age']].corr()
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Matrix Heatmap')
plt.show()
```

In [ ]:
```
corr_matrix = df_all_enrolled[['dropout_dummy','high_school_average_grade','state_matriculation_e
                               'additional_points', 'total_points_for_study', 'english_language_A_
                               'croatian_language_%', 'mathematics_A_%','mathematics_B_%', 'inform
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Matrix Heatmap')
plt.show()
```

Number of students for each enrollment year

In [ ]:
```
# using value_counts() to count the number of students for each enrollment year
enrollment_counts = df_all_enrolled['enrollment_year'].value_counts().sort_index()
enrollment_counts
```

Distribution by gender

In [ ]:
```
# plotting the pie chart with labels, title, and percentage display
plt.pie(df_all_enrolled['gender'].value_counts(),
        labels=df_all_enrolled['gender'].value_counts().index,
        autopct='%1.1f%%',
        startangle=90,
        colors=['lightcoral','skyblue'])
plt.title('Raspodjela studenata po spolovima')
plt.axis('equal')  # ensures the pie chart is a circle
plt.show()
```

Distribution by gender - by years

In [ ]:
```
# colors for each gender
color_mapping = {'Ž': 'lightcoral', 'M': 'skyblue'}
# get unique enrollment years
enrollment_year_list = df_all_enrolled['enrollment_year'].unique()

# set up the grid layout for subplots: adjust rows and cols according to the number of enrollment
rows = (len(enrollment_year_list) // 3) + 1  # adjust the number of rows based on number of years
cols = 3

# create the figure and axes for subplots
fig, axes = plt.subplots(rows, cols, figsize=(12, 12))
# flatten the axes array for easier iteration
axes = axes.flatten()

# loop through each enrollment year and plot a pie chart
for i, year in enumerate(enrollment_year_list):
    # filter the data for the specific enrollment year
    data_for_year = df_all_enrolled[df_all_enrolled['enrollment_year'] == year]
    # count the number of males and females for the year
    gender_counts = data_for_year['gender'].value_counts()
    # get the corresponding colors for the pie chart based on the gender labels
    colors = [color_mapping[gender] for gender in gender_counts.index]
    # plot the pie chart in the corresponding subplot
    axes[i].pie(gender_counts,
                labels=gender_counts.index,
                autopct='%1.1f%%',
                startangle=90,
                colors=colors)
    axes[i].set_title(f'Godina {year}')
    axes[i].axis('equal')
# remove unused subplots (if any)
for j in range(i + 1, rows * cols):
    fig.delaxes(axes[j])
# set the main title for the entire figure
plt.suptitle('Raspodjela studenata po spolovima po upisanim godinama', fontsize=16)
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```

How many students per student status

In [ ]:
```
df_all_enrolled['student_status'].value_counts()
```

How many students per student status - by years

In [ ]:
```
# colors for each status
color_mapping = {'diploma': 'yellow', 'ispis': 'lightcoral', 'aktivan 2023/24': 'skyblue'}
# looping through each enrollment year and plotting a pie chart
enrollment_year_list = df_all_enrolled['enrollment_year'].unique()

# set up the grid layout for subplots: adjust rows and cols according to the number of enrollment
rows = (len(enrollment_year_list) // 3) + 1  # Adjust the number of rows based on number of years
cols = 3

# create the figure and axes for subplots
fig, axes = plt.subplots(rows, cols, figsize=(12, 12))
# flatten the axes array for easier iteration
axes = axes.flatten()

for i, year in enumerate(enrollment_year_list):
    # filter the data for the specific enrollment year
    data_for_year = df_all_enrolled[df_all_enrolled['enrollment_year'] == year]
    # count the number
    status_counts = data_for_year['student_status'].value_counts()
    # get the corresponding colors for the pie chart based on the labels
    colors = [color_mapping[s] for s in status_counts.index]
    # plot the pie chart in the corresponding subplot
    axes[i].pie(status_counts,
                labels=status_counts.index,
                autopct='%1.1f%%',
                startangle=90,
                colors=colors)
    axes[i].set_title(f'Godina {year}')
    axes[i].axis('equal')
# remove unused subplots (if any)
for j in range(i + 1, rows * cols):
    fig.delaxes(axes[j])
# set the main title for the entire figure
plt.suptitle('Raspodjela studenata po statusima po upisanim godinama', fontsize=16)
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```

How many students per high school category

In [ ]:
```
df_all_enrolled['high_school_category'].value_counts()
```

How many students per high school category - by years

In [ ]:
```
# colors for each status
color_mapping = {'gimnazija':'yellow','strukovna':'red','umjetnička':'skyblue','no data':'green'}
# looping through each enrollment year and plotting a pie chart
enrollment_year_list = df_all_enrolled['enrollment_year'].unique()

# Set up the grid layout for subplots: adjust rows and cols according to the number of enrollment
rows = (len(enrollment_year_list) // 3) + 1  # Adjust the number of rows based on number of years
cols = 3

# create the figure and axes for subplots
fig, axes = plt.subplots(rows, cols, figsize=(12, 12))
# flatten the axes array for easier iteration
axes = axes.flatten()

for i, year in enumerate(enrollment_year_list):
    # filter the data for the specific enrollment year
    data_for_year = df_all_enrolled[df_all_enrolled['enrollment_year'] == year]
    # count the number
    school_counts = data_for_year['high_school_category'].value_counts()
    # get the corresponding colors for the pie chart based on the labels
    colors = [color_mapping[s] for s in school_counts.index]
    axes[i].pie(school_counts,
                labels=school_counts.index,
                autopct='%1.1f%%',
                startangle=90,
                colors=colors)
    axes[i].set_title(f'Godina {year}')
    axes[i].axis('equal')
# remove unused subplots (if any)
for j in range(i + 1, rows * cols):
    fig.delaxes(axes[j])
# set the main title for the entire figure
plt.suptitle('Raspodjela studenata po kategoriji srednje škole po upisnoj godini', fontsize=16)
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```

High school location

In [ ]:
```
# grouping the data by 'high_school_location' and counting the number of students
location_counts = df_all_enrolled['high_school_location'].value_counts()

# plotting the data
plt.figure(figsize=(10, 6))
location_counts.plot(kind='bar', color='blue')

# adding title and labels
plt.title('Broj studenata prema lokacijima srednjih škola', fontsize=14)
plt.xlabel('Lokacija srednje škole', fontsize=12)
plt.ylabel('Broj studenata', fontsize=12)

# rotating x-axis labels for better readability
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

In [ ]:
```
# grouping the data by 'high_school_location' and counting the number of students
location_counts = df_all_enrolled['high_school_location'].value_counts()

most_students_location = location_counts.idxmax()
most_students_count_location = location_counts.max()
print(f"The location of the high school where the largest number of students is: {most_students_l
```

Average grade at high school

In [ ]:
```
df_all_enrolled['high_school_average_grade'].describe()
```

In [ ]:
```
# find the most common age at enrollment
most_common_grade = df_all_enrolled['high_school_grade_category'].mode()[0]
print(f"The most common grade point average from the high school belongs to the grade: {most_comm
```

In [ ]:
```
# count the number of students in each grade category
grade_counts = df_all_enrolled['high_school_grade_category'].value_counts()
# Plotting the pie chart
plt.figure(figsize=(4, 4))
plt.pie(grade_counts,
        labels=grade_counts.index,
        autopct='%1.1f%%',
        startangle=90,
        colors=['lightcoral','skyblue','lightgreen','gold'])
plt.title('Raspodjela studenata po ocjenama iz srednje škole')
plt.axis('equal')
plt.show()
```

Student status according to high school final grades

In [ ]:
```
# Group by 'enrollment_year', 'high_school_grade_category', and 'student_status'
grouped = df_all_enrolled.groupby(['enrollment_year','high_school_grade_category','student_status

# colors for each student status
status_colors = {'aktivan 2023/24':'green','ispis':'red','diploma':'blue'}
# get the unique enrollment years
enrollment_years = df_all_enrolled['enrollment_year'].unique()

# set up the grid layout for subplots
rows = (len(enrollment_years) // 3) + 1
cols = 3
fig, axes = plt.subplots(rows, cols, figsize=(15, 10), constrained_layout=True)
axes = axes.flatten()

# loop through each enrollment year and plot a histogram in the grid
for idx, year in enumerate(enrollment_years):
    if year not in grouped.index:
        continue  # skip years with no data
    year_data = grouped.loc[year]  # get the data for the specific year
    # get the unique high school grade categories
    high_school_grades = df_all_enrolled['high_school_grade_category'].unique()
    ax = axes[idx]
    for i, grade in enumerate(high_school_grades):
        if grade not in year_data.index:
            continue  # skip grades with no data for the year
        grade_data = year_data.loc[grade]
        # extract counts for each student status
        studying = grade_data.get('aktivan 2023/24', 0)
        graduated = grade_data.get('diploma', 0)
        dropped_out = grade_data.get('ispis', 0)
        # bar width and positions
        width = 0.25  # Bar width
        positions = [i - width, i, i + width]  # positions for bars on x-axis
        # plot bars for each student status
        ax.bar(i - width, studying, width, label='aktivan', color=status_colors['aktivan 2023/24'
        ax.bar(i, graduated, width, label='diploma', color=status_colors['diploma'], alpha=0.6)
        ax.bar(i + width, dropped_out, width, label='ispis', color=status_colors['ispis'], alpha=

