UNIVERZA V LJUBLJANI
EKONOMSKA FAKULTETA

MAGISTRSKO DELO

RAZVOJ PROGRAMSKE REŠITVE ZA ANALIZIRANJE CEN DELNIC

Ljubljana, december 2024 ROK ZUPAN

IZJAVA O AVTORSTVU

Podpisani Rok Zupan, študent Ekonomske fakultete Univerze v Ljubljani, avtor predloženega dela z naslovom Razvoj programske rešitve za analiziranje cen delnic, pripravljenega v sodelovanju s svetovalcem red. prof. dr. Mirom Gradišarjem

IZJAVLJAM

1. da sem predloženo delo pripravil samostojno;

2. da je tiskana oblika predloženega dela istovetna njegovi elektronski obliki;

3. da je besedilo predloženega dela jezikovno korektno in tehnično pripravljeno v skladu z Navodili za izdelavo zaključnih nalog Ekonomske fakultete Univerze v Ljubljani, kar pomeni, da sem poskrbel, da so dela in mnenja drugih avtorjev oziroma avtoric, ki jih uporabljam oziroma navajam v besedilu, citirana oziroma povzeta v skladu z Navodili za izdelavo zaključnih nalog Ekonomske fakultete Univerze v Ljubljani;

4. da se zavedam, da je plagiatorstvo – predstavljanje tujih del (v pisni ali grafični obliki) kot mojih lastnih – kaznivo po Kazenskem zakoniku Republike Slovenije;

5. da se zavedam posledic, ki bi jih na osnovi predloženega dela dokazano plagiatorstvo lahko predstavljalo za moj status na Ekonomski fakulteti Univerze v Ljubljani v skladu z relevantnim pravilnikom;

6. da sem pridobil vsa potrebna dovoljenja za uporabo podatkov in avtorskih del v predloženem delu in jih v njem jasno označil;

7. da sem pri pripravi predloženega dela ravnal v skladu z etičnimi načeli in, kjer je to potrebno, za raziskavo pridobil soglasje etične komisije;

8. da soglašam, da se elektronska oblika predloženega dela uporabi za preverjanje podobnosti vsebine z drugimi deli s programsko opremo za preverjanje podobnosti vsebine, ki je povezana s študijskim informacijskim sistemom članice;

9. da na Univerzo v Ljubljani neodplačno, neizključno, prostorsko in časovno neomejeno prenašam pravico shranitve predloženega dela v elektronski obliki, pravico reproduciranja ter pravico dajanja predloženega dela na voljo javnosti na svetovnem spletu prek Repozitorija Univerze v Ljubljani;

10. da hkrati z objavo predloženega dela dovoljujem objavo svojih osebnih podatkov, ki so navedeni v njem in v tej izjavi;

11. da sem preveril verodostojnost informacij, ki izhajajo iz zapisov na podlagi uporabe orodij umetne inteligence.

V Ljubljani, dne 23.01.2025 Podpis študenta: Rok zupan

KAZALO

1 UVOD 1
2 PREGLED LITERATURE 3
2.1 Trendi v trgovanju z delnicami 3
2.1.1 Visokofrekvenčno trgovanje 3
2.1.2 Trgovanje s pomočjo umetne inteligence in strojnega učenja 3
2.1.3 Povečanje individualnega vlaganja in spletnih trgovalnih platform 4
2.2 Algoritmi za napovedovanje cen delnic 5
2.3 Pridobivanje in obdelava podatkov s spleta 8
3 METODOLOGIJA 11
3.1 Metode razvoja programske opreme 12
3.2 Izbor tehnologij in orodij 13
3.2.1 Programski jezik 13
3.2.2 Spletno ogrodje 20
3.2.3 Programske knjižnice 22
3.2.3.1 Pandas 22
3.2.3.2 YFinance 23
3.2.3.3 Requests 23
3.2.3.4 Scikit-learn 24
3.2.3.5 Keras 24
3.2.4 Integrirano razvojno okolje 25
4 RAZVOJ PROGRAMSKE REŠITVE 25
4.1 Implementacija prototipa in osnovni opis 26
4.2 Pridobivanje podatkov 28
4.3 Uporabniški vmesnik 30
4.4 Obdelava podatkov 35
4.5 Analiza podatkov 39
4.6 Algoritmi za napovedovanje 42
5 TESTIRANJE IN REZULTATI 53
5.1 Preverjanje veljavnosti modelov 53
5.2 Analiza rezultatov 54

6 SKLEP 59
LITERATURA IN VIRI 61
PRILOGA 65

KAZALO TABEL

Tabela 1: Struktura aplikacije Dash 21
Tabela 2: Spremenljivke in ključi delniških informacij 36
Tabela 3: Tehnični kazalniki delnice 47
Tabela 4: Izračun metrik natančnosti modela za delnico IBM 55
Tabela 5: Izračun metrik natančnosti modela za delnico AAPL 55
Tabela 6: Izračun metrik natančnosti modela za delnico GOOGL 56
Tabela 7: Izračun metrik natančnosti modela za delnico AMZN 57
Tabela 8: Rezultati testiranja LSTM-modela 58
Tabela 9: Dodatni rezultati testiranja LSTM-modela 59

KAZALO SLIK

Slika 1: Prikaz LSTM-celice 8
Slika 2: Tehnike obdelave podatkov 11
Slika 3: Diagram poteka if-else pogojnega stavka 17
Slika 4: Diagram poteka while pogojnega stavka 17
Slika 5: Definicija funkcije 19
Slika 6: Implementacija prototipa 28
Slika 7: Uporabniški vmesnik 30
Slika 8: Zaslonska slika tretjega zavihka 41
Slika 9: Zaslonska slika prvega zavihka 43
Slika 10: Združitev spremenljivk df in inflation 46
Slika 11: Združitev spremenljivk df in unemployment 47
Slika 12: Zaslonska slika drugega zavihka 52

KAZALO PRILOG

Priloga 1: Programska koda 1

SEZNAM KRATIC

angl. - angleško
API – (angl. application programming interface); aplikacijsko-programski vmesnik
ATR – (angl Average True Range); povprečni pravi razpon
BDP – bruto domači proizvod
CSS – (angl. Cascading Style Sheets); kaskadne stilske podloge
CSV – (angl. Comma-Separated Values); vrednosti, ločene z vejico
EMA – (ang. Exponential Moving Average); eksponentno drseče povprečje
FDD – (angl. Feature-Driven development); funkcionalno voden razvoj
FINRA – (angl. Financial Industry Regulatory Authority); Ameriška agencija za nadzor nad finančno industrijo
FRED – (angl. Federal Reserve Economic Data); federalni rezervni ekonomski podatki
HTML – (angl. Hyper Text Markup Language); jezik za označevanje nadbesedila
HTTP – (angl. HyperText Transfer Protocol); protokol za prenos besedila
IKT – informacijsko-komunikacijska tehnologija
IoT – (angl. Internet of Things); internet stvari
JSON – (angl. JavaScript Object Notation); zapis predmeta JavaScript
LSTM – (angl. Long Short-Term Memory); dolgotrajno kratkoročni spomin
MACD – (ang. Moving Average Convergence/Divergence); konvergenca/divergenca drsečih povprečij
MAE – (angl. Mean absolute error); povprečna absolutna napaka
MAPE – (angl. Mean Absolute Percentage Error); povprečna absolutna odstotna napaka
OBV – (angl. On-Balance Volume); ravnovesni volumen
R^2 – (angl. Coefficient of determination); koeficient determinacije
REPL – (angl. Read–Eval–Print Loop); beri-oceni-natisni zanki
RMSE – (angl. Root-Mean-Square Deviation); povprečni kvadratni odklon
RSI – (angl. Relative Strength Index); indeks relativne moči
SMA – (ang. Simple Moving Average); preprosto drseče povprečje
SSL – (angl. Secure Sockets Layer); varnostna vtična plast
SQL – (angl. Structured Query Language); strukturirani povpraševalni jezik za delo s podatkovnimi bazami
SU – strojno učenje
UI – umetna inteligenca
TDD – (angl. Test Driven Development); testno voden razvoj
XML – (angl. Extensible Markup Language); razširljivi označevalni jezik
ZDA – Združene države Amerike

1 UVOD

Tehnološki napredek je oblikoval finančni svet. Pred izumom telegrafa leta 1844 in telefona leta 1876 je bila komunikacija na trgu vrednostnih papirjev primitivna, saj so za prenos informacij med trgi uporabljali ljudi in golobe (Markham, 2002).

Vloga tehnologije v finančnem sektorju se je od sredine 20. stoletja močno povečala. Hardy (1980) je v svoji študiji raziskoval povezavo med bruto domačim proizvodom (v nadaljevanju BDP) in številom telefonskih linij na osebo v 45 državah med letoma 1960 in 1973. Ugotovil je pomembno povezavo med širjenjem telefonskih storitev in gospodarsko rastjo, pri čemer večja dostopnost in uporaba telefonskih storitev prispevata k izboljšanju komunikacijskih zmogljivosti. To pozitivno vpliva na trgovanje, produktivnost in poslovanje.

Njegova študija je postavila temelje za nadaljnji razvoj, ki je hitro sledil z razširitvijo osebnih računalnikov in digitalnih komunikacij. Tehnološke inovacije so ustvarile močno povezan svet, kakršnega se pozna danes. Leff (1984) je raziskoval, kako lahko telekomunikacije zmanjšajo stroške prenosa informacij, kar vodi do boljših in bolj racionalnih odločitev, saj so ključne informacije lažje dostopne tistim, ki sprejemajo odločitve.

Ker bo programska rešitev, ki jo izdelam v tem magistrskem delu, del telekomunikacijske infrastrukture, je smiselno proučiti raziskavo Maddena in Savaga (1998). Madden in Savage se osredotočata na pomen vlaganj v telekomunikacijsko infrastrukturo kot ključnega dejavnika za spodbujanje gospodarske rasti v tranzicijskih gospodarstvih Srednje in Vzhodne Evrope. Avtorji predlagajo nadgradnjo in širitev telekomunikacijskega omrežja, saj bi to lahko spodbudilo gospodarsko rast.

Svetovni splet je v 90. letih prejšnjega stoletja začel revolucionirati tudi finančni sektor. Z razširjenostjo interneta so se finančne institucije hitro prilagodile novim tehnologijam. Pojavila se je elektronska trgovina, ki je omogočila nakup in prodajo finančnih instrumentov prek spleta, kar je drastično povečalo hitrost in obseg trgovanja. Banke in borze so začele uporabljati spletne platforme za komunikacijo s strankami, izvedbo transakcij in ponudbo finančnih storitev na daljavo.

V zadnjih letih pa se za prenos informacij o trgovanju uporabljajo optični kabli in mikrovalovni stolpi, kar omogoča prenos informacij s svetlobno hitrostjo. Obstaja skupina trgovcev, oboroženih z zapletenimi algoritmi, ki so pripravljeni porabiti znatne vsote denarja za dostop do teh najsodobnejših zmogljivosti. Plačujejo, da postavijo svoje strežnike blizu ali znotraj borz, kar jim omogoča hitrostno prednost, ki jo potrebujejo za svoje trgovalne strategije (Zaharudin in drugi, 2022).

Glede na to, da je razvoj interneta in informacijske tehnologije pospešil pretok informacij, kapitala in idej, je pomembno upoštevati tudi študijo Beila in drugih (2005). Ta raziskava, v nasprotju s prejšnjo študijo Maddena in Savaga (1998), ugotavlja enosmerno pozitivno

povezavo med BDP in investicijami v telekomunikacije, medtem ko vpliv investicij v informacijsko-komunikacijsko tehnologijo (v nadaljevanju IKT) na BDP ni bil tako izrazit. Avtorji so proučevali stanje na trgu v Združenih državah Amerike (v nadaljevanju ZDA) med letoma 1947 in 1996.

Čeprav obstoječe spletne platforme, kot sta Yahoo Finance in Bloomberg, ponujajo obsežne finančne informacije, novice in analize o različnih finančnih trgih in podjetjih, imata ti rešitvi omejitve, ki vplivajo na učinkovitost uporabnikov.

Pomanjkljivosti v obstoječih rešitvah so naslednje:

- Uporabniška izkušnja: uporabniški vmesniki so pogosto zapleteni in neintuitivni, kar vpliva na uporabniško izkušnjo in povečuje verjetnost napak. Dizajn uporabniškega vmesnika lahko vpliva na odločitve uporabnikov (Antoniou, 2021).
- Nedostopnost naprednih analitičnih orodij: nekatere platforme ne nudijo naprednih analitičnih orodij, kot so modeli strojnega učenja za napovedovanje cen, kar vlagateljem otežuje analizo tržnih trendov (Khafaji, 2024).
- Neizkoriščenost vrednosti podatkov: s pomočjo podatkovne analize se lahko izboljša sprejemanje odločitev na podlagi opisnih in napovednih vpogledov, kar omogoči pravi izkoristek podatkov (Khan, 2024).

Namen magistrskega dela je izboljšati informacijsko podporo pri trgovanju z vrednostnimi papirji. Želim razviti novo programsko rešitev, ki bo vlagateljem omogočila enostaven dostop do ključnih podatkov iz različnih virov in integracijo teh podatkov z ekonomskimi kazalniki. Rešitev bo vključevala orodja za napovedovanje cen delnic, ki temeljijo na zgodovinskih podatkih in modelu za strojno učenje, kar bo vlagateljem omogočilo informirane in natančne trgovalne odločitve (Eldomiaty in drugi, 2020; Fischer in Krauss, 2018).

Cilje magistrskega dela sem razdelil glede na postopek izdelave programske rešitve:

1. Integracija in analiza podatkov z več virov: vlagateljem omogočiti enostaven dostop do podatkov in njihove analize z različnih spletnih mest.
2. Uporabniku prijazen vmesnik in ureditev podatkov: razviti enostaven in intuitiven uporabniški vmesnik ter urediti podatke v obliko, ki jo uporabnik zlahka razume.
3. Izračun ekonomskih kazalnikov: na podlagi urejenih podatkov izračunati ekonomske kazalnike za napovedovanje cen delnic.
4. Izdelava modela napovedovanja cen delnic: z združevanjem tržnih podatkov in ekonomskih kazalnikov omogočiti razvoj naprednih trgovalnih strategij.

Uporabil sem metodo pregleda literature in temeljil na elementih različnih agilnih metod razvoja programske opreme. Začel sem s pregledovanjem literature o možnih tehnologijah in orodjih. Nadaljeval sem s pregledom literature o delnicah in trgovanju ter se seznanil z

delovanjem algoritmov za napovedovanje cen delnic ter kako zanesljivo pridobivati podatke s spleta. Nato sem se lotil razvoja spletne rešitve in pridobivanja ustreznih podatkov za izračun ekonomskih kazalnikov. S pomočjo pripravljenih in prečiščenih podatkov sem izdelal modele za napoved cen delnic. Sledila sta testiranje modelov in preverjanje njihove veljavnosti. Prekrivanje med posameznimi koraki je bilo pričakovano; vsak korak pri pisanju naloge in razvoju rešitve sem podkrepil z branjem strokovne literature. Pisanje magistrskega dela in razvoj programske rešitve sta potekala vzporedno. Poleg strokovne literature sem si pri razvoju pomagal tudi s spletnimi mesti, forumi in skupnostmi, ki jih programerji uporabljajo za medsebojno sodelovanje. Pri pripravi programske kode sem si pomagal tudi z orodji generativne umetne inteligence

2 PREGLED LITERATURE

2.1 Trendi v trgovanju z delnicami

Po navedbah Ameriške komisije za vrednostne papirje in borzo je sodobna tehnologija vplivala na razvoj borznega trga, kar omogoča vlagateljem, da pridobijo informacije o podjetjih in njihovih delnicah. Internet zagotavlja ažurne cene delnic in novice o podjetjih, ki izdajajo delnice. Te zadnje informacije je mogoče posredovati strankam, podjetja pa lahko spremljajo uspešnost svojih delnic. Posledica teh informacij je, da vlagateljem, trgovcem in svetovalcem omogočajo popoln vpogled v situacijo (Chang, 2021).

2.1.1 Visokofrekvenčno trgovanje

Trend, ki se je pojavil na trgu trgovanja z delnicami, je visokofrekvenčno trgovanje, ki pa nima uradne jasne definicije. Različne institucije, kot so Ameriška komisija za vrednostne papirje in borzo, Nizozemski organ za finančne trge, Avstralska komisija za vrednostne papirje in naložbe in Londonska borza, vse drugače definirajo visokofrekvenčno trgovanje. Čeprav ni skupne definicije visokofrekvenčnega trgovanja, obstajajo nekatere značilnosti, ki ga ločujejo od drugih oblik trgovanja, kot so uporaba sofisticiranih računalniških programov in sistemov, zelo kratki časovni intervali držanja pozicij ter velika količina naročil z zelo majhnimi maržami dobička. V ZDA je tržni delež visokofrekvenčnega trgovanja dosegel vrhunec okoli 60 % celotnega trgovanja z delnicami leta 2009, kar je tržni delež od leta 2005 povečalo za približno 20 %. Delež se je postopoma zmanjšal na približno 50 % leta 2013 in ostal na podobni ravni tudi leta 2018 – 52 % (Zaharudin in drugi, 2022).

2.1.2 Trgovanje s pomočjo umetne inteligence in strojnega učenja

Umetna inteligenca (v nadaljevanju UI) je študija oblikovanja algoritmov, ki izvajajo naloge ali vedenja, za katera bi oseba menila, da zahtevajo inteligenco, če bi jih opravil človek. Inteligentni sistemi lahko imajo različne oblike: lahko gre za sistem, ki je zasnovan tako, da

ga ljudje ne zaznajo; govorne pomočnike, kot so Alexa, Siri, Cortana ali Google Assistant; avtonomna vozila; priporočila na spletnih trgovskih mestih. Ko so ti sistemi sposobni samostojno sprejeti nekatere odločitve na podlagi določenih ciljev, se imenujejo agenti (Riedl, 2019).

Strojno učenje (v nadaljevanju SU) se nanaša na uporabo računalniških algoritmov, ki omogočajo sistemom, da se samodejno učijo in izboljšajo podatke, da napovedujejo ali razvrščajo naravo podatkov s pomočjo vzorcev. Na splošno je SU področje znotraj UI, ki omogoča, da sistemi samostojno sprejemajo odločitve brez zunanje podpore. Odločitev se sprejme z iskanjem skritih plasti vzorcev znotraj kompleksnih podatkov (Sen in drugi, 2021).

Število delničarjev in podjetij na borzi raste, mnogi so poskusili najti rešitev za napoved prihodnjih trendov na borzi, kar predstavlja izziv zaradi zapletenih dejavnikov, ki vplivajo na cene. Tu nastopi UI, ki lahko igra ključno vlogo. V okviru UI se razvijajo algoritmi SU, ki jih raziskovalci uporabljajo za namen ugotavljanja vzorcev podatkov, izračunov tveganja in napoved prihodnjih naložb. Obstajata dve glavni teoretični hipotezi na tem področju:

- Hipoteza učinkovitih trgov trdi, da je trenutna tržna cena odziv na nedavno objavljene novice, zaradi česar so tržne cene vedno nepredvidljive.
- Hipoteza prilagodljivih trgov poskuša najti povezave med hipotezo učinkovitih trgov in psihološkimi teorijami, ki poskušajo razložiti trende na trgih.

Na podlagi hipoteze prilagodljivih trgov obstajata dve glavni analitični metodi: temeljna analiza in tehnična analiza. Temeljna analiza ocenjuje intrinzično vrednost delnice na podlagi podatkov, kot so bilanca stanja, ekonomski kazalniki in vedenje potrošnikov. Tehnična analiza pa proučuje zgodovinske podatke o cenah delnic in matematične kazalnike, kot so indeks relativne moči, premikajoče povprečje konvergence in divergence in indeks denarnega toka in sprejema odločitve o trgovanju. Na splošno je pri temeljni analizi podatke težje obdelati za učenje modelov strojnega učenja zaradi njihove neurejenosti, v nasprotju s tehnično analizo, kjer so zgodovinski podatki o cenah strukturirani in javno dostopni (Mokhtari in drugi, 2021).

2.1.3 Povečanje individualnega vlaganja in spletnih trgovalnih platform

Individualni vlagatelji so pomembni in koristni udeleženci na delniških trgih, inovacije v zadnjih letih pri dostopu individualnih vlagateljev do trgov pa so pripeljale do večjega števila in raznolikosti vlagateljev. Dejstvo, da mali vlagatelji ravnajo drugače kot institucionalni vlagatelji in včasih tudi različno med seboj, je lahko še posebej dragoceno v obdobjih tržne nestabilnosti. Kadar institucionalna likvidnost usahne, lahko trgovanje malih vlagateljev pripomore k znižanju razlik med ponudbenimi in povpraševalnimi cenami ter zmanjša vpliv poslov na ceno (Schulp, 2021)

Načelo »več je bolje« – čeprav ni brezhibno – običajno velja, ko je govora o udeležencih na delniškem trgu. Kljub slabšalnim izrazom za male vlagatelje mali vlagatelji veliko prispevajo k trgovanju. Več vlagateljev pomeni več informacij, kar koristi vsem tržnim udeležencem, saj pomaga vzpostaviti bolj učinkovite cene. Več kapitala, ki ga vlagatelji vložijo, pomaga financirati gospodarsko rast. Vlagatelji z večjo toleranco do tveganja, kar je pogosto kritizirano, so nujni za financiranje inovacij in podjetništva, ki zagotavljajo prihodnjo gospodarsko rast (Schulp, 2021).

Nazadnje je vlaganje v delniški trg pomembna pot do bogastva za posamezne vlagatelje. Čeprav vrednost delnic ne raste vedno, je povprečna letna donosnost S&P 500 v zadnjih 60 letih približno 8 %. Takšne donose močno presegajo donosnosti mnogih drugih varčevalnih ali naložbenih možnosti, ki so na voljo posameznikom, kar dolgoročnim vlagateljem ponuja dobre priložnosti za povečanje premoženja s pomočjo delniških naložb (Schulp, 2021)

Večina analitikov to povečanje pripisuje dostopnosti tako imenovanega trgovanja brez provizij, kjer vlagatelji ne plačajo neposrednih provizij za trgovanje. Čeprav je Robinhood Financial začel ponujati trgovanje brez provizij že leta 2015, je ta model postal standard na trgih konec leta 2019. Poleg tega so k povečanju prispevali tudi drugi dejavniki, kot so enostavnejši uporabniški vmesniki aplikacij za trgovanje, možnost odpiranja računov brez minimalnega stanja in trgovanje z delci delnic. V letu 2020 je individualno trgovanje predstavljalo petino trgovanja na delniških trgih. Po pandemiji je ta trend padel, ampak vseeno ostal višje kot pred pandemijo (Schulp, 2021).

Študije so potrdile, da novi vlagatelji predstavljajo bolj raznoliko populacijo kot prejšnji vlagatelji. Na primer raziskava, ki jo je izvedla Ameriška agencija za nadzor nad finančno industrijo (angl. Financial Industry Regulatory Authority – FINRA) v sodelovanju z Univerzo v Chicagu, je pokazala, da so vlagatelji, ki so prvič odprli naložbeni račun leta 2020, mlajši, z nižjimi dohodki in bolj rasno raznoliki v primerjavi s prejšnjimi vlagatelji. Mnogi med njimi so začeli vlagati z manjšimi zneski, saj so citirali, da je možnost vlaganja z majhnim začetnim kapitalom glavni razlog za odprtje računa (Schulp, 2021).

2.2 Algoritmi za napovedovanje cen delnic

Algoritem zagotavlja rešitev za računski problem. Primer računskega problema je zmnožek dveh števil. Najbolj pomembna lastnost algoritma je, da deluje pravilno (Knebl, 2020).

Strojno učenje je širok spekter, ki pokriva razne algoritme za prepoznavanje vzorcev in odločanje. Strojno učenje se deli na tri veje: nadzorovano učenje, nenadzorovano učenje in spodbujevalno učenje. Nadzorovano učenje proučuje označene podatke.

Cilj nadzorovanega učenja je ugotoviti povezavo med X in Y. Vsak x_i je lastnostni vektor, medtem ko y_i je odziv. V nenadzorovanem učenju so podatki neoznačeni $x_1, x_2, \ldots, x_n$ in cilj

je ugotoviti raziskovalne informacije o podatkih in ugotoviti skrite vzorce. Tretja veja je spodbujevalno učenje in je algoritmični pristop, kjer se definira niz stanja in dejanja kot odgovor na spreminjajoč se režim, da se maksimira kumulativna nagrada (Dixon in drugi, 2020).

Pogosto uporabljeni algoritmi za napovedovanje cen delnic vključujejo (Mokhtari in drugi, 2021):

- linearno regresijo: za iskanje povezav med zgodovinskimi podatki in prihodnjimi cenami,
- model LSTM (angl. Long Short-Term Memory): model, ki je posebej zasnovan za delo s časovnimi vrstami in lahko zaznava vzorce preteklih podatkov.

Regresijska analiza je statistična metoda za preiskovanje kavzalnega odnosa med dvema ali več spremenljivkami. Statistični model je preprost opis stanja procesa. Model ni ne hipoteza ne teorija. Model se ne vrednoti z eksperimentom, ampak generira dobre testne hipoteze, relevantne pomembnim problemom. Linearna regresija zahteva, da je model linearen v regresijskih parametrih. Enostaven linearen regresijski model se zapiše z enačbo (1):

$$y = \beta_0 + \beta_0 x + \varepsilon, \quad (1)$$

kjer y predstavlja odvisno spremenljivko, β_0 predstavlja presek, β_1 predstavlja gradient regresijske premice, x predstavlja neodvisno spremenljivko in ε je naključna napaka, ki se običajno predpostavi kot normalno porazdeljena $E(\varepsilon) = 0$ (Yan in drugi, 2009).

Poleg enostavnega linearnega regresijskega modela obstaja tudi model multiple linearne regresije, ki je linearni regresijski model z eno odvisno spremenljivko in več kot eno neodvisno spremenljivko. Multipla linearna regresija predpostavlja, da je odvisna spremenljivka linearna funkcija modela s parametri in da je več kot le ena neodvisna spremenljivka v modelu. Splošna oblika modela večkratne linearne regresije se zapiše z enačbo (2):

$$y = \beta 0 + \beta 1 x 1 + \cdots + \beta p x p + \varepsilon, \quad (2)$$

kjer y predstavlja odvisno spremenljivko, $\beta_0, \beta_1, \ldots, \beta_p$ so regresijski koeficienti, $x_1, x_2, \ldots, x_n$ so neodvisne spremenljivke modela. Naključno napako se označi z ε, ki se običajno predpostavi kot normalno porazdeljeno $E(\varepsilon) = 0$ (Yan in drugi, 2009).

Tretji tip regresijskega modela se imenuje nelinearen regresijski model, ki predpostavlja, da je odnos med odvisno spremenljivko in neodvisno spremenljivko nelinearen v regresijskih parametrih. Primer nelinearnega regresijskega modela se zapiše z enačbo (3):

$$y = \frac{\alpha}{1 + e^{\beta t}} + \varepsilon, \quad (3)$$

kjer y predstavlja rast določenega organizma kot funkcijo časa t, α in β sta parametra modela in ε je naključna napaka (Yan in drugi, 2009).

Kot sem že opisal, je SU področje znotraj UI. Če kopljem še malo globlje, je globoko učenje koncept znotraj SU, ki temelji na umetni nevronski mreži.

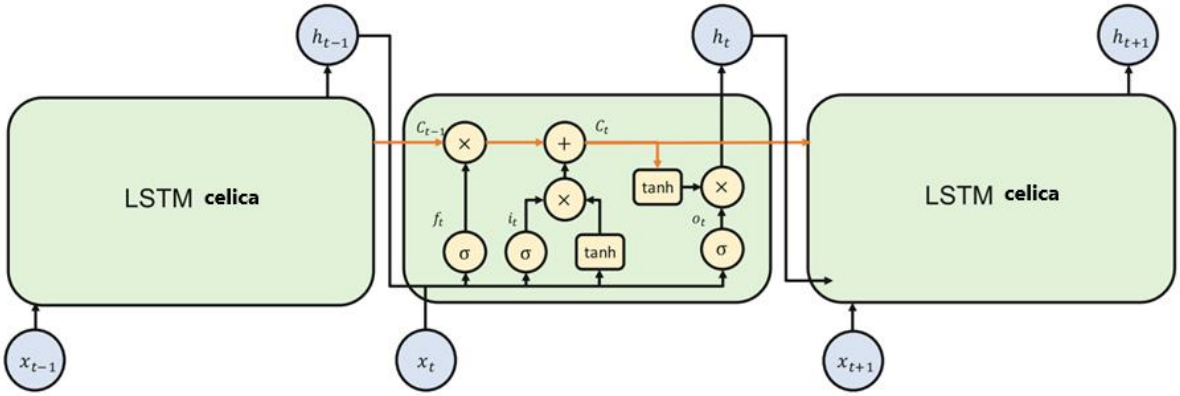
Znotraj globokega učenja je več različnih vrst nevronskih mrež. Najbolj uporaben algoritem za napovedovanje cen delnic je algoritem LSTM, ki je del nevronske mreže s povratno zanko, ki je posebej ustvarjena za sekvenčne podatkovne strukture, kot so podatki časovnih vrst, sekvenčni dogodki in generiranje naravnega jezika. LSTM-algoritem je bolj sofisticirana arhitektura z naprednimi detajlnimi mehanizmi znotraj nevronske mreže s povratno zanko (Janiesch in drugi, 2020).

Za razliko od preproste celice ima LSTM-celica dve stanji: stanje celice C_t in skrito stanje h_t. Proces posodobitve stanja celice formira informacijsko pot, ki teče skozi celotno sekvenco s preprostimi izračuni. Ta funkcionalnost omogoča lahek pretok informacij skozi sekvenco tako, da se lahko odvisnost med dvema vrednostnima, ki sta locirani daleč stran med seboj v sekvenci, pravilno upošteva. Medtem je skrito stanje vključeno v izračune. Vrata kontrolirajo, ali se doda ali se pozabi informacija pretoku, in so implementirana s sigmoidno funkcijo. Izhod te funkcije je omejen med 0 in 1. Ko je izhod funkcije 1, informacija mora biti v popolnosti obdržana, medtem ko je izhod funkcije 0, mora informacija biti pozabljena. V LSTM-celici obstajajo tri vrste vrat: pozabljena vrata, vhodna vrata in izhodna vrata (Dong in drugi, 2020).

Na sliki 1 je simbol H_{t-1} skrito stanje iz prejšnjega časovnega koraka, ki vsebuje kratkoročne informacije in vpliva na obdelavo trenutnega vhoda. H_t je skrito stanje trenutnega časovnega koraka, ki vsebuje nove posodobljene informacije in se prenese naprej. X_t je trenutni vhodni podatek, ki ga celica obravnava v tem časovnem koraku. C_{t-1} je stanje celice iz prejšnjega časovnega koraka, ki hrani dolgotrajne informacije. C_t je posodobljeno stanje celice v trenutnem časovnem koraku, ki vključuje nove informacije in vpliva na naslednje korake.