    # set titles and labels
    ax.set_title(f'Upisna godina: {year}')
    ax.set_ylabel('Broj studenata')
    ax.set_xticks(range(len(high_school_grades)))
    ax.set_xticklabels(high_school_grades, rotation=45)
    ax.legend(['aktivan','diploma','ispis'])

# remove unused subplots if the number of years is less than the number of grid cells
for idx in range(len(enrollment_years), len(axes)):
    fig.delaxes(axes[idx])

# set the main title for the entire figure
plt.suptitle('Status studenata po završnim ocjenama srednje škole po upisnoj godini', fontsize=16
plt.show()
```

Division by gender by status by enrollment years

In [ ]:
```
# Group by 'enrollment_year', 'gender', and 'status' to count students
grouped = df_all_enrolled.groupby(['enrollment_year', 'gender', 'student_status']).size().unstack

# Define colors for each gender
colors = {'M': 'blue', 'Ž': 'pink'}
# get the unique enrollment years
enrollment_years = sorted(df_all_enrolled['enrollment_year'].unique())

# set up the grid layout for subplots
rows = (len(enrollment_years) // 3) + int(len(enrollment_years) % 3 != 0)
cols = 3
fig, axes = plt.subplots(rows, cols, figsize=(15, 5 * rows), constrained_layout=True)
axes = axes.flatten()

# plot data for each year
for idx, year in enumerate(enrollment_years):
    if year not in grouped.index:
        continue  # skip years with no data

    year_data = grouped.loc[year]
    status_list = df_all_enrolled['student_status'].unique()
    ax = axes[idx]
    width = 0.35
    positions = range(len(status_list))
    for i, status in enumerate(status_list):
        male_count = year_data.loc['M', status] if 'M' in year_data.index else 0
        female_count = year_data.loc['Ž', status] if 'Ž' in year_data.index else 0
        # plot bars for male and female students
        ax.bar(i - width / 2, male_count, width, label='M', color=colors['M'])
        ax.bar(i + width / 2, female_count, width, label='Ž', color=colors['Ž'])

    # set titles and labels
    ax.set_title(f'Upisna godina: {year}')
    ax.set_ylabel('Broj studenata')
    ax.set_xticks(positions)
    ax.set_xticklabels(status_list, rotation=45)
    ax.legend(['M','Ž'])
# hide any unused subplots
for idx in range(len(enrollment_years), len(axes)):
    fig.delaxes(axes[idx])
# set the main title for the entire figure
plt.suptitle('Raspodjela po spolu za svaki status studenta prema upisnim godinama', fontsize=16)
plt.show()
```

Gender distribution by high school category by enrollment years

In [ ]:
```
grouped_school = df_all_enrolled.groupby(['enrollment_year', 'gender', 'high_school_category']).s

# colors for each gender
colors = {'M': 'blue', 'Ž': 'pink'}
# get the unique enrollment years
enrollment_years = sorted(df_all_enrolled['enrollment_year'].unique())

# set up the grid layout for subplots
rows = (len(enrollment_years) // 3) + int(len(enrollment_years) % 3 != 0)  # Calculate the number
cols = 3  # Number of columns for subplots
fig, axes = plt.subplots(rows, cols, figsize=(15, 5 * rows), constrained_layout=True)
axes = axes.flatten()

# plot data for each year
for idx, year in enumerate(enrollment_years):
    if year not in grouped_school.index:
        continue  # skip years with no data

    year_data = grouped_school.loc[year]
    school_list = df_all_enrolled['high_school_category'].unique()
    ax = axes[idx]
    width = 0.35
    positions = range(len(school_list))

    for i, status in enumerate(school_list):
        male_count = year_data.loc['M', status] if 'M' in year_data.index else 0
        female_count = year_data.loc['Ž', status] if 'Ž' in year_data.index else 0
        # plot bars for male and female students
```

```
        ax.bar(i - width / 2, male_count, width, label='M', color=colors['M'])
        ax.bar(i + width / 2, female_count, width, label='Ž', color=colors['Ž'])

    # set titles and labels
    ax.set_title(f'Upisna godina: {year}')
    ax.set_ylabel('Broj studenata')
    ax.set_xticks(positions)
    ax.set_xticklabels(school_list, rotation=45)
    ax.legend(['M','Ž'])

# hide any unused subplots
for idx in range(len(enrollment_years), len(axes)):
    fig.delaxes(axes[idx])
# set the main title for the entire figure
plt.suptitle('Raspodjela po spolu za kategoriji srednje škole prema upisnim godinama', fontsize=1
plt.show()
```

Import data about enrolled courses

In [6]:
```
data_path = "../data/processed/student_courses_preprocessed.csv"
df_courses = pd.read_csv(data_path)
```

In [ ]:
```
df_courses.isna().sum()
```

In [ ]:
```
# number of unique students
df_courses['JMBAG'].nunique()
```

In [ ]:
```
# get student courses for students that are in main data
df_courses_all = df_courses.loc[df_courses['JMBAG'].isin(df_all_enrolled['JMBAG'])]
df_courses_all.shape
```

In [ ]:
```
# number of unique students
df_courses_all['JMBAG'].nunique()
```

Mandatory vs elective courses

Mandatory courses

In [11]:
```
# df for mandatory courses
df_courses_all_mandatory = df_courses_all.loc[df_courses_all['mandatory_or_elective_course']=='Ob
```

How many times each mandatory course was enrolled (not counting the first time student enrolled)

In [ ]:
```
# how many times each course was enrolled by each student
# group by JMBAG and course_name, and count occurrences
enrollment_counts_mandatory = df_courses_all_mandatory.groupby(['JMBAG', 'course_name']).size().r
```

In [13]:
```
enrollment_counts_mandatory_repeat = enrollment_counts_mandatory.loc[enrollment_counts_mandatory[
course_counts_mandatory_repeat = enrollment_counts_mandatory_repeat.groupby(['course_name']).size
course_counts_mandatory_repeat.sort_values(by='count', ascending=False,inplace=True)
```

In [ ]:
```
# courses that have more than 10 students that enrolled again
course_counts_mandatory_repeat_10 = course_counts_mandatory_repeat.loc[course_counts_mandatory_re
```

In [ ]:
```
fig = plt.figure(figsize=(8,12))
sns.barplot(x='count', y='course_name', data=course_counts_mandatory_repeat_10, color="c")
plt.title('Koliko se puta obavezan predmet upisivao >= 10 (van prvog puta)')
plt.xlabel('Broj ponavljanja')
plt.ylabel('')
plt.show()
```

Elective courses

In [22]:
```
# df for elective courses
df_courses_all_elective = df_courses_all.loc[df_courses_all['mandatory_or_elective_course']=='Izb
```

How many times each elective course was enrolled (not counting the first time student enrolled)

In [23]:
```
# how many times each course was enrolled by each student
# group by JMBAG and course_name, and count occurrences
enrollment_counts_elective = df_courses_all_elective.groupby(['JMBAG', 'course_name']).size().res
```

In [24]:
```
enrollment_counts_elective_repeat = enrollment_counts_elective.loc[enrollment_counts_elective['co
course_counts_elective_repeat = enrollment_counts_elective_repeat.groupby(['course_name']).size()
course_counts_elective_repeat.sort_values(by='count', ascending=False,inplace=True)
```

In [ ]:
```
fig = plt.figure(figsize=(8,8))
sns.barplot(x='count', y='course_name', data=course_counts_elective_repeat, color="c")
plt.title('Koliko se puta izborni predmet upisivao (van prvog puta)')
plt.xlabel('Broj ponavljanja')
plt.ylabel('')
plt.show()
```

Import data for graduated students with matura

In [ ]:
```
data_path = "../data/processed/graduated_students_w_matura.csv"
df_end_w_matura = pd.read_csv(data_path)
df_end_w_matura.head()
```

In [ ]:
```
df_end_w_matura.shape
```

In [ ]:
```
# number of unique students
df_end_w_matura['JMBAG'].nunique()
```

Distribution of years of study among graduated students

In [ ]:
```
# rounding the years of study to the nearest higher whole number
df_end_w_matura['study_duration_bin'] = np.ceil(df_end_w_matura['studying_years']).astype(int)
# grouping by number of years
year_distribution = df_end_w_matura['study_duration_bin'].value_counts().sort_index()
print(year_distribution)

# display of the distribution in a histogram
plt.figure(figsize=(8, 5))
plt.bar(year_distribution.index, year_distribution.values, color='blue')
plt.title('Distribucija godina studiranja među završenim studentima')
plt.xlabel('Broj godina studiranja')
plt.ylabel('Broj studenata')
plt.xticks([3, 4, 5, 6, 7])
plt.show()
```

In [32]:
```
# merging df with student data with study data for active and for the rest of the students
df_end_w_matura_add = pd.merge(df_end_w_matura, df_main_student_study, on="JMBAG", how="left")
```

In [ ]:
```
df_end_w_matura_add.shape
```

In [ ]:
```
# group by study years and gender
grouped = df_end_w_matura_add.groupby(['study_duration_bin', 'gender']).size().unstack(fill_value
```

```python
# ensure both male and female columns are present
if 'M' not in grouped.columns:
    grouped['M'] = 0
if 'Ž' not in grouped.columns:
    grouped['Ž'] = 0

# plotting side-by-side bars
bar_width = 0.35
index = np.arange(len(grouped))
fig, ax = plt.subplots()
# create bar plots for male and female
bar1 = ax.bar(index, grouped['M'], bar_width, label='M',color='blue')
bar2 = ax.bar(index + bar_width, grouped['Ž'], bar_width, label='Ž',color='pink')
ax.set_xlabel('Broj godina studiranja')
ax.set_ylabel('Broj studenata')
ax.set_title('Raspodjela po spolovima po godinama studiranja')
ax.set_xticks(index + bar_width / 2)
ax.set_xticklabels(grouped.index)
ax.legend()
plt.tight_layout()
plt.show()
```

U nastavku slijedi kod za primjenu modela.

In [75]:
```
# library
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score, f1_s
pd.set_option('display.max_columns',None)
pd.set_option('display.max_rows',None)
```

Import data

In [ ]:
```
data_path = "../data/interim/main_student_data_w_study.csv"
df_main_student_study = pd.read_csv(data_path)
df_main_student_study.shape
```

Adding course enrollment data

In [77]:
```
data_path = "../data/processed/student_courses_preprocessed.csv"
df_courses = pd.read_csv(data_path)
```

In [ ]:
```
# get student courses for students that are in main data
df_courses_all = df_courses.loc[df_courses['JMBAG'].isin(df_main_student_study['JMBAG'])]
df_courses_all.shape
```

In [79]:
```
# get mandatory courses
df_courses_all_mandatory = df_courses_all.loc[df_courses_all['mandatory_or_elective_course']=='Ob
```

In [80]:
```
# how many times each course was enrolled by each student
# group by JMBAG and course_name, and count occurrences
enrollment_counts_mandatory = df_courses_all_mandatory.groupby(['JMBAG', 'course_name']).size().r
```

Adding data about how many times each student enrolled in courses (Matematika II, Matematika I, Programiranje II) on main data

In [81]:
```
# list of specific courses you want to keep
specific_courses = ['Matematika II', 'Matematika I', 'Programiranje II']
# filter the DataFrame to include only the specific courses
filtered_courses = enrollment_counts_mandatory[enrollment_counts_mandatory['course_name'].isin(sp
# pivot the table to have specific course names as columns and the count as values
course_pivot = filtered_courses.pivot_table(index='JMBAG', columns='course_name', values='count',
```

In [ ]:
```
# merge this new table with the main student df on 'JMBAG'
df_students_with_courses = pd.merge(df_main_student_study, course_pivot, on='JMBAG', how='left')
```

Division by gender

In [ ]:
```
# df for male only
df_main_M = df_students_with_courses.loc[df_students_with_courses['gender']=='M']
df_main_M.shape
```

In [ ]:
```
# df for female only
df_main_F = df_students_with_courses.loc[df_students_with_courses['gender']=='Ž']
df_main_F.shape
```

Division by high school category

In [ ]:
```
# gymnasiums
df_main_gimn = df_students_with_courses.loc[df_students_with_courses['gymnasiums_dummy']==1]
df_main_gimn.shape
```

In [ ]:
```
# vocational
df_main_struk = df_students_with_courses.loc[df_students_with_courses['vocational_dummy']==1]
df_main_struk.shape
```

In [ ]:
```
# art
df_main_art = df_students_with_courses.loc[df_students_with_courses['art_dummy']==1]
df_main_art.shape
```

Division by location

In [ ]:
```
# Split
df_main_st = df_students_with_courses.loc[df_students_with_courses['high_school_location']=='Spli
df_main_st.shape
```

In [ ]:
```
# ouside of Croatia
df_main_van = df_students_with_courses.loc[df_students_with_courses['high_school_location']=='izv
df_main_van.shape
```

In [ ]:
```
# outside Split in all Croatia
df_main_rh = df_students_with_courses.loc[(df_students_with_courses['high_school_location']!='Spl
df_main_rh.shape
```

Functions

In [72]:
```
def calculate_metrics (y_test,y_pred):
    # Calculate accuracy
    accuracy = accuracy_score(y_test,y_pred)
    print(f"Accuracy: {accuracy:.2f}")
    # Calculate precision
    precision = precision_score(y_test,y_pred, average='binary')
    print(f"Precision: {precision:.2f}")
    # Calculate recall
    recall = recall_score(y_test,y_pred, average='binary')
    print(f"Recall: {recall:.2f}")
    # Calculate F1-score
    f1 = f1_score(y_test,y_pred, average='binary')
    print(f"F1-Score: {f1:.2f}")
    # Calculate AUC
    auc = roc_auc_score(y_test,y_pred)
    print(f"AUC: {auc:.2f}")
    conf_matrix = confusion_matrix(y_test,y_pred)
    plt.figure(figsize=(3, 2))
    sns.set(font_scale=1.0)
    sns.heatmap(conf_matrix,annot = True,fmt='d',cbar=False)
    plt.xlabel('Točna vrijednost')
    plt.ylabel('Predviđena vrijednost')
```

First set of input features

Based on literature:

- 'enrollment_year',
- 'birth_year',
- 'gender_dummy',

- 'gymnasiums_dummy',
- 'vocational_dummy'
- 'art_dummy'
- 'enrollment_age'
- 'high_school_average_grade'
- 'outside_cro_high_school_dummy'

In [114…
```
# get the names of features and target value
feature_names = ['enrollment_year', 'birth_year','gender_dummy','gymnasiums_dummy', 'vocational_d
                 'enrollment_age', 'high_school_average_grade','outside_cro_high_school_dummy']
target_value = ['dropout_dummy']
```

All data

In [115…
```
# split data into features (x) and target (y)
X = df_main_student_study[feature_names]
y = df_main_student_study[target_value]
```

Logistic regression

In [ ]:
```
# split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
# initialize the Logistic Regression Model
log_reg = LogisticRegression()
# train the Model
log_reg.fit(X_train, y_train)
# make Predictions
y_pred = log_reg.predict(X_test)
calculate_metrics(y_test, y_pred)
```

Decision tree

In [ ]:
```
# split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
# initialize the Decision Tree Classifier Model
decision_tree = DecisionTreeClassifier(random_state=42)
# train the Model
decision_tree.fit(X_train, y_train)
# make Predictions
y_pred = decision_tree.predict(X_test)
calculate_metrics(y_test, y_pred)
```

Random forest

In [ ]:
```
# split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
# initialize the Random Forest Classifier Model
rf = RandomForestClassifier(n_estimators=200, max_depth=10, random_state=42)
# train the Model
rf.fit(X_train, y_train)
# make Predictions
y_pred = rf.predict(X_test)
calculate_metrics(y_test, y_pred)
```

Gradient boosting

In [ ]:
```
# split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
# initialize and train the Gradient Boosting Classifier
gb_clf = GradientBoostingClassifier(random_state=42)
gb_clf.fit(X_train, y_train)
# make predictions and evaluate the model
y_pred = gb_clf.predict(X_test)
calculate_metrics(y_test, y_pred)
```

SVM

In [ ]:
```
# split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
# define the parameter grid for classification
param_grid = {'C': [0.1, 1, 10],
              'gamma': ['scale','auto'],
              'kernel': ['linear', 'rbf']}
# initialize GridSearchCV
grid = GridSearchCV(SVC(), param_grid, refit=True, cv=3, n_jobs=-1)
grid.fit(X_train, y_train)
print("Best parameters found: ", grid.best_params_)
# evaluate the tuned model
grid_predictions = grid.predict(X_test)
calculate_metrics(y_test, grid_predictions)
```

By location

In [19]:
```
# split data into features (x) and target (y)
X_st = df_main_st[feature_names]
y_st = df_main_st[target_value]
X_van = df_main_van[feature_names]
y_van = df_main_van[target_value]
X_rh = df_main_rh[feature_names]
y_rh = df_main_rh[target_value]
```

Logistic regression

In [ ]:
```
# split into train and test sets
X_train_st, X_test_st, y_train_st, y_test_st = train_test_split(X_st, y_st, test_size=0.25, rando
# initialize the Logistic Regression Model
log_reg_st = LogisticRegression()
# train the Model
log_reg_st.fit(X_train_st, y_train_st)
# make Predictions
y_pred_st = log_reg_st.predict(X_test_st)
print("For Split:\n")
calculate_metrics(y_test_st, y_pred_st)
```

In [ ]:
```
# split into train and test sets
X_train_van, X_test_van, y_train_van, y_test_van= train_test_split(X_van, y_van, test_size=0.25,
log_reg_van = LogisticRegression()
log_reg_van.fit(X_train_van, y_train_van)
y_pred_van = log_reg_van.predict(X_test_van)
print("Outside Croatia:\n")
calculate_metrics(y_test_van, y_pred_van)
```

In [ ]:
```
# split into train and test sets
X_train_rh, X_test_rh, y_train_rh, y_test_rh= train_test_split(X_rh, y_rh, test_size=0.25, random
log_reg_rh = LogisticRegression()
log_reg_rh.fit(X_train_rh, y_train_rh)
y_pred_rh = log_reg_rh.predict(X_test_rh)
print("Outside Split:\n")
calculate_metrics(y_test_rh, y_pred_rh)
```

Decision tree

In [ ]:
```
# split into train and test sets
```