F_t so pozabljena vrata, ki določijo, katere informacije iz prejšnjega stanja celice je treba izbrisati ali ohraniti, odvisno od pomembnosti. I_t so vhodna vrata, ki odločajo, katere nove informacije iz trenutnega vhoda je treba dodati v stanje celice za dolgoročno shranjevanje. Izhodna vrata o_t odločajo, katere informacije iz trenutnega stanja celice bodo poslane naprej kot del skritega stanja. H_{t+1} in X_{t+1} sta povezana z naslednjim časovnim korakom in predstavljata naravno nadaljevanje procesiranja podatkov v LSTM-modelu. Znak σ na sliki predstavlja sigmoidno funkcijo, ki pretvarja vhodne vrednosti v interval med 0 in 1. *Tanh* predstavlja tangens hiperbolično funkcijo, ki transformira vhodne vrednosti v interval med -1 in 1, kar omogoča bolj stabilne izračune – normalizacijo podatkov. Znak × predstavlja elementno množenje med dvema vektorja. V našem primeru pozabljena vrata množijo vrednost celice iz prejšnjega stanja z vrednostmi sigmoidne funkcije, kar določa, katere informacije ohraniti in katere zavreči. Znak + predstavlja združitev informacij.

Slika 1: Prikaz LSTM-celice



Vir: prirejeno po Dong in drugi (2020).

V raziskovalnem članku so proučevali, da morajo vlagatelji imeti dostop do pravočasnih in natančnih informacij, da lahko učinkovito trgujejo z delnicami. Glede na to, da se na borzi trguje s številnimi podjetji, različni dejavniki vplivajo na odločanje. Prav tako je težko napovedati, kako se bodo gibale cene delnic. Zaradi tega je tehnika napovedi cen delnic ključnega pomena in predstavlja izziv. Iskanje napovednega modela z najmanjšo stopnjo napake in največjo stopnjo natančnosti lahko postane raziskovalna tema. Razvili so model, ki temelji na tehnični analizi in uporablja LSTM-algoritem za napovedovanje cene delnice podjetja za naslednjih 30 dni. Ta študija mi je pomagala kot iztočnica za ustvarjanje lastnega modela za analiziranje cen delnic (Khatri in drugi, 2023).

2.3 Pridobivanje in obdelava podatkov s spleta

Pridobivanje podatkov s spleta, znano tudi kot spletno strganje, je proces avtomatiziranega zbiranja podatkov z različnih spletnih strani. To vključuje uporabo orodij in tehnik za pridobivanje in obdelavo informacij, ki so na voljo na spletnih mestih, ter njihovo prilagajanje specifičnim potrebam, bodisi za osebno uporabo ali poslovne namene (Chapagain, 2019).

Spletno strganje je ključni proces v različnih industrijah, saj omogoča zbiranje surovih podatkov, ki so osnova za analizo, raziskave in sprejemanje odločitev. Pomembno je upoštevati pravne omejitve in politiko spletnih strani, kot so pogoji uporabe in politika zasebnosti, preden se izvede katerakoli aktivnost strganja ali druga aktivnost, ki je tesno povezano s strganjem, vendar se osredotoča na indeksiranje povezav na spletnih straneh.

Tak pristop je ključen za pridobivanje strukturiranih podatkov iz neorganiziranih virov, kot so spletna mesta, in se uporablja v širokem spektru raziskovalnih in poslovnih dejavnosti.

BeautifulSoup je priljubljena Python knjižnica, ki se uporablja predvsem za strganje podatkov s spletnih strani, predvsem za obdelavo HTML- ali XML-dokumentov.

Razvijalcem omogoča, da enostavno izvlečejo podatke s spletnih strani, saj omogoča navigacijo po strukturi HTML. Ena izmed prednosti uporabe knjižnice BeautifulSoup pred drugimi knjižnicami je tudi, da omogoča razčlenjevanje nepopolnih HTML-dokumentov z manjkajočimi ali nepopolnimi oznakami (Chapagain, 2019).

Poleg načina strganja podatkov s spleta se do podatkov dostopa tudi na druge načine. Podatki s spletne strani Yahoo Finance se lahko pridobijo s pomočjo knjižnic, ki vsebujejo podatke te spletne strani, v programskem jeziku Python.

Po pridobitvi podatkov s spleta jih bo treba tudi obdelati. To se imenuje obdelava podatkov. Obdelava podatkov je ključni korak v procesu analize podatkov, saj se ukvarja s pripravo podatkov za nadaljnje analize in izboljšanje njihove kakovosti. V resničnem svetu so podatki pogosto nepopolni in nekonsistentni, kar lahko negativno vpliva na rezultate analiz. Obdelava podatkov vključuje različne tehnike, ki pomagajo izboljšati kakovost podatkov in povečati natančnost analitičnih metod. Preden se uporabijo tehnike za izboljšanje kakovosti podatkov, je treba določiti, h kakšnim podatkom se stremi.

Razlog, zakaj se obdeluje podatke, preden se jih analizira, je, da se zagotovi kakovost podatkov. Kakovost podatkov se zagotovi s številnimi dejavniki (Han in drugi, 2011):

- Točnost: podatki morajo odražati resnične vrednosti in biti brez napak. Napake v podatkih, kot so napačne vrednosti ali napačno zabeleženi podatki, lahko vodijo do napačnih zaključkov. Točnost je bistvena za sprejemanje pravilnih odločitev.
- Popolnost: popolnost podatkov pomeni, da ne manjkajo ključni podatki. Nepopolni podatki, kot so manjkajoče vrednosti, lahko zmanjšajo natančno analiz ali celo onemogočijo izvedbo nekaterih analiz.
- Konsistentnost: konsistentni podatki pomenijo, da med različnimi viri ni neskladij. Podatki iz različnih virov morajo biti usklajeni, da ne prihaja do nasprotujočih si informacij.
- Pravočasnost: podatki morajo biti ažurni in zbrani v pravem trenutku. Podatki, ki so zastareli, ne odražajo trenutne realnosti in lahko vodijo k napačnim odločitvam.
- Verodostojnost: podatki morajo biti zanesljivi in uporabniki jim morajo zaupati. Če podatki izvirajo iz nezanesljivih virov ali so v preteklosti vsebovali napake, uporabniki morda ne bodo zaupali njihovi veljavnosti, tudi če so sedaj natančni.
- Razumljivost: podatki morajo biti enostavni za razumevanje. Če so podatki zapleteni ali slabo strukturirani, je uporabnikom težko razumeti njihovo vsebino ali jih pravilno interpretirati.

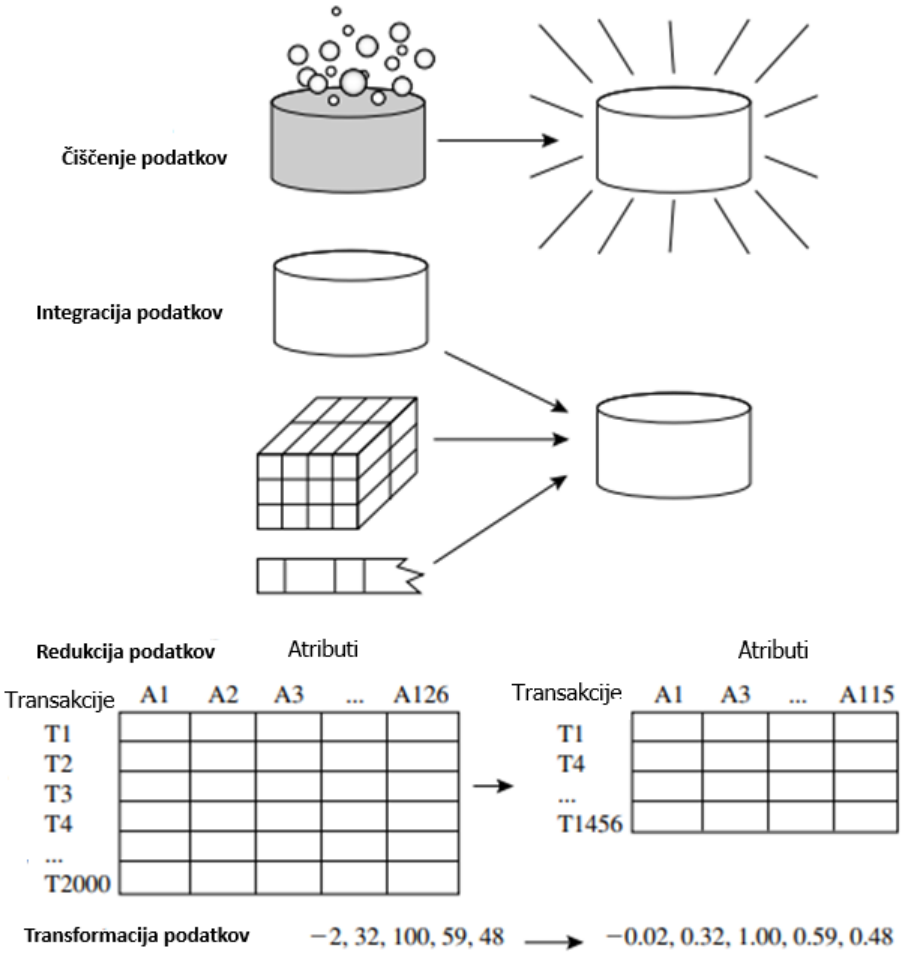
Po razlagi, kaj zagotavlja kakovost podatkov in kateri so glavni dejavniki za zagotovitev kakovosti podatkov, sledi opis tehnik, s katerimi se doseže želeni cilj.

Med osnovnimi tehnikami obdelave podatkov so (Han in drugi, 2011):

- Čiščenje podatkov: odpravljanje napak in zapolnjevanje manjkajočih vrednosti. Čiščenje podatkov se izvede s polnjenjem manjkajočih vrednosti, glajenjem izstopajočih podatkov, iskanjem in odstranjevanjem izstopajočih vrednosti in reševanjem nekonsistentnosti.
- Integracija podatkov: združevanje podatkov iz različnih virov v enotno skladišče podatkov. Skladišče podatkov je lahko baza podatkov, združenih v medsebojno odvisne tabele. Prav tako je lahko skladišče podatkov aplikacija, ki absorbira podatke iz različnih virov in jih omogoča uporabniku za nadaljnjo uporabo.
- Redukcija podatkov: zmanjšanje velikosti podatkov s pomočjo združevanja, odstranjevanja odvečnih lastnosti ali uporabe metod, kot sta klasifikacija in združevanje v skupine. Z redukcijo se zmanjša reprezentacija originalnih podatkov, vendar se lahko z redukcijskimi podatki dosežejo enaki analitični rezultati. Redukcija podatkov se razdeli na dimenzionalno redukcijo in številčno redukcijo:
  - Dimenzionalna redukcija označuje zmanjšano število atributov – stolpcev. To se lahko naredi z odstranjevanjem odvečnih atributov, združitvijo več atributov v en atribut ali z majhno izbiro nabora atributov, ki služijo za nadaljnjo analizo.
  - Številčna redukcija je izmenjava podatkov z alternativno, manjšo reprezentacijo podatkov z uporabo parametričnih modelov (linearnih ali nelinearnih modelov), z uporabo neparametričnih modelov (histogrami, vzorčenjem ali združevanji v skupine).
- Transformacija podatkov: pretvorba podatkov v ustrezne oblike, kot je normalizacija, kjer so podatki prilagojeni določenemu območju (npr. od 0,0 do 1,0), kar lahko izboljša natančnost algoritmov. Transformacija podatkov zajema tudi pretvorbo podatkov v želeno obliko podatkov. Spreminja se lahko oblika, v kateri se zajamejo datumi za namen lažje berljivosti uporabnika.

Na sliki 2 so grafično prikazane tehnike obdelave podatkov, ki sem jih opisal v prejšnjem odstavku.

Slika 2: Tehnike obdelave podatkov



Vir: prirejeno po Han in drugi (2012).

Te tehnike se pogosto uporabljajo skupaj, da zagotovijo podatke visoke kakovosti, ki so bistveni za natančne in učinkovite analize v nadaljnjih fazah, kot je rudarjenje podatkov ali SU (Han in drugi, 2011).

3 METODOLOGIJA

Računalniško programiranje je pisanje ukazov s pomočjo računalnika, da naredi nekaj s pomočjo jezika, ki ga razume. Računalnik je zelo močno orodje, a ne more razmišljati sam zase. Potrebuje točno določene ukaze: kako rešiti nalogo; kako interpretirati pogoj, na podlagi katerega se odloči, kateri poti bo sledil; kaj narediti s podatki, ki pridejo iz naprave, kot je omrežje ali disk; kako se odzvati, ko se zgodi nekaj, kar je razvijalec predvidel, ampak je zunaj normalnega delovanja (Romano in Kruger, 2023).

3.1 Metode razvoja programske opreme

Življenjski cikel razvoja programske opreme se nanaša na metodologijo, ki se uporablja za načrtovanje, ustvarjanje, testiranje, uvedbo in vzdrževanje programske opreme skozi njen celoten življenjski cikel do njene ukinitve. Obstajajo različne metodologije, ki jih je mogoče uporabiti in kombinirati v življenjskem ciklu razvoja programske opreme (Swanlund in drugi, 2024).

Metodologije razvoja programske opreme zagotavljajo ogrodje za načrtovanje, izvajanje in upravljanje procesa razvoja programske opreme (Vijayasarathy in Butler, 2016).

Razvoj programske opreme se lahko deli na pristope in ne samo na posamezne vrste metodologije. Osnovno deljenje pristopov razvoja programske opreme je na: agilni, tradicionalni, iterativni in hibridni pristop. V študiji Swanlund in drugi (2024) so prišli do ugotovitve, da je hibridni pristop najpogosteje uporabljen s kar 45 %, sledi mu agilni pristop s 33 %, nato tradicionalni pristop s 14 % in najmanj uporabljen je iterativni pristop z 8 %. Karakteristike različnih pristopov se opredelijo glede na organizacijske, projektne in ekipne. Za agilne pristope velja, da je organizacija majhno število zaposlenih s srednjimi prihodki, gre za projekte z nizkim proračunom in srednjo do visoko stopnjo kritičnosti ter majhno ekipo. Pri tradicionalnem pristopu gre običajno za organizacijo z visokimi prihodki in velikim številom zaposlenih. Projekt ima velik proračun in visoko stopnjo kritičnosti. Ekip je več v tradicionalnem pristopu ali pa je le ena srednje velika ekipa. Iterativni pristop, ki je najmanj uporabljen, se uporablja v organizaciji z majhnim številom zaposlenih. Projekt ima srednji proračun in srednjo do visoko stopnjo kritičnosti, ekipa pa je majhna. Hibridni pristop združuje zgornje pristope, pri tem velikost organizacije ni pomembna. Projekt ima običajno srednji proračun in visoko stopnjo kritičnosti, ekipa pa je majhna.

Leta 2001 je bila napisana knjiga »Manifesto for Agile Software Development«, ki je spremenila način razvoja programske opreme organizacij. V knjigi se agilni razvoj programske opreme definira kot možnost razvoja programske opreme hitro, z možnostjo hitrega prilagajanja zahtev in potreb. Agilna metodologija je bila zgrajena na ramenih dveh v tistem času popularnih metod za razvoj programske opreme: ekstremnega programiranja in arhitekturne agilnosti. Ekstremno programiranje je metoda, ki spodbuja tesno sodelovanje med strankami in razvojnim timom. Stranke so v nenehnem stiku z razvojem in sproti prilagajajo zahteve skozi uporabniške zgodbe. Vsak cikel razvoja traja le dva tedna in na koncu vsake iteracije ekipa strankam predstavi sistem za povratne informacije. Ekstremno programiranje se osredotoča na hiter razvoj, minimalno arhitekturo in delo v kratkih ciklih, kjer razvijalci pišejo dovolj kode, dokler se testi ne končajo. Upravljanje delovne obremenitve poteka prek točkovanja uporabniških zgodb, kar omogoča boljše načrtovanje stroškov in časa za prihodnje iteracije. Arhitekturna agilnost pa se osredotoča na integracijo arhitekturnega razmišljanja v agilne procese. Gre za metodo, ki omogoča fleksibilne arhitekturne odločitve ob pravem času, pri čemer poudarja tehnični dolg in analiziranje

odvisnosti med komponentami. Ta metoda skuša najti ravnovesje med hitrim razvojem in dolgoročnim vzdrževanjem arhitekture.

Scrum je ena od najbolj priljubljenih agilnih metodologij. Poudarja kratke iteracije, imenovane šprinti, ki trajajo določeno časovno obdobje (na primer dva tedna). Na koncu vsakega sprinta ekipa predstavi novo funkcionalnost. Scrum spodbuja stalno komunikacijo z naročnikom in vključuje različne sestanke, kot so dnevni »stand-up« sestanki, sprint načrtovanja, predstavitve in retrospektive. Scrum ima jasno opredeljene vloge: lastnik izdelka, Scrum mojster in razvojna ekipa (Swanlund in drugi, 2024).

Testno voden razvoj (angl. Test Driven Development, v nadaljevanju TDD) je metoda, pri kateri razvijalci najprej pišejo teste za posamezne funkcionalnosti, preden napišejo dejansko kodo. Ta pristop omogoča, da se več pozornosti nameni kakovosti kode in zmanjšanju napak. TDD pogosto spremlja uporabo avtomatiziranih testnih orodij in se pogosto uporablja skupaj z drugimi agilnimi metodologijami (Swanlund in drugi, 2024).

Funkcionalno voden razvoj (angl. Feature-driven Development – v nadaljevanju FDD) se osredotoča na hitre iteracije, imenovane funkcije, kjer vsaka iteracija dodaja novo funkcionalnost. Vsaka funkcija je opisana kot kratka naloga, na primer »prikaži košarico stranki«. FDD omogoča modularni razvoj in spodbuja timsko delo z jasnimi vlogami. Metoda pa odvrne od prekomernega popravljanja, saj se osredotoča na dodajanje vrednosti končnemu izdelku (Swanlund in drugi, 2024).

3.2 Izbor tehnologij in orodij

3.2.1 Programski jezik

V magistrskem delu programsko rešitev razvijem v programskem jeziku Python. Je visokoravni večnamenski programski jezik, znan po svoji zelo preprosti sintaksi. Sintaksa poudarja lahko berljivost in izražanje, kar olajša pisanje in vzdrževanje kode. Ta programski jezik omogoča tudi obsežno standardno knjižnico in veliko izbiro knjižnic zunanjih razvijalcev. Vse te knjižnice podpira zelo aktivna Pythonova skupnost (Marvin in drugi, 2018).

Python je ustvaril Guido Van Rossum, nizozemski računalniški znanstvenik in matematik, ki se je med božičem leta 1989 poigral z idejo, ki je nato prerasla v kreacijo programskega jezika. Prvič se je Python pojavil v javnosti okoli leta 1991 in od takrat naprej se je razvil v enega od današnjih vodilnih programskih jezikov (Romano in Kruger, 2023).

Čeprav je delo s Pythonom enostavno na beri-oceni-natisni zanki (angl. read-evaluate-print loop – REPL), je cilj ustvarjanje delujočih Python aplikacijskih datotek. Python datoteka je skripta, kar pomeni, da naredi neko uporabno delo, ko je izvedena s Python programom. Datoteka je lahko modul, kar pomeni, da je ustvarjena tako, da je lahko uvožena za

zagotavljanje uporabnih definicij. Direktorij Pythonovih modulov je paket. To so formalne definicije programskega jezika Python.

Poleg formalnih definicij obstajajo tudi neformalni izrazi, kot so knjižnica, aplikacija in ogrodje. Ti pogosto uporabljeni koncepti se lahko definirajo tako (Lott, 2015):

- Knjižnica: je zbirka modulov ali paketov.
- Aplikacija: je v najpreprostejši definiciji skripta. Zahtevnejša aplikacija vsebuje skripto in nekaj dodatnih modulov in paketov.
- Ogrodje: je aplikacija, v katero se vključijo dodatni prilagojeni moduli in paketi. Mnogo ogrodij vključuje tudi datoteke, ki niso Python skripte. Spletno ogrodje vključuje mnogo HTML- in CSS-datotek.

Python podpira več programskih paradigem. V grobem se programske paradigme deli na imperativno in deklarativno programiranje. Imperativno programiranje se nato dalje deli še na objektno usmerjeno in proceduralno programiranje. Deklarativno programiranje pa se deli na funkcionalno, podatkovno-potenčno, reaktivno in logično programiranje. Večina modernih programskih jezikov je večparadigmatičnih z eno prevladujočo paradigmo. V Pythonu se paradigme deli na (Dyer in Chauhan, 2022):

- Imperativno programiranje: v imperativnem programiranju razvijalci natančno določijo, kako naj se nekaj izračuna z uporabo izjav, ki spreminjajo stanje programa. Stavek se deklarira kot imperativen samo, ko le-ta eksplicitno ne uporablja lastnosti drugih paradigem.
- Proceduralno programiranje: ta paradigma programiranja organizira kodo v ponovno uporabne elemente, imenovane procedure, v Pythonu imenovane funkcije.
- Objektno usmerjeno programiranje: temelji na konceptu razredov in objektov. Razredi so kot načrti, ki opisujejo, kakšni bodo objekti, medtem ko so objekti primeri razredov.
- Funkcionalno programiranje: je deklarativen stil programiranja v primerjavi s prejšnjimi tremi paradigmami, ki so vse imperativne. To pomeni, da se razvijalec osredotoča bolj na to, kaj naj program naredi, in ne na to, kako naj program to naredi.

V študiji Dyer in Chauhan (2022) je bilo ugotovljeno, da je objektno usmerjena paradigma najbolj uporabljena – 40 %. Sledi ji proceduralna paradigma z malo več kot 30 %. Kratke skripte so najpogosteje napisane v proceduralni paradigmi in služijo kot uporabnostne skripte. Funkcionalna paradigma je najmanj uporabljena, ampak funkcionalni stavki so ena izmed najpogosteje uporabljenih lastnosti.

Razvoj v Pythonu je hiter, saj gre za interpretiran jezik. To pomeni, da se ukazi izvajajo v času delovanja, kar odpravi potrebo po predhodni pretvorbi programa v strojni jezik. Ta lastnost omogoča hitro testiranje in eksperimentiranje.

Ta vsestranskost, skupaj z zmogljivostjo Pythona, da deluje na vseh operacijskih sistemih, kot so Windows, GNU/Linux in mac OS, prispeva k njegovi priljubljenosti. Uporablja se za

pisanje avtomatiziranih skript, strojno učenje, znanstvene izračune, obdelavo velikih podatkov, spletne aplikacije, programiranje grafičnih uporabniških vmesnikov in naprave interneta stvari (angl. Internet of Things, v nadaljevanju IoT). Za uporabo so ga sprejela tudi velika tehnološka podjetja, kot so: Google, Uber, Facebook in Mozilla (Marvin in drugi, 2018)

Spremenljivke so reference vrednosti v spominu. Spremenljivke v Pythonu lahko deklarirajo različne podatkovne tipe, kot so nizi, cela števila, števila s plavajočo vejico, logične vrednosti ter različne podatkovne vrste in podatkovne strukture. V nasprotju z jeziki, kot so Java ali C++, ki zahtevajo, da predhodno deklarirate tip spremenljivke, Python tega ne zahteva, saj določi tip spremenljivke med izvajanjem programa (Marvin in drugi, 2018).

Spremenljivko si je mogoče predstavljati kot škatlo z nalepko. Sama škatla nima vrednosti, vendar postane dragocena, ko se v njo nekaj postavi. Na podoben način spremenljivka predstavlja vrednost, ki jo vsebuje. Poleg tega se lahko vrednost in tip spremenljivke med izvajanjem spremenita (Marvin in drugi, 2018).

Čeprav Python ne zahteva deklaracije podatkovnega tipa spremenljivk, pa ima vsak podatkovni tip določen format:

- Številski podatkovni tipi:
  - int (cela števila): uporablja se za delo s celimi števili. Primer: x = 10;
  - float (števila s plavajočo vejico): uporablja se za decimalna števila. Primer: y = 10.5;
  - complex (kompleksna števila): Python podpira tudi kompleksna števila, kjer j označuje imaginarni del. Primer: z = 2 + 3j.
- Znakovni niz (angl. string): niz je zaporedje znakov, ki je zapisano med enojnimi (') ali dvojnimi narekovaji ("). Primer: ime = "Python".
- Logične vrednosti (angl. bool): ta podatkovni tip ima dve vrednosti: True ali False, kar je uporabno pri pogojnih stavkih in logičnih operacijah. Primer: x = True.
- Seznami (angl. list): seznam je zbirka elementov, ki so lahko različnih tipov. Seznam je spremenljiv, kar pomeni, da lahko njegove elemente spreminjaš. Seznam je zapisan med oglatimi oklepaji. Primer: seznam = [1, 2, "Python", 4.5].
- Tabele (angl. tuples): tuple je nespremenljiva zbirka elementov. Ko se enkrat določi elemente v tabeli, jih ni mogoče spreminjati. Tabele so zapisane med okroglimi oklepaji. Primer: tabela = (1, 2, "Python").
- Slovarji (angl. dictionary): slovarji vsebujejo pare ključ-vrednost. Vsakemu ključu je dodeljena določena vrednost, pri čemer so ključi unikatni. Slovarji so zapisani med zavitimi oklepaji. Primer: slovar = {"ime": "Python", "leto": 1991}.
- Množice (angl. set): množica je nespremenljiv, neurejen seznam unikatnih elementov. Elementi v množici ne morejo biti podvojeni. Zapisana je med zavitimi oklepaji, podobno kot slovar, vendar brez parov ključ-vrednost. Primer: množica = {1, 2, 3, 4}.

Python, kot ostali programski jeziki, ima nekaj pravil glede poimenovanja identifikatorjev, kot so imena spremenljivk, imena razredov, imena funkcij, ime modulov, in drugih objektov. Nekatere so strogo določene ob interpretiranju, medtem ko so druge določene s konvencijo in jih imajo razvijalci možnost ignorirati. Pravila so določena zato, da ob interpretaciji ne pride do zmede in da je koda bolj berljiva za ljudi (Marvin in drugi, 2018).

Zelo priročna funkcija, ki jo omogoča Python, je *input()*. Ko je ta funkcija klicana, dovoli uporabniku vnesti vhodni signal v programsko kodo z uporabo tipkovnice. Izvedba programa se zaustavi, dokler uporabnik ne pritisne tipke *Enter* po tem, ko je zaključil z vnosom. Vhodni signal je vnesen v program kot znakovni niz, ki ga lahko razvijalec uporabi v programski kodi. Ker je vhodni signal sprva vedno znakovni niz, je mogoče pretvoriti znakovni niz v druge podatkovne tipe (Marvin in drugi, 2018).

Komentarji so integralni del programiranja, so opombe, ki dajo več informacij o kodi. Razložijo, zakaj so nekatere odločitve bile sprejete v kodi, kaj so lahko izboljšave v prihodnosti, in razložijo poslovno logiko. Obstajajo tri različne vrste komentarjev (Marvin in drugi, 2018):

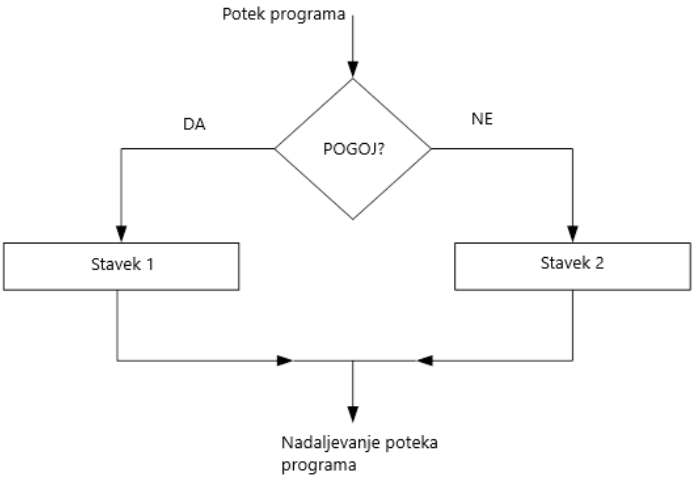
- Dokumentacijski niz: gre za dobesedni niz, ki se uporablja kot komentar v Pythonu. Piše se znotraj trojnih narekovajev. Dokumentacijski nizi se pogosto uporabljajo za dokumentiranje modulov, funkcij in definicij razredov.
- Blokovni komentar: nanaša se na kodo, ki sledi, in mora biti enako zamaknjen kot koda.
- Vrstični komentarji: so napisani v isti vrstici kot ukaz, ki ga pojasnjujejo, in so zaradi svoje nepreglednosti najmanj uporabljeni.

Potek programa opiše pot, kjer so stavki v kodi izvedeni. Python zasleduje preprost potek programa od zgoraj navzdol. Vsaka vrstica mora počakati, da so se vse vrstice nad njo izvedle, nato lahko začne z izvedbo. Potek programa od zgoraj navzdol je lahko razumljiv in preprost za reševanje napak v kodi (Marvin in drugi, 2018).

Po definiciji poteka programa se lahko definira kontrolni stavek. Kontrolni stavek je struktura v kodi, ki pogojno spremeni potek programa. Kontrolni stavek to doseže tako, da pogojno izvede različne dele kode. Kontrolni stavek je lahko uporabljen ponavljajoče in pogojno izvede del kode. V Pythonu obstajata dva glavna kontrolna stavka: if in while (Marvin in drugi, 2018).

Stavek if ti dovoli izvesti del kode, če je pogoj izpolnjen. V drugem primeru izvede alternativni del kode else, ki pa ni obvezen. Prav tako se lahko uporabi več if stavkov enega za drugim in se tako doda več pogojnih stavkov. Na sliki 3 je grafični prikaz opisa if-else pogojnega stavka, ki sem ga opisal.

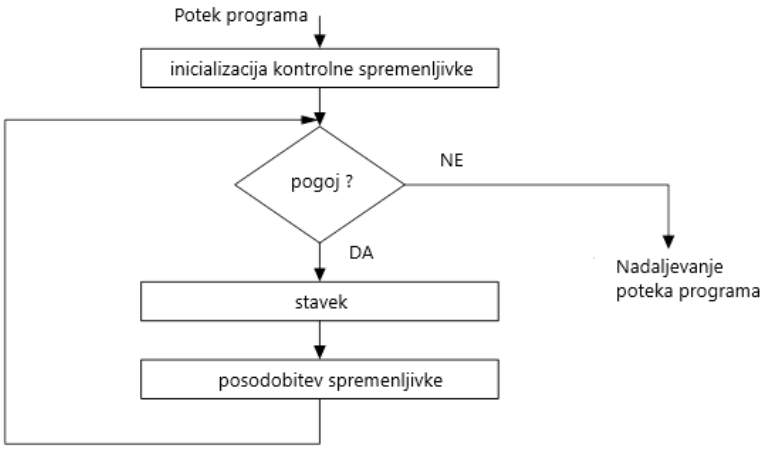
Slika 3: Diagram poteka if-else pogojnega stavka



Vir: prirejeno po CodeSansar (2024a).

Stavek while ti dovoljuje, da se del kode izvede večkrat, dokler izpolnjuješ pogoj. While stavek ima lahko tudi else alternativni del kode, ki se bo izvedel, ko pogoj ne bo več izpolnjen. Na sliki 4 je predstavljen grafični prikaz while stavka, ki sem ga opisal.

Slika 4: Diagram poteka while pogojnega stavka



Vir: prirejeno po CodeSansar (2024b).

Glavna razlika med if in while stavkom je, da ti if stavek ponudi priložnost razvejati izvedbo kode glede na pogoj. Izvedba kode v if stavku je lahko izvedena samo enkrat in nato nadaljuje s potekom program ali konča z izvajanjem. Pri stavku while se izvaja del kode večkrat, dokler je pogoj izpolnjen. To pomeni, da bo stavek while izvajal del kode, dokler bo vrednost A večja od vrednosti B, in nadaljeval s potekom programa samo, ko A ne bo več večji od B. Stavek while se lahko obravnava tudi kot zanka.