```
X_train_st, X_test_st, y_train_st, y_test_st = train_test_split(X_st, y_st, test_size=0.25, rando
# initialize the Decision Tree Classifier Model
decision_tree_st = DecisionTreeClassifier(random_state=42)
# train the Model
decision_tree_st.fit(X_train_st, y_train_st)
# make Predictions
y_pred_st = decision_tree_st.predict(X_test_st)
print("For Split:\n")
calculate_metrics(y_test_st, y_pred_st)
```

In [ ]:
```
# split into train and test sets
X_train_van, X_test_van, y_train_van, y_test_van= train_test_split(X_van, y_van, test_size=0.25,
decision_tree_van = DecisionTreeClassifier(random_state=42)
decision_tree_van.fit(X_train_van, y_train_van)
# make Predictions
y_pred_van = decision_tree_van.predict(X_test_van)
print("Outside Croatia:\n")
calculate_metrics(y_test_van, y_pred_van)
```

In [ ]:
```
# split into train and test sets
X_train_rh, X_test_rh, y_train_rh, y_test_rh= train_test_split(X_rh, y_rh, test_size=0.25, random
decision_tree_rh = DecisionTreeClassifier(random_state=42)
decision_tree_rh.fit(X_train_rh, y_train_rh)
# make Predictions
y_pred_rh = decision_tree_rh.predict(X_test_rh)
print("Outside Split:\n")
calculate_metrics(y_test_rh, y_pred_rh)
```

Random forest

In [ ]:
```
# split into train and test sets
X_train_st, X_test_st, y_train_st, y_test_st = train_test_split(X_st, y_st, test_size=0.25, rando
# initialize the Random Forest Classifier Model
rf_st = RandomForestClassifier(n_estimators=200, max_depth=10, random_state=42)
# train the Model
rf_st.fit(X_train_st, y_train_st)
# make Predictions
y_pred_st = rf_st.predict(X_test_st)
print("For Split:\n")
calculate_metrics(y_test_st, y_pred_st)
```

In [ ]:
```
# split into train and test sets
X_train_van, X_test_van, y_train_van, y_test_van= train_test_split(X_van, y_van, test_size=0.25,
rf_van = RandomForestClassifier(n_estimators=200, max_depth=10, random_state=42)
rf_van.fit(X_train_van, y_train_van)
# make Predictions
y_pred_van = rf_van.predict(X_test_van)
print("Outside Croatia:\n")
calculate_metrics(y_test_van, y_pred_van)
```

In [ ]:
```
# split into train and test sets
X_train_rh, X_test_rh, y_train_rh, y_test_rh= train_test_split(X_rh, y_rh, test_size=0.25, random
rf_rh = RandomForestClassifier(n_estimators=200, max_depth=10, random_state=42)
rf_rh.fit(X_train_rh, y_train_rh)
# make Predictions
y_pred_rh = rf_rh.predict(X_test_rh)
print("Outside Split:\n")
calculate_metrics(y_test_rh, y_pred_rh)
```

Gradient boosting

In [ ]:
```
# split into train and test sets
X_train_st, X_test_st, y_train_st, y_test_st = train_test_split(X_st, y_st, test_size=0.25, rando
# initialize and train the Gradient Boosting Classifier
gb_clf_st = GradientBoostingClassifier(random_state=42)
gb_clf_st.fit(X_train_st, y_train_st)
# make predictions and evaluate the model
y_pred_st = gb_clf_st.predict(X_test_st)
print("For Split:\n")
calculate_metrics(y_test_st, y_pred_st)
```

In [ ]:
```
# split into train and test sets
X_train_van, X_test_van, y_train_van, y_test_van=train_test_split(X_van, y_van, test_size=0.25, r
# initialize and train the Gradient Boosting Classifier
gb_clf_van = GradientBoostingClassifier(random_state=42)
gb_clf_van.fit(X_train_van, y_train_van)
# make predictions and evaluate the model
y_pred_van = gb_clf_van.predict(X_test_van)
print("Outside Croatia:\n")
calculate_metrics(y_test_van, y_pred_van)
```

In [ ]:
```
# split into train and test sets
X_train_rh, X_test_rh, y_train_rh, y_test_rh= train_test_split(X_rh, y_rh, test_size=0.25, random
# initialize and train the Gradient Boosting Classifier
gb_clf_rh = GradientBoostingClassifier(random_state=42)
gb_clf_rh.fit(X_train_rh, y_train_rh)
# make predictions and evaluate the model
y_pred_rh = gb_clf_rh.predict(X_test_rh)
print("Outside Split:\n")
calculate_metrics(y_test_rh, y_pred_rh)
```

SVM

In [ ]:
```
# split into train and test sets
X_train_st, X_test_st, y_train_st, y_test_st = train_test_split(X_st, y_st, test_size=0.25, rando
# define the parameter grid for classification
param_grid = {'C': [0.1, 1, 10],
              'gamma': ['scale','auto'],
              'kernel': ['linear', 'rbf']}
# initialize GridSearchCV
grid = GridSearchCV(SVC(), param_grid, refit=True, cv=3, n_jobs=-1)
grid.fit(X_train_st, y_train_st)
print("Best parameters found: ", grid.best_params_)
# evaluate the tuned model
grid_predictions_st = grid.predict(X_test_st)
print("For Split:\n")
calculate_metrics(y_test_st, grid_predictions_st)
```

In [ ]:
```
# split into train and test sets
X_train_van, X_test_van, y_train_van, y_test_van= train_test_split(X_van, y_van, test_size=0.25,
# define the parameter grid for classification
param_grid = {'C': [0.1, 1, 10],
              'gamma': ['scale','auto'],
              'kernel': ['linear', 'rbf']}
# initialize GridSearchCV
grid = GridSearchCV(SVC(), param_grid, refit=True,cv=3, verbose=2)
grid.fit(X_train_van, y_train_van)
print("Best parameters found: ", grid.best_params_)
# evaluate the tuned model
grid_predictions_van = grid.predict(X_test_van)
print("Outside Croatia:\n")
calculate_metrics(y_test_van, grid_predictions_van)
```

In [ ]:
```
# split into train and test sets
X_train_rh, X_test_rh, y_train_rh, y_test_rh= train_test_split(X_rh, y_rh, test_size=0.25, random
# define the parameter grid for classification
param_grid = {'C': [0.1, 1, 10],
              'gamma': ['scale','auto'],
              'kernel': ['linear', 'rbf']}
# initialize GridSearchCV
```