Zanka je način za izvedbo dela kode programa večkrat. Zanke se uporabljajo za prehajanje skozi t. i. iterativne objekte. Za potrebe tega poglavja se lahko iterativni objekt opredeli tako:

- karkoli, kar se lahko obravnava kot zbirka elementov, skozi katere se lahko prehaja,
- karkoli, kar se lahko pojavi na desni strani zanke for, na primer: for x in iterable.

Nekateri pogosti primeri iterativnih objektov so: nizi, seznami, slovarji, datoteke. Iterativni objekt si je mogoče predstavljati kot zbirko homogenih stvari, ki so združene v večjo celoto.

Iterativni objekt lahko predstavim s primerjavo avtomobila na parkirišču salona. Avtomobilsko parkirišče je zbirka (iterativni objekt), posamezni avtomobili pa so člani te zbirke. Če bi iskali avto za nakup, bi verjetno imeli nekaj lastnosti, ki jih želite. Vstopili bi na parkirišče in začeli pregledovati avto za avtom, dokler ne bi našli tistega, ki najbolj ustreza vašim pogojem. Dejstvo, da prehajate od enega avtomobila do drugega in ponavljate ta pregled, je v bistvu to, kar pomeni iteracija ali zanka. Zanke omogočajo, da se iterativni objekti razdelijo in je mogoče izvesti operacije na njihovih posameznih članih ali se jih pretvori v nove podatkovne strukture (Marvin in drugi, 2018).

Če želi razvijalec slediti funkcionalnemu načinu programiranja, so funkcije ena izmed najpomembnejših pojmov. Funkcije so sestavni del programskega jezika Python in številnih drugih programskih jezikov. Funkcije so enostaven način, kako združiti nekaj vrstic kode, ki izvajajo neko funkcionalnost. To je še posebej koristno, če je koda večkrat uporabljena v različnih delih programa. Funkcije omogočajo skriti zapleteno kodo, ki jo potrebujete v svoji programski kodi. Funkcije si je mogoče predstavljati kot majhne programe znotraj večjega programa, ki izvajajo specifične naloge (Marvin in drugi, 2018).

Pomembno je, da čeprav je možno v eno funkcijo vstaviti veliko funkcionalnosti, je bolje napisati funkcije, ki izvajajo samo eno specifično nalogo. To olajša spreminjanje kode v posamezne module, dolgoročno pa je koda bolj vzdržna in lažja za odpravljanje napak. Funkcije lahko sprejmejo vhodne podatke in lahko (vendar ni nujno) vračajo vrednost ali več vrednosti. Tri glavne vrste funkcij v Pythonu so: vgrajene funkcije; funkcije, ki jih določijo uporabniki, in anonimne funkcije (Marvin in drugi, 2018).

Vgrajene funkcije so že vključene ob namestitvi Python programa. Uporablja se jih lahko kjerkoli v kodi, brez potrebe po uvozu dodatnih modulov. Funkcija *input* je primer, ki sem ga vpisal, in je ena izmed vgrajenih funkcij. Najbolj osnovna vgrajena funkcija pa je *print*, ki izpiše predmete v terminal.

Funkcije, ki jih določi uporabnik, so namenjene temu, da uporabnik doseže določen cilj s pisanjem prilagojene funkcije. Glavni namen funkcij je organizirati program v logične dele, ki skupaj rešujejo specifičen problem. Na sliki 5 je definicija funkcije v Pythonu.

Slika 5: Definicija funkcije

```
def ime_funkcije(parameter_en, parameter_dva, parameter_n):
    # Logika funkcije
    return
```

Vir: lastno delo.

Koraki za definiranje funkcije so naslednji:

1. Ključna beseda def, ki ji sledi ime funkcije.
2. Sledijo parametri, če so potrebni in so zapisani znotraj oklepajev. Definicijo funkcije se zaključi z dvopičjem.
3. Zapis logike funkcije, ki je premaknjena za zamik desno.
4. Uporaba ključne besede return, da vrne rezultat funkcije. Ta korak je neobvezen in če ni vključen, funkcija privzeto vrne rezultat None.

Klic funkcije pomeni izvajanje logike funkcije, ki je definirana za podpičjem. Pri klicanju funkcije imena argumentov ne rabijo biti enaka imenom parametrov, ki jih funkcija pričakuje. Pomembno je, da karkoli se poda funkciji ob klicu, funkcija to interno obravnava kot določen parameter.

Spremenljivke, definirane v logiki funkcije, se imenujejo lokalne spremenljivke, saj so dostopne le znotraj funkcije. Imajo lokalno območje delovanja. Spremenljivke, definirane zunaj telesa funkcije, se imenujejo globalne spremenljivke, saj so dostopne tako zunaj kot znotraj funkcij.

Parametri so informacije, ki morajo biti vključeni v funkcijo, da ta deluje. Bolj pogosto se parametre naslovi z argumenti. Z argumenti se označujejo dejanske vrednosti oziroma reference, ki se določijo spremenljivkam parametrov, ko je funkcija klicana. Argumenti so funkcijam to, kar so sestavine receptu. Python podpira naslednje argumente:

- Obvezen argument: ta argument mora biti prisoten ob klicu funkcije. Ti tipi argumentov morajo biti v pravilnem vrstnem redu, da funkcija deluje pravilno.
- Argument s ključno besedo: v primeru, da je vrstni red v funkciji pomemben, se lahko uporabi argument s ključno besedo. Tako v primeru, da so argumenti podani v napačnem vrstnem redu, funkcija po ključni besedi najde pravilni argument in funkcija bo dobila parametre v pravilnem vrstnem redu in vrnila pravilni rezultat.
- Privzeti argument: ta argument se uporabi, ko ni podane vrednosti argumenta ob klicu funkcije. Takrat se argumentu pripiše privzeta vrednost. V primeru, da je dodeljena privzeta vrednost in se ob klicu funkcije deklarira novo vrednost argumenta, bo funkcija upoštevala vrednost, ki se uporabi ob klicu funkcije.
- Spremenljivo število argumentov: Python omogoča, da se lahko funkciji določi poljubno število spremenljivk, ki jih je mogoče obravnavati. To se naredi s ključno sintakso *.

Anonimne funkcije v Pythonu se imenujejo lambda funkcije. Tako so imenovane, ker za razliko od drugih funkcij ne zahtevajo imena v svoji definiciji. Te funkcije so začasne in se uporabljajo samo na mestu, kjer so definirane, zato jih ni treba klicati na drugih mestih v kodi. Pogosto nadomestijo običajne funkcije takrat, ko se funkcija uporabi samo enkrat, ali takrat, ko se uporabijo skupaj z drugimi funkcijami, kot so *map(), reduce(), filter()* (Marvin in drugi, 2018).

V nekaterih primerih v aplikaciji uporabim defenzivno programiranje. Tako se lahko soočim z napakami ali izjemami. Najbolj preprost način za defenzivno programiranje je blok *try* in *except*. Koda se bo poskusila izvesti po stavku *try* in v primeru, da naleti na napako, bo koda izvedla stavek *except*. Tako se lahko predvidijo napake in se je mogoče izogniti temu, da napaka zaustavi program (Marvin in drugi, 2018).

3.2.2 Spletno ogrodje

V Pythonu uporabim spletno ogrodje, imenovano Ploty Dash, ki se uporablja za ustvarjanje analitičnih spletnih aplikacij. Zgrajeno je na osnovi priljubljene Python knjižnice Plotly, ki se uporablja za ustvarjanje interaktivnih in vizualno privlačnih podatkovnih vizualizacij, ter spletnega razvojnega ogrodja Flask. Ta okvir ponuja različne možnosti prilagajanja za vizualizacije (Lombardo, 2024).

Flask je enostavno ogrodje za spletne aplikacije, napisano v Pythonu. Uporablja fleksibilnost Pythona, da omogoča relativno preprost razvoj spletnih aplikacij. Flask omogoča pisanje preprostih enostranskih aplikacij, hkrati pa ima tudi zmogljivost za njihovo razširitev in izdelavo večjih aplikacij brez težav (Aggarwal, 2019).

Dash se ne uporablja le za raziskovanje podatkov; uporablja se lahko za vse faze procesa podatkovne analize, od raziskovalnih do operacijskih produkcijskih okolij. Dash je sestavljen iz štirih komponent: Flask, ki se uporablja za zaledno programiranje; Plotly, ki se uporablja kot knjižnica za izrisovanje grafov; React za interaktivne komponente in Dash aplikacija, kjer gre za osnovno aplikacijo Reacta. Enostavna arhitektura Dasha je ena od glavnih prednosti, saj ti dovoljuje ustvarjati podatkovne, interaktivne spletne aplikacije in vmesnike samo z uporabo Pythona, brez potrebe po uporabi HTML, CSS ali JavaScripta (Dabbas, 2021).

Dash ni le en velik paket, ki vsebuje vse, vendar gre za nekaj paketov, ki vsak skrbi za svoj določen vidik. Najpogosteje uporabljeni paketi znotraj Dasha so:

- Dash: to je glavni paket, ki zagotavlja hrbtenico vsaki aplikaciji prek dash.Dash objekta. Prav tako zagotavlja nekatera orodja za vodenje interaktivnosti in izjem.
- Dash Core Components: paket, ki zagotavlja interaktivnost komponent, ki jih lahko uporabnik spreminja. Spustni nivoji, gumb za izbiro datumov in drsniki so primeri komponent, ki so vključene v ta paket.

- Dash HTML Components: ta paket vsebuje vse HTML-značke kot Python razrede. Preprosto pretvori Python v HTML. Primer: dash_html_components.H1('Hello, World') v Pythonu se pretvori v <h1>Hello, World</h1> HTML-komponento in jo tako obravnava v brskalniku.
- Dash Bootstrap Components: to je zunanji paket, ki doda Bootstrap funkcionalnosti spletnemu ogrodju Dash. Paket in njegove komponente poskrbijo za opcije izgleda aplikacije in vizualnih signalov. Elementi se lahko na podlagi tega paketa prekrivajo, njihova velikost se specificira glede na velikost zaslona brskalnika, in zagotovijo se lahko različne barve za boljšo komunikacijo z uporabniki (Dabbas, 2021).

V tabeli 1 so v levem stolpcu našteti deli aplikacije, ki definirajo strukturo Dasha.

Tabela 1: Struktura aplikacije Dash

Deli aplikacije	app.py
Uvozi	import dash import dash_html_components as html import dash_core_compontents as dcc
Instanca aplikacije	app = dash.dash(__name__)
Izgled aplikacije: seznam html interaktivnih komponent	app.layout = html.div([dcc.dropdown() dcc.graph() ...])
Povratne funkcije	@app.callback() … @app.callback() …
Zagon aplikacije	if __name__ == '__main__': app.run_server()

Vir: prirejeno po Dabbas (2021).

Običajno je datoteka *app.py* mesto, kjer se nahajajo vsaj instanca aplikacije, osnovni uvozi in zagon aplikacije. Na začetku se uvozi zahtevane pakete in se kreira spremenljivka *app*. Uporaba *__name__* vrednosti za parameter *name* je uporabljena zato, da Dash z lahkoto locira statična sredstva, uporabljena v aplikaciji. Za izgled aplikacije se uporabljajo HTML-dokumenti. Povratne funkcije se uporabljajo za povezovanje vidnih HTML-elementov aplikacije med seboj. Funkcije so načeloma neodvisne med seboj in njihov vrstni red ni pomemben (Dabbas, 2021).

Funkcija je v svojem osnovnem pojmu procedura. Lahko vzame enega ali več argumentov (vhodov), naredi nekaj z njimi in vrne enega ali več izhodov. Da funkcija lahko deluje znotraj aplikacije, mora funkcija vedeti, na kaj se sklicujejo vhodi in izhodi. Vsaka Dash komponenta ima svoj *id* parameter, ki se lahko enostavno definira zato, da se ga lahko unikatno identificira. Za določanje vhodov in izhodov funkcije se uporablja modul

dash.Dependencies tako, da se uvozi *Input, Output* razreda. Razlika med zalednim in čelnim delom programa je v Dashu zelo enostavno razlikovana. Vse, kar se nahaja v *app.layout,* se smatra kot čelni del programa, medtem pa je vse, kar se nahaja v povratnih funkcijah, del zalednega dela programa. Vhod povratne funkcije je lahko tudi seznam. Nato se lahko prehaja skozi seznam in se sprejema več vhodnih vrednosti z enim vhodom. Vrstni red vhodov in izhodov je pomemben, najprej na vrsto pridejo izhodi, nato se deklarira še vhode. Tudi vrstni red vhodov je pomemben, saj tako, kot so napisani vhodi, tako bodo določeni tudi argumenti funkcije. Tako kot bodo napisane spremenljivke v *return* funkciji, tako so določeni tudi izhodi. Še en opcijski parameter povratnih funkcij je *State,* ki ne proži povratne funkcije, vendar se ob proženju funkcije drugih vhodov uporabi kot stanje določenega elementa (Dabbas, 2021).

3.2.3 Programske knjižnice

3.2.3.1 Pandas

Pandas je knjižnica, ki vsebuje visokonivojske podatkovne strukture in orodja, zasnovana za pomoč programerjem Pythona pri izvajanju manipulacij podatkov in odkrivanju informacij na preprost in hiter način. Učinkovita analiza podatkov zahteva sposobnost indeksiranja, pridobivanja, urejanja, preoblikovanja, združevanja, rezanja in izvajanja analiz na enodimenzionalnih in večdimenzionalnih podatkih, vključno s heterogenimi podatki, ki so samodejno usklajeni z oznakami indeksov. Za omogočanje teh funkcij Pandas ponuja naslednje značilnosti:

- visokozmogljive strukture za enovrstične in tabelarne podatke, kot sta objekta Series in DataFrame, ki omogočata predstavitev homogenih in heterogenih podatkovnih naborov;
- prilagodljivo preoblikovanje podatkovne strukture, ki omogoča vstavljanje in brisanje vrstic in stolpcev;
- hierarhično indeksiranje podatkov vzdolž več osi, kar omogoča več oznak na posamezen podatek;
- oznake za serije in tabelarne podatke za lažje indeksiranje in samodejno usklajevanje podatkov;
- enostavno prepoznavanje in odpravljanje manjkajočih podatkov;
- rezanje in izbiro podnaborov podatkov, vključno z združevanjem in povezovanjem podatkov s podobnimi operacijami kot jezik SQL;
- obširne funkcije za uvoz in izvoz podatkov iz več formatov, kot so CSV, Excel, baze podatkov in JSON;
- podporo za funkcionalnost, specifično za časovne vrste, kot so ustvarjanje datumskih razponov, premiki časa, statistika z drsnimi okni ipd.;
- vgrajeno podporo za pridobivanje podatkov iz spletnih virov, kot so Yahoo!, Google Finance, Svetovna banka in drugi.

Pandas ponuja odlično orodje za učenje manipulacije s podatki za Python programerje in ljubitelje podatkovne znanosti. Čeprav Pandas ni orodje za podatkovno znanost, omogoča integracijo z drugimi knjižnicami v Python ekosistemu, kot sta SciPy in scikit-learn (Heydt, 2015).

Gre za programsko knjižnico v Pythonu, ki je kompatibilna z drugimi knjižnicami, kot sta NumPy in Matplotlib (Campesato, 2022).

3.2.3.2 YFinance

YFinance je programska knjižnica, ki omogoča pridobivanje zgodovinskih podatkov o delnicah neposredno s spletne strani Yahoo Finance z uporabo tehnik spletnega strganja (Pagar in drugi, 2023).

Poleg strganja zgodovinskih podatkov je knjižnica odlična za pridobivanje podrobnih podatkov podjetja, ki ga uporabnik želi analizirati. Podatki so osveženi dnevno, kar je pomembno pri izdelavi napovednih modelov (Raipitam in drugi, 2023).

Knjižnica omogoča enostavno integracijo zgodovinskih podatkov, kar je ključno za modeliranje napovednih algoritmov, zlasti v aplikaciji, kot so napovedi gibanj cen delnic z uporabo metod strojnega učenja. Ta proces se pogosto uporablja za tehnično analizo delnic, kjer se proučuje zgodovinsko gibanje cen. Postopek običajno vključuje naslednje korake: konfiguracijo, ekstrakcijo podatkov, prečiščevanje in filtriranje ter oblikovanje v podatkovno vrsto DataFrame, ki je del programske knjižnice Pandas (Pagar in drugi, 2023).

3.2.3.3 Requests

Requests je HTTP-knjižnica v Pythonu. Z Requests se lahko pošilja različne vrste zahtev (GET, POST, PUT, DELETE) na strežnike, avtomatizira spletne interakcije, kot so prijava, pridobivanje RSS-virov, delo s piškotki in SSL-verifikacija, vse z nekaj vrsticami kode. Requests poenostavi delo v primerjavi s knjižnicami urllib in urllib2, ki zahtevata daljše in bolj zapleteno kodiranje. Glavne prednosti knjižnice Requests so:

- samodejno kodiranje parametrov in dekodiranje odgovorov,
- enostavno pošiljanje zahtev s shranjevanjem seje in podporo za piškotke,
- podporo za JSON-format, kjer ima Requests vgrajen dekoder,
- podporo za SSL-verifikacijo in proxy nastavitve,
- upravljanje z izjemami, kot so napake povezave, http-napake in časovne omejitve.

Requests poenostavi naloge, ki bi sicer zahtevale več modulov in kompleksno kodo, zaradi česar je dostop do spleta v Pythonu hitrejši in bolj intuitiven (Bansai, 2014).

3.2.3.4 Scikit-learn

Scikit-learn je Python knjižnica, zasnovana za znanstvene izračune, in vsebuje več naprednih algoritmov strojnega učenja za klasifikacijo, regresijo, razvrščanje v skupine, zmanjševanje dimenzij, izbiro modelov in pripravo podatkov. Njeno ime izhaja iz SciPy Toolkita, ki je razširitev modula SciPy. SciPy je zbirka Python modulov za znanstvene izračune in vključuje več osnovnih paketov, kot so NumPy, Matplotlib, IPython in drugi. Scikit-learn zahteva delujočo namestitev knjižnic NumPy in SciPy. Scikit-learn lahko namestite z upraviteljem paketov pip z naslednjim ukazom: *pip install scikit-learn* (Weiming, 2019).

Je osnovna knjižnica za strojno učenje, ki omogoča enostaven in enoten vmesnik ter veliko izbiro modelov. Nastala je iz SciPy kot samostojna knjižnica zaradi naraščajoče kompleksnosti metod strojnega učenja. Njeno ime izhaja iz izraza »SciPy kit-learn«. Postala je industrijski standard za strojno učenje v Pythonu, po katerem so se zgledovale druge knjižnice, kot sta *xgboost* in *fbprophet*, za zamenjavo in kombinacijo različnih algoritmov (Kats, 2019).

Scikit-learn ponuja obsežen nabor orodij za strojno učenje, vključno z:

- pripravo podatkov (standardizacija in transformacije),
- izborom modelov (navzkrižno preverjanje veljavnosti, optimizacija nastavitvenih parametrov in procesne linije),
- različnimi metričnimi funkcijami in funkcijami za oceno napak,
- zmanjšanjem dimenzionalnosti,
- razvrščanjem v skupine,
- regresijo in klasifikacijo z različnimi modeli.

Scikit-learn deluje z dvodimenzionalnimi strukturami podatkov, podobnimi NumPy tabelam, zato so tako NumPy kot Pandas podatkovni okviri primerni za uporabo (Kats, 2019).

3.2.3.5 Keras

Keras je odprtokodna knjižnica za globoko učenje v Pythonu, zasnovana tako, da je visokonivojska, uporabniku prijazna, modularna in razširljiva. Zaradi velike skupnosti z več kot 200.000 uporabniki je ena izmed najbolj priljubljenih knjižnic za globoko učenje. Uradna stran za dokumentacijo Keras je na naslovu https://keras.io. Najenostavnejši način za namestitev Keras je z naslednjim ukazom v terminalu: *pip install keras*. Privzeto Keras uporablja TensorFlow kot knjižnico za obdelavo tenzorjev, vendar je mogoče konfigurirati tudi drugo implementacijo v ozadju (Weiming, 2019).

Keras je visokonivojski aplikacijski programski vmesnik za nevronske mreže, napisan v Pythonu in zmožen delovati na podlagi TensorFlow. Knjižnica TensorFlow in vmesnik Keras sta močno povezana (Silaparasetty, 2020).

3.2.4 Integrirano razvojno okolje

Programsko kodo razvijam v Visual Studio Code. Najpogosteje se imenuje VS Code. Osredotoča se na hitrost in zadošča potrebam širše razvojne skupnosti. Razvijalci so včasih iskali razširjena razvojna okolja, kjer so lahko integrirali celoten življenjski cikel razvoja programske opreme – nekaj, kar sem že definiral. Ta okolja so običajno podpirala en programski jezik. Z razvojem panoge v bolj spletno usmerjene aplikacije so se spremenila tudi razvojna okolja. Microsoft je zato leta 2015 izdal prvi večjezični enostavni urejevalnik, imenovan VS Code, ki sledi konceptu »vzemi, kar potrebuješ«. VS Code je torej hibrid med urejevalnikom in integriranim razvojnim okoljem.

Urejevalniki podpirajo različne programske jezike, omogočajo delo z datotekami in mapami, vendar so omejeni v smislu projektov in rešitev. Integrirana razvojna okolja omogočajo urejanje in izvajanje kode ter odpravljanje napak. Ponujajo obsežne funkcije za povečanje produktivnosti, vendar so omejeni na določen nabor jezikov in ogrodij.

VS Code se nahaja med urejevalnikom in integriranim razvojnim okoljem. Je lahek in hiter, hkrati pa podpira več programskih jezikov s pomočjo razširitev za VS Code. Gre za urejevalnik, ki deluje na več platformah in podpira operacijske sisteme Windows, mac OS in Linux ter ponuja funkcije za celoten razvojni življenjski cikel (Mehboob Ahmed Khan in Habib, 2020).

4 RAZVOJ PROGRAMSKE REŠITVE

Ob začetku razvoja programske rešitve sem se najprej odločil, katere metode razvoja programske opreme bom uporabil. Izbral sem kombinacijo več metod, saj sem projekt razvijal sam in prilagajal razvoj glede na specifične potrebe. Začel sem z metodo razvoja prototipa, ki ga bom podrobneje opisal v poglavju 4.1. Ko sem programsko rešitev razvil do stopnje prototipa, sem za nadaljnji razvoj uporabil načela agilnih metod.

Med agilnimi metodami sem črpal elemente iz metodologij scrum, TDD in FDD, kar se je izkazalo za najbolj primerno za iterativni razvoj. Čeprav scrum običajno vključuje ekipo, sem se osredotočil na uporabo nekaterih njegovih ključnih načel, kot so kratki cikli razvoja (sprinti), ki omogočajo postopno dodajanje in izboljševanje funkcionalnosti. Na primer, najprej sem na prvem zavihku dodajal nove kazalnike, nato sem na drugem zavihku izboljševal model, na tretjem pa sem izboljšal preglednost predstavljenih podatkov. Prilagodljivost na spremembe, ki je značilna za scrum, sem prav tako vpeljal v razvoj. Ob

morebitnih nepričakovanih težavah, kot so omejitve dostopa do podatkov, sem razvoj prilagodil, da sem našel ustrezne rešitve.

Načela testno vodenega razvoja sem uporabljal skozi celoten cikel razvoja. Kodo sem testiral sproti in na podlagi rezultatov testov sproti prilagajal in izboljševal funkcionalnosti. Preden sem nadaljeval z razvojem, sem vedno preveril, ali aplikacija deluje ustrezno. Poleg tega sem upošteval tudi načela funkcionalno vodenega razvoja, ki temeljijo na iterativnem dodajanju posameznih funkcionalnosti in osredotočenosti na hitro implementacijo novih komponent. Po dodajanju funkcionalnosti sem jih testiral, nato pa nadaljeval z razvojem naslednjih elementov aplikacije. Čeprav sem projekt izvajal sam, sem združil ključna načela teh metodologij in jih prilagodil tako, da so ustrezala naravi in obsegu mojega dela.

4.1 Implementacija prototipa in osnovni opis

Najprej sem želel vzpostaviti osnovni pogled uporabnika aplikacije, kar sem določil kot prototip. Prototip bo torej končan, ko bo uporabnik lahko preskakoval med zavihki in bo vzpostavljeno spletno okolje. Zaradi preglednosti sem se odločil razdeliti programsko rešitev na štiri Python skripte:

- app.py – to je osnovna skripta aplikacije. Na tej strani se nahajajo instanca aplikacije, osnovni uvozi in zagon aplikacije. Prav tako sem na tej strani uvozil osnovni paket *Dash*. Kreiral sem spremenljivko *app*, na katero se vežejo vse povratne funkcije, tudi tiste z drugim skriptom oziroma v našem primeru tudi tiste iz drugih zavihkov.
- stock_information.py – ta skripta se nanaša na prvi zavihek aplikacije. Imenoval sem ga *Stock Information.* Uporabnik na tem zavihku lahko dostopa do grafičnega pogleda zgodovinskih podatkov želene delnice. Do teh podatkov dostopa tako, da v vnosno polje vpiše znak delnice in pritisne na gumb *Get Stock Information*. V tabelo pod vnosnim poljem se shranijo osnovni podatki o delnici. V tabeli je lahko več delnic. Uporabnik lahko podatke posameznih delnic s pritiskom na gumb *X* odstrani iz tabele. Uporabnik nadaljuje z izbiro periode in intervala s pomočjo dveh spustnih menijev. Perioda se nanaša na to, kako daleč v preteklost želi uporabnik videti zgodovinske podatke, interval pa se nanaša na to, kako pogosti naj bodo ti zgodovinski podatki. Po izbiri periode in intervala uporabnik pritisne na gumb *Get Historical Data*. Uporabniku se pod spustnima menijema prikaže graf, na katerem lahko vidi graf vseh delnic, ki so se ob kliku na gumb *Get Historical Data* nahajale v tabeli. Po tem se lahko uporabnik odloči za izbiro kazalnika, ki bi ga želel prikazati na grafu, z izbiro prek spustnega menija.
- prediction_model.py – ta skripta se nanaša na drugi zavihek aplikacije. Imenoval sem ga *Prediction Model.* Uporabnik na tem zavihku dostopa do cene delnice v prihodnosti na podlagi različnih modelov strojnega učenja. Uporabnik na tem zavihku sam določi model, ki ga bo izbral za napovedovanje cene delnice v prihodnosti. Prav tako sam določi periodo zgodovinskih podatkov delnice, ki bodo vplivali na izračun cene, in določi, na kateri dan v prihodnosti želi napovedati ceno delnice. Enako kot v prejšnjem zavihku je

v tem zavihku vnosno polje, kamor uporabnik vnese delniški znak in gumb za pridobivanje podatkov o delnici. Osnovni podatki o delnici se vnesejo v tabelo pod vnosnim poljem in gumbom. Pod tabelo se nahajajo spustni meni za izbiro modela, gumb za izbiro periode zgodovinski podatkov, gumb za izbiro dneva, za katerega bom naredil napoved cene delnice, in gumb *Train Model.* S pritiskom na slednji gumb se izriše graf, ki vsebuje zgodovinske cene delnice, kako se v nekem obdobju cene delnice razlikujejo od napovedane cene delnice, ceno delnice v prihodnjem obdobju in evalvacijo modela, ki ga je uporabnik izbral.

- stock_news.py – ta skripta vsebuje kodo tretjega zavihka. Imenuje se *Stock News.* Uporabnik prek vnosnega polja vpiše delniški znak, nato s klikom na gumb pridobi podatke o novicah delnice, s katero se želi spoznati. Podatki o novicah delnice se zabeležijo v tabelo. V tabeli niso napisane celotne novice oziroma članki delnice, ampak samo naslov, vir, datum, predogled in povezava do celotnega članka.

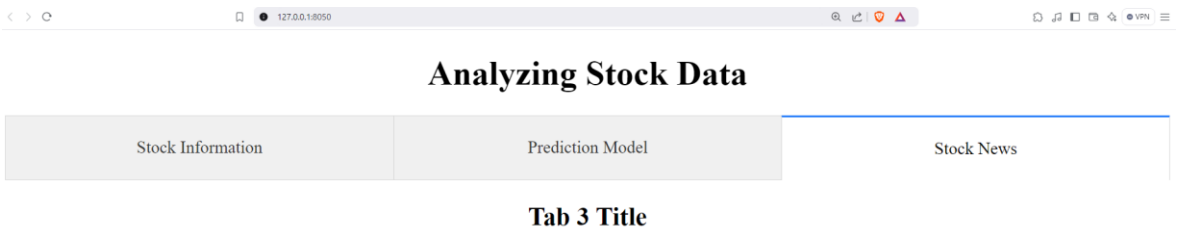
Za to, da sem implementiral prototip oziroma vzpostavil spletno okolje in razdelil zavihke med tri različne skripte, je bilo treba začeti s skripto *app.py*, ostale skripte pa sem napolnil le z naslovom zavihka, tako da ob implementaciji prototipa zavihki niso imeli nobene. V skripti *app.py* sem poleg zgoraj omenjenega osnovnega paketa *Dash* uvozil vse tri skripte, ki so se morale nahajati v isti mapi kot skripta *app.py*. Uvozil sem tudi še dva druga osnovna paketa Dash spletnega ogrodja. To sta *Dash Core Components* in *Dash HTML Components*, ki se iz paketa *dash* uvozi kot *dcc* in *html.* Prvi mi je služil za uvoz elementa *Tabs*, saj sem želel, da se mi na vrhu prikazujejo trije zavihki, med katerimi bodo uporabniki lahko preklopili. Element *Tabs* pa ima za vsak zavihek tudi svoj element *Tab* znotraj parametra *children*, kjer sem definiral imena zavihkov in njihovo unikatno vrednost, ki jo potem uporabljam za sklicevanje na ta zavihek. V tej skripti so definirani tudi trije HTML-elementi:

- celoten *Div*, znotraj katerega so definirani vsi ostali elementi,
- naslov aplikacije, ki je definiran na vrhu,
- *Div*, ki prikazujejo vsebino trenutno izbranega zavihka.

Za prikazovanje vsebine trenutno izbranega zavihka sem v skripto moral vključiti povratne funkcije. Povratne funkcije sem lahko začel uporabljati tako, da sem iz modula *dash.dependencies* uvozil *Input*, *Output* razreda. *Input* poskrbi za proženje povratne funkcije in *Output* je tisto, kar povratna funkcija vrne. Da sem dosegel vsebino trenutnega zavihka, sem napisal en *if* stavek in dva *elif* stavka. Ko je vhod enak določeni unikatni vrednosti zavihka, naj povratna funkcija vrne spremenljivko *layout* tiste skripte, ki se nanaša na ta zavihek. *Layout* spremenljivka v vsaki izmed teh treh skript vsebuje vse *dcc*- in *html*-elemente in tako prikazuje vizualne komponente. Te tri skripte sem uvozil na vrhu skripte *app.py*, kar sem opisal v prejšnjem odstavku. Elementu *Tabs* sem dodal še dva parametra, ki izboljšata uporabniško izkušnjo. Nastavil sem *persistence* na stanje *True* in *persistence_type* na stanje *session*. Tako uporabnik ohrani pogled tistega zavihka, ki ga je izbral v primeru osvežitve brskalnika. Zavihek se ohrani do konca seje aplikacije.