```
grid = GridSearchCV(SVC(), param_grid, refit=True, cv=3,n_jobs=-1)
grid.fit(X_train_rh, y_train_rh)
print("Best parameters found: ", grid.best_params_)
# evaluate the tuned model
grid_predictions_rh = grid.predict(X_test_rh)
print("Outside Split:\n")
calculate_metrics(y_test_rh, grid_predictions_rh)
```

By high school category

In [ ]:
```
# split data into features (x) and target (y)
X_g = df_main_gimn[feature_names]
y_g = df_main_gimn[target_value]
X_s = df_main_struk[feature_names]
y_s = df_main_struk[target_value]
```

Logistic regression

In [ ]:
```
# split into train and test sets
X_train_g, X_test_g, y_train_g, y_test_g = train_test_split(X_g, y_g, test_size=0.25, random_stat
# initialize the Logistic Regression Model
log_reg_g = LogisticRegression()
# train the Model
log_reg_g.fit(X_train_g, y_train_g)
# make Predictions
y_pred_g = log_reg_g.predict(X_test_g)
print("Gimnazija:\n")
calculate_metrics(y_test_g, y_pred_g)
```

In [ ]:
```
# split into train and test sets
X_train_s, X_test_s, y_train_s, y_test_s= train_test_split(X_s, y_s, test_size=0.25, random_state
log_reg_s = LogisticRegression()
log_reg_s.fit(X_train_s, y_train_s)
# make Predictions
y_pred_s = log_reg_s.predict(X_test_s)
print("Strukovna:\n")
calculate_metrics(y_test_s, y_pred_s)
```

Decision tree

In [ ]:
```
# split into train and test sets
X_train_g, X_test_g, y_train_g, y_test_g = train_test_split(X_g, y_g, test_size=0.25, random_stat
# initialize the Decision Tree Classifier Model
decision_tree_g = DecisionTreeClassifier(random_state=42)
# train the Model
decision_tree_g.fit(X_train_g, y_train_g)
# make Predictions
y_pred_g = decision_tree_g.predict(X_test_g)
print("Gimnazija:\n")
calculate_metrics(y_test_g, y_pred_g)
```

In [ ]:
```
# split into train and test sets
X_train_s, X_test_s, y_train_s, y_test_s= train_test_split(X_s, y_s, test_size=0.25, random_state
decision_tree_s = DecisionTreeClassifier(random_state=42)
decision_tree_s.fit(X_train_s, y_train_s)
# make Predictions
y_pred_s = decision_tree_s.predict(X_test_s)
print("Strukovna:\n")
calculate_metrics(y_test_s, y_pred_s)
```

Random forest

In [ ]:
```
# split into train and test sets
X_train_g, X_test_g, y_train_g, y_test_g = train_test_split(X_g, y_g, test_size=0.25, random_stat
# initialize the Random Forest Classifier Model
rf_g = RandomForestClassifier(n_estimators=200, max_depth=10, random_state=42)
# train the Model
rf_g.fit(X_train_g, y_train_g)
# make Predictions
y_pred_g = rf_g.predict(X_test_g)
print("Gimnazija:\n")
calculate_metrics(y_test_g, y_pred_g)
```

In [ ]:
```
# split into train and test sets
X_train_s, X_test_s, y_train_s, y_test_s= train_test_split(X_s, y_s, test_size=0.25, random_state
rf_s = RandomForestClassifier(n_estimators=200, max_depth=10, random_state=42)
rf_s.fit(X_train_s, y_train_s)
# make Predictions
y_pred_s = rf_s.predict(X_test_s)
print("Strukovna:\n")
calculate_metrics(y_test_s, y_pred_s)
```

Gradient boosting

In [ ]:
```
# split into train and test sets
X_train_g, X_test_g, y_train_g, y_test_g = train_test_split(X_g, y_g, test_size=0.25, random_stat
# initialize and train the Gradient Boosting Classifier
gb_clf_g = GradientBoostingClassifier(random_state=42)
gb_clf_g.fit(X_train_g, y_train_g)
# make predictions and evaluate the model
y_pred_g = gb_clf_g.predict(X_test_g)
print("Gimnazija:\n")
calculate_metrics(y_test_g, y_pred_g)
```

In [ ]:
```
# split into train and test sets
X_train_s, X_test_s, y_train_s, y_test_s= train_test_split(X_s, y_s, test_size=0.25, random_state
# initialize and train the Gradient Boosting Classifier
gb_clf_s = GradientBoostingClassifier(random_state=42)
gb_clf_s.fit(X_train_s, y_train_s)
# make predictions and evaluate the model
y_pred_s = gb_clf_s.predict(X_test_s)
print("Strukovna:\n")
calculate_metrics(y_test_s, y_pred_s)
```

SVM

In [ ]:
```
# split into train and test sets
X_train_g, X_test_g, y_train_g, y_test_g = train_test_split(X_g, y_g, test_size=0.25, random_stat
# define the parameter grid for classification
param_grid = {'C': [0.1, 1, 10],
              'gamma': ['scale','auto'],
              'kernel': ['linear', 'rbf']}
# initialize GridSearchCV
grid = GridSearchCV(SVC(), param_grid, refit=True, cv=3, n_jobs=-1)
grid.fit(X_train_g, y_train_g)
print("Best parameters found: ", grid.best_params_)
# evaluate the tuned model
grid_predictions_g = grid.predict(X_test_g)
print("Gimnazija:\n")
calculate_metrics(y_test_g, grid_predictions_g)
```

In [ ]:
```
# split into train and test sets
X_train_s, X_test_s, y_train_s, y_test_s= train_test_split(X_s, y_s, test_size=0.25, random_state
# define the parameter grid for classification
param_grid = {'C': [0.1, 1, 10],
              'gamma': ['scale','auto'],
              'kernel': ['linear', 'rbf']}
# initialize GridSearchCV
```

```
grid = GridSearchCV(SVC(), param_grid, refit=True, cv=3, n_jobs=-1)
grid.fit(X_train_s, y_train_s)
print("Best parameters found: ", grid.best_params_)
# evaluate the tuned model
grid_predictions_s = grid.predict(X_test_s)
print("Strukovna:\n")
calculate_metrics(y_test_s, grid_predictions_s)
```

By gender

In [ ]:
```
# split data into features (x) and target (y)
X_male = df_main_M[feature_names]
y_male = df_main_M[target_value]
X_female = df_main_F[feature_names]
y_female = df_main_F[target_value]
```

Logistic regression

In [ ]:
```
# split into train and test sets
X_train_m, X_test_m, y_train_m, y_test_m = train_test_split(X_male, y_male, test_size=0.25, rando
# initialize the Logistic Regression Model
log_reg_m = LogisticRegression()
# train the Model
log_reg_m.fit(X_train_m, y_train_m)
# make Predictions
y_pred_m = log_reg_m.predict(X_test_m)
print("Metrics for male:\n")
calculate_metrics(y_test_m, y_pred_m)
```

In [ ]:
```
# split into train and test sets
X_train_f, X_test_f, y_train_f, y_test_f = train_test_split(X_female, y_female, test_size=0.25, r
# initialize the Logistic Regression Model
log_reg_f = LogisticRegression()
# train the Model
log_reg_f.fit(X_train_f, y_train_f)
# make Predictions
y_pred_f = log_reg_f.predict(X_test_f)
print("Metrics for female:\n")
calculate_metrics(y_test_f, y_pred_f)
```

Decision tree

In [ ]:
```
# split into train and test sets
X_train_m, X_test_m, y_train_m, y_test_m = train_test_split(X_male, y_male, test_size=0.25, rando
# initialize the Decision Tree Classifier Model
decision_tree_m = DecisionTreeClassifier(random_state=42)
# train the Model
decision_tree_m.fit(X_train_m, y_train_m)
# make Predictions
y_pred_m = decision_tree_m.predict(X_test_m)
print("Metrics for male:\n")
calculate_metrics(y_test_m, y_pred_m)
```

In [ ]:
```
# split into train and test sets
X_train_f, X_test_f, y_train_f, y_test_f = train_test_split(X_female, y_female, test_size=0.25, r
# initialize the Decision Tree Classifier Model
decision_tree_f = DecisionTreeClassifier(random_state=42)
# train the Model
decision_tree_f.fit(X_train_f, y_train_f)
# make Predictions
y_pred_f = decision_tree_f.predict(X_test_f)
print("Metrics for female:\n")
calculate_metrics(y_test_f, y_pred_f)
```

Random forest

In [ ]:
```
# split into train and test sets
X_train_m, X_test_m, y_train_m, y_test_m = train_test_split(X_male, y_male, test_size=0.25, rando
# initialize the Random Forest Classifier Model
rf_m = RandomForestClassifier(n_estimators=200, max_depth=10, random_state=42)
# train the Model
rf_m.fit(X_train_m, y_train_m)
# make Predictions
y_pred_m = rf_m.predict(X_test_m)
print("Metrics for male:\n")
calculate_metrics(y_test_m, y_pred_m)
```

In [ ]:
```
# split into train and test sets
X_train_f, X_test_f, y_train_f, y_test_f = train_test_split(X_female, y_female, test_size=0.25, r
# initialize the Random Forest Classifier Model
rf_f = RandomForestClassifier(n_estimators=200, max_depth=10, random_state=42)
# train the Model
rf_f.fit(X_train_f, y_train_f)
# make Predictions
y_pred_f = rf_f.predict(X_test_f)
print("Metrics for female:\n")
calculate_metrics(y_test_f, y_pred_f)
```

Gradient boosting

In [ ]:
```
# split into train and test sets
X_train_m, X_test_m, y_train_m, y_test_m = train_test_split(X_male, y_male, test_size=0.25, rando
# initialize and train the Gradient Boosting Classifier
gb_clf_m = GradientBoostingClassifier(random_state=42)
gb_clf_m.fit(X_train_m, y_train_m)
# make predictions and evaluate the model
y_pred_m = gb_clf_m.predict(X_test_m)
print("Metrics for male:\n")
calculate_metrics(y_test_m, y_pred_m)
```

In [ ]:
```
# split into train and test sets
X_train_f, X_test_f, y_train_f, y_test_f = train_test_split(X_female, y_female, test_size=0.25, r
# initialize and train the Gradient Boosting Classifier
gb_clf_f = GradientBoostingClassifier(random_state=42)
gb_clf_f.fit(X_train_f, y_train_f)
# make predictions and evaluate the model
y_pred_f = gb_clf_f.predict(X_test_f)
print("Metrics for female:\n")
calculate_metrics(y_test_f, y_pred_f)
```

SVM

In [ ]:
```
# split into train and test sets
X_train_m, X_test_m, y_train_m, y_test_m = train_test_split(X_male, y_male, test_size=0.25, rando
# define the parameter grid for classification
param_grid = {'C': [0.1, 1, 10],
              'gamma': ['scale','auto'],
              'kernel': ['linear', 'rbf']}
# initialize GridSearchCV
grid = GridSearchCV(SVC(), param_grid, refit=True, cv=3, n_jobs=-1)
grid.fit(X_train_m, y_train_m)
print("Best parameters found: ", grid.best_params_)
# evaluate the tuned model
grid_predictions_m = grid.predict(X_test_m)
print("Metrics for male:\n")
calculate_metrics(y_test_m, grid_predictions_m)
```

In [ ]:
```
# split into train and test sets
```