Na sliki 6 je prikazana implementacija prototipa, ko uporabnik izbere tretji zavihek. Na vrhu je prikazan naslov aplikacije, sledijo zavihki, nato pa sledi še naslov tretjega zavihka, ki je trenutno izbran. Vzrok, da je besedilo na zaslonski sliki v angleščini, je to, da je spletna aplikacija narejena v angleščini.

Slika 6: Implementacija prototipa



Vir: lastno delo.

4.2 Pridobivanje podatkov

V poglavju 2.3 sem že definiral pridobivanje podatkov s strganjem s spleta. Za ta način pridobivanja podatkov sem se odločil na začetku. S pomočjo knjižnic *BeafutifulSoup* in *requests* sem poslal klic na spletno stran Yahoo Finance in pridobil osnovne podatke o izbrani delnici, ki sem jo izbral z vnosno masko. Po tem, ko sem napisal kodo, ki vsebuje pridobivanje HTML-strukture in nato ekstrakcijo želenih podatkov v aplikacijo, sta pretekla približno dva meseca, ko sem nadaljeval z razvojem. V tem času sem opazil, da je spletna platforma Yahoo Finance spremenila svojo HTML-strukturo, kar pomeni, da bi za pridobivanje istih podatkov o delnici in shranjevanje v osnovno tabelo moral posodobiti svojo kodo. Ta način pridobivanja podatkov ni konsistenten in pravočasen, zato ni primeren kot končna rešitev pridobivanja osnovnih podatkov, prav tako pa ni dobro izhodišče za nadaljnje pridobivanje zgodovinskih podatkov.

Podatke o delnicah bi lahko pridobil tudi s katerih drugih spletnih platform, ampak Yahoo Finance je znana kot najbolj ažurna platforma s širokim naborom delnic. Prav tako bi se pridobivanja podatkov lahko lotil z ročnim prenosom naborov želenih delnic in uvažanjem v Excel, kar pa bi zmanjšalo fleksibilnost in enostavnost pridobivanja podatkov za uporabnike. Prav tako bi bilo treba ročno ali z avtomatizacijo skript skrbeti za pridobivanje podatkov za določen nabor delnic, kar bi zmanjšalo celoten spekter izbire uporabnika. Možna uporaba bi bili tudi neuradni aplikacijski programski vmesniki (angl. API), kot so *RapidAPI*, kjer z registracijo pridobiš API-ključ in se ključ uporabi v lastni kodi za

odklepanje dostopa do podatkov. Ta način je boljši od ročnega pridobivanja izbranega nabora podatkov, saj omogoča širši nabor, vendar se za ta način nisem odločil, saj lahko z večjim številom poizvedb vključuje stroške. Yahoo Finance je svoj uradni aplikacijski programski vmesnik ukinil leta 2017.

Aplikacijski programski vmesnik je komunikacijski vmesnik, ki omogoča prenos zahtevkov in odgovorov med uporabniki aplikacije in strežniškim sistemom v ozadju. Deluje kot most med različnimi komponentami programske opreme, kar omogoča aplikacijam, da komunicirajo in delujejo usklajeno. API ponuja definirane metode in protokole za pošiljanje zahtevkov ter prejemanje odgovorov, kar zagotavlja standardiziran način interakcije med aplikacijami ali storitvami (Chan in drugi, 2019)

V tem času pa se je prav zaradi močne Pythonove skupnosti, omenjene v poglavju 3.2.1, razvila knjižnica, imenovana *yfinance*. V tem primeru knjižnica *yfinance* deluje kot vmesnik med spletno platformo Yahoo Finance in aplikacijo, ki jo bom razvil. V zavihkih *Stock Information* in *Prediction Model* sem uvozil knjižnico in jo shranil kot alias *yf*. Ta knjižnica ima razred, imenovan *Ticker*, ki sprejme kot argument tisti delniški znak, ki sem ga vpisal v vnosni maski. Nato sem z uporabo ekstrakcije slovarja *info* pridobil osnovne podatke o delnici, ki sem jo vpisal v vnosni maski in shranil v spremenljivko.

Poleg pridobivanja osnovnih podatkov o delnici pa je bilo treba v zavihkih *Stock Information* in *Prediction Model* pridobiti tudi zgodovinske podatke o cenah delnic. V zavihku *Stock Information* sem na podlagi zgodovinskih podatkov lahko izračunal ekonomske kazalnike. V zavihku *Prediction Model* pa sem uporabili zgodovinske podatke kot vhodne podatke za napoved cene delnice v prihodnosti. Do zgodovinskih podatkov sem prišel prek funkcije *history* znotraj razreda *Ticker*. Argumenti, ki sem jih podal tej funkciji, so bili delniški znak, interval zgodovinskih cen in perioda zgodovinskih cen.

Poleg pridobivanja podatkov o delnicah je bilo treba v zavihku *Prediction Model* pridobiti ekonomske kazalnike. Kazalnike je bilo treba pridobiti iz drugega vira, kot je Yahoo Finance, zato sem podatke zajel drugače. Primerna spletna stran, ki vsebuje podatke obrestne mere, stopnje brezposelnosti in inflacije, je FRED. Omenjene podatke sem pridobil za ameriški trg, saj tudi Yahoo Finance vsebuje največ ameriških delnic. Tudi v primeru pridobivanja podatkov s spletne strani FRED že obstaja Python knjižnica *fredapi*. Ta knjižnica vsebuje razred *Fred*, ki mu je treba v argument zapisati API-ključ. Torej, da lahko dostopam do ekonomskih kazalnikov, ki sem jih naštel zgoraj, je treba opraviti registracijo na spletni strani FRED in pridobiti API-ključ. Ta ključ sem nato vpisal kot argument znotraj razreda *Fred*.

V tretjem zavihku *Stock News* sem pridobil podatke prek spletne strani *finnhub.io*. Ta spletna stran omogoča pridobivanje novic o posameznih delnicah prek aplikacijsko programskega vmesnika. Novice, ki se pridobijo s te spletne strani, so pridobljene z različnih virov, tako da ta API razvijalcu olajša delo. Z registracijo na njihovi spletni strani sem pridobil

brezplačni API-ključ. Z brezplačnim ključem se lahko pridobijo podatki za leta nazaj za katerokoli delnico.

Finnhub pridobiva tržne podatke neposredno z borz, temeljne podatke iz poročil podjetij, ETF-podatke pa iz poročil izdajateljev. Podatki se razlikujejo od tistih pri Bloombergu, Reutersu in drugih, saj uporabljajo lastne metode za pridobivanje in standardizacijo (Finnhub, 2024).

4.3 Uporabniški vmesnik

Uporabniški vmesnik (angl. User Interface) je osrednja študija interakcije med človekom in računalnikom. Uporabniški vmesniki so dostopne točke, kjer uporabnik komunicira z dizajnom. To je najbolj razširjena definicija. Oblikovalčeva naloga je ustvariti uporabniški vmesnik, kjer najbolje izkoristi dostopne točke za uporabo, in ugotoviti, kakšne interakcije so mogoče (Morales Díaz, 2022).

Na sliki 7 sta grafično prikazani interakcija med človekom in računalnikom ter vloga uporabniškega vmesnika. Odnos se lahko definira v šestih korakih, ki se med seboj povezujejo. Kot začetni korak se interpretirajo možgani, saj definicija izhaja iz človekove percepcije. Možgani zaznajo informacije o stanju računalnika in z motoričnimi gibi opravijo vnos informacij. Ta vnos vpliva na spremembo stanja računalnika in lahko privede do akcije, ki jo računalnik opravi. To se imenuje izhod računalnika. Človek s svojimi čutnimi sposobnostmi zazna spremembo, ki jo je opravil računalnik, in jo ovrednoti v možganih. Sedaj lahko človek opravi nov motorični gib in ponovno vzpostavi povezavo med sabo in računalnikom.

Slika 7: Uporabniški vmesnik



Vir: prirejeno po Morales Díaz (2022).

Definicija, ki sem jo opredelil zgoraj, se v zadnjih letih opušča, saj ni dovolj celovita. V porastu je naslednja definicija. Uporabniški vmesnik je artefakt, zasnovan z namenom, da povezuje računalniško napravo in človeškega uporabnika, omogoča učinkovito povezavo med motoričnimi, senzornimi, kognitivnimi in voljnimi sposobnostmi človeka ter vhodnimi, procesnimi in izhodnimi zmogljivosti naprave za določen nabor nalog, povezanih z določenimi predmeti (Morales Díaz, 2022).

V programski rešitvi, ki sem jo razvil, uporabnik za vnos podatkov uporablja prste, tipkovnico in miško. V prvem zavihku uporabnik lahko komunicira z računalnikom prek naslednjih elementov:

- *dcc.Input* – gre za vnosno polje. Elementu se določi unikatni identifikator (*id*), ki mi pomaga v kodi, da dostopam do tega elementa. Temu elementu sem določil tudi tip podatka, ki ga pričakuje. V mojem primeru je to tekstovni zapis. Določil sem tudi dva parametra, ki sem ju opisal zgoraj. Nastavil sem *persistence* na stanje *True* in *persistence_type* na stanje *session.* Podatkovni vnos ostane v elementu, dokler se ne vpiše novega podatka ali pa se končna seja.
- *html.Button* – gre za gumb, na katerega pritisne uporabnik, da pridobi podatke o tistem delniškem znaku, ki ga je vpisal v vnosno polje. Tudi temu elementu sem določil unikatni identifikator (*id*). Določil sem tudi parameter *n_clicks* na 0, saj ima gumb takšno število pritiskov ob začetku seje aplikacije.
- *DataTable* – gre za tabelo, v kateri lahko uporabnik odstranjuje že vpisane podatke o posamezni delnici. Tabela ima unikatni identifikator (*id*), našteta imena stolpcev v definiciji, parameter *row_deletable* je nastavljen na *True*, zato da lahko uporabnik odstrani vrstico. Tabeli sem s parametri nastavil tudi izgled in obliko. Imena stolpcev v tabeli so: *Stock Ticker, Stock Name, Current Price, Open, Previous Close, Volume, Avg. Volume, PE Ratio, EPS.* Vsak stolpec ima unikatni identifikator (*id*).
- *dcc.Dropdown* – gre za spustni meni, kjer uporabnik določi periodo uporabe zgodovinskih podatkov. Spustni meni ima unikatni identifikator (*id*). Opcije, ki so ponujene uporabniku, so zapisane kot parameter. Gre za seznam slovarjev. Parameter *clearable* je nastavljen na *False,* saj uporabnik mora vedno izbirati med eno izmed danih opcij. Ta element ima kot dcc.Input nastavljena parametra *persistence* na stanje *True* in *persistence_type* na stanje *session.*
- *dcc.Dropdown* – gre za spustni meni, kjer uporabnik določi interval natančnosti zajema zgodovinskih podatkov. Spustni meni ima unikatni identifikator (*id*). Opcije, ki so ponujene uporabniku, so zapisane kot parameter. Gre za seznam slovarjev. Parameter *clearable* je nastavljen na *False,* saj uporabnik mora vedno izbirati med eno izmed danih opcij. Ta element ima nastavljena parametra *persistence* na stanje *True* in *persistence_type* na stanje *session.*
- *html.Button* – gre za gumb, na katerega pritisne uporabnik, ko želi prikazati grafe s podatki v tabeli in izbranimi opcijami v zgoraj opisanih spustnih menijih. Tudi ta gumb

ima unikatni identifikator (*id*). Določil sem tudi parameter *n_clicks* na 0, saj ima gumb takšno število pritiskov ob začetku seje aplikacije.

- *dcc.Dropdown* – gre za spustni meni, kjer uporabnik določi izbrani kazalnik, ki ga želi prikazati. Spustni meni ima unikatni identifikator (*id*). Opcije, ki so ponujene uporabniku, so zapisane kot parameter. Gre za seznam slovarjev. Parameter *clearable* je nastavljen na *True*, saj lahko uporabnik odstrani kazalanik, če ga ne želi odstraniti z grafa. Ta element ima nastavljena parametra *persistence* na stanje *True* in *persistence_type* na stanje *session*.
- *dcc.Graph* – gre za zadnji element, s katerim lahko uporabnik operira v tem zavihku. Uporabi lahko približevanje grafa in se tako osredotoči bolj na nekatera časovna obdobja, ki jih želi proučiti. Tudi ta element ima unikatni identifikator (*id*).

Na zgornjem seznamu omenjenih elementov v prvem zavihku, s katerimi ima uporabnik lahko interakcijo, jih večina lahko hrani podatke z vgrajenimi parametri. Le gumba, ki sta omenjena, ne moreta hraniti vidnih podatkov na zaslonu uporabnika. Ker želim uporabniku omogočiti čim boljšo uporabniško izkušnjo z enostavnim uporabniškim vmesnikom, sem razvil programsko kodo tako, da se vrednosti, ki jih je izbral, vpisal ali prikazal uporabnik, ohranijo, dokler jih ne spremeni sam ali pa se seja zaključi. Večina elementov, ki hranijo informacije, ima za to že vgrajena parametra *persistence* in *persistence_type*. Elementa, ki shranita podatke in nimata omenjenih parametrov, sta *DataTable* in *dcc.Graph.*

Za ta dva elementa sem moral razviti povratni funkciji, ki sta shranili podatke teh dveh elementov v elementa, ki lahko hranita podatke skozi sejo, in se bosta ob osvežitvi strani prožili tako, da bosta vrnili podatke iz elementov, ki hranijo vrednost, v *DataTable* in *dcc.Graph*. Element, ki lahko hrani vrednost, se imenuje *dcc.Store.* Ta element ni grafično prikazan v uporabniškem vmesniku, vendar je naveden v spremenljivki *layout*. Določi se mu unikatni identifikator (*id*) in parameter *storage_type* na vrednost *session.* Uporabil sem dva elementa *dcc.Store*, enega za shranjevanje tabele in enega za shranjevanje grafa.

Prvo povratno funkcijo sem poimenoval *update_row_data_with_stored_data.* Ta funkcija vzame podatke iz elementa *dcc.Store* kot *Input* in tabeli *DataTable* pripiše te podatke kot *Output.* Ta povratna funkcija se je sprožila vedno, ko sem klical ta zavihek ali osvežil aplikacijo. V kodo sem dodal tudi *if* stavek, ki funkciji ne vrne *Output* v primeru, da ni podatkov v elementu *dcc.Store.*

Drugo povratno funkcijo sem poimenoval *load_graph_from_store*. Ta funkcija vzame podatke iz elementa *dcc.Store* kot *Input,* te podatke nato pretvori v grafično obliko modula *go* in razreda *Figure* v primeru, da so podatki v elementu *dcc.Store*. Nato se prilagodi višina grafov v funkciji in vrne grafična oblika kot *Output*. Izhod je torej *dcc.Graph* in na ta način se ohrani graf v tem zavihku.

Še ena pomembna povratna funkcija, ki sem jo razvil v prvem zavihku, je *update_store_data_on_row_deletion*. To povratno funkcijo je bilo treba razviti zato, ker ima

DataTable element omogočen brisanje vrstic tabele. Doseči je bilo treba to, da se v primeru odstranitve vrstice v tabeli odstrani tudi vrstica v *dcc.Store* elementu, ki hrani podatke o tej vrstici. Če te povratne funkcije ne bi bilo, bi, kadar bi izbrisali vrstico v tabeli in nato zamenjali zavihek ali osvežili stran, prišlo do nalaganja te vrstice ponovno v tabelo s proženjem povratne funkcije *update_row_data_with_stored_data.* V tej povratni funkciji sta dva vhodna argumenta. To sta trenutno stanje podatkov tabele in prejšnje stanje podatkov tabele. To, da se ohrani prejšnje stanje podatkov tabele, je odlična lastnost *DataTable* elementa. V primeru, da je dolžina podatkov trenutnega stanja tabele manjša od dolžine podatkov prejšnjega stanja tabele, se trenutni podatki ohranijo. Tej povratni funkciji je bilo treba vključiti parameter *prevent_initial_call* na stanje *True*, kar pomeni, da se ob osvežitvi ni avtomatsko prožila. Prav tako je bilo na izhodni argument – *Output,* ki je *dcc.Store* za shranjevanje podatkov tabele, treba nastaviti parameter *allow_duplicate* na stanje *True*. To je treba narediti vedno, ko je v več različnih povratnih funkcijah nastavljen enak izhodni argument. Zaenkrat druge povratne funkcije, kjer je *dcc.Store* za shranjevanje podatkov tabele razvit kot izhodni argument, še nisem opisal. Opisane tri povratne funkcije so bistvo zagotavljanja konsistentnih podatkov v prvem zavihku skozi celotno sejo aplikacije.

Zadnja povratna funkcija v prvem zavihku, ki obsega del uporabniškega vmesnika, se imenuje *update_interval_options,* ker knjižnica *yfinance* ne omogoča pridobivanja podatkov na minutnem ali 60-minutnem intervalu za periodni čas, daljši od enega dne. Vhodni argument te povratne funkcije je izbrana perioda. Če je perioda enaka vrednosti enega dne, potem ima uporabnik na voljo intervale na minutni in 60-minutni ravni. V primeru, da perioda ni enaka enemu dnevu, ima uporabnik na voljo dnevne in mesečne intervale. Izhodni argument povratne funkcije je seznam intervalnih možnosti, ki se uporabniku prikažejo v spustnem meniju.

V drugem zavihku, ki sem ga poimenoval *Prediction Model,* sem uporabljal zelo podobne elemente. Zaradi zahtevnosti in časa trajanja izvajanja napovedi modela sem omogočil le en prikaz zgodovinskih in prihodnjih cen delnic grafa. Uporabnik v tem zavihku uporablja naslednje elemente za komunikacijo z računalnikom:

- *dcc.Input* – gre za vnosno polje delniškega znaka. Za razliko od prvega zavihka se podatke o delnici pridobi samo z enim gumbom in na tej strani se ne pridobivajo osnovne informacije o delnici. Temu elementu se določi unikatni identifikator (*id*), pričakovan tekstovni podatkovni tip in parametra *persistence* na stanje *True* in *persistence_type* na stanje *session.*
- *dcc.Dropdown* – gre za spustni meni, kjer uporabnik določi model napovedovanja cene delnice v prihodnosti. Spustni meni ima unikatni identifikator (*id*). Ta element ima parametra *persistence* na stanje *True* in *persistence_type* na stanje *session*. Uporabnik izbira med vnaprej določenimi modeli, ki morajo biti konfigurirani v kodi.
- *dcc.RadioItems* – gre za element, kjer uporabnik odloča med vnaprej določenimi možnostmi. V našem primeru uporabnik odloča o časovnem obdobju zgodovinske cene delnice, ki bo vključeno v model za napovedovanje cene. Element ima unikatni

identifikator (*id*), parameter *persistence* in parameter *persitence_type* konfigurirana enako kot pri ostalih elementih.

- *dcc.RadioItems* – gre za element, kjer uporabnik odloča, za koliko dni vnaprej želi napovedati ceno delnice. Element ima unikatni identifikator (*id*), parameter *persistence* in parameter *persitence_type* konfigurirana enako kot pri ostalih elementih.
- *html.Button* – gre za element, ki proži povratno funkcijo, ki izračuna in prikaže napoved cene v prihodnosti. Ta element ima unikatni identifikator (*id*). Določil sem tudi parameter *n_clicks* na 0, saj ima gumb takšno število pritiskov ob začetku seje aplikacije.
- *dcc.Graph* – s tem elementom si uporabnik prikaže gibanje cene delnic – predhodno in napovedno. Ta element ima unikatni identifikator (*id*). Za razliko od grafa na prvem zavihku se ta element nahaja znotraj še enega elementa, ki izboljšuje uporabniško izkušnjo. Imenuje se *dcc.Loading*, kar omogoča, da dokler model izračunava vrednosti, uporabnik na zaslonu vidi prikaz izvajanja kalkulacije. Tako uporabnik ve, da poteka izračun in da aplikacija deluje v skladu s pričakovanji. Ko se izračun zaključi, se graf prikaže in prikaz izvajanja kalkulacije izgine.

Podobno kot v prvem zavihku je tudi v drugem zavihku element *dcc.Graph*, ki ne vsebuje parametrov za shranjevanje skozi sejo. Razvil sem podobno povratno funkcijo *load_graph_from_store,* ki se kliče ob osvežitvi strani ali zavihka in tako kliče podatke iz *dcc.Store*, ki ima parametra unikatni identifikator (*id*) in *storage_type* nastavljena na stanje *session*. Ti podatki tega elementa so argumenti povratne funkcije, pretvorijo se v objekt modula *go* in razreda *Figure*. *Output* te povratne funkcije pa je graf *dcc.Graph*.

V tretjem zavihku, ki sem ga poimenoval *Stock News*, sem uporabil naslednje elemente, ki so vplivali na uporabniško interakcijo:

- *dcc.Input* – gre za element vnosnega polja, v katerega se vnese delniški znak. Nastavil sem mu tudi prvotno vrednost s parametrom *placeholder*: Enter stock ticker. Tako uporabnik dobi navodila in ve, kaj mora storiti, da bo pridobil novice o delnici. Spremenil sem tudi stil temu vnosnemu polju in mu priredil unikatni identifikator (*id*). Prav tako sem s parametrom *type*, ki sem ga nastavil na vrednost *text*, določil, katero vrsto podatka pričakuje vnosno polje.
- *html.Button* – na tem gumbu piše *Get News.* Uporabnik s tem gumbom pridobi novice o delnici, katere delniški znak je vpisal v vnosno polje. Tudi ta gumb ima unikatni identifikator (*id*), da se ga lahko uporabi v povratnih funkcijah.
- *html.Div* – temu elementu sem nastavil unikatni identifikator (id). Kasneje ga bom uporabil zato, da v njega shranim delniški znak, za katerega novice o delnici sem pridobil. Tako bo uporabnik, tudi če bo zamenjal zavihek, vedel, katere delnice so prikazane v tabeli.
- *DataTable* – gre za tabelo, v katero bom shranil novice delnic, ki sem jih pridobil. Tej tabeli sem priredil unikatni identifikator (*id*), nato sem dodal parameter *columns*, kjer sem znotraj seznama naštel več slovarjev. En slovar predstavlja en stolpec v tabeli. Ima

dve vrednosti ključev *name* in *id*. Vsak ključ ima svojo vrednost, tako sem vsakemu stolpcu določil ime in unikatni identifikator. Parameter *row_deletable* sem nastavil na stanje False. Nato sem s parametri style nastavil obliko in izgled tabele ter s parametrom *markdown_options* določil, da lahko v tabelo shranim elemente HTML-datotek. Tako bo uporabnik lahko iz tabele pritisnil na povezavo, ki ga bo pripeljala do novice na spletu.

Uporabniško izkušnjo sem izboljšal v tretjem zavihku z eno povratno funkcijo, ki sem jo poimenoval *update_news_table*. Ta funkcija kot vhodni argument sprejme *dcc.Store* element, ki hrani slovar podatkov. Ta slovar podatkov ima dva ključa, imenovana *news* in *title*. Vrednosti teh dveh ključev se nato vrneta v izhod povratne funkcije. Izhoda (*Output*) povratne funkcije sta dva: podatki tabele *DataTable* in pa *html.Div,* ki shranjuje naslov strani. V naslovu je shranjen delniški znak, zato uporabnik ve, katera delnica se prikazuje v tabeli. V primeru, da ničesar ni shranjenega v *dcc.Store,* rezultat vrne prazno tabelo in naslov.

4.4 Obdelava podatkov

V poglavju 2.3 sem definiral obdelavo podatkov s pomočjo raziskave literature. V tem delu magistrskega dela se osredotočim na obdelavo podatkov v programski rešitvi, ki sem jo razvil. Pomembno je opozoriti, da pri dinamični programski rešitvi ne gre za vnaprej izvedeno obdelavo podatkov. Uporabnik z ukazi na tipkovnici ali miški sproži obdelavo podatkov, ki pa mora biti predvidena in izvedena v skladu s pričakovanji. Razvijalec torej pripravi programsko kodo, ki pričakuje podatke v neki obliki. Programska koda procesira te podatke in jih vrne uporabniku.

Uporabniški vmesnik prvega zavihka sem opisal v poglavju 4.3. Sledi opis povratnih funkcij prvega zavihka, kjer se osredotočim na programsko kodo potovanja podatkov. Ko uporabnik vpiše delniški znak v vnosno polje in pritisne gumb *Get Stock Information,* se izvede obdelava podatkov. S pritiskom na gumb se sproži povratna funkcija *update_output.* Ta povratna funkcija ima tri vhodne argumente – *Input* in dva izhodna argumenta – *Output*. Cilj te povratne funkcije je pridobiti osnovne informacije delnice podjetja v vrstico tabele, ki jih uporabnik želi pridobiti z vpisom delniškega znaka. Prvi vhodni argument je gumb *Get Stock Information,* druga dva vhodna argumenta sta stanji, ki se jih v spletnem ogrodju Dash označi s *State*. Ob kliku na gumb *Get Stock Information* se v drugi v drugi in tretji vhodni argument shrani stanje delniškega znaka v vnosnem polju *dcc.Input* in stanje tabele, shranjeno v elementu *dcc.Store.* Povratna funkcija najprej pridobi podatke na način, ki sem ga opisal v poglavju 4.2 Pridobivanje podatkov. Ko so osnovne informacije o delnici shranjene v spremenljivki *stock_info,* ki je slovar, se iz slovarja pridobijo posamezne informacije o delnici, razdelijo in shranijo vsako v svojo spremenljivko. To se izvede z metodo *get.* Vsem spremenljivkam se v primeru, da ne pridobijo informacije iz slovarja *stock_info,* pripiše vrednost *N/A.*

V tabeli 2 je v prvem stolpcu napisana posamezna spremenljivka, v katero sem shranil vrednosti, ki jih koda pridobi iz slovarja *stock_info* z metodo *get*. V drugem stolpcu so napisani ključi slovarja *stock_info*, iz katerih so bile pridobljene informacije z zgoraj omenjeno metodo.

Tabela 2: Spremenljivke in ključi delniških informacij

posamezna spremenljivka	stock_info ključ
open_price	open
prev_close	previousclose
current_price	currentprice
volume	volume
avg_volume	averagevolume
pe_ratio	trailingpe
eps	trailingeps

Vir: lastno delo.

Ko so vrednosti shranjene v posameznih spremenljivkah, sem kreiral novo spremenljivko imenovano *new_row*. Gre za slovar, kjer sem ključu določil enako vrednost unikatnega identifikatorja (*id*) stolpcev v tabeli *DataTable*. Kot vrednosti sem določil novo določene spremenljivke.

Nadaljeval sem s preverjanjem, ali je stanje trenutne vrednosti tabele v elementu *dcc.Store* enako *None*. Če je pogoj izpolnjen, sem to stanje spremenil v prazen list, saj sem v naslednjem koraku temu listu dodal spremenljivko *new_row*. Nato sem zaključil povratno funkcijo in z ukazom *return* vrnil list v oba izhodna argumenta. Izhodna elementa sta bila:

- podatki v elementu *dcc.Store,*
- podatki v elementu *DataTable.*

Tema dvema izhodnima elementoma v tej povratni funkciji je bilo treba določiti še parameter *allow_duplicate* na stanje *True.* To pa zato, ker je element *dcc.Store*, ki hrani podatke tabele skozi celotno sejo, že zapisan kot *Output* v drugi povratni funkciji *update_store_data_on_row_deletion*, ki se sproži ob brisanju vrstice v tabeli. Element *DataTable* pa se uporabi kot *Output* v povratni funkciji *update_row_data_with_stored_data*, ki se kliče ob osvežitvi strani ali preklopu zavihka. Povratna funkcija *update_output* shrani osnovne informacije delnice, ki so z delniškim znakom zapisane v vnosnem polju, v tabelo.

Naslednja povratna funkcija v prvem zavihku se imenuje *update_historical_data.* Gre za povratno funkcijo, ki ima štiri vhodne argumente in tri izhodne argumente. Bistvo te povratne funkcije je, da izriše graf zgodovinskih cen delnice in ima parameter *prevent_initial_call* na stanje *True*, da preveri izvedbo ob osvežitvi strani ali preklopu zavihka. Vhodni argumenti te povratne funkcije so razdeljeni na en argument *Input* in tri argumente *State. Input* argument je klik na gumb *Get Historical Data.* Trije *State* argumenti so stanje podatkov v tabeli *DataTable,* stanje vrednosti v periodnem spustnem meniju in

stanje vrednosti v intervalnem spustnem meniju. Povratna funkcija se začne izvajati, če je pritisnjen gumb *Get Historical Data* in stanje v tabeli *DataTable* ni *None*. *None* je stanje, ko podatkov v tabeli ni. Nato sem deklariral spremenljivko *tickers* tako, da sem vanjo shranil vse delniške znake iz tabele. To sem naredil z zanko *for*, ki sem jo poslal skozi vse vrstice stolpca Stock Ticker. *Plotly* modul ima znotraj sebe modul *subplots*, iz katerega sem uvozil funkcijo *make_subplots*. Ta funkcija pričakuje večje število grafov tako, da se poda število stolpcev in vrstic. Želel sem imeti grafe prikazane enega pod drugim, zato sem določil število en stolpec in enačil število vrstic s toliko, kolikor je dolžina spremenljivke *tickers*. Določil sem parameter *shared_axes* na stanje *False*, saj ima vsaka delnica svojo os. Določil sem tudi naslov vsakega od teh grafov, ki bodo prikazani eden pod drugim. Naslove sem kreiral tako, da sem zanko *for* poslal skozi vse delniške znake in jim zraven dodal trenutno stanje spustnega menija periode in trenutno stanje spustnega menija intervala. Nadaljeval sem z določitvijo razmaka med grafi na vrednost 0,1. Celotno funkcijo *make_suplots* z zgoraj izbranimi parametri sem shranil v spremenljivko *fig*. Nato sem določil črno barvo vsem zaključnim cenam delnice.

Po tem sem se lotil razvoja nove *for* zanke. V tej zanki sem oštevilčil vse delniške znake, prek katerih skozi zanko pridobim vse zgodovinske podatke delnic. Vsakemu delniškemu znaku prek modula *yf* in razreda *Ticker* ter funkcije *history* pridobim zgodovinske podatke in jih shranim v spremenljivko *hist_data*. Funkciji *history* je bilo treba deklarirati tudi argumente periode in intervala, ki sem ju enačil s trenutnim stanjem spustnih menijev. Nato sem spisal pogojni stavek *if*, če v primeru, da funkcija ne vrne zgodovinskih podatkov, ignorira ta delniški znak in ga ne prikaže. Sledi dodajanje zaključnih cen delnice tistemu znaku, ki se obravnava v *for* zanki. Na X-osi sem deklariral indeks spremenljivke *hist_data*, na Y-osi pa sem deklariral zaključno ceno delnice spremenljivke *hist_data*. Ko se for zanka zaključi, spremenljivki *fig* spremenim višino s funkcijo *update_layout*. Višina vsakega grafa mora biti enaka ne glede na to, koliko grafov prikažem.

Ko je graf shranjen v spremenljivki *fig*, se lahko ta spremenljivka vrne kot argument v element *dcc.Graph* in se prikaže graf. Temu izhodu sem dodal tudi parameter *allow_duplicate* na stanje *True*, saj se ob klicu preklopa zavihka ali osvežitvi strani kliče povratna funkcija *load_graph_from_store*, kjer se že uporablja ta izhod. Drugi izhodni argument je *dcc.Store*, kamor se shrani tudi ta spremenljivka *fig*, le da jo shrani kot slovar. Tretji izhodni argument je vrednost indikatorja, ki se ji pripiše vrednost *None*. Pod naslednjim poglavjem analize podatkov bom opisal še zadnjo povratno funkcijo prvega zavihka *apply_indicator*.

V tem poglavju bi lahko opisal tudi povratno funkcijo *train_model* drugega zavihka, saj gre predvsem za obdelavo podatkov, ampak bom tu povratno funkcijo opisal v poglavju 4.6 Algoritmi za napovedovanje, saj je bistvo te povratne funkcije izračun napovedne cene delnice.

V nadaljevanju opišem obdelavo podatkov, ki se zgodi v tretjem zavihku *Stock News*. V prejšnjem poglavju 4.3 Obdelava podatkov sem opisal elemente tretjega zavihka in povratno funkcijo, ki se sproži ob osvežitvi zavihka. Sedaj opišem še obdelavo podatkov, ki se zgodi med pridobivanjem podatkov iz API spletne strani *finnhub.io* s shranjevanjem v tabelo *DataTable*. Povratno funkcijo, ki se sproži z gumbom *Get News,* sem poimenoval *fetch_and_store_news*. Ta funkcija ima dva vhodna argumenta in tri izhodne argumente. Gumb Get News je prvi vhodni argument – Input, medtem ko je stanje – *State* vnosnega polja *dcc.Input* drug vhodni argument. V spremenljivko *news_data* shranim vrednost, ki jo vrne funkcija *fetch_stock_news.* Ta funkcija kot argument vzame stanje vnosnega polja *ticker*, ki sem ga omenil prej. V vnosno polje se vpiše delniški znak. V funkciji *fetch_stock_news* najprej definiram API-ključ. Spremenljivki *api_key* se priredi vrednost ključa, ki sem ga pridobil na spletni strani *finnhub.io.* Nato v spremenljivko *today* shranim današnji dan z uvoženo knjižnico *datetime,* razredom *date* in funkcijo *today*. Nato s knjižnico *datetime* in razredom *timedelta,* ki mu dodelim argument *days* enak 365, pridobim datum eno leto nazaj in ga shranim v spremenljivko *one_year_ago*. Nato v spremenljivko *url* shranim povezavo za pridobivanje podatkov želene delnice. V to povezavo dinamično integriram spremenljivke *today, one_year, api_key* in *ticker*. Nato v spremenljivko *response* shranim vse novice, ki jih lahko pridobim za to delnico. To naredim tako, da knjižnico *requests*, ki sem jo uvozil, in funkcijo *get* z argument *url* enačim s to spremenljivko *response*. Nato sem napisal pogojni stavek, ki v primeru, da je *status_code*, ki se pridobi iz *response* spremenljivke, enak 200, vrne spremenljivko *response* v obliki JSON. V primeru, da je *status_code* nekaj drugega, se v terminalu prikaže napaka, da ni bilo mogoče pridobiti novic o delnici. Tako se zaključi funkcija *fetch_stock_news* in se vrne spremenljivka *response* v obliki JSON in se shrani v spremenljivko *news_data.*

Nato sem kreiral spremenljivko *formatted_news*, ki je seznam, in znotraj tega seznama naštel več slovarjev. Vsak slovar ima določen ključ, ki se v tem primeru ujema z imenom stolcpa v tabeli *DataTable*. S pomočjo *for* zanke se iz spremenljivke *news_data* pridobi spremenljivki *index* in *item* za vsako novico. V stolpec *number*, ki označuje številko novice po vrsti, se shrani *index* in se mu prišteje plus ena. Tako se prva novica z indeksom nič označi s številko ena. Naslov novice se pridobi iz spremenljivke *item*, iz katere se pridobi tudi vrednost *headline*. Vir novice se prav tako pridobi iz spremenljivke *item*, iz katere se pokliče vrednost *source*. Datum se shrani kot podatkovni tip datuma s knjižnico *datetime*, razredom *datetime* in funkcijo *fromtimestamp*, kateremu se kot argument poda vrednost *datetime* iz spremenljivke *item*. Pred klicem funkcije *timestamp* koda spremeni še vrednost *datetime* v znakovni niz. Predogled besedila sem pridobil iz spremenljivke *item* in vrednosti *summary*. Nato sem povezavo novice pridobil iz spremenljivke *item* in vrednosti *url*. Da ne prikazujem celotne povezave, sem za vsako vrstico priredil vrednost – *Read more*. Če uporabnik klikne na to vrednost, ga povezava vodi do spletne strani novice. Nato se v spremenljivko *title* shrani znakovni niz, ki je sestavljen iz delniškega znaka. Sledi kreiranje spremenljivke *session_data*, ki je slovar, ki ima kot prvi ključ *news* in vrednost spremenljivko *formatted_news,* kamor sem shranil vse novice. Drugi ključ je *title*, v katerega

sem shranil enako vrednost kot vrednost spremenljivke *title* znakovnega niza. Nato vrnem tri spremenljivke kot izhodne argumente povratne funkcije *fetch_and_store_news*.

Izhodni argument *formatted_news* pošljem v tabelo DataTable, da prikažem vse novice. Drugi argument je element *dcc.Store*, kamor shranim vse novice in naslov, tako da pošljem spremenljivko *session_data*. Tretji argument je element *html.Div*, s katerim prikažem naslov zavihka, ki prikazuje delniški znak delnice, katere novice so prikazane. Prvi in tretji argument se bosta ob osvežitvi izgubila. Vrednost teh dveh elementov se bo napolnila s povratno funkcijo *update_news_table,* ki sem jo opisal v prejšnjem poglavju. S tem sem zaključil opis tretjega zavihka in na sliki 8 je prikazan zaslonski pogled tretjega zavihka. Vzrok, da je besedilo na zaslonski sliki v angleščini, je to, da je spletna aplikacija narejena v angleščini.

4.5 Analiza podatkov

Analiza podatkov je postopek sistematičnega zbiranja, čiščenja, preoblikovanja, opisovanja, modeliranja in interpretacije podatkov, pri čemer se pogosto uporabljajo statistične metode. Analiza podatkov je pomemben del znanstvenih raziskav in poslovanja, kjer se je v zadnjih letih povečalo povpraševanje po odločanju na podlagi podatkov. Tehnike analize podatkov se uporabljajo za pridobivanje uporabnih vpogledov iz zbirk podatkov, ki jih je mogoče uporabiti za operativne odločitve ali usmerjanje prihodnjih raziskav. Zaradi porasta velikih podatkov, shranjevanja velikih količin podatkov v obsežnih bazah in skladiščih podatkov se povečuje potreba po uporabi tehnik analize podatkov za generiranje vpogledov iz količin podatkov, ki so prevelike, da bi jih lahko obdelovali instrumenti z nizko zmogljivostjo za obdelavo informacij (Eldridge, 2024).

Postopek zbiranja, čiščenja, preoblikovanja in opisovanja podatkov sem opisal že v prejšnjih poglavjih. To sem storil z opisom funkcij, ki so privedle do podatkov različnih izhodišč v različnih zavihkih. Sledi opis interpretacije podatkov, kjer opišem povratno funkcijo *apply_indicator,* ki prvemu zavihku omogoči zaključeno sliko. Uporabnik lahko tu sam ustvari analizo zgodovinskih podatkov, ki jih je zajel in prikazal na grafu. Grafu lahko doda kazalnike in s pomočjo kazalnikov interpretira rezultate.

Povratna funkcija *apply_indicator* ima štiri vhodne argumente in dva izhodna argumenta. Cilj povratne funkcije je grafični prikaz kazalnika, ki je izbran v spustnem meniju. Štirje vhodni argumenti se delijo na en vhodni sprožilec funkcije *Input* in tri vhodna stanja *State* ob začetku izvajanja funkcije. Vhodni sprožilec je spustni meni, ki vsebuje različne kazalnike. Tri vhodna stanja *State* so: stanje podatkov v tabeli *DataTable,* stanje izbire vrednosti v spustnem meniju periode in stanje izbire vrednosti v spustnem meniju intervala. Povratna funkcija gre nato čez podoben postopek kot povratna funkcija *update_historical_data,* saj se na novo pridobijo zgodovinski podatki. V primeru, da je vrednost spremenljivke kazalnika *selected_indicator*, s katerim se sproži povratna funkcija,

enaka *None*, potem se konča izvajanje povratne funkcije in povratna funkcija vrne enak rezultat, kot če bi se izvedla funkcija *update_historical_data*.

Slika 8: Zaslonska slika tretjega zavihka



#	Title	Source	Date	Preview Text	Link
1	Japan to propose $65 billion plan to aid domestic chip industry, draft shows	Finnhub	2024-11-11 08:06:22	Japan's government will propose a $65 billion plan to boost its chip industry with subsidies and other financial assistance over a period of multiple years, a draft seen on Monday by Reuters showed. ...	Read more
2	IBM faces suit over Weather Channel site users' data, Bloomberg says	Yahoo	2024-11-09 12:42:13	IBM (IBM), which owns The Weather Channel website, is facing a proposed class action alleging the company violated federal privacy law by sharing personal information of users who watched videos online to advertisers without consent, Bloomberg's Cassandre Coyer reports, citing the suit. The complaint claimed IBM deployed tracking technology on its weather.com site that disclosed users' personally identifiable information and video viewing data, including their location and viewing history, with	Read more
3	Walt Disney, Cisco, And Shopify Headline Latest Earnings Week	SeekingAlpha	2024-11-09 12:00:00	Stay ahead in the financial market with Seeking Alpha's Wall Street Week Ahead newsletter.	Read more
4	Dividend Champion, Contender, And Challenger Highlights: Week Of November 10	SeekingAlpha	2024-11-09 01:57:09	Stay up to date on dividend activity with our weekly summary for Dividend Champions, Contenders, and Challengers, including ex-dividend and pay dates.	Read more
5	International Business Machines Corporation (NYSE:IBM) Partners with Cognizant on AI and Cloud Innovation, New FinOps Center of Excellence to Drive Enterprise Modernization	Yahoo	2024-11-09 00:06:20	We recently published a list of 12 AI News and Ratings You Should Not Miss. In this article, we are going to take a look at where International Business Machines Corporation (NYSE:IBM) stands against other trending AI stocks on the latest analyst ratings and news. CNBC's Deidre Bosa joined 'Money Movers' to discuss the potential […]	Read more
6	IBM, Cognizant team up to tame rising enterprise cloud bills	Yahoo	2024-11-08 15:58:10	The two companies launched a FinOps Center of Excellence initiative to help ease hybrid and multicloud cost management.	Read more
7	IBM "AI in Action" Report Identifies Key Characteristics of Businesses That Consider Themselves Leaders in AI	Yahoo	2024-11-08 15:21:00	IBM (NYSE: IBM) released a new report called AI in Action that highlights artificial intelligence strategies of businesses considered "AI Leader The report is based on a survey, conducted on behalf of IBM by Harris Poll, of 2,000 companies across the U.S., U.K., India, Japan and Germany.	Read more

Vir: lastno delo.

V primeru, da vrednost spremenljivke kazalnika *selected_indicator* ni enaka *None*, potem je enaka vrednosti, ki ustreza posameznim kazalnikom. Vsak kazalnik v spustnem meniju ima svoj unikatni identifikator (*id*). Za vsak indikator je napisan pogojni stavek *if* in v primeru, da se izbira ujema z znakovnim nizom v pogojnem stavku, se začne izvajati, kar je znotraj pogojnega stavka. Za vsak kazalnik sem spisal funkcijo, ki kot argument prejme zgodovinske podatke tiste delnice, ki se trenutno izvaja v *for* zanki. Funkcije kazalnikov, ki sem jih spisal, pa so: *calculate_keltner_channels, calculate_moving_average, calculate_acceleration_bands.*

Pospeševalni pasovi so tehnični kazalnik, ki ga je leta 2002 razvil Price Headley. Merijo nihajnost v določenem obdobju. Pasovi so narisani okoli enostavnega drsečega povprečja, ki služi kot sredinska črta, zgornji in spodnji pas pa sta enako oddaljena od te črte, podobno kot pri Bollingerjevih pasovih. Pospeševalni pasovi želijo prepoznati trenutek, ko se vrednostni papir začne premikati v določeni smeri, vendar še ni dosegel pomembnih cenovnih premikov (Buckley, 2024).

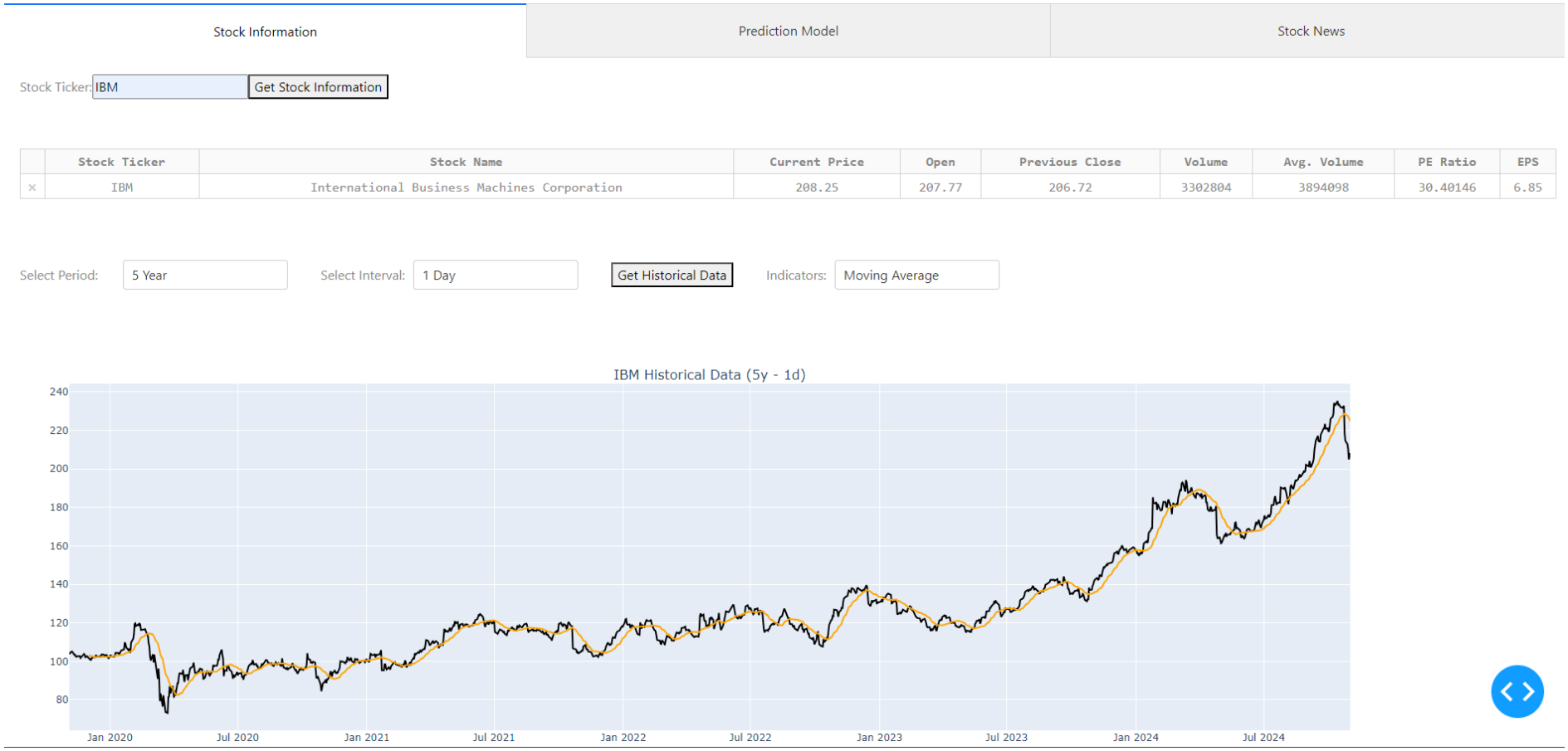
Keltnerjevi kanali so tehnični kazalnik, ki temelji na nihajnosti in se uporablja za analizo gibanja cen. Postavljeni so na obeh straneh cene delnice in pomagajo določiti smer trenda (Mitchell, 2024).

Vrednost omenjenih funkcij se shrani v spremenljivko, ki se nato doda spremenljivki *fig.* Nato se naslednji spremenljivki *fig* popravi višina s funkcijo *update_layout* tako, da so vsi grafi iste višine in se spremenljivka *fig* pošlje kot izhod celotne funkcije *apply_indicator*. Izhodna argumenta te funkcije sta torej spremenljivka *fig* in spremenljivka *fig*, ki je prevedena v slovar z ukazom *to_dict*. Prva se pošlje kot *Output* na element *dcc.Graph*, medtem ko se druga pošlje kot *Output* na element *dcc.Store*, ki nato shrani graf z izbranim kazalnikom. Na sliki 9 je prikazana zaslonska slika prvega zavihka zgodovinskih podatkov delnice in izbranega kazalnika. Vzrok, da je besedilo na zaslonski sliki v angleščini, je to, da je spletna aplikacija narejena v angleščini.

4.6 Algoritmi za napovedovanje

Bistvo drugega zavihka *Prediction Model* so algoritmi za napovedovanje. Uporabnik izbira med več algoritmi za napoved cene v prihodnosti. Prvo povratno funkcijo sem opisal že v poglavju *load_graph_from_store* uporabniški vmesnik. Druga povratna funkcija se imenuje *train_model* in sestavlja celotno ostalo kodo v skripti *prediction_model.py* z izjemo nekaterih funkcij, ki se kličejo v tej povratni funkciji.

Slika 9: Zaslonska slika prvega zavihka



Vir: lastno delo.

Povratna funkcija *train_model* ima dodan parameter *prevent_initial_call* na stanje *True*. Ta funkcija se kliče le ob proženi spremembi stanja *Input*, kar sem dosegel z razvojem pogojnega stavka *if*. Povratna funkcija ima en vhodni argument *Input* in štiri vhodne argumente *State*. *Input* je gumb *Train Model*, medtem ko so vhodni argumenti *State*:

- *dcc.Input* – ob pritisku na gumb *Train Model* povratna funkcija *train_model* vzame vrednost, ki je zapisana v vnosnem polju. Polje pričakuje vrednost delniškega znaka.
- *dcc.Dropdown* – izbira se lahko med štirimi modeli v spustnem modelu. Te možnosti so linearna regresija, ridge regresija, lasso regresija in LSTM-model.
- *dcc.RadioButton* – s klikom se izbere vrednost količine zgodovinskih podatkov. Na voljo so vrednosti: eno leto, dve leti, pet let in deset let.
- *dcc.RadioButton* – s klikom se izbira, na kateri dan v prihodnosti uporabnik želi napovedati ceno delnice. Na voljo so vrednosti: 30 dni, 45 dni in 60 dni.

Ko je povratna funkcija sprožena, se najprej začne pridobivanje podatkov. S funkcijo, ki sem jo spisal, *get_stock_data*, se najprej v spremenljivko *df* shrani zgodovinske podatke delnice. Tej funkciji se kot argument pošlje stanje delniškega znaka v vnosnem polju in stanje periode zgodovinskih podatkov. Iz pridobljenih podatkov, shranjenih v spremenljivki *df*, se iz indeksa pridobijo vsi datumi, za katere obstaja vrednost delnice. Nato se s pogojnim stavkom in funkcijo *is_datetime64_any_dtype* prepričam, da je indeks datumski podatkovni tip. V primeru, da ni, ga koda spremeni vanj. Koda se nadaljuje s pridobivanjem ekonomskih podatkov. To so stopnja inflacije, stopnja brezposelnosti in obrestna mera.

Inflacija označuje splošno rast cen blaga in storitev v gospodarstvu, kar zmanjša kupno moč valute. Za razliko od sprememb cen posameznih artiklov inflacija predstavlja širši dvig cen, kar pomeni, da se lahko z eno enoto valute kupi manj dobrin in storitev kot prej. S tem se vrednost valute znižuje skozi čas (European Central Bank, 2023).

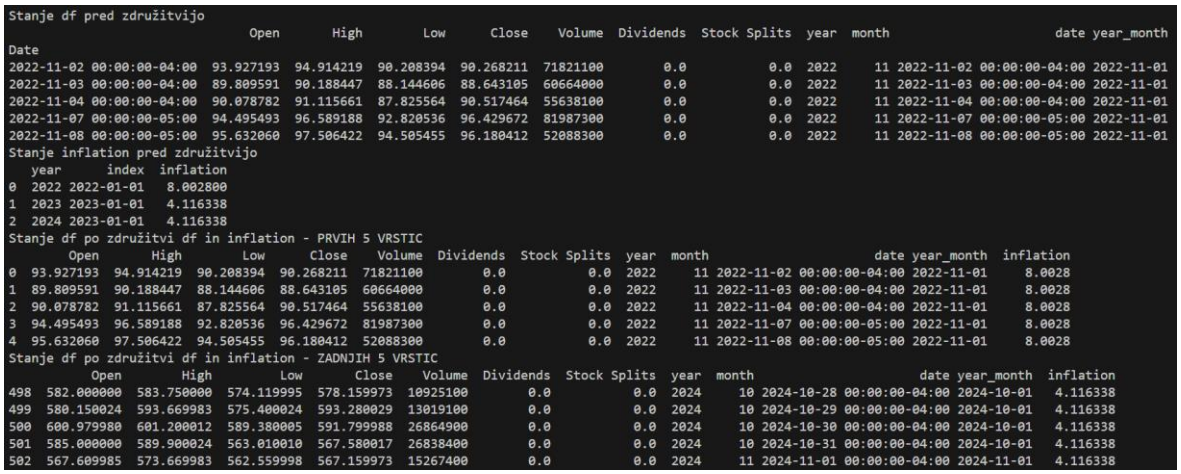
Stopnja brezposelnosti je ključen pokazatelj za razumevanje razmer na trgu dela in gospodarskega stanja. Brezposelnost nastopi, ko je nekdo pripravljen in sposoben delati, a nima plačane zaposlitve. Stopnja brezposelnosti je odstotek ljudi v delovni sili, ki so brezposelni, in se izračuna na podlagi podatkov o zaposlenosti in brezposelnosti (Reserve Bank of Australia, 2023).

Obrestna mera na medbančnem trgu v ZDA je obrestna mera, po kateri depozitne institucije med seboj trgujejo s presežnimi sredstvi, ki jih hranijo pri federalnih rezervnih bankah. Ko ima neka institucija presežek sredstev na svojem rezervnem računu, jih lahko posodi drugi banki, ki potrebuje večja sredstva. Gre za osrednjo obrestno mero na finančnem trgu ZDA in vpliva na druge obrestne mere, ki jih banke zaračunavajo svojim komitentom z višjimi bonitetnimi ocenami, ter posredno vpliva na dolgoročne obrestne mere, kot so hipotekarni krediti, posojila in varčevalni računi, ki so ključni za premoženje in zaupanje potrošnikov (Board of Governors of the Federal Reserve System, 2024).

Te tri ekonomske kazalnike sem pridobil s spletne strani FRED. Pridobivanje teh podatkov sem opisal v poglavju 4.2 Pridobivanje podatkov. Pridobivanje podatkov sem vključil v prilagojeno funkcijo, ki sem jo spisal – *fetch_economic_data*. Ta funkcija kot argument *stock_dates* sprejme indeks spremenljivke *df*, ki so vsi zgodovinski datumi delnice, ki sem jo vnesel. Iz spremenljivke *stock_dates* sem s funkcijama *min*() in *max*() pridobil začetne in končne datume periode delnice. Nato sem za vse tri ekonomske kazalnike pridobil podatke v tem časovnem obdobju tako, da sem funkciji *fred*, ki sem jo uvozil, dodal metodo *get_series* in z argumenti začetnega in končnega datuma ter *id* ekonomskega kazalnika, ki sem ga pridobil na FRED spletni strani. Vsakega od kazalnikov sem shranil v svojo spremenljivko in ga spremenil v podatkovni tip *DataFrame* ter mu kot indeks določil datum. Te tri ekonomske kazalnike sem poslal kot izhodne argumente funkcije *fetch_economic_data*. Nato sem za nadaljnjo obdelavo podatkov iz indeksa spremenljivke *df* pridobil leto in mesec in jih shranil kot nova dva stolpca *year* in *month* te spremenljivke. Pogostost pridobivanja podatkov se razlikuje med različnimi kazalniki. Za inflacijo sem lahko pridobil le letno vrednost, zato sem v nadaljevanju vsem datumom istega leta dodelil tisto vrednost inflacije, ki sem jo na letni ravni pridobil s spletne strani FRED.

Pridobivanje vrednosti inflacije vsem datumom, za katere so pridobljeni podatki delnice, sem začel s tem, da sem iz spremenljivke *df* pridobil vsa unikatna leta, za katera obstajajo podatki o ceni delnice. Ta leta se razlikujejo glede na to, kaj uporabnik izbere v elementu *dcc.RadioButtons* periode zgodovinskih podatkov. Iz indeksa spremenljivke *inflation* sem prav tako dodal stolpec spremenljivki *inflation*, imenovan *year*. Nato sem ustvaril novo spremenljivko *full_year* kot DataFrame, ki ima kot stolpec *year*, vsa unikatna leta stolpca *year* v spremenljivki *df*. Nato sem spremenljivki *inflation* odstranil duplikate, nastavil stolpec *year* kot indeks in zamenjal ta indeks s stolpcem *year* spremenljivke *full_years* in opravil zapolnjevanje vrednosti vnaprej. To pomeni, da ker za leto 2024 še ni podatka glede inflacije, sem mu priredil vrednost 2023 s funkcijo *ffill*. Nato sem stolpca spremenljivke *inflation – year* in *inflation* združil s spremenljivko *df* s funkcijo *merge* na skupnem ključu *year* tako, da sem obdržal vse podatke spremenljivke *df*. Tako sem imel za vse datume določenega leta tisto vrednost, ki jo je FRED določil na letni ravni v tistem letu. Za trenutno leto pa sem priredil podatke prejšnjega leta. Ta način bo deloval tudi v prihodnosti in bo trenutnemu letu v primeru, da ni podatka inflacije, priredil vrednost prejšnjega leta. Na sliki 10 je prikazanih prvih pet vrstic za tri različne *DataFrame* spremenljivke in zadnjih pet vrstic ene spremenljivke *DataFrame*. Prva je spremenljivka *df* pred združitvijo z *inflation* spremenljivko, druga je *inflation* spremenljivka pred združitvijo s spremenljivko *df*. Tretja je združena spremenljivka *df* s prikazom zadnjih petih vrstic.

Slika 10: Združitev spremenljivk df in inflation



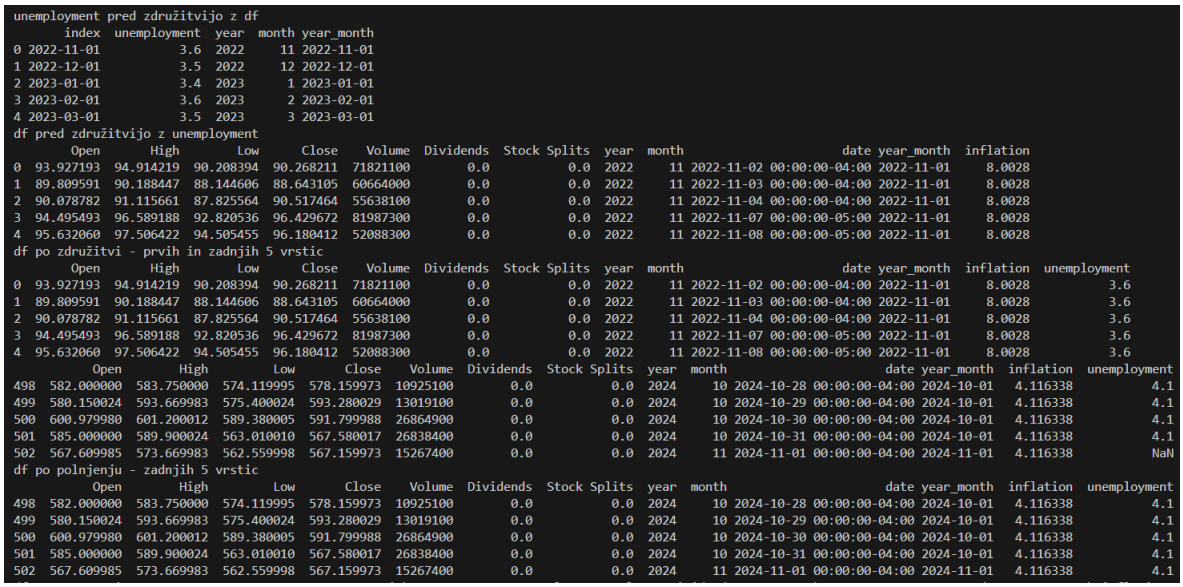
```
Stanje df pred združitvijo
                                Open       High        Low      Close    Volume  Dividends  Stock Splits  year  month                       date year_month
Date
2022-11-02 00:00:00-04:00  93.927193  94.914219  90.208394  90.268211  71821100        0.0           0.0  2022     11 2022-11-02 00:00:00-04:00 2022-11-01
2022-11-03 00:00:00-04:00  89.809591  90.188447  88.144606  88.643105  60664000        0.0           0.0  2022     11 2022-11-03 00:00:00-04:00 2022-11-01
2022-11-04 00:00:00-04:00  90.078782  91.115661  87.825564  90.517464  55638100        0.0           0.0  2022     11 2022-11-04 00:00:00-04:00 2022-11-01
2022-11-07 00:00:00-05:00  94.495493  96.589188  92.820536  96.429672  81987300        0.0           0.0  2022     11 2022-11-07 00:00:00-05:00 2022-11-01
2022-11-08 00:00:00-05:00  95.632060  97.506422  94.505455  96.180412  52088300        0.0           0.0  2022     11 2022-11-08 00:00:00-05:00 2022-11-01
Stanje inflation pred združitvijo
   year      index  inflation
0  2022 2022-01-01   8.002800
1  2023 2023-01-01   4.116338
2  2024 2023-01-01   4.116338
Stanje df po združitvi df in inflation - PRVIH 5 VRSTIC
       Open       High        Low      Close    Volume  Dividends  Stock Splits  year  month                       date year_month  inflation
0  93.927193  94.914219  90.208394  90.268211  71821100        0.0           0.0  2022     11 2022-11-02 00:00:00-04:00 2022-11-01     8.0028
1  89.809591  90.188447  88.144606  88.643105  60664000        0.0           0.0  2022     11 2022-11-03 00:00:00-04:00 2022-11-01     8.0028
2  90.078782  91.115661  87.825564  90.517464  55638100        0.0           0.0  2022     11 2022-11-04 00:00:00-04:00 2022-11-01     8.0028
3  94.495493  96.589188  92.820536  96.429672  81987300        0.0           0.0  2022     11 2022-11-07 00:00:00-05:00 2022-11-01     8.0028
4  95.632060  97.506422  94.505455  96.180412  52088300        0.0           0.0  2022     11 2022-11-08 00:00:00-05:00 2022-11-01     8.0028
Stanje df po združitvi df in inflation - ZADNJIH 5 VRSTIC
          Open        High         Low       Close    Volume  Dividends  Stock Splits  year  month                       date year_month  inflation
498  582.000000  583.750000  574.119995  578.159973  10925100        0.0           0.0  2024     10 2024-10-28 00:00:00-04:00 2024-10-01   4.116338
499  580.150024  593.669983  575.400024  593.280029  13019100        0.0           0.0  2024     10 2024-10-29 00:00:00-04:00 2024-10-01   4.116338
500  600.979980  601.200012  589.380005  591.799988  26864900        0.0           0.0  2024     10 2024-10-30 00:00:00-04:00 2024-10-01   4.116338
501  585.000000  589.900024  563.010010  567.580017  26838400        0.0           0.0  2024     10 2024-10-31 00:00:00-04:00 2024-10-01   4.116338
502  567.609985  573.669983  562.559998  567.159973  15267400        0.0           0.0  2024     11 2024-11-01 00:00:00-04:00 2024-11-01   4.116338
```

Vir: lastno delo.