```
X_train_f, X_test_f, y_train_f, y_test_f = train_test_split(X_female, y_female, test_size=0.25, r
# define the parameter grid for classification
param_grid = {'C': [0.1, 1, 10],
              'gamma': ['scale','auto'],
              'kernel': ['linear', 'rbf']}
# initialize GridSearchCV
grid = GridSearchCV(SVC(), param_grid, refit=True,  cv=3, n_jobs=-1)
grid.fit(X_train_f, y_train_f)
print("Best parameters found: ", grid.best_params_)
# evaluate the tuned model
grid_predictions_f = grid.predict(X_test_f)
print("Metrics for female:\n")
calculate_metrics(y_test_f, grid_predictions_f)
```

Adding new features (course enrollment)

In [40]:
```
# get the names of features and target value
feature_names = ['enrollment_year', 'birth_year','gender_dummy','gymnasiums_dummy', 'vocational_d
                 'art_dummy','enrollment_age', 'high_school_average_grade','outside_cro_high_scho
                 'Matematika I','Matematika II', 'Programiranje II']
target_value = ['dropout_dummy']
```

In [41]:
```
# split data into features (x) and target (y)
X = df_students_with_courses[feature_names]
y = df_students_with_courses[target_value]
```

Logistic regression

In [ ]:
```
# split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
# initialize the Logistic Regression Model
log_reg = LogisticRegression()
# train the Model
log_reg.fit(X_train, y_train)
# make Predictions
y_pred = log_reg.predict(X_test)
calculate_metrics(y_test, y_pred)
```

Decision tree

In [ ]:
```
# split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
# initialize the Decision Tree Classifier Model
decision_tree = DecisionTreeClassifier(random_state=42)
# train the Model
decision_tree.fit(X_train, y_train)
# make Predictions
y_pred = decision_tree.predict(X_test)
calculate_metrics(y_test, y_pred)
```

Random forest

In [ ]:
```
# split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
# initialize the Random Forest Classifier Model
rf = RandomForestClassifier(n_estimators=200, max_depth=10, random_state=42)
# train the Model
rf.fit(X_train, y_train)
# make Predictions
y_pred = rf.predict(X_test)
calculate_metrics(y_test, y_pred)
```

Gradient boosting

In [ ]:
```
# split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
# initialize and train the Gradient Boosting Classifier
gb_clf = GradientBoostingClassifier(random_state=42)
gb_clf.fit(X_train, y_train)
# make predictions and evaluate the model
y_pred = gb_clf.predict(X_test)
calculate_metrics(y_test, y_pred)
```

SVM

In [ ]:
```
# split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
# define the parameter grid for classification
param_grid = {'C': [0.1, 1, 10],
              'gamma': ['scale','auto'],
              'kernel': ['linear', 'rbf']}
# initialize GridSearchCV
grid = GridSearchCV(SVC(), param_grid, refit=True, cv=3, n_jobs=-1)
grid.fit(X_train, y_train)
print("Best parameters found: ", grid.best_params_)
# evaluate the tuned model
grid_predictions = grid.predict(X_test)
calculate_metrics(y_test, grid_predictions)
```

Second set of input features

Based on correlations with dropout variable:

- 'enrollment_year',
- 'birth_year',
- 'gender_dummy',
- 'gymnasiums_dummy',
- 'high_school_average_grade'
- 'state_matriculation_exam_points'
- 'total_points_for_study'
- 'english_language_A_%'
- 'english_language_B_%'
- 'croatian_language_%'

In [88]:
```
# get the names of features and target value
feature_names = ['enrollment_year', 'birth_year','gender_dummy','gymnasiums_dummy', 'high_school_
                 'total_points_for_study','english_language_A_%','english_language_B_%','croatian
target_value = ['dropout_dummy']
```

All data

In [ ]:
```
# split data into features (x) and target (y)
X = df_main_student_study[feature_names]
y = df_main_student_study[target_value]
```

Logistic regression

In [ ]:
```
# split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
# initialize the Logistic Regression Model
```

```
log_reg = LogisticRegression()
# train the Model
log_reg.fit(X_train, y_train)
# make Predictions
y_pred = log_reg.predict(X_test)
calculate_metrics(y_test, y_pred)
```

Decision tree

In [ ]:
```
# split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
# initialize the Decision Tree Classifier Model
decision_tree = DecisionTreeClassifier(random_state=42)
# train the Model
decision_tree.fit(X_train, y_train)
# make Predictions
y_pred = decision_tree.predict(X_test)
calculate_metrics(y_test, y_pred)
```

Random forest

In [ ]:
```
# split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
# initialize the Random Forest Classifier Model
rf = RandomForestClassifier(n_estimators=200, max_depth=10, random_state=42)
# train the Model
rf.fit(X_train, y_train)
# make Predictions
y_pred = rf.predict(X_test)
calculate_metrics(y_test, y_pred)
```

Gradient boosting

In [ ]:
```
# split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
# initialize and train the Gradient Boosting Classifier
gb_clf = GradientBoostingClassifier(random_state=42)
gb_clf.fit(X_train, y_train)
# make predictions and evaluate the model
y_pred = gb_clf.predict(X_test)
calculate_metrics(y_test, y_pred)
```

SVM

In [ ]:
```
# split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
# define the parameter grid for classification
param_grid = {'C': [0.1, 1, 10],
              'gamma': ['scale','auto'],
              'kernel': ['linear', 'rbf']}
# initialize GridSearchCV
grid = GridSearchCV(SVC(), param_grid, refit=True, cv=3, n_jobs=-1)
grid.fit(X_train, y_train)
print("Best parameters found: ", grid.best_params_)
# evaluate the tuned model
grid_predictions = grid.predict(X_test)
calculate_metrics(y_test, grid_predictions)
```

By location

In [ ]:
```
# split data into features (x) and target (y)
X_st = df_main_st[feature_names]
y_st = df_main_st[target_value]
X_van = df_main_van[feature_names]
y_van = df_main_van[target_value]
X_rh = df_main_rh[feature_names]
y_rh = df_main_rh[target_value]
```

Logistic regression

In [ ]:
```
# split into train and test sets
X_train_st, X_test_st, y_train_st, y_test_st = train_test_split(X_st, y_st, test_size=0.25, rando
# initialize the Logistic Regression Model
log_reg_st = LogisticRegression()
# train the Model
log_reg_st.fit(X_train_st, y_train_st)
# make Predictions
y_pred_st = log_reg_st.predict(X_test_st)
print("For Split:\n")
calculate_metrics(y_test_st, y_pred_st)
```

In [ ]:
```
# split into train and test sets
X_train_van, X_test_van, y_train_van, y_test_van= train_test_split(X_van, y_van, test_size=0.25,
log_reg_van = LogisticRegression()
log_reg_van.fit(X_train_van, y_train_van)
y_pred_van = log_reg_van.predict(X_test_van)
print("Outside Croatia:\n")
calculate_metrics(y_test_van, y_pred_van)
```

In [ ]:
```
# split into train and test sets
X_train_rh, X_test_rh, y_train_rh, y_test_rh= train_test_split(X_rh, y_rh, test_size=0.25, random
log_reg_rh = LogisticRegression()
log_reg_rh.fit(X_train_rh, y_train_rh)
y_pred_rh = log_reg_rh.predict(X_test_rh)
print("Outside Split:\n")
calculate_metrics(y_test_rh, y_pred_rh)
```

Decision tree

In [ ]:
```
# split into train and test sets
X_train_st, X_test_st, y_train_st, y_test_st = train_test_split(X_st, y_st, test_size=0.25, rando
# initialize the Decision Tree Classifier Model
decision_tree_st = DecisionTreeClassifier(random_state=42)
# train the Model
decision_tree_st.fit(X_train_st, y_train_st)
# make Predictions
y_pred_st = decision_tree_st.predict(X_test_st)
print("For Split:\n")
calculate_metrics(y_test_st, y_pred_st)
```

In [ ]:
```
# split into train and test sets
X_train_van, X_test_van, y_train_van, y_test_van= train_test_split(X_van, y_van, test_size=0.25,
decision_tree_van = DecisionTreeClassifier(random_state=42)
decision_tree_van.fit(X_train_van, y_train_van)
# make Predictions
y_pred_van = decision_tree_van.predict(X_test_van)
print("Outside Croatia:\n")
calculate_metrics(y_test_van, y_pred_van)
```

In [ ]:
```
# split into train and test sets
X_train_rh, X_test_rh, y_train_rh, y_test_rh= train_test_split(X_rh, y_rh, test_size=0.25, random
decision_tree_rh = DecisionTreeClassifier(random_state=42)
decision_tree_rh.fit(X_train_rh, y_train_rh)
# make Predictions
y_pred_rh = decision_tree_rh.predict(X_test_rh)
print("Outside Split:\n")
calculate_metrics(y_test_rh, y_pred_rh)
```

Random forest

In [ ]:
```
# split into train and test sets
X_train_st, X_test_st, y_train_st, y_test_st = train_test_split(X_st, y_st, test_size=0.25, rando
# initialize the Random Forest Classifier Model
rf_st = RandomForestClassifier(n_estimators=200, max_depth=10, random_state=42)
# train the Model
rf_st.fit(X_train_st, y_train_st)
# make Predictions
y_pred_st = rf_st.predict(X_test_st)
print("For Split:\n")
calculate_metrics(y_test_st, y_pred_st)
```

In [ ]:
```
# split into train and test sets
X_train_van, X_test_van, y_train_van, y_test_van= train_test_split(X_van, y_van, test_size=0.25,
rf_van = RandomForestClassifier(n_estimators=200, max_depth=10, random_state=42)
rf_van.fit(X_train_van, y_train_van)
# make Predictions
y_pred_van = rf_van.predict(X_test_van)
print("Outside Croatia:\n")
calculate_metrics(y_test_van, y_pred_van)
```

In [ ]:
```
# split into train and test sets
X_train_rh, X_test_rh, y_train_rh, y_test_rh= train_test_split(X_rh, y_rh, test_size=0.25, random
rf_rh = RandomForestClassifier(n_estimators=200, max_depth=10, random_state=42)
rf_rh.fit(X_train_rh, y_train_rh)
# make Predictions
y_pred_rh = rf_rh.predict(X_test_rh)
print("Outside Split:\n")
calculate_metrics(y_test_rh, y_pred_rh)
```

Gradient boosting

In [ ]:
```
# split into train and test sets
X_train_st, X_test_st, y_train_st, y_test_st = train_test_split(X_st, y_st, test_size=0.25, rando
# initialize and train the Gradient Boosting Classifier
gb_clf_st = GradientBoostingClassifier(random_state=42)
gb_clf_st.fit(X_train_st, y_train_st)
# make predictions and evaluate the model
y_pred_st = gb_clf_st.predict(X_test_st)
print("For Split:\n")
calculate_metrics(y_test_st, y_pred_st)
```

In [ ]:
```
# split into train and test sets
X_train_van, X_test_van, y_train_van, y_test_van=train_test_split(X_van, y_van, test_size=0.25, r
# initialize and train the Gradient Boosting Classifier
gb_clf_van = GradientBoostingClassifier(random_state=42)
gb_clf_van.fit(X_train_van, y_train_van)
# make predictions and evaluate the model
y_pred_van = gb_clf_van.predict(X_test_van)
print("Outside Croatia:\n")
calculate_metrics(y_test_van, y_pred_van)
```

In [ ]:
```
# split into train and test sets
X_train_rh, X_test_rh, y_train_rh, y_test_rh= train_test_split(X_rh, y_rh, test_size=0.25, random
# initialize and train the Gradient Boosting Classifier
gb_clf_rh = GradientBoostingClassifier(random_state=42)
gb_clf_rh.fit(X_train_rh, y_train_rh)
# make predictions and evaluate the model
y_pred_rh = gb_clf_rh.predict(X_test_rh)
print("Outside Split:\n")
calculate_metrics(y_test_rh, y_pred_rh)
```

SVM

In [ ]:
```
# split into train and test sets
X_train_st, X_test_st, y_train_st, y_test_st = train_test_split(X_st, y_st, test_size=0.25, rando
# define the parameter grid for classification
param_grid = {'C': [0.1, 1, 10],
              'gamma': ['scale','auto'],
              'kernel': ['linear', 'rbf']}
# initialize GridSearchCV
grid = GridSearchCV(SVC(), param_grid, refit=True, cv=3, n_jobs=-1)
grid.fit(X_train_st, y_train_st)
print("Best parameters found: ", grid.best_params_)
# evaluate the tuned model
grid_predictions_st = grid.predict(X_test_st)
print("For Split:\n")
calculate_metrics(y_test_st, grid_predictions_st)
```

In [ ]:
```
# split into train and test sets
X_train_van, X_test_van, y_train_van, y_test_van= train_test_split(X_van, y_van, test_size=0.25,
# define the parameter grid for classification
param_grid = {'C': [0.1, 1, 10],
              'gamma': ['scale','auto'],
              'kernel': ['linear', 'rbf']}
# initialize GridSearchCV
grid = GridSearchCV(SVC(), param_grid, refit=True,cv=3, verbose=2)
grid.fit(X_train_van, y_train_van)
print("Best parameters found: ", grid.best_params_)
# evaluate the tuned model
grid_predictions_van = grid.predict(X_test_van)
print("Outside Croatia:\n")
calculate_metrics(y_test_van, grid_predictions_van)
```

In [ ]:
```
# split into train and test sets
X_train_rh, X_test_rh, y_train_rh, y_test_rh= train_test_split(X_rh, y_rh, test_size=0.25, random
# define the parameter grid for classification
param_grid = {'C': [0.1, 1, 10],
              'gamma': ['scale','auto'],
              'kernel': ['linear', 'rbf']}
# initialize GridSearchCV
grid = GridSearchCV(SVC(), param_grid, refit=True, cv=3,n_jobs=-1)
grid.fit(X_train_rh, y_train_rh)
print("Best parameters found: ", grid.best_params_)
# evaluate the tuned model
grid_predictions_rh = grid.predict(X_test_rh)
print("Outside Split:\n")
calculate_metrics(y_test_rh, grid_predictions_rh)
```

By high school category

In [ ]:
```
# split data into features (x) and target (y)
X_g = df_main_gimn[feature_names]
y_g = df_main_gimn[target_value]
X_s = df_main_struk[feature_names]
y_s = df_main_struk[target_value]
```

Logistic regression

In [ ]:
```
# split into train and test sets
X_train_g, X_test_g, y_train_g, y_test_g = train_test_split(X_g, y_g, test_size=0.25, random_stat
# initialize the Logistic Regression Model
log_reg_g = LogisticRegression()
# train the Model
log_reg_g.fit(X_train_g, y_train_g)
# make Predictions
```