Vidi se, da se podatki inflacije pridobijo za vse datume periode zgodovinskih podatkov izbrane delnice. Te podatke sem v terminalu prikazal s funkcijama *head* in *tail*. *Head* prikaže prvih pet vrstic spremenljivke, *tail* pa prikaže zadnjih pet vrstic spremenljivke.

Na podoben način sem se lotil tudi pridobivanja podatkov brezposelnosti in obrestne mere. Pri teh dveh ekonomskih kazalnikih so podatki na mesečni ravni. Da sem omogočil podobno združitev kot pri inflaciji, sem tu spremenljivki *unemployment* ustvaril stolpec *year_month.* Spremenljivki *df* sem to opravil že prej tako, da sem združil stolpca *year* in *month* v skupni stolpec *year_month*, to je prikazano tudi na prejšnji sliki. Ta stolpca sta podana v znakovni obliki, kjer prvi štiri znaki predstavljajo leto, sledita dva znaka za mesec, dva znaka za dan pa sta določena na vrednost 01. Na sliki 11 je prikazanih prvih pet vrstic spremenljivke *unemployment* pred združitvijo s spremenljivko *df*, prvih pet vrstic spremenljivke *df* pred združitvijo s spremenljivko *unemployment*. Nato je prikazanih prvih in zadnjih pet vrstic po združitvi spremenljivke *df*. Na koncu je prikazanih še zadnjih pet vrstic spremenljivke *df*, ki se zapolnijo manjkajoče vrednosti s podatki, ki so na voljo s funkcijo *ffill*.

Slika 11: Združitev spremenljivk df in unemployment



```
unemployment pred združitvijo z df
       index  unemployment  year  month year_month
0 2022-11-01           3.6  2022     11 2022-11-01
1 2022-12-01           3.5  2022     12 2022-12-01
2 2023-01-01           3.4  2023      1 2023-01-01
3 2023-02-01           3.6  2023      2 2023-02-01
4 2023-03-01           3.5  2023      3 2023-03-01
df pred združitvijo z unemployment
        Open       High        Low      Close    Volume  Dividends  Stock Splits  year  month                       date year_month  inflation
0  93.927193  94.914219  90.208394  90.268211  71821100        0.0           0.0  2022     11 2022-11-02 00:00:00-04:00 2022-11-01     8.0028
1  89.809591  90.188447  88.144606  88.643105  60664000        0.0           0.0  2022     11 2022-11-03 00:00:00-04:00 2022-11-01     8.0028
2  90.078782  91.115661  87.825564  90.517464  55638100        0.0           0.0  2022     11 2022-11-04 00:00:00-04:00 2022-11-01     8.0028
3  94.495493  96.589188  92.820536  96.429672  81987300        0.0           0.0  2022     11 2022-11-07 00:00:00-05:00 2022-11-01     8.0028
4  95.632060  97.506422  94.505455  96.180412  52088300        0.0           0.0  2022     11 2022-11-08 00:00:00-05:00 2022-11-01     8.0028
df po združitvi - prvih in zadnjih 5 vrstic
        Open       High        Low      Close    Volume  Dividends  Stock Splits  year  month                       date year_month  inflation  unemployment
0  93.927193  94.914219  90.208394  90.268211  71821100        0.0           0.0  2022     11 2022-11-02 00:00:00-04:00 2022-11-01     8.0028           3.6
1  89.809591  90.188447  88.144606  88.643105  60664000        0.0           0.0  2022     11 2022-11-03 00:00:00-04:00 2022-11-01     8.0028           3.6
2  90.078782  91.115661  87.825564  90.517464  55638100        0.0           0.0  2022     11 2022-11-04 00:00:00-04:00 2022-11-01     8.0028           3.6
3  94.495493  96.589188  92.820536  96.429672  81987300        0.0           0.0  2022     11 2022-11-07 00:00:00-05:00 2022-11-01     8.0028           3.6
4  95.632060  97.506422  94.505455  96.180412  52088300        0.0           0.0  2022     11 2022-11-08 00:00:00-05:00 2022-11-01     8.0028           3.6
          Open        High         Low       Close    Volume  Dividends  Stock Splits  year  month                       date year_month  inflation  unemployment
498  582.000000  583.750000  574.119995  578.159973  10925100        0.0           0.0  2024     10 2024-10-28 00:00:00-04:00 2024-10-01   4.116338           4.1
499  580.150024  593.669983  575.400024  593.280029  13019100        0.0           0.0  2024     10 2024-10-29 00:00:00-04:00 2024-10-01   4.116338           4.1
500  600.979980  601.200012  589.380005  591.799988  26864900        0.0           0.0  2024     10 2024-10-30 00:00:00-04:00 2024-10-01   4.116338           4.1
501  585.000000  589.900024  563.010010  567.580017  26838400        0.0           0.0  2024     10 2024-10-31 00:00:00-04:00 2024-10-01   4.116338           4.1
502  567.609985  573.669983  562.559998  567.159973  15267400        0.0           0.0  2024     11 2024-11-01 00:00:00-04:00 2024-11-01   4.116338           NaN
df po polnjenju - zadnjih 5 vrstic
          Open        High         Low       Close    Volume  Dividends  Stock Splits  year  month                       date year_month  inflation  unemployment
498  582.000000  583.750000  574.119995  578.159973  10925100        0.0           0.0  2024     10 2024-10-28 00:00:00-04:00 2024-10-01   4.116338           4.1
499  580.150024  593.669983  575.400024  593.280029  13019100        0.0           0.0  2024     10 2024-10-29 00:00:00-04:00 2024-10-01   4.116338           4.1
500  600.979980  601.200012  589.380005  591.799988  26864900        0.0           0.0  2024     10 2024-10-30 00:00:00-04:00 2024-10-01   4.116338           4.1
501  585.000000  589.900024  563.010010  567.580017  26838400        0.0           0.0  2024     10 2024-10-31 00:00:00-04:00 2024-10-01   4.116338           4.1
502  567.609985  573.669983  562.559998  567.159973  15267400        0.0           0.0  2024     11 2024-11-01 00:00:00-04:00 2024-11-01   4.116338           4.1
```

Vir: lastno delo.

Popolnoma enako kot za združitev spremenljivk *df* in *unemployment* sem opravil za združitev spremenljivk *df* in *interest_rate*. Po vsaki združitvi dveh podatkovnih tipov *DataFrame* je prišlo do ponastavitve indeksa spremenljivke *df*. Ker pa sem v indeksu potreboval datum delnice, sem pred pridobivanjem ekonomskih kazalnikov indeks spremenljivke *df* shranil v stolpec *date*. Ko sem imel vse ekonomske kazalnike shranjene v spremenljivki *df*, sem nastavil stolpec *date* ponovno kot indeks te spremenljivke.

Nato sem napisal stavek *try* in začel z deklaracijo spremenljivk *X_train*, *X_test*, *y_train* in *y_test*, ki so bili enaki izhodnim argumentom funkcije *prepare_data*, ki sem jo napisal. Ta funkcija je kot vhodni argument sprejela spremenljivko *df*.

V tej funkciji sem začel z izračunom tehničnih ekonomskih kazalnikov cen delnice. V tabeli 3 so prikazani stolpci spremenljivke *df*, kako sem prišel do stolpcev in ime ekonomskega kazalnika v stolpcu.

Tabela 3: Tehnični kazalniki delnice

Ime stolpca	Način izračuna	Ekonomski kazalnik
SMA_20	Z uporabo funkcij rolling in mean za perido 20 dni	Enostavno drseče povprečje
SMA_50	Z uporabo funkcij rolling in mean za perido 50 dni	Enostavno drseče povprečje
EMA	Z razvito funkcijo calculate_ema	Eksponentno drseče povprečje
MACD	Z razvito funkcijo calculate_macd	Konvergenca/divergenca drsečih povprečij
OBV	Z razvito funkcijo calculate_obv	Ravnovesni volumen

se nadaljuje

Ime stolpca	Način izračuna	Ekonomski kazalnik
RSI	Z razvito funkcijo calculate_rsi	Indeks relativne moči
Bollinger_Upper	Z razvito funkcijo calculate bollinger_bands	Zgornji bollingerjev pas
Bollinger_Lower	Z razvito funkcijo calculate bollinger_bands	Spodnji bollingerjev pas
Momentum	Z razvito funkcijo calculate_momentum	Zagon
Stochastic	Z razvito funkcijo calculate_stochastic_oscillator	Stohastični oscilator
ATR	Z razvito funkcijo calculate_atr	Povprečni pravi razpon
Volume_Oscillator	Z razvito funkcijo calculate_volume_oscillator	Oscilator volumna

Vir: lastno delo.

Enostavno drseče povprečje je časovna serija, ustvarjena z izračunom povprečja več zaporednih vrednosti druge časovne serija. Je posebna vrsta glajenja, ki zmanjšuje naključna nihanja in poudarja osnovni trend. Eksponentno drseče povprečje je variacija drsečega povprečja, kjer uteži eksponentno padajo za starejše podatke. Gre za enostavno eksponentno glajenje in se pogosto uporablja v napovedovanju časovnih serij (Hyndman, 2009).

Stohastični oscilator je tehnični kazalnik, ki meri trenutne zaključne cene delnice glede na njihov cenovni razpon v določenem časovnem obdobju**.** Konvergenca/divergenca drsečih povprečij je tehnični kazalnik, ki meri razliko med dvema eksponentnima drsečima povprečjema (12-dnevnim in 26-dnevnim) ter vključuje signalno linijo (devetdnevno eksponentno povprečje), da identificira trende, zagon in morebitne preobrate (Ramlall, 2016).

Indeks relativne moči je tehnični kazalnik, ki meri hitrost in velikost nedavnih sprememb cene, da prepozna, ali je delnica prekupljena ali preprodana. Je momentni oscilator, katerega vrednosti se gibljejo med 0 in 100. Najpogosteje se uporablja za identifikacijo ekstremnih vrednosti in opozarjanje na možne preobrate trenda (Ramlall, 2016).

Bollingerjevi pasovi so tehnični kazalnik, ki meri nihajnost cen delnice in pomaga določiti, ali je cena visoka ali nizka glede na nedavno gibanje. Sestavljeni so iz srednjega pasu, ki je 20-dnevno drseče povprečje, ter zgornjega in spodnjega pasu, ki sta odmaknjena za dve standardni deviaciji od srednjega pasu. Ko je cena blizu zgornjega pasu, je delnica lahko preveč kupljena, ko je blizu spodnjega pasu, pa je lahko preprodana (Ramlall, 2016).

Tehnični kazalnik zagon sem razvil tako, da sem od zaključne cene odštel zaključno ceno pred desetimi dnevi.

Ravnovesni volumen je tehnični kazalnik, ki sledi spremembam volumna za napovedovanje gibanja cen. Ravnovesni volumen se izračuna tako, da se volumen doda prejšnji vrednosti ravnovesnega volumna, če je zaključna cena višja od prejšnje, oziroma odšteje, če je

zaključna cena nižja. Ključna logika ravnovesnega volumna je, da volumen pogosto prehiteva spremembe cen, kar omogoča identifikacijo trendov (Biberdorf, 2024a).

Povprečni pravi razpon je tehnični kazalnik, ki meri nihajnost trga, ne da bi upošteval smer gibanja cen. Razvil ga je J. Welles Wilder Jr. in se izračuna kot povprečje največjih razlik cene za določeno obdobje (Biberdorf, 2024b).

Večino funkcij za izračun ekonomskih tehničnih kazalnikov sem moral razviti sam. Enačbe za te izračune niso zahtevne. Pri teh izračunih je zanimivo, da sem klical funkcijo znotraj funkcije. To pomeni, da sem ob pritisku na gumb *Train Model* klical povratno funkcijo *train_model*. Ta funkcija je znotraj sebe klicala funkcijo *prepare_data*, ki je klicala funkcijo *calculate_macd*, ki je klicala funkcijo *calculate_ema*. Tudi funkcija *calculate_ema* je znotraj sebe klicala funkcijo *ewm*.

Po tem, ko sem izračunal vse tehnične kazalnike, sem spremenljivki *df* odstranil vse vrstice, ki imajo prazne vrednosti. V tem primeru sem dobil prazne vrednosti pri nekaterih tehničnih kazalnikih, saj za te nisem mogel izračunati vrednosti, ker je vrednost odvisna od preteklih kazalnikov. Nato sem kot *X* spremenljivko shranil vse stolpce, omenjene v zgornji tabeli, in tri ostale stolpce: *inflation, interest_rate* in *unemployment*. Kot *y* spremenljivko sem določil končno ceno delnice naslednji dan. Nato sem s funkcijo *iloc* določil, da imata spremenljivki enako število vrstic.

Izhodni argumenti funkcije *prepare_data* so zgoraj omenjeni *X_train*, *X_test*, *y_train* in *y_test*. Te sem pridobil tako, da sem iz knjižnice *scikit-learn* uvozil funkcijo *train_test_split*, ki kot argumente sprejme spremenljivko *X*, spremenljivko *y*, *test_size* in *shuffle*. Velikost *test_size* sem nastavil na 0.2, kar predstavlja 20 %. Stanje spremenljivke *shuffle* sem nastavil na *False*. Kar je vrnila funkcija *train_test_split*, sem uporabil tudi kot izhod funkcije *prepare_data*.

Nato sem uporabil pogojni stavek *if*, ki je preveril, če je izbrani model v spustnem meniju enak LSTM-modelu. Z uporabo funkcije MinMaxScaler sem normaliziral vhodne podatke. Normalizacija omogoča, da LSTM-model bolj učinkovito obravnava podatke, saj je občutljiv na vrednostne skale. Vhodne podatke (*X_train* in *X_test*) in ciljne spremenljivke (*y_train* in *y_test*) normalizira ločeno, saj se njihova porazdelitev razlikuje. LSTM zahteva podatke v obliki zaporedij, zato jih je treba pretvoriti v segmente dolžine spremenljivke *time_steps*. Vsak segment (*X_train_lstm*) predstavlja vhodne podatke za model, zadnji element segmenta (*y_train_lstm*) pa je ciljna vrednost, ki jo model napoveduje. Na podoben način se pripravijo tudi testni podatki za oceno modela. Model ima dva sloja LSTM, vsakega z enakim številom nevronov. Pri testiranju sem spreminjal število nevronov med 64, 128 in 256. Prvi sloj vrne sekvence, kar drugemu sloju omogoča dostop do celotnega časovnega zaporedja. Drugi sloj vrne samo končni rezultat. Med vsakim slojem je Dropout sloj (0.4), ki preprečuje preveliko naučenost tako, da naključno izključi 40 % nevronov. Vrednost Dropout sem med testiranjem spreminjal. Model se zaključi z dvema gostima (Dense)

slojema: prvi ima 64 nevronov in uporablja aktivacijsko funkcijo *ReLU*. Zadnji sloj ima en izhod, ki predstavlja napovedano ceno. Model je optimiziran z algoritmom *Adam*, ki omogoča hitrejše in stabilnejše učenje, in uporablja funkcijo izgube *mean_squared_error*, saj je cilj minimizirati napake v napovedanih vrednostih. Model se usposobi na trening podatkih z določenim številom iteracij in velikostjo *batch_size*. Iteracije določajo, kolikokrat model obdeluje celoten nabor podatkov. Batch_size določa, koliko primerov model obdela hkrati. Postopek usposabljanja je prilagojen, da se model čim boljše nauči prepoznati vzorce iz zgodovinskih podatkov. Po treningu model napove vrednosti za testni nabor podatkov (*X_test_lstm*). Napovedane vrednosti so sprva še normalizirane, zato se jih razveljavi, da se jih lahko primerja z dejanskimi vrednostmi. Model omogoča tudi napovedovanje prihodnjih dni na podlagi zadnjih znanih podatkov. Število *time_steps* normaliziranih podatkov se uporabi kot začetni vhod. Model iterativno napoveduje vrednosti za prihodnje dni. Prihodnje napovedi se stabilizirajo z uporabo eksponentnega glajenja, kjer parameter *alpha* določa, koliko teže se pripiše trenutni napovedi v primerjavi s prejšnjimi. Nato se model oceni z metrikami MAE, RMSE, MAPE in R^2. Testiranje modela LSTM opišem tudi v naslednjem poglavju.

Koda se nadaljuje z uporabo pogojnega stavka *else*, ki preveri, če je izbrani model za napovedovanje drugačen od LSTM-modela. V tem delu ima uporabnik tri možnosti: *linear, ridge* in *lasso*, ki predstavljajo različne regresijske metode za napovedovanje. Če je eden od teh modelov izbran, se koda nadaljuje s pripravo modela, njegovim usposabljanjem in napovedovanjem.

Če je izbran linearni regresijski model (*linear*), se požene razred *LinearRegression* iz knjižnice *scikit-learn*. Nato se model usposobi z metodo *fit*, kjer se podajo vhodne spremenljivke *X_train* in ciljne vrednosti *y_train*. Linearna regresija preprosto ustvari linearno funkcijo, ki poskuša najbolje približati podatke.

Če je izbran model *ridge*, se uporabi funkcija *fine_tune_model*, ki se prilagodi za ridge regresijo.

Ridge regresija, znana tudi kot regularizacija L2, doda kazen na vsoto kvadratov regresijskih koeficientov. Cilj te metode je zmanjšati vpliv kolinearnosti med neodvisnimi spremenljivkami in preprečiti prevelike vrednosti koeficientov, kar bi lahko vodilo do prevelike naučenosti modela. Parameter α določa moč kazni, višje vrednosti α povzročijo večje zmanjšanje koeficientov. Ridge regresija je uporabna, kadar so vse spremenljivke pomembne, saj zmanjša velikost koeficientov, vendar jih ne nastavi na nič. Tako je primerna za modele z veliko povezanimi spremenljivkami, kjer je pomembno ohraniti vse značilnosti v analizi (Weiming, 2019).

Če je izbran model *lasso*, se uporabi funkcija *fine_tune_model* za prilagajanje lasso regresijskega modela.

Lasso regresija, znana tudi kot regularizacija L1, doda kazen na vsoto absolutnih vrednosti koeficientov. Ključna lastnost te metode je, da lahko nastavi nekatere regresijske koeficiente na nič, kar pomeni, da izloči manj pomembne spremenljivke iz modela. Parameter α tudi tukaj nadzira moč kazni, pri čemer višje vrednosti α povečajo število koeficientov, ki so nastavljeni na nič. Lasso regresija je še posebej uporabna pri modelih z veliko značilnostmi, kjer so nekatere nepomembne, saj omogoča avtomatsko izbiro najbolj pomembnih spremenljivk. Ta metoda je primerna za redke modele, kjer je le majhen del značilnosti ključen za napovedovanje (Weiming, 2019).

Po usposabljanju modela se izvede napoved na testnem nizu *X_test* z metodo *predict*, kar vrne napovedane vrednosti *y_pred*. S tem se oceni, kako natančen je model glede na dejanske vrednosti v *y_test*. Nato se izračunajo različne meritve za oceno uspešnosti modela: MAE, RMSE, MAPE in R^2.

Celoten graf in meritve uspešnosti modela se shranijo v spremenljivko *fig* in ta spremenljivka se shrani v elementu *dcc.Graph* in *dcc.Store*. To se naredi v primeru, da je izbrani model LSTM ali katerikoli drug enostavnejši model. Prvi element *dcc.Graph* prikazuje graf in meritve uspešnosti do osvežitve strani, po osvežitvi strani se graf in meritve naložijo iz elementa *dcc.Store*.

S tem sem zaključil opis kode v drugem zavihku in posledično opis celotne kode vseh zavihkov. Na sliki 12 je prikaz drugega zavihka z izbrano delnico podjetja IBM, linearnim modelom, zgodovinskimi podatki delnice za dve leti in izračun cene delnice podjetja čez 45 dni. Vzrok, da je besedilo na zaslonski sliki v angleščini, je to, da je spletna aplikacija narejena v angleščini.

Slika 12: Zaslonska slika drugega zavihka



Vir: lastno delo.

5 TESTIRANJE IN REZULTATI

Po tem, ko sem zaključil razvoj kode v vseh treh zavihkih, sem se lotil testiranja modelov drugega zavihka. Modelom sem preverjal veljavnost z ocenami uspešnosti modela. V tem poglavju najprej opišem različne načine preverjanje veljavnosti modela in nato interpretiram rezultate.

5.1 Preverjanje veljavnosti modelov

Pri napovedovanju cen delnic je ključnega pomena oceniti natančnost modela z uporabo ustreznih ocenjevalnih metrik. Štiri najpogostejše uporabljene metrike za regresijske modele so MAE, RMSE, MAPE in R^2.

- Povprečna absolutna napaka (angl. Mean absolute error, v nadaljevanju MAE)

MAE meri povprečno absolutno razliko med dejanskimi in napovedanimi vrednostmi. Ta metrika kaže povprečno velikost napak, pri čemer zanemarja smer teh napak. Manjša vrednost MAE pomeni boljšo natančnost modela. MAE se izračuna po naslednji enačbi (4):

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |y_i - y_i'|, \tag{4}$$

kjer je n število podatkov, y_i dejanska vrednost in y_i' napovedana vrednost za i-ti eksperiment (Mokhtari in drugi, 2021).

- Povprečni kvadratni odklon (angl. Root-Mean-Square Deviation, v nadaljevanju RMSE)

Koren srednje kvadratne napake predstavlja standardni odklon pri napovedovanju. Metrika pokaže, kako razpršeni so preostanki glede na regresijsko črto. RMSE je občutljiv na večje napake, saj jih zaradi kvadriranja kaznuje bolj kot manjše. Nižje vrednosti RMSE pomenijo boljšo natančnost napovedi. Enačba (5) za RMSE je:

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_i' - y_i)^2}, \tag{5}$$

kjer je n število podatkov, y_i dejanska vrednost in y_i' napovedana vrednost (Mokhtari in drugi, 2021).

- Povprečna absolutna odstotna napaka (angl. Mean Absolute Percentage Error, v nadaljevanju MAPE)

MAPE ocenjuje povprečno relativno napako modela v odstotkih. Pokaže, koliko odstotkov so napovedi v povprečju napačne glede na dejanske vrednosti. Je asimetrična metrika, saj višje kaznuje napake, ko so napovedane vrednosti večje od dejanskih, kot pa obratno.

Enačba (6) za MAPE je:

$$MAPE = \frac{1}{n}\sum_{i=1}^{n} \left| \frac{y_i - y_i'}{y_i} \right| \times 100, \quad (6)$$

kjer je *n* število podatkov, y_i dejanska vrednost in y_i' napovedana vrednost. Manjša vrednost MAPE pomeni boljšo natančnost napovedi (Mokhtari in drugi, 2021).

- Koeficient determinacije (R^2)

R^2, imenovan tudi koeficient determinacije, meri, koliko variabilnosti v ciljni spremenljivki pojasnjujejo neodvisne spremenljivke. Njegova vrednost se giblje med 0 in 1, kjer vrednost bližje 1 pomeni boljšo natančnost modela. Višji R^2 kaže na to, da je model dobro prilagojen podatkom. Če je $R^2 < 0{,}3$, se to šteje za šibek model; vrednosti med 0,3 in 0,7 pomenijo zmerno moč; vrednosti večje od 0,7 pa kažejo na močan model. Enačba (7) za R^2 je:

$$R^2 = 1 - \frac{\sum(y_i - y_i')^2}{\sum(y_i - y_i')^2}, \quad (7)$$

kjer je *n* število podatkov, y_i dejanska vrednost in y_i' napovedana vrednost (Mokhtari in drugi, 2021). Vrednost koeficienta determinacije je lahko tudi negativna v primeru, da podatki ne ustrezajo modelu.

Te metrike sem vpeljal v kodo za evalvacijo natančnosti modela, tako da sem uvozil funkcije *mean_absolute_error*, *root_mean_squared_error, calculate_mape* in *r2_score*. Metrike sem uporabil tako za napovedi, pridobljene z regresijskimi modeli (linear, ridge in lasso), kot tudi za kompleksnejši LSTM-model. Rezultate sem nato prikazal kot del vizualizacije na grafu, da bi uporabnikom omogočil boljšo interpretacijo uspešnosti modela.

5.2 Analiza rezultatov

V tem poglavju opišem rezultate testiranja natančnosti modela za napovedane cene delnice v drugem zavihku. Testiranje natančnosti modela sem razdelil na dva dela. V prvem delu testiranja med seboj primerjam enostavnejše regresijske modele (linear, ridge in lasso). V drugem delu testiranja testiram model LSTM in kako se rezultati tega testiranja razlikujejo glede na izbrane vhodne argumente.

Enostavnejše regresijske modele sem testiral na delnicah IBM, AAPL, GOOGL in AMZN. Zgodovina podatkov predstavlja vrednost, koliko zgodovinskih podatkov se je pridobilo za posamezno zaključno ceno delnice in ekonomske kazalnike. Dan napovedi pomeni, za kateri dan v prihodnosti od trenutnega datuma se napove cena delnice.

V tabeli 4 je prikazan izračun metrik natančnosti modela za delnico IBM z različnimi modeli in vhodnimi parametri.

Tabela 4: Izračun metrik natančnosti modela za delnico IBM

Model	Zgodovina podatkov, dan napovedi	MAE	RMSE	MAPE	R^2	Napovedana cena delnice
Linearen	1 leto, 30 dni	8,74	10,11	4,05 %	–0,16	223,31
	2 leti, 45 dni	2,44	3,43	1,18 %	0,95	209,20
	5 let, 45 dni	3,87	4,70	2,05 %	0,95	203,30
	5 let, 60 dni	3,79	4,62	2,01 %	0,95	196,59
	10 let, 45 dni	2,21	3,05	1,33 %	0,99	202,75
	10 let, 60 dni	0	3,05	1,33 %	0,99	202,13
Ridge	1 leto, 30 dni	5,28	6,83	2,47 %	0,52	208,95
	2 leti, 45 dni	2,41	3,41	1,17 %	0,95	205,84
	5 let, 45 dni	3,79	4,62	2,00 %	0,95	198,02
	5 let, 60 dni	3,79	4,62	2,01 %	0,95	196,79
	10 let, 45 dni	2,18	3,03	1,31 %	0,99	204,08
	10 let, 60 dni	2,18	3,03	1,31 %	0,99	203,77
Lasso	1 leto, 30 dni	7,22	8,51	3,35 %	0,21	225,60
	2 leti, 45 dni	2,41	3,42	1,17 %	0,95	199,48
	5 let, 45 dni	3,69	4,51	1,95 %	0,95	192,39
	5 let, 60 dni	3,69	4,51	1,95 %	0,95	189,29
	10 let, 45 dni	2,22	3,06	1,34 %	0,99	202,14
	10 let, 60 dni	2,22	3,06	1,34 %	0,99	201,44

Vir: lastno delo.

Trenutna cena delnice ob izvedbi testiranja: 15. 11. 2024 je bila vrednost 204,99 ameriškega dolarja. Tudi napovedana cena delnice je prikazana v ameriških dolarjih. IBM je delniški znak ameriškega podjetja International Business Machines Corporation.

V tabeli 5 je prikazan izračun metrik natančnosti modela za delnico AAPL z različnimi modeli in vhodnimi parametri.

Tabela 5: Izračun metrik natančnosti modela za delnico AAPL

Model	Zgodovina podatkov, dan napovedi	MAE	RMSE	MAPE	R^2	Napovedana cena delnice
Linearen	1 leto, 30 dni	8,73	9,58	3,84 %	–5,37	233,42
	2 leti, 45 dni	2,93	3,68	1,32 %	0,56	218,01
	5 let, 45 dni	2,62	3,48	1,31 %	0,98	222,21
	5 let, 60 dni	2,62	3,48	1,31 %	0,98	221,78
	10 let, 45 dni	2,36	3,11	1,29 %	0,99	226,50
	10 let, 60 dni	2,36	3,11	1,29 %	0,99	228,44
Ridge	1 leto, 30 dni	3,55	4,11	1,55 %	-0,17	219,23
	2 leti, 45 dni	2,59	3,44	1,16 %	0,65	222,62
	5 let, 45 dni	2,60	3,46	1,30 %	0,98	222,89
	5 let, 60 dni	2,60	3,46	1,30 %	0,98	222,71

se nadaljuje

Model	Zgodovina podatkov, dan napovedi	MAE	RMSE	MAPE	R^2	Napovedana cena delnice
	10 let, 45 dni	2,36	3,11	1,29 %	0,99	226,51
	10 let, 60 dni	2,36	3,11	1,29 %	0,99	228,45
Lasso	1 leto, 30 dni	3,61	4,17	1,58 %	-0,21	218,71
	2 leti, 45 dni	2,51	3,28	1,12 %	0,68	215,51
	5 let, 45 dni	2,60	3,46	1,30 %	0,98	215,06
	5 let, 60 dni	2,60	3,46	1,30 %	0,98	211,87
	10 let, 45 dni	2,33	3,07	1,28 %	0,99	227,10
	10 let, 60 dni	2,33	3,07	1,28 %	0,99	227,00

Vir: lastno delo.

Trenutna cena delnice ob izvedbi testiranja: 18. 11. 2024 je bila vrednost 225 ameriških dolarjev. Tudi napovedana cena delnice je prikazana v ameriških dolarjih. AAPL je delniški znak ameriškega podjetja Apple.

V tabeli 6 je prikazan izračun metrik natančnosti modela za delnico GOOGL z različnimi modeli in vhodnimi parametri

Tabela 6: Izračun metrik natančnosti modela za delnico GOOGL

Model	Zgodovina podatkov, dan napovedi	MAE	RMSE	MAPE	R^2	Napovedana cena delnice
Linearen	1 leto, 30 dni	6,90	7,37	4,10 %	-0,52	178,65
	2 leti, 45 dni	2,46	3,25	1,49 %	0,83	172,49
	5 let, 45 dni	2,15	2,97	1,36 %	0,96	177,23
	5 let, 60 dni	2,15	2,97	1,36	0,96	177,48
	10 let, 45 dni	2,19	2,90	1,64 %	0,99	175,47
	10 let, 60 dni	2,19	2,90	1,64 %	0,99	175,75
Ridge	1 leto, 30 dni	2,95	3,60	1,75 %	0,64	179,58
	2 leti, 45 dni	2,54	3,37	1,54 %	0,82	179,00
	5 let, 45 dni	2,15	2,96	1,36 %	0,96	177,06
	5 let, 60 dni	2,15	2,96	1,36 %	0,96	177,20
	10 let, 45 dni	2,19	2,89	1,64 %	0,99	174,84
	10 let, 60 dni	2,19	2,89	1,64 %	0,99	174,96
Lasso	1 leto, 30 dni	2,92	3,55	1,74 %	0,65	177,80
	2 leti, 45 dni	2,55	3,38	1,54 %	0,82	186,41
	5 let, 45 dni	2,17	2,97	1,37 %	0,96	182,52
	5 let, 60 dni	2,17	2,97	1,37 %	0,96	184,54
	10 let, 45 dni	2,18	2,88	1,63 %	0,99	176,74
	10 let, 60 dni	2,18	2,88	1,63 %	0,99	177,33

Vir: lastno delo.