```
y_pred_g = log_reg_g.predict(X_test_g)
print("Gimnazija:\n")
calculate_metrics(y_test_g, y_pred_g)
```

In [ ]:
```
# split into train and test sets
X_train_s, X_test_s, y_train_s, y_test_s= train_test_split(X_s, y_s, test_size=0.25, random_state
log_reg_s = LogisticRegression()
log_reg_s.fit(X_train_s, y_train_s)
# make Predictions
y_pred_s = log_reg_s.predict(X_test_s)
print("Strukovna:\n")
calculate_metrics(y_test_s, y_pred_s)
```

Decision tree

In [ ]:
```
# split into train and test sets
X_train_g, X_test_g, y_train_g, y_test_g = train_test_split(X_g, y_g, test_size=0.25, random_stat
# initialize the Decision Tree Classifier Model
decision_tree_g = DecisionTreeClassifier(random_state=42)
# train the Model
decision_tree_g.fit(X_train_g, y_train_g)
# make Predictions
y_pred_g = decision_tree_g.predict(X_test_g)
print("Gimnazija:\n")
calculate_metrics(y_test_g, y_pred_g)
```

In [ ]:
```
# split into train and test sets
X_train_s, X_test_s, y_train_s, y_test_s= train_test_split(X_s, y_s, test_size=0.25, random_state
decision_tree_s = DecisionTreeClassifier(random_state=42)
decision_tree_s.fit(X_train_s, y_train_s)
# make Predictions
y_pred_s = decision_tree_s.predict(X_test_s)
print("Strukovna:\n")
calculate_metrics(y_test_s, y_pred_s)
```

Random forest

In [ ]:
```
# split into train and test sets
X_train_g, X_test_g, y_train_g, y_test_g = train_test_split(X_g, y_g, test_size=0.25, random_stat
# initialize the Random Forest Classifier Model
rf_g = RandomForestClassifier(n_estimators=200, max_depth=10, random_state=42)
# train the Model
rf_g.fit(X_train_g, y_train_g)
# make Predictions
y_pred_g = rf_g.predict(X_test_g)
print("Gimnazija:\n")
calculate_metrics(y_test_g, y_pred_g)
```

In [ ]:
```
# split into train and test sets
X_train_s, X_test_s, y_train_s, y_test_s= train_test_split(X_s, y_s, test_size=0.25, random_state
rf_s = RandomForestClassifier(n_estimators=200, max_depth=10, random_state=42)
rf_s.fit(X_train_s, y_train_s)
# make Predictions
y_pred_s = rf_s.predict(X_test_s)
print("Strukovna:\n")
calculate_metrics(y_test_s, y_pred_s)
```

Gradient boosting

In [ ]:
```
# split into train and test sets
X_train_g, X_test_g, y_train_g, y_test_g = train_test_split(X_g, y_g, test_size=0.25, random_stat
# initialize and train the Gradient Boosting Classifier
gb_clf_g = GradientBoostingClassifier(random_state=42)
gb_clf_g.fit(X_train_g, y_train_g)
# make predictions and evaluate the model
y_pred_g = gb_clf_g.predict(X_test_g)
print("Gimnazija:\n")
calculate_metrics(y_test_g, y_pred_g)
```

In [ ]:
```
# split into train and test sets
X_train_s, X_test_s, y_train_s, y_test_s= train_test_split(X_s, y_s, test_size=0.25, random_state
# initialize and train the Gradient Boosting Classifier
gb_clf_s = GradientBoostingClassifier(random_state=42)
gb_clf_s.fit(X_train_s, y_train_s)
# make predictions and evaluate the model
y_pred_s = gb_clf_s.predict(X_test_s)
print("Strukovna:\n")
calculate_metrics(y_test_s, y_pred_s)
```

SVM

In [ ]:
```
# split into train and test sets
X_train_g, X_test_g, y_train_g, y_test_g = train_test_split(X_g, y_g, test_size=0.25, random_stat
# define the parameter grid for classification
param_grid = {'C': [0.1, 1, 10],
              'gamma': ['scale','auto'],
              'kernel': ['linear', 'rbf']}
# initialize GridSearchCV
grid = GridSearchCV(SVC(), param_grid, refit=True, cv=3, n_jobs=-1)
grid.fit(X_train_g, y_train_g)
print("Best parameters found: ", grid.best_params_)
# evaluate the tuned model
grid_predictions_g = grid.predict(X_test_g)
print("Gimnazija:\n")
calculate_metrics(y_test_g, grid_predictions_g)
```

In [ ]:
```
# split into train and test sets
X_train_s, X_test_s, y_train_s, y_test_s= train_test_split(X_s, y_s, test_size=0.25, random_state
# define the parameter grid for classification
param_grid = {'C': [0.1, 1, 10],
              'gamma': ['scale','auto'],
              'kernel': ['linear', 'rbf']}
# initialize GridSearchCV
grid = GridSearchCV(SVC(), param_grid, refit=True, cv=3, n_jobs=-1)
grid.fit(X_train_s, y_train_s)
print("Best parameters found: ", grid.best_params_)
# evaluate the tuned model
grid_predictions_s = grid.predict(X_test_s)
print("Strukovna:\n")
calculate_metrics(y_test_s, grid_predictions_s)
```

By gender

In [ ]:
```
# split data into features (x) and target (y)
X_male = df_main_M[feature_names]
y_male = df_main_M[target_value]
X_female = df_main_F[feature_names]
y_female = df_main_F[target_value]
```

Logistic regression

In [ ]:
```
# split into train and test sets
X_train_m, X_test_m, y_train_m, y_test_m = train_test_split(X_male, y_male, test_size=0.25, rando
# initialize the Logistic Regression Model
log_reg_m = LogisticRegression()
# train the Model
log_reg_m.fit(X_train_m, y_train_m)
# make Predictions
```