Trenutna cena delnice ob izvedbi testiranja: 18. 11. 2024 je bila vrednost 175,30 ameriškega dolarja. Tudi napovedana cena delnice je prikazana v ameriških dolarjih. GOOGL je delniški znak ameriškega podjetja Alphabet Inc.

V tabeli 7 je prikazan izračun metrik natančnosti modela za delnico AMZN z različnimi modeli in vhodnimi parametri.

Tabela 7: Izračun metrik natančnosti modela za delnico AMZN

Model	Zgodovina podatkov, dan napovedi	MAE	RMSE	MAPE	R^2	Napovedana cena delnice
Linearen	1 leto, 30 dni	4,99	6,12	2,57 %	0,54	198,33
	2 leti, 45 dni	3,77	5,07	2,08 %	0,79	206,96
	5 let, 45 dni	3,12	4,12	1,73 %	0,92	205,89
	5 let, 60 dni	3,12	4,12	1,73 %	0,92	206,77
	10 let, 45 dni	2,81	3,67	1,92 %	0,99	205,55
	10 let, 60 dni	2,81	3,67	1,92 %	0,99	206,20
Ridge	1 leto, 30 dni	3,47	4,64	1,76 %	0,73	198,10
	2 leti, 45 dni	3,79	5,19	2,09 %	0,78	205,74
	5 let, 45 dni	3,02	4,01	1,68 %	0,92	204,95
	5 let, 60 dni	3,02	4,01	1,68 %	0,92	205,42
	10 let, 45 dni	2,68	3,54	1,84 %	0,99	204,84
	10 let, 60 dni	2,68	3,54	1,84 %	0,99	205,24
Lasso	1 leto, 30 dni	3,73	4,80	1,90 %	0,72	197,42
	2 leti, 45 dni	3,86	5,33	2,13 %	0,77	207,40
	5 let, 45 dni	3,09	4,09	1,72 %	0,92	216,41
	5 let, 60 dni	3,09	4,09	1,72 %	0,92	222,61
	10 let, 45 dni	2,38	3,23	1,66 %	0,99	209,89
	10 let, 60 dni	2,38	3,23	1,66 %	0,99	211,11

Vir: lastno delo.

Trenutna cena delnice ob izvedbi testiranja: 18. 11. 2024 je bila vrednost 201,70 ameriškega dolarja. Tudi napovedana cena delnice je prikazana v ameriških dolarjih. AMZN je delniški znak ameriškega podjetja Amazon.com, Inc.

Rezultat izračuna metrik natančnosti enostavnejših modelov sem izvedel na štirih delnicah ameriških podjetij. Vse štiri delnice so tehnološki giganti, ki so dosegli veliko rast v zadnjih dveh desetletjih. Za vsako delnico sem testiral vse tri modele s petimi različnimi vhodnimi parametri zgodovinskih podatkov in dneva napovedi.

Ob izbranih parametrih zgodovinskih podatkov enega leta in napovedi delnice trideset dni vnaprej sem zaradi premajhnega števila podatkov dobil slabe rezultate. Pri vseh štirih obravnavanih delnicah se je vrednost koeficienta determinacije močno izboljšala ob vpeljavi lasso ali ridge regresije namesto linearnega modela. Za te parametre sem samo pri lasso in ridge regresiji delnice podjetja Amazon dosegel sprejemljivo vrednost koeficienta determinacije. Ostale tri metrike natančnosti izračuna modela so se vse izboljšale z vpeljavo

ridge ali lasso modela v primerjavi z linearnim modelom. Ridge regresija se je obnesla najbolje pri kratkih napovedih, kjer se z uvedbo regularizacije L2 zmanjša prevelika naučenost, hkrati pa se ohranijo vse značilnosti, kar omogoča boljšo stabilnost in natančnost napovedi.

Z večanjem obsega zgodovinskih podatkov se je uspešnost vseh modelov znatno izboljšala. Pri krajših časovnih obdobjih (1–2 leti) sta ridge in lasso regresija pokazali boljšo stabilnost in natančnost kot linearni model, saj sta zmanjšali preveliko naučenost. Pri srednje dolgih obdobjih (5 let) so vsi modeli postali robustnejši, pri čemer je ridge regresija še vedno izstopala z najnižjimi napakami. Pri dolgoročnih podatkih (10 let) so vsi modeli dosegli visoko natančnost, pri čemer sta ridge in lasso regresija ostala nekoliko bolj stabilna in robustna v primerjavi z linearnim modelom. Pri testiranjih, kjer je koeficient determinacije negativen, pomeni da podatki ne ustrezajo modelu.

Testiral sem tudi LSTM-model, ki pa je kompleksnejši in težji za umerjanje. Pri tem modelu sem testiral podatke na delnici IBM, s podatki 5 let, dan napovedi 45 dni. Spreminjal sem vhodne parametre modela LSTM in opravil 10 testov. Cena delnice IBM na dan 23. 11. 2024 znaša 229,97 ameriškega dolarja.

V tabeli 8 so prikazani rezultati testiranja LSTM-modela in različni parametri, ki sem jih uporabil za izračun teh rezultatov.

Tabela 8: Rezultati testiranja LSTM-modela

test	time_ steps	units	Dropout	epochs	batch_size	alpha	MAE	RMSE	MAPE	R^2	Napoved
1	30	128	0,3	50	32	0,6	12,83	15,51	6,44 %	0,36	131,04
2	20	64	0,2	50	32	0,5	25,05	27,38	12,92 %	–0,89	133,39
3	50	256	0,3	30	64	0,7	6,97	8,83	3,51 %	0,81	194,75
4	10	128	0,5	100	16	0,6	5,94	7,72	3,17 %	0,86	175,10
5	70	128	0,3	50	32	0,3	21,95	25,22	10,97 %	–0,42	149,90
6	30	256	0,2	50	64	0,5	9,53	11,36	4,85 %	0,66	156,65
7	20	128	0,3	30	32	0,7	10,96	13,32	5,56 %	0,55	142,13
8	50	64	0,3	100	16	0,5	25,90	31,01	12,86 %	–1,37	160,42
9	30	128	0,2	50	32	0,7	8,80	10,98	4,45 %	0,68	167,34
10	30	256	0,5	50	32	0,6	12,97	15,46	6,56 %	0,37	135,30

Vir: lastno delo.

Na podlagi rezultatov v zgornji tabeli sta izstopala testa 3 in 4. Odločil sem se za dodatno testiranje še petih testov in prilagodil parametre glede na najboljše rezultate.

V tabeli 9 so prikazani rezultati testiranja LSTM-modela in različni parametri, ki sem jih določil naknadno po pridobljenih rezultatih v tabeli 8.

Tabela 9: Dodatni rezultati testiranja LSTM-modela

test	time_ steps	units	Dropout	epochs	batch_size	alpha	MAE	RMSE	MAPE	R^2	Napoved
1	10	256	0,5	100	16	0,6	26,07	28,63	13,99 %	– 0,95	203,08
2	30	128	0,4	50	32	0,5	11,30	13,93	5,69 %	0,49	128,58
3	50	256	0,2	70	16	0,7	10,30	12,74	5,16 %	0,60	109,84
4	30	256	0,3	60	32	0,6	7,09	9,12	3,63 %	0,78	149,10
5	20	128	0,4	100	16	0,5	7,42	8,72	4,02 %	0,81	188,37

Vir: lastno delo.

Z novimi petimi testi nisem izboljšal rezultatov metrik natančnosti modela. Po testiranju lahko določim, da je najboljši LSTM-model vseboval naslednje parametre: time_steps na vrednosti 10, units na vrednosti 128, Dropout na vrednosti 0,5, število iteracij na vrednosti 100, batch_size na vrednosti 16. Dejavnik glajenja sem v tem primeru nastavil na vrednost 0,6.

6 SKLEP

V tem poglavju opišem pregled izpolnjevanja ciljev, ki sem si jih zadal v dispoziciji in uvodu magistrskega dela. Poleg opisa izpolnitev ciljev opišem, kako bi lahko programsko rešitev nadgradili in kakšne so možnosti za nadaljnje delo.

Prvi cilj, ki sem ga izpolnil pri razvoju programske rešitve, sta bili integracija in analiza podatkov z več virov. Integriral sem podatke s spletnih strani Yahoo Finance, FRED in finnhub.io. Podatke s spletne strani Yahoo Finance sem dobil prek uvoza knjižnice *yfinance*, ki se posodablja sproti. Podatke o ekonomskih kazalnikih v državi, kot so brezposelnost, obrestna mera in inflacije, sem pridobil prek aplikacijskega programskega vmesnika s spletne strani, ki hrani ekonomske podatke in se imenuje FRED. Tu sem moral zaprositi za brezplačni ključ, ki sem ga nato uporabil v svoji kodi. Tudi tukaj sem moral uporabiti prilagojeno Python knjižnico *fredapi*, ki skupaj s ključem, ki ga pridobiš na spletni strani, omogoča dostop do podatkov. Tretji vir podatkov, kjer sem v programsko rešitev integriral novice o delnicah, sem pridobil iz aplikacijskega programskega vmesnika. V tem primeru na spletni strani *finnhub.io* pridobiš ključ, ki ga nato navedeš v povezavi, ki te pripelje do novic posamezne delnice. Na tej povezavi dinamično nastaviš izbrani delniški znak in se tako izogneš uporabi knjižnice kot v primeru *fredapi*. Uporabniku je v vsakem zavihku omogočena analiza, ki jo opravi sam.

Drugi cilj, ki sem ga izpolnil, je razvoj uporabniku prijaznega vmesnika in ureditev podatkov v enostavno obliko. Eden izmed načinov, kako sem omogočil uporabniku prijazen vmesnik, je, da sem omogočil shranjevanje podatkov v posameznem zavihku ob osvežitvi strani ali

izbiri drugega zavihka. To sem omogočil z uporabo parametra *persistence* na stanju *True* in *persistence_type* na stanju session na elementih, ki so to omogočali. Če element ni imel tega parametra, sem moral razviti posebno funkcijo, ki je iz elementa *dcc.Store* pridobila podatke ob osvežitvi strani. Uporabniški vmesnik sem prav tako poenostavil uporabniku z izbiro spustnih menijev in vnosnih polj, ki so omogočala enostaven izbor oziroma vnos. Preden sem podatke prikazal v končnih elementih aplikacije, sem moral pridobljene podatke integrirati med seboj, jih urediti in nato pretvoriti v obliko, primerno za prikaz. Pri pretvorbi sem si pomagal z že vgrajenimi Pythonovimi knjižnicami in uvoženo knjižnico *pandas*.

Tretji cilj, izračun ekonomskih kazalnikov, sem razvil v prvem in drugem zavihku. V prvem zavihku sem omogočil, da uporabnik po tem, ko pridobi zgodovinske podatke o ceni delnice in si prikaže graf poteka cene delnice skozi čas, lahko doda tudi ekonomske kazalnike, ki se izračunajo glede na ceno delnice. Nekateri ekonomski kazalniki, ki sem jih omogočil, so premikajoče povprečje, keltnerjevi kanali in pospeševalni pasovi. Po izbiri kazalnika v spustnem meniju in pritisku na gumb se s pomočjo pogojnih stavkov in razvitih funkcij izračuna posamezen kazalnik. Ekonomske kazalnike sem izračunal tudi v drugem zavihku. Tu sem na podlagi cene delnice izračunal ekonomske kazalnike, ki sem jih potem uporabil kot vhodno spremenljivko *X* v napovednem modelu.

Četrti cilj, izdelavo modela za napovedovanje cen delnic, sem razvil v drugem zavihku. Uporabnik lahko izbere med štirimi modeli: linearna regresija, lasso regresija, ridge regresija in LSTM-model. Z združitvijo in integracijo tržnih podatkov posamezne delnice in federalnih rezervnih ekonomskih kazalnikov sem razvil programsko kodo, ki omogoča napoved cene delnice. Uporabnik izbere vhodne parametre modela, ki so čas zgodovinskih podatkov, dan napovedi delnice v prihodnosti, delniški znak in vrsta modela. Bistvo tukaj je, da sem uporabniku omogočili izbiro parametrov in enostaven dostop do modela za napoved cen delnic.

V magistrskem delu sem izpolnil cilje, ki sem si jih zadal v dispoziciji. Uvozil in integriral sem podatke z več spletnih mest, ustvaril sem uporabniku prijazen vmesnik, omogočil sem izračun ekonomskih podatkov in izdelal model za napoved cen delnic. To programsko rešitev bi bilo mogoče nadgraditi na več načinov. Prvi način je izboljšanje LSTM-modela, saj gre za močno orodje za napoved cen delnic. To bi lahko naredil tako, da bi pripeljal nove vhodne spremenljivke v model. Pri konfiguraciji modela bi lahko podatkovni znanstvenik, ki se ukvarja izključno z modeli, pripomogel k bolj pravilnim vhodnim parametrom modela. V model bi lahko vključil spremenljivke sentimenta novic ali ga prilagodil glede na različne industrije delnic. Sentiment novic delnic bi lahko naredil v tretjem zavihku, kjer se pridobivajo novice o posamezni delnici. Prav tako bi lahko uporabniku omogočil shranjevanje analiz v Excelu.

LITERATURA IN VIRI

1. Aggarwal, S. (2019). *Flask Framework Cookbook*. (2. izd.). Packt Publishing, Limited.
2. Antoniou, J. (2021). Modern technology interfaces: Usability, sociability challenges and decision-making. *EAI/Springer Innovations in Communication and Computing* (str. 55–77). Springer, Cham. https://doi.org/10.1007/978-3-030-52559-0_5
3. Bansai, S. (2014). Python Requests: Interacting with the Web Made Easy. *Open Source for You*. Athena Information Solutions.
4. Beil, R. O., Ford, G. in Jackson, J. D. (2005). On the relationship between telecommunications investment and economic growth in the United States. *International economic journal*, *19*(1), 3–9.
5. Biberdorf, J. (2024a). *Harnessing the Power of On-Balance Volume Indicator in Trading.* Newstex.
6. Biberdorf, J. (2024b). *Master the Average True Range (ATR) Indicator.* Newstex.
7. Board of Governors of the Federal Reserve System. (2024). *FEDFUNDS* https://fred.stlouisfed.org/series/FEDFUNDS
8. Buckley, D. (2024). *Acceleration Bands*. https://www.daytrading.com/acceleration-bands
9. Campesato, O. (2022). *Pandas Basics*. (1. izd.). Mercury Learning & Information. https://doi-org.nukweb.nuk.uni-lj.si/10.1515/9781683928256
10. Chan, J., Chung, R. in Huang, J. (2019). *Python API Development Fundamentals: Develop a Full-Stack Web Application with Python and Flask*. Packt Publishing.
11. Chang, R. (2021). Financial Technology: China's Stock Markets vs U.S. Stock Markets. *E3S Web of Conferences*, let. 275. EDP Sciences. https://doi.org/10.1051/e3sconf/202127501006
12. Chapagain, A. (2019). *Web Scraping Fundamentals*. Packt Publishing.
13. Dabbas, E. (2021). *Interactive Dashboards and Data Apps with Plotly and Dash* (1. izd.). Packt Publishing.
14. CodeSansar. (2024a, 5. oktober). *Flowchart if-else.* https://www.codesansar.com/c-programming/if-else-statement-with-examples.htm
15. CodeSansar. (2024b, 5. oktober). *While loop flowchart.* https://www.codesansar.com/c-programming/while-loop-with-examples.htm
16. Dixon, M. F., Halperin, I. in Bilokon, P. (2020). *Machine learning in finance: From theory to practice*. Elsevier Scopus. https://doi.org/10.1007/978-3-030-41068-1
17. Dong, H., Ding, Z. in Zhang, S. (2020). *Deep reinforcement learning: Fundamentals, research and applications*. Elsevier Scopus. https://doi.org/10.1007/978-981-15-4095-0
18. Dyer, R. in Chauhan, J. (2022). An exploratory study on the predominant programming paradigms in Python code. *ESEC/FSE 2022 - Proceedings of the 30th ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (str. 684–695). https://doi.org/10.1145/3540250.3549158

19. Eldomiaty, T. I., Saeed, Y., Hammam, R., in AboulSoud, S. (2020). The associations between stock prices, inflation rates, interest rates are still persistent: Empirical evidence from stock duration model. *Journal of Economics, Finance and Administrative Science*, *25*(49), 149–161.
20. Eldridge, S. (2024, 19. oktober). Data Analysis. V Encyclopedia Britannica. https://www.britannica.com/science/data-analysis
21. European Central Bank. (2024, 20. oktober). *What is inflation?* https://www.ecb.europa.eu/ecb-and-you/explainers/tell-me-more/html/what_is_inflation.en.html
22. Finnhub. (2024, 20. oktober). *FAQ*. https://finnhub.io/faq
23. Fischer, T. in Krauss, C. (2018). Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, *270*(2), 654–669.
24. Han, J., Pei, J. in Kamber, M. (2011). *Data Mining: Concepts and Techniques* (2. izd.). Morgan Kaufmann.
25. Hardy, A. P. (1980). The role of the telephone in economic development. *Telecommunications policy*, *4*(4), 278–286. https://doi.org/10.1016/0308-5961(80)90044-0
26. Heydt, M. (2015). *Learning pandas* (1. izd.). Packt Publishing.
27. Hyndman, R. J. (2009). Moving averages. https://robjhyndman.com/papers/moving_averages.pdf
28. Janiesch, C., Zschech, P. in Heinrich, K. (2021). Machine learning and deep learning. *EM*, *31*(3), 685–695.
29. Kats, D. K. P. (2019). *Learn Python by Building Data Science Applications* (1. izd.). Packt Publishing, Limited.
30. Khafaji, S. (2024). Role of machine learning and big data mining in financial decisions. *The American Journal of Management and Economics Innovations*, *6*(7), 20–38. https://doi.org/10.37547/tajmei/volume06issue07-03
31. Khan, A. H. (2024). Effective decision making using data analytics. *Interantional Journal of Scientific Research in Engineering and Management*, *8*(4), 1–5. https://doi.org/10.55041/IJSREM32598
32. Khatri, S., Kasab, A., Dahake, S. in Hiwale, N. (2023). Stock Price Prediction using Technical Analysis. *International Journal of Advanced Research in Science, Communication and Technology*, *3*(7), 308–315. https://doi.org/10.48175/IJARSCT-9510
33. Knebl, H. (2020). *Algorithms and data structures: Foundations and probabilistic methods for design and analysis*. Elsevier Scopus. https://doi.org/10.1007/978-3-030-59758-0
34. Leff, N. H. (1984). Externalities, Information Costs, and Social Benefit-Cost Analysis for Economic Development: An Example from Telecommunications. *Economic development and cultural change*, *32*(2), 255–276. https://doi.org/10.1086/451385

35. Lombardo, R. (2024). Python and Plotly Dash, A Quick and Convenient Way to Develop Web Apps for Teaching Physical Chemistry Models. *Journal of chemical education*, *101*(11), 4661–4670. American Chemical Society and Division of Chemical Education. https://doi.org/10.1021/acs.jchemed.3c01167
36. Lott, S. F. (2015). Python essentials. V *modernize existing Python code and plan code migrations to Python using this definitive guide* (1. izd.). Packt Publishing.
37. Madden, G. in Savage, S. J. (1998). CEE telecommunications investment and economic growth. *Information economics and policy*, *10*(2), 173–195. https://doi.org/10.1016/S0167-6245(97)00020-6
38. Markham, J. W. (2002). *From Christopher Columbus to the Robber Barons: A Financial History of the United States 1492–1900* (1. izd.). Routledge. https://doi.org/10.4324/9781003247081
39. Marvin, R., Ng'ang'a, M. in Omondi, A. (2018). *Python Fundamentals* (1. izd.). Packt Publishing, Limited.
40. Mehboob Ahmed Khan, O. in Habib, K. (2020). Developing Multi-Platform Apps with Visual Studio Code. *Get up and Running with vs Code by Building Multi-Platform, Cloud-native, and Microservices-based Apps* (1. izd.). Packt Publishing, Limited.
41. Mitchell, C. (2024). *Keltner Channel: Definition, How It Works, and How to Use*. https://www.investopedia.com/terms/k/keltnerchannel.asp
42. Mokhtari, S., Yen, K. K. in Liu, J. (2021). Effectiveness of Artificial Intelligence in Stock Market Prediction based on Machine Learning. *International journal of computer applications*, *183*(7), 1–8. https://doi.org/10.5120/ijca2021921347
43. Morales Díaz, L. V. (2022). What is a User Interface, again? A Survey of Definitions of User Interface. *ACM International Conference Proceeding Series* (str. 1–7). https://doi.org/10.1145/3565494.3565504
44. Pagar, S., Jaiswal, A., Auti, A., Purohit, V. in Mishra, V. (2023). Technical Analysis of Stock Market using Data Science and its tools. *2023 7th International Conference On Computing, Communication, Control And Automation, ICCUBEA 2023* (str. 1-5). doi: 10.1109/ICCUBEA58933.2023.10392184.
45. Raipitam, S. K., Kumar, S., Dhanani, T., Bilgaiyan, S. in Gourisaria, M. K. (2023). Comparative Study on Stock Market Prediction using Generic CNN-LSTM and Ensemble Learning. *2023 International Conference on Network, Multimedia and Information Technology (NMITCON)* (str. 1–6). IEEE. https://doi.org/10.1109/NMITCON58196.2023.10275849
46. Ramlall, I. (2016). *Applied Technical Analysis for Advanced Learners and Practitioners*, (1. izd.). Emerald Publishing Limited. https://doi.org/10.1108/9781786356338
47. Reserve Bank of Australia. (2024, 20. oktober). *Unemployment: Its Measurement and Types*. https://www.rba.gov.au/education/resources/explainers/unemployment-its-measurement-and-types.html.
48. Riedl, M. O. (2019). Human-centered artificial intelligence and machine learning. *Human behavior and emerging technologies*, *1*(1), 33–36. https://doi.org/10.1002/hbe2.117

49. Romano, F. in Kruger, H. (2023). *Learn Python Programming* (3. izd.). Packt Publishing.
50. Schulp, J. J. (2021). GameStop and the Rise of Retail Trading. *The Cato journal*, *41*(3), 511–527. https://doi.org/10.36009/CJ.41.3.3
51. Sen, J., Mehtab, S., Sen, R., Dutta, A., Kherwa, P., Saheel, A., Berry, P., Khurana, S., Singh, S. Cadotte, D. W. W., Anderson, D. W., Ost, K. J., Akinbo, R. S., Daramola, O. A. in Lainjo, B. (2021). Machine Learning - Algorithms, Models and Applications. *arXiv.org*. Cornell University Library. https://doi.org/10.5772/intechopen.94615
52. Silaparasetty, V. (2020). *Deep Learning Projects Using TensorFlow 2: Neural Network Development with Python and Keras*. Apress Berkeley. https://doi.org/10.1007/978-1-4842-5802-6
53. Swanlund, S., Wilson, V., Kapoor, V., Gabbard, M. in Srinath, C. (2024). Software Development Life Cycle. *Journal of the Society for Clinical Data Management*, *4*(1), 1–24. https://doi.org/10.47912/jscdm.343
54. Weiming, J. M. (2019). *Mastering Python for Finance* (2. izd.). Packt Publishing Limited.
55. Vijayasarathy, L. R. in Butler, C. W. (2016). Choice of Software Development Methodologies: Do Organizational, Project, and Team Characteristics Matter?. *IEEE software*, *33*(5), 86–94. https://doi.org/10.1109/MS.2015.26
56. Yan, X., Yan, X. in Su, X. G. (2009). *Linear regression analysis*. World Scientific Publishing. https://doi.org/10.1142/6986
57. Zaharudin, K. Z., Young, M. R. in Hsu, W. (2022). High-frequency trading: Definition, implications, and controversies. *Journal of economic surveys*, *36*(1), 75–107. https://doi.org/10.1111/joes.12434