```
y_pred_m = log_reg_m.predict(X_test_m)
print("Metrics for male:\n")
calculate_metrics(y_test_m, y_pred_m)
```

In [ ]:
```
# split into train and test sets
X_train_f, X_test_f, y_train_f, y_test_f = train_test_split(X_female, y_female, test_size=0.25, r
# initialize the Logistic Regression Model
log_reg_f = LogisticRegression()
# train the Model
log_reg_f.fit(X_train_f, y_train_f)
# make Predictions
y_pred_f = log_reg_f.predict(X_test_f)
print("Metrics for female:\n")
calculate_metrics(y_test_f, y_pred_f)
```

Decision tree

In [ ]:
```
# split into train and test sets
X_train_m, X_test_m, y_train_m, y_test_m = train_test_split(X_male, y_male, test_size=0.25, rando
# initialize the Decision Tree Classifier Model
decision_tree_m = DecisionTreeClassifier(random_state=42)
# train the Model
decision_tree_m.fit(X_train_m, y_train_m)
# make Predictions
y_pred_m = decision_tree_m.predict(X_test_m)
print("Metrics for male:\n")
calculate_metrics(y_test_m, y_pred_m)
```

In [ ]:
```
# split into train and test sets
X_train_f, X_test_f, y_train_f, y_test_f = train_test_split(X_female, y_female, test_size=0.25, r
# initialize the Decision Tree Classifier Model
decision_tree_f = DecisionTreeClassifier(random_state=42)
# train the Model
decision_tree_f.fit(X_train_f, y_train_f)
# make Predictions
y_pred_f = decision_tree_f.predict(X_test_f)
print("Metrics for female:\n")
calculate_metrics(y_test_f, y_pred_f)
```

Random forest

In [ ]:
```
# split into train and test sets
X_train_m, X_test_m, y_train_m, y_test_m = train_test_split(X_male, y_male, test_size=0.25, rando
# initialize the Random Forest Classifier Model
rf_m = RandomForestClassifier(n_estimators=200, max_depth=10, random_state=42)
# train the Model
rf_m.fit(X_train_m, y_train_m)
# make Predictions
y_pred_m = rf_m.predict(X_test_m)
print("Metrics for male:\n")
calculate_metrics(y_test_m, y_pred_m)
```

In [ ]:
```
# split into train and test sets
X_train_f, X_test_f, y_train_f, y_test_f = train_test_split(X_female, y_female, test_size=0.25, r
# initialize the Random Forest Classifier Model
rf_f = RandomForestClassifier(n_estimators=200, max_depth=10, random_state=42)
# train the Model
rf_f.fit(X_train_f, y_train_f)
# make Predictions
y_pred_f = rf_f.predict(X_test_f)
print("Metrics for female:\n")
calculate_metrics(y_test_f, y_pred_f)
```

Gradient boosting

In [ ]:
```
# split into train and test sets
X_train_m, X_test_m, y_train_m, y_test_m = train_test_split(X_male, y_male, test_size=0.25, rando
# initialize and train the Gradient Boosting Classifier
gb_clf_m = GradientBoostingClassifier(random_state=42)
gb_clf_m.fit(X_train_m, y_train_m)
# make predictions and evaluate the model
y_pred_m = gb_clf_m.predict(X_test_m)
print("Metrics for male:\n")
calculate_metrics(y_test_m, y_pred_m)
```

In [ ]:
```
# split into train and test sets
X_train_f, X_test_f, y_train_f, y_test_f = train_test_split(X_female, y_female, test_size=0.25, r
# initialize and train the Gradient Boosting Classifier
gb_clf_f = GradientBoostingClassifier(random_state=42)
gb_clf_f.fit(X_train_f, y_train_f)
# make predictions and evaluate the model
y_pred_f = gb_clf_f.predict(X_test_f)
print("Metrics for female:\n")
calculate_metrics(y_test_f, y_pred_f)
```

SVM

In [ ]:
```
# split into train and test sets
X_train_m, X_test_m, y_train_m, y_test_m = train_test_split(X_male, y_male, test_size=0.25, rando
# define the parameter grid for classification
param_grid = {'C': [0.1, 1, 10],
              'gamma': ['scale','auto'],
              'kernel': ['linear', 'rbf']}
# initialize GridSearchCV
grid = GridSearchCV(SVC(), param_grid, refit=True, cv=3, n_jobs=-1)
grid.fit(X_train_m, y_train_m)
print("Best parameters found: ", grid.best_params_)
# evaluate the tuned model
grid_predictions_m = grid.predict(X_test_m)
print("Metrics for male:\n")
calculate_metrics(y_test_m, grid_predictions_m)
```

In [ ]:
```
# split into train and test sets
X_train_f, X_test_f, y_train_f, y_test_f = train_test_split(X_female, y_female, test_size=0.25, r
# define the parameter grid for classification
param_grid = {'C': [0.1, 1, 10],
              'gamma': ['scale','auto'],
              'kernel': ['linear', 'rbf']}
# initialize GridSearchCV
grid = GridSearchCV(SVC(), param_grid, refit=True,  cv=3, n_jobs=-1)
grid.fit(X_train_f, y_train_f)
print("Best parameters found: ", grid.best_params_)
# evaluate the tuned model
grid_predictions_f = grid.predict(X_test_f)
print("Metrics for female:\n")
calculate_metrics(y_test_f, grid_predictions_f)
```

Adding new features (course enrollment)

In [47]:
```
# get the names of features and target value
feature_names = ['enrollment_year', 'birth_year','gender_dummy','gymnasiums_dummy', 'high_school_
                 'state_matriculation_exam_points', 'total_points_for_study','english_language_A_
                 'english_language_B_%','croatian_language_%',
                 'Matematika I','Matematika II', 'Programiranje II']
target_value = ['dropout_dummy']
```

In [48]:
```
# split data into features (x) and target (y)
X = df_students_with_courses[feature_names]
y = df_students_with_courses[target_value]
```

Logistic regression

In [ ]:
```
# split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
# initialize the Logistic Regression Model
log_reg = LogisticRegression()
# train the Model
log_reg.fit(X_train, y_train)
# make Predictions
y_pred = log_reg.predict(X_test)
calculate_metrics(y_test, y_pred)
```

Decision tree

In [ ]:
```
# split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
# initialize the Decision Tree Classifier Model
decision_tree = DecisionTreeClassifier(random_state=42)
# train the Model
decision_tree.fit(X_train, y_train)
# make Predictions
y_pred = decision_tree.predict(X_test)
calculate_metrics(y_test, y_pred)
```

Random forest

In [ ]:
```
# split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
# initialize the Random Forest Classifier Model
rf = RandomForestClassifier(n_estimators=200, max_depth=10, random_state=42)
# train the Model
rf.fit(X_train, y_train)
# make Predictions
y_pred = rf.predict(X_test)
calculate_metrics(y_test, y_pred)
```

Gradient boosting

In [ ]:
```
# split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
# initialize and train the Gradient Boosting Classifier
gb_clf = GradientBoostingClassifier(random_state=42)
gb_clf.fit(X_train, y_train)
# make predictions and evaluate the model
y_pred = gb_clf.predict(X_test)
calculate_metrics(y_test, y_pred)
```

SVM

In [ ]:
```
# split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
# define the parameter grid for classification
param_grid = {'C': [0.1, 1, 10],
              'gamma': ['scale','auto'],
              'kernel': ['linear', 'rbf']}
# initialize GridSearchCV
grid = GridSearchCV(SVC(), param_grid, refit=True, cv=3, n_jobs=-1)
grid.fit(X_train, y_train)
print("Best parameters found: ", grid.best_params_)
```

```
# evaluate the tuned model
grid_predictions = grid.predict(X_test)
calculate_metrics(y_test, grid_predictions)
```