PRILOGA

Priloga 1: Programska koda

```
import dash
from dash import dcc, html
from dash.dependencies import Input, Output
import prediction_model
import stock_news  # Import the stock_news module
import stock_info
import dash_bootstrap_components as dbc


# Initialize the Dash app
app = dash.Dash(__name__, suppress_callback_exceptions=True,
external_stylesheets=[dbc.themes.MINTY])

# Register the callbacks for the stock_info module and prediction_model
stock_info.register_callbacks(app)
prediction_model.register_callbacks(app)
stock_news.register_callbacks(app)  # Register the callbacks for the Stock
News tab

# Define the layout of the app with custom theme
app.layout = html.Div([
    html.H1("Analyzing Stock Data", style={'text-align': 'center'}),
    dcc.Tabs(
        id='tabs',
        value='stock-info',
        persistence=True,  # Enable persistence on the tabs
        persistence_type='session',  # Ensure the selected tab persists
within the session
        children=[
            dcc.Tab(label='Stock Information', value='stock-info',
style={'backgroundColor': '#f0f0f0', 'color': '#333'}),
            dcc.Tab(label='Prediction Model', value='prediction-model',
style={'backgroundColor': '#f0f0f0', 'color': '#333'}),
            dcc.Tab(label='Stock News', value='stock-news',
style={'backgroundColor': '#f0f0f0', 'color': '#333'}),  # Correct the value
to match 'stock-news'
        ]
    ),
    html.Div(id='tabs-content')
])


# Callback to update tab content based on user selection
@app.callback(
    Output('tabs-content', 'children'),
    [Input('tabs', 'value')]
)
```

```python
def render_content(tab):
    if tab == 'stock-info':
        return stock_info.layout
    elif tab == 'prediction-model':
        return prediction_model.layout
    elif tab == 'stock-news':
        return stock_news.layout

if __name__ == '__main__':
    app.run_server(debug=True)
```

```python
from dash import html, dcc, Input, Output, State, no_update
from dash.dash_table import DataTable
import yfinance as yf
import plotly.graph_objs as go
from dash.exceptions import PreventUpdate
from plotly.subplots import make_subplots
import pandas as pd


# Define the layout of the Stock Info tab
layout = html.Div([
    dcc.Store(id='stored-graph-data', storage_type='session'),  # Data
persists across tabs
    dcc.Store(id='stored-data', storage_type='session'),  # Store data
persistently across sessions
    html.Div([
        html.Label("Stock Ticker:"),
        dcc.Input(id='stock-ticker', type='text', persistence=True,
persistence_type='session'),
        html.Button('Get Stock Information', id='submit-val', n_clicks=0),
    ], style={'text-align': 'left', 'width': '30%', 'padding': '20px'}),

    html.Div(id='output-container-button'),
    html.Div(id='table-container', children=[
        DataTable(
            id='stock-table',
            columns=[
                {"name": "Stock Ticker", "id": "ticker"},
                {"name": "Stock Name", "id": "name"},
                {"name": "Current Price", "id": "current_price"},
                {"name": "Open", "id": "open"},
                {"name": "Previous Close", "id": "prev_close"},
                {"name": "Volume", "id": "volume"},
                {"name": "Avg. Volume", "id": "avg_volume"},
                {"name": "PE Ratio", "id": "PE Ratio"},
                {"name": "EPS", "id": "EPS"}
```

```
            ],
            data=[],  # Initial empty data
            row_deletable=True,
            style_table={'width': '100%', 'margin': '20px auto'},
            style_cell={'text-align': 'center'},
            style_header={'fontWeight': 'bold'},
        )
    ], style={'text-align': 'center', 'padding': '20px'}),
    # Historical Data Section
    html.Div([
        # Div for Select Period label and dropdown
        html.Div([
            html.Label("Select Period:", style={'margin-right': '10px',
'vertical-align': 'middle', 'padding': '20px'}),
            dcc.Dropdown(
                id='period-dropdown',
                options=[
                    {'label': '1 Day', 'value': '1d'},
                    {'label': '6 Month', 'value': '6mo'},
                    {'label': '1 Year', 'value': '1y'},
                    {'label': '5 Year', 'value': '5y'},
                    {'label': 'Max', 'value': 'max'}
                ],
                clearable=False,
                persistence=True,
                persistence_type='session',
                style={'width': '200px', 'display': 'inline-block',
'vertical-align': 'middle'}  # Inline and width control
            ),
        ], style={'display': 'inline-block', 'margin-right': '40px',
'vertical-align': 'middle'}),  # Adjust margin for spacing

        # Div for Select Interval label and dropdown
        html.Div([
            html.Label("Select Interval:", style={'margin-right': '10px',
'vertical-align': 'middle'}),
            dcc.Dropdown(
                id='interval-dropdown',
                options=[
                    {'label': '1 Minute', 'value': '1m'},
                    {'label': '60 Minutes', 'value': '60m'},
                    {'label': '1 Day', 'value': '1d'},
                    {'label': '1 Month', 'value': '1mo'}
                ],
                clearable=False,
                persistence=True,
                persistence_type='session',
                style={'width': '200px', 'display': 'inline-block',
'vertical-align': 'middle'}  # Inline and width control
```

```
            ),
        ], style={'display': 'inline-block', 'margin-right': '40px',
'vertical-align': 'middle'}),  # Adjust margin for spacing

        # Button for fetching historical data
        html.Button('Get Historical Data', id='get-historical-data',
n_clicks=0,
                    style={'display': 'inline-block', 'vertical-align':
'middle', 'margin-right': '40px'}),

        # Div for Indicators label and dropdown, properly aligned
        html.Div([
            html.Label("Indicators:", style={'margin-right': '10px',
'display': 'inline-block', 'vertical-align': 'middle'}),
            dcc.Dropdown(
                id='indicators-dropdown',
                options=[
                    {'label': 'Keltner Channels', 'value': 'keltner'},
                    {'label': 'Moving Average', 'value': 'ma'},
                    {'label': 'Acceleration Bands', 'value': 'acceleration'}
                ],
                clearable=True,
                persistence=True,
                persistence_type='session',
                value=None,  # No indicator selected by default
                style={'width': '200px', 'display': 'inline-block',
'vertical-align': 'middle'}
            ),
        ], id='indicators-div', style={'display': 'inline-block', 'vertical-
align': 'middle'})  # Ensures everything is in one line


    ], style={'margin-top': '20px'}),

    dcc.Graph(
        id='historical-graph',
        style={'width': '90%'}
    )
])

def calculate_moving_average(data, period=20):
    """
    Calculate Moving Average (MA)

    Parameters:
    - data: DataFrame with stock data
    - period: Period for the moving average (default is 20)

    Returns:
```

```
    - Series with the Moving Average
    """
    return data['Close'].rolling(window=period).mean()

def calculate_keltner_channels(data, period=20, multiplier=2):
    """
    Calculate Keltner Channels (EMA, upper band, lower band)

    Parameters:
    - data: DataFrame with stock data
    - period: Period for EMA and ATR (default is 20)
    - multiplier: Multiplier for ATR to create the bands (default is 2)

    Returns:
    - DataFrame with EMA, upper band, and lower band columns
    """
    # Calculate the Exponential Moving Average (EMA)
    ema = data['Close'].ewm(span=period, adjust=False).mean()

    # Calculate the True Range (TR)
    high_low = data['High'] - data['Low']
    high_close = (data['High'] - data['Close'].shift()).abs()
    low_close = (data['Low'] - data['Close'].shift()).abs()
    true_range = pd.concat([high_low, high_close, low_close],
axis=1).max(axis=1)

    # Calculate the Average True Range (ATR)
    atr = true_range.rolling(window=period).mean()

    # Calculate the upper and lower Keltner Channels
    upper_band = ema + (multiplier * atr)
    lower_band = ema - (multiplier * atr)

    return pd.DataFrame({
        'EMA': ema,
        'Upper Band': upper_band,
        'Lower Band': lower_band
    })

def calculate_acceleration_bands(data, period=20, multiplier=4):
    """
    Calculate Acceleration Bands using the standardized ATR-based formula.

    Parameters:
    - data: DataFrame with stock data
    - period: Period for SMA and ATR (default is 20)
    - multiplier: Multiplier for ATR to create the bands (default is 4)

    Returns:
```

```
    - DataFrame with Upper Band and Lower Band
    """
    # Calculate the Simple Moving Average (SMA)
    sma = data['Close'].rolling(window=period).mean()

    # Calculate the True Range (TR)
    high_low = data['High'] - data['Low']
    high_close = (data['High'] - data['Close'].shift()).abs()
    low_close = (data['Low'] - data['Close'].shift()).abs()
    true_range = pd.concat([high_low, high_close, low_close],
axis=1).max(axis=1)

    # Calculate the Average True Range (ATR) using a rolling window
    atr = true_range.rolling(window=period).mean()

    # Calculate the Upper and Lower Acceleration Bands based on ATR
    upper_band = sma * (1 + (multiplier * atr / sma))
    lower_band = sma * (1 - (multiplier * atr / sma))

    return pd.DataFrame({
        'Upper Band': upper_band,
        'Lower Band': lower_band
    })


def register_callbacks(app):
    # Callback for fetching and displaying stock information (same as
before)
    @app.callback(
        Output('stored-data', 'data', allow_duplicate=True),
        Output('stock-table', 'data', allow_duplicate=True),
        [Input('submit-val', 'n_clicks')],
        [State('stock-ticker', 'value'), State('stored-data', 'data')],
        prevent_initial_call=True
    )
    def update_output(n_clicks, ticker, stored_data):
        if n_clicks > 0 and ticker:
            # Fetch data using yfinance
            ticker_data = yf.Ticker(ticker.strip())

            # Extract relevant data
            stock_info = ticker_data.info

            open_price = stock_info.get('open', 'N/A')
            prev_close = stock_info.get('previousClose', 'N/A')
            current_price = stock_info.get('currentPrice', 'N/A')
            volume = stock_info.get('volume', 'N/A')
            avg_volume = stock_info.get('averageVolume', 'N/A')
            pe_ratio = stock_info.get('trailingPE', 'N/A')
```

```
            eps = stock_info.get('trailingEps', 'N/A')
            stock_name = stock_info.get('longName', 'N/A')

            new_row = {
                "ticker": ticker.strip(),
                "name": stock_name,
                "current_price": current_price,
                "open": open_price,
                "prev_close": prev_close,
                "volume": volume,
                "avg_volume": avg_volume,
                "PE Ratio": pe_ratio,
                "EPS": eps
            }

            if stored_data is None:
                stored_data = []

            stored_data.append(new_row)
            return stored_data, stored_data
        else:
            raise PreventUpdate


# Callback to update DataTable and synchronize stored data
    @app.callback(
    Output('stored-data', 'data', allow_duplicate=True),
    Input('stock-table', 'data'),
    Input('stock-table', 'data_previous'),
    prevent_initial_call=True
    )
    def update_store_data_on_row_deletion(data, data_previous):
        #print('data', data)
        print('data_previous', data_previous)

        # If the table_data has fewer rows than stored_data, assume rows
were deleted

        if data is not None and data_previous is not None:
            if len(data) < len(data_previous):
                return data # Sync the stored data with the table

        # Otherwise, no deletion occurred, just keep the stored_data as is
        else: return no_update

# Callback to update DataTable and synchronize stored data
    @app.callback(
    Output('stock-table', 'data'),
    Input('stored-data', 'data'),
```

```
    )
    def update_row_data_with_stored_data(stored_data):
        if stored_data is not []:
            return stored_data


    # Callback to update the interval dropdown based on the selected period
    @app.callback(
        Output('interval-dropdown', 'options'),
        [Input('period-dropdown', 'value')]
    )
    def update_interval_options(selected_period):
        # For '1d' period, allow 1m and 60m intervals
        if selected_period == '1d':
            return [
                {'label': '1 Minute', 'value': '1m'},
                {'label': '60 Minutes', 'value': '60m'},
            ]

        # For longer periods, restrict to >=1d intervals
        else:
            return [
                {'label': '1 Day', 'value': '1d'},
                {'label': '1 Month', 'value': '1mo'}
            ]


    # Load the graph with indicators (if applied) from stored-graph-data
    @app.callback(
        Output('historical-graph', 'figure'),  # Load the graph figure
        [Input('stored-graph-data', 'data')]  # Restore the graph with
indicators from storage
    )
    def load_graph_from_store(stored_data):
        if stored_data:
            # Load the stored figure and ensure the height is maintained
            fig = go.Figure(stored_data)
            if 'layout' in stored_data and 'height' not in
stored_data['layout']:
                # Set height if it's not explicitly stored
                fig.update_layout(height=600 * len(stored_data['data']))
            return fig
        else:
            return {'data': [], 'layout': go.Layout(title="No historical
data available")}



    @app.callback(
```

```
        [Output('stored-graph-data', 'data'),  # Store graph data for
persistence
        Output('historical-graph', 'figure', allow_duplicate=True),  #
Output the graph figure
        Output('indicators-dropdown', 'value')],  # Clear the indicators
dropdown
        [Input('get-historical-data', 'n_clicks')],  # Trigger when "Get
Historical Data" is clicked
        [State('stock-table', 'data'),  # Get tickers from the stock table
        State('period-dropdown', 'value'),  # Get selected period
        State('interval-dropdown', 'value')],  # Get selected interval
        prevent_initial_call=True
    )
    def update_historical_data(n_clicks, stock_table_data, selected_period,
selected_interval):
        if n_clicks > 0 and stock_table_data:
            tickers = [row['ticker'] for row in stock_table_data if 'ticker'
in row]

            # Create subplots for each ticker with independent x-axis
            fig = make_subplots(
                rows=len(tickers), cols=1,
                shared_xaxes=False,  # Disable shared x-axis to allow
independent time ranges
                subplot_titles=[f"{ticker} Historical Data
({selected_period} - {selected_interval})" for ticker in tickers],
                vertical_spacing=0.1  # Adjust vertical spacing between
graphs
            )

            # Restore the original traces (closing prices)
            closing_price_color = '#000000'  # Black for all closing prices
            for i, ticker in enumerate(tickers):
                ticker_data = yf.Ticker(ticker.strip())
                hist_data = ticker_data.history(period=selected_period,
interval=selected_interval)

                if hist_data.empty:
                    continue

                # Add closing price trace (black)
                fig.add_trace(go.Scatter(
                    x=hist_data.index,
                    y=hist_data['Close'],
                    mode='lines',
                    line=dict(color=closing_price_color),
                    name=f'{ticker} Closing Price',
                    showlegend=False
                ), row=i+1, col=1)
```

```
            # Set the figure height: 600 pixels per ticker
            fig.update_layout(
                height=600 * len(tickers),  # 600 pixels per ticker
                showlegend=False  # Disable global legend
            )

            # Clear the Indicators dropdown when "Get Historical Data" is
clicked
            return fig.to_dict(), fig, None  # Reset indicators dropdown to
None

        else:
            raise PreventUpdate


    # Apply indicators and update stored graph data
    @app.callback(
        [Output('historical-graph', 'figure', allow_duplicate=True),  #
Update the graph figure
        Output('stored-graph-data', 'data', allow_duplicate=True)],  # Store
the graph (including indicators)
        [Input('indicators-dropdown', 'value')],  # Trigger when the
indicator dropdown is selected/cleared
        [State('stock-table', 'data'),  # Get tickers from the stock table
        State('period-dropdown', 'value'),  # Get selected period
        State('interval-dropdown', 'value')],  # Get selected interval
        prevent_initial_call=True
    )
    def apply_indicator(selected_indicator, stock_table_data,
selected_period, selected_interval):
        if stock_table_data:
            tickers = [row['ticker'] for row in stock_table_data if 'ticker'
in row]
            fig = make_subplots(
                rows=len(tickers), cols=1,
                shared_xaxes=False,  # Disable shared x-axis to allow
independent time ranges
                subplot_titles=[f"{ticker} Historical Data
({selected_period} - {selected_interval})" for ticker in tickers],
                vertical_spacing=0.1  # Adjust vertical spacing between
graphs
            )

            # Make closing prices black
            closing_price_color = '#000000'

            # Restore the original traces (closing prices)
            for i, ticker in enumerate(tickers):
```

```
                ticker_data = yf.Ticker(ticker.strip())
                hist_data = ticker_data.history(period=selected_period,
interval=selected_interval)

                if hist_data.empty:
                    continue

                # Add closing price trace (black)
                fig.add_trace(go.Scatter(
                    x=hist_data.index,
                    y=hist_data['Close'],
                    mode='lines',
                    line=dict(color=closing_price_color),
                    name=f'{ticker} Closing Price',
                    showlegend=False
                ), row=i+1, col=1)

            # If the dropdown is cleared (None), do not add any indicators,
return only the closing prices
            if selected_indicator is None:
                # Set the figure height: 600 pixels per ticker
                fig.update_layout(height=600 * len(tickers))  # Explicitly
set the height
                return fig, fig.to_dict()  # Update and store the graph
without any indicators

            # Apply selected indicator if dropdown is not cleared
            for i, ticker in enumerate(tickers):
                ticker_data = yf.Ticker(ticker.strip())
                hist_data = ticker_data.history(period=selected_period,
interval=selected_interval)

                if hist_data.empty:
                    continue

                if selected_indicator == 'keltner':
                    # Calculate Keltner Channels
                    keltner_data = calculate_keltner_channels(hist_data)

                    # Add Keltner Channels to the correct row of the graph
                    fig.add_trace(go.Scatter(
                        x=hist_data.index,
                        y=keltner_data['EMA'],
                        mode='lines',
                        line=dict(color='blue'),
                        name=f'{ticker} EMA',
                        showlegend=False
                    ), row=i+1, col=1)
```

```
                fig.add_trace(go.Scatter(
                    x=hist_data.index,
                    y=keltner_data['Upper Band'],
                    mode='lines',
                    line=dict(color='red', dash='dash'),
                    name=f'{ticker} Upper Keltner Band',
                    showlegend=False
                ), row=i+1, col=1)

                fig.add_trace(go.Scatter(
                    x=hist_data.index,
                    y=keltner_data['Lower Band'],
                    mode='lines',
                    line=dict(color='green', dash='dash'),
                    name=f'{ticker} Lower Keltner Band',
                    showlegend=False
                ), row=i+1, col=1)

            elif selected_indicator == 'ma':
                # Calculate Moving Average
                ma_data = calculate_moving_average(hist_data)

                # Add Moving Average
                fig.add_trace(go.Scatter(
                    x=hist_data.index,
                    y=ma_data,
                    mode='lines',
                    line=dict(color='orange'),
                    name=f'{ticker} {len(ma_data)}-Period MA',
                    showlegend=False
                ), row=i+1, col=1)

            elif selected_indicator == 'acceleration':
                # Calculate Acceleration Bands (using the new formula)
                acc_bands = calculate_acceleration_bands(hist_data)

                # Add Acceleration Bands to the correct row of the graph
                fig.add_trace(go.Scatter(
                    x=hist_data.index,
                    y=acc_bands['Upper Band'],
                    mode='lines',
                    line=dict(color='purple', dash='dash'),
                    name=f'{ticker} Upper Acceleration Band',
                    showlegend=False
                ), row=i+1, col=1)

                fig.add_trace(go.Scatter(
                    x=hist_data.index,
                    y=acc_bands['Lower Band'],
```

```
                        mode='lines',
                        line=dict(color='cyan', dash='dash'),
                        name=f'{ticker} Lower Acceleration Band',
                        showlegend=False
                    ), row=i+1, col=1)

            # Set the figure height: 600 pixels per ticker
            fig.update_layout(height=600 * len(tickers))  # Explicitly set
the height
            return fig, fig.to_dict()  # Update and store the graph with
indicators

        else:
            raise PreventUpdate



# You must call this function in your app.py file
# stock_info.register_callbacks(app
```

```
from dash import html, dcc, Input, Output, State
import yfinance as yf
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.metrics import root_mean_squared_error, r2_score,
mean_absolute_error
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import GridSearchCV
from dash.exceptions import PreventUpdate
from keras import models, layers
from sklearn.preprocessing import MinMaxScaler
import plotly.graph_objs as go
import numpy as np
import pandas as pd
from fredapi import Fred
# Initialize the FRED API with your API key
fred = Fred(api_key='546111fc45e588f667ed386a648cc1c7')


# Define the layout for the Prediction Model tab
layout = html.Div([
    dcc.Store(id='stored-graph-prediction', storage_type='session'),  #
Store the graph data

    # Model selection, radio buttons, and button (horizontally aligned but
radio buttons vertically stacked)
    html.Div([
```

```
        # Model selection dropdown with box and inline label
        html.Div([
            html.Div([
                html.Label("Stock Ticker:"),
                dcc.Input(id='stock-ticker-prediction', type='text',
persistence=True, persistence_type='session'),
            ], style={'text-align': 'left', 'padding': '5px'}),  # Left-
aligned input
            html.Div([
                html.Label("Select Model:", style={'margin-right': '10px',
'vertical-align': 'middle'}),
                dcc.Dropdown(
                    id='model-selection',
                    options=[
                        {'label': 'Linear Regression', 'value': 'linear'},
                        {'label': 'Ridge Regression', 'value': 'ridge'},
                        {'label': 'Lasso Regression', 'value': 'lasso'},
                        {'label': 'LSTM', 'value': 'lstm'}
                    ],
                    value='linear',  # Default to linear regression
                    clearable=False,
                    persistence=True,  # Enable persistence
                    persistence_type='session', # Store across sessions
                    style={'width': '170px', 'vertical-align': 'middle'}  #
Align with label
                ),
            ], style={'display': 'inline-flex', 'align-items': 'center',
'text-align': 'left'}),
        ], style={'border': '1px solid #ddd', 'padding': '15px', 'border-
radius': '5px', 'width': '20%', 'display': 'inline-block', 'text-align':
'left'}),

        # Radio button for selecting the historical data period (with box
and left alignment)
        html.Div([
            html.Div([
                html.Label("Historical Data:"),
                dcc.RadioItems(
                    id='historical-data',
                    options=[
                        {'label': '1 Year', 'value': '1y'},
                        {'label': '2 Years', 'value': '2y'},
                        {'label': '5 Years', 'value': '5y'},
                        {'label': '10 Years', 'value': '10y'}
                    ],
                    value='2y',  # Default selection
                    persistence=True,  # Enable persistence
                    persistence_type='session', # Store across sessions
```

```
                    labelStyle={'display': 'block', 'text-align': 'left'}  # 
Align labels to the left
                ),
            ], style={'padding': '10px'})
        ], style={'border': '1px solid #ddd', 'padding': '15px', 'border-
radius': '5px', 'margin-right': '20px', 'width': '20%', 'display': 'inline-
block'}),

        # Radio button for selecting the prediction days (with box and left 
alignment)
        html.Div([
            html.Div([
                html.Label("Prediction Days:"),
                dcc.RadioItems(
                    id='prediction-days',
                    options=[
                        {'label': '30 Days', 'value': 30},
                        {'label': '45 Days', 'value': 45},
                        {'label': '60 Days', 'value': 60}
                    ],
                    value=30,  # Default selection
                    persistence=True,  # Enable persistence
                    persistence_type='session', # Store across sessions
                    labelStyle={'display': 'block', 'text-align': 'left'}  # 
Align labels to the left
                ),
            ], style={'padding': '10px'})
        ], style={'border': '1px solid #ddd', 'padding': '15px', 'border-
radius': '5px', 'margin-right': '20px', 'width': '20%', 'display': 'inline-
block'}),

        # Button to train the model with box
        html.Div([
            html.Button('Train Model', id='train-lr', n_clicks=0)
        ], style={'border': '1px solid #ddd', 'padding': '15px', 'border-
radius': '5px', 'width': '10%', 'display': 'inline-block'}),

    ], style={'text-align': 'center', 'padding': '20px', 'width': '100%', 
'display': 'flex', 'justify-content': 'space-around', 'align-items': 
'center'}),

    html.Div([
        dcc.Loading(
            id="loading-spinner",
            children=[dcc.Graph(id='price-prediction-graph')],
            type="default"  # You can choose different spinner types
        )
    ])
```

```
])


def fill_missing_data_with_avg(data_series):
    """
    Fills missing data by averaging the previous and next available values.
    If data is missing at the beginning or end, it fills with the closest
available value.

    Parameters:
    - data_series: Pandas Series of daily economic data (e.g., Treasury
Yield)

    Returns:
    - A Pandas Series with missing values filled.
    """
    # Fill missing values by taking the average of the previous and next
available values
    for i in range(1, len(data_series) - 1):
        if np.isnan(data_series[i]):  # If the current value is NaN
            prev_value = data_series[i - 1]
            next_value = data_series[i + 1]

            # Check if the next value is also NaN, if not, take the average
            if not np.isnan(prev_value) and not np.isnan(next_value):
                data_series[i] = (prev_value + next_value) / 2
            elif np.isnan(next_value):  # If the next value is NaN, fill
with the previous value
                data_series[i] = prev_value

    # Fill the missing values at the beginning or end with the closest
available value
    data_series.fillna(method='ffill', inplace=True)  # Forward fill
    data_series.fillna(method='bfill', inplace=True)  # Backward fill

    return data_series


def fetch_economic_data(stock_dates):
    """
    Fetches inflation, unemployment, interest rate, and daily Treasury Yield
data from FRED API
    for the given stock date range.

    Handles missing Treasury Yield data by filling NaN values using the
average of the previous
    and next available values.

    Parameters:
```

```
    - stock_dates: Index of stock data dates (usually daily).

    Returns:
    - Four DataFrames: inflation, unemployment, interest_rate, and
treasury_yield.
    """
    # Determine the start and end date from the stock data
    start_date = stock_dates.min().strftime('%Y-%m-%d')
    end_date = stock_dates.max().strftime('%Y-%m-%d')

    # Fetch the economic data from FRED API for the given date range
    inflation = fred.get_series('FPCPITOTLZGUSA',
observation_start=start_date, observation_end=end_date)
    unemployment = fred.get_series('UNRATE', observation_start=start_date,
observation_end=end_date)
    interest_rate = fred.get_series('FEDFUNDS',
observation_start=start_date, observation_end=end_date)
    treasury_yield = fred.get_series('DGS10', observation_start=start_date,
observation_end=end_date)  # Daily Data

    # Convert the series to DataFrames
    inflation = inflation.to_frame(name='inflation').reset_index()
    unemployment = unemployment.to_frame(name='unemployment').reset_index()
    interest_rate =
interest_rate.to_frame(name='interest_rate').reset_index()
    treasury_yield =
treasury_yield.to_frame(name='treasury_yield').reset_index()

    # Convert the index to datetime for each DataFrame
    inflation['index'] = pd.to_datetime(inflation['index'])
    unemployment['index'] = pd.to_datetime(unemployment['index'])
    interest_rate['index'] = pd.to_datetime(interest_rate['index'])
    treasury_yield['index'] = pd.to_datetime(treasury_yield['index'])

    # Handle missing Treasury Yield data: fill missing values by averaging
the neighboring values
    treasury_yield.set_index('index', inplace=True)  # Set the date as the
index
    treasury_yield = treasury_yield.reindex(pd.date_range(start_date,
end_date, freq='D'))  # Reindex daily
    treasury_yield['treasury_yield'].fillna(method='ffill', inplace=True)  #
Forward-fill first
    treasury_yield['treasury_yield'].fillna(method='bfill', inplace=True)  #
Backward-fill for any remaining

    # Print the DataFrames to inspect them
    print("Inflation DataFrame:")
    print(inflation)
```

```
    print("Unemployment DataFrame:")
    print(unemployment)

    print("Interest Rate DataFrame:")
    print(interest_rate)

    print("Treasury Yield DataFrame:")
    print(treasury_yield)

    # Return the four DataFrames
    return inflation, unemployment, interest_rate, treasury_yield



# Function to get historical stock data
def get_stock_data(ticker, period, interval="1d"):
    ticker_data = yf.Ticker(ticker)
    hist_data = ticker_data.history(period=period, interval=interval)

    return hist_data

# Calculate EMA
def calculate_ema(data, period=20):
    """
    Calculate Exponential Moving Average (EMA).

    Parameters:
    - data: DataFrame with stock data
    - period: Period for the EMA (default is 20)

    Returns:
    - Series with the EMA
    """
    ema = data['Close'].ewm(span=period, adjust=False).mean()
    return ema

# Calculate MACD
def calculate_macd(data, short_period=12, long_period=26):
    """
    Calculate the MACD (Moving Average Convergence Divergence).

    Parameters:
    - data: DataFrame with stock data
    - short_period: Short period EMA (default is 12)
    - long_period: Long period EMA (default is 26)

    Returns:
    - Series with MACD values
    """
```

```
    ema_short = calculate_ema(data, period=short_period)
    ema_long = calculate_ema(data, period=long_period)
    macd = ema_short - ema_long
    return macd

# Calculate OBV (On-Balance Volume)
def calculate_obv(data):
    """
    Calculate On-Balance Volume (OBV).

    Parameters:
    - data: DataFrame with stock data

    Returns:
    - Series with OBV values
    """
    obv = [0]  # Start with the first OBV as 0
    for i in range(1, len(data)):
        if data['Close'][i] > data['Close'][i-1]:
            obv.append(obv[-1] + data['Volume'][i])
        elif data['Close'][i] < data['Close'][i-1]:
            obv.append(obv[-1] - data['Volume'][i])
        else:
            obv.append(obv[-1])

    return pd.Series(obv, index=data.index)

def calculate_bollinger_bands(data, period=20, std_dev_factor=2):
    sma = data['Close'].rolling(window=period).mean()
    rolling_std = data['Close'].rolling(window=period).std()
    upper_band = sma + (rolling_std * std_dev_factor)
    lower_band = sma - (rolling_std * std_dev_factor)

    return upper_band, lower_band

def calculate_momentum(data, period=10):
    momentum = data['Close'].diff(period)
    return momentum

def calculate_stochastic_oscillator(data, period=14):
    low_min = data['Low'].rolling(window=period).min()
    high_max = data['High'].rolling(window=period).max()
    stoch = 100 * (data['Close'] - low_min) / (high_max - low_min)

    return stoch

def calculate_atr(data, period=14):
    high_low = data['High'] - data['Low']
    high_close = abs(data['High'] - data['Close'].shift())
```

```
    low_close = abs(data['Low'] - data['Close'].shift())
    true_range = pd.concat([high_low, high_close, low_close],
axis=1).max(axis=1)
    atr = true_range.rolling(window=period).mean()

    return atr

def calculate_volume_oscillator(data, short_period=5, long_period=20):
    short_vol_ma = data['Volume'].rolling(window=short_period).mean()
    long_vol_ma = data['Volume'].rolling(window=long_period).mean()
    vol_osc = ((short_vol_ma - long_vol_ma) / long_vol_ma) * 100

    return vol_osc

# Prepare the data for training
def prepare_data(df):
    df['SMA_20'] = df['Close'].rolling(window=20).mean()
    df['SMA_50'] = df['Close'].rolling(window=50).mean()
    df['EMA_20'] = calculate_ema(df, period=20)
    df['MACD'] = calculate_macd(df)
    df['OBV'] = calculate_obv(df)
    df['RSI'] = calculate_rsi(df['Close'])
    df['Bollinger_Upper'], df['Bollinger_Lower'] =
calculate_bollinger_bands(df)
    df['Momentum'] = calculate_momentum(df)
    df['Stochastic'] = calculate_stochastic_oscillator(df)
    df['ATR'] = calculate_atr(df)
    df['Volume_Oscillator'] = calculate_volume_oscillator(df)

    df.dropna(inplace=True)  # Remove rows with NaN values due to rolling
window

    # Define the feature set and target
    X = df[['SMA_20', 'SMA_50', 'EMA_20', 'MACD', 'OBV', 'RSI',
'Bollinger_Upper', 'Bollinger_Lower',
            'Momentum', 'Stochastic', 'ATR', 'Volume_Oscillator',
'inflation', 'unemployment', 'interest_rate']]
    y = df['Close'].shift(-1)  # Predict the next day's closing price

    print('y', y)
    print('X', X)
    X = X.iloc[:-1]
    y = y.iloc[:-1]

    print('y', y)
    print('X', X)

    return train_test_split(X, y, test_size=0.2, shuffle=False)
```

```python
# RSI Calculation (already implemented)
def calculate_rsi(series, window=14):
    delta = series.diff(1)
    gain = (delta.where(delta > 0, 0)).rolling(window=window).mean()
    loss = (-delta.where(delta < 0, 0)).rolling(window=window).mean()
    rs = gain / loss
    return 100 - (100 / (1 + rs))


# Function to calculate MAPE
def calculate_mape(y_true, y_pred):
    """
    Calculate Mean Absolute Percentage Error (MAPE).

    Parameters:
    - y_true: Array of actual values
    - y_pred: Array of predicted values

    Returns:
    - MAPE value
    """
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100



def update_features_for_prediction(features, predicted_close_price):
    """
    Update the technical indicators for the next day based on the predicted
closing price.
    This is a simplified example; in reality, you'd also need the high/low
prices, etc.
    """
    # Update the moving averages with the predicted price
    features['SMA_20'] = (features['SMA_20'] * 19 + predicted_close_price) /
20
    features['SMA_50'] = (features['SMA_50'] * 49 + predicted_close_price) /
50

    # Update RSI (use np.where to handle Series values)
    delta = predicted_close_price - features['SMA_20']  # Approximation for
simplicity
    gain = np.where(delta > 0, delta, 0)  # Element-wise operation for gain
    loss = np.where(delta < 0, -delta, 0)  # Element-wise operation for loss

    # Avoid division by zero with np.where
    rs = np.where(loss != 0, gain / loss, np.inf)  # Handle division by zero
    features['RSI'] = 100 - (100 / (1 + rs))

    return features
```

```
def fine_tune_model(model_type, X_train, y_train):
    """
    Fine-tune the Ridge or Lasso model using GridSearchCV with 5-fold cross-
validation.

    Parameters:
    - model_type: 'ridge' or 'lasso'
    - X_train: Training feature set
    - y_train: Training labels

    Returns:
    - The best model found by GridSearchCV.
    """
    # Define the parameter grid for Ridge and Lasso models
    if model_type == 'ridge':
        param_grid = {
            'alpha': [0.01, 0.1, 1, 10, 100],
            'max_iter': [1000, 5000, 10000],
            'tol': [1e-3, 1e-4, 1e-5],
            'solver': ['auto', 'svd', 'cholesky']
        }
        model = Ridge()
    elif model_type == 'lasso':
        param_grid = {
            'alpha': [0.01, 0.1, 1, 10, 100],
            'max_iter': [1000, 5000, 10000],
            'tol': [1e-3, 1e-4, 1e-5]
        }
        model = Lasso()

    # GridSearchCV for hyperparameter tuning with 5-fold cross-validation
    grid_search = GridSearchCV(model, param_grid, cv=5,
scoring='neg_mean_squared_error')
    grid_search.fit(X_train, y_train)

    # Get the best parameters and model from GridSearchCV
    best_model = grid_search.best_estimator_

    print(f"Best parameters for {model_type}: {grid_search.best_params_}")

    return best_model

# Register callbacks for the Prediction Model tab
def register_callbacks(app):

    @app.callback(
    Output('price-prediction-graph', 'figure'),
```

```
    [Input('stored-graph-prediction', 'data')],
    )
    def load_graph_from_store(stored_graph_data):
        if stored_graph_data:
            return go.Figure(stored_graph_data)
        else:
            return {'data': [], 'layout': go.Layout(title="No prediction
data available.")}



    @app.callback(
        [Output('price-prediction-graph', 'figure', allow_duplicate=True),
        Output('stored-graph-prediction', 'data')],
        [Input('train-lr', 'n_clicks')],
        [State('model-selection', 'value'),
        State('stock-ticker-prediction', 'value'),
        State('historical-data', 'value'),
        State('prediction-days', 'value')],
        prevent_initial_call=True)
    def train_model(n_clicks, selected_model, ticker_value, historical_data,
prediction_days):
        if n_clicks > 0:

                ticker = ticker_value

                df = get_stock_data(ticker_value, period=historical_data)

                # Get the stock dates from the df
                stock_dates = df.index
                # Ensure the Date index is in Datetime format
                if not pd.api.types.is_datetime64_any_dtype(df.index):
                    df.index = pd.to_datetime(df.index)

                # Fetch economic data for the stock date range
                inflation, unemployment, interest_rate, treasury_yield =
fetch_economic_data(stock_dates)

                # Extract distinct years from the stock data
                df['year'] = df.index.year  # Extract year from stock dates
                df['month'] = df.index.month  # Extract month from stock
dates
                df['date'] = df.index
                distinct_years = df['year'].unique()

                # Handle Inflation (forward-fill for missing years)
                inflation['year'] = inflation['index'].dt.year  # Extract
year from inflation dates
```

```
                # Create a DataFrame with all years from the stock data to
ensure inflation covers all years
                full_years = pd.DataFrame({'year': distinct_years})

                # Drop duplicates in inflation to handle only unique years,
and forward-fill missing future years
                inflation =
inflation.drop_duplicates(subset='year').set_index('year').reindex(full_year
s['year']).ffill().reset_index()

                print('inflation123', inflation)
                # Create 'year_month' column from year and month
                df['year_month'] = pd.to_datetime(df['year'].astype(str) +
'-' + df['month'].astype(str), format='%Y-%m')

                print('Stanje df pred združitvijo')
                print(df.head(5))
                print('Stanje inflation pred združitvijo')
                print(inflation.head(5))

                # Merge inflation with stock data on the year
                df = pd.merge(df, inflation[['year', 'inflation']],
on='year', how='left')

                print('Stanje df po združitvi df in inflation - PRVIH 5
VRSTIC')
                print(df.head(5))
                print('Stanje df po združitvi df in inflation - ZADNJIH 5
VRSTIC')
                print(df.tail(5))

                # Handle Unemployment
                unemployment['year'] = unemployment['index'].dt.year  #
Extract year from unemployment dates
                unemployment['month'] = unemployment['index'].dt.month  #
Extract month from unemployment dates

                # Create a 'year_month' column in the format YYYY-MM from
year and month
                unemployment['year_month'] =
pd.to_datetime(unemployment['year'].astype(str) + '-' +
unemployment['month'].astype(str), format='%Y-%m')

                print('unemployment pred združitvijo z df')
                print(unemployment.head(5))
                print('df pred združitvijo z unemployment')
                print(df.head(5))
```

```
                # Merge unemployment with stock data on the 'year_month'
column
                df = pd.merge(df, unemployment[['year_month',
'unemployment']], on='year_month', how='left')

                print('df po združitvi - prvih in zadnjih 5 vrstic')
                print(df.head(5))
                print(df.tail(5))

                # Forward-fill missing unemployment data for the current
month
                df['unemployment'] = df['unemployment'].ffill()

                print('df po polnjenju - zadnjih 5 vrstic')
                print(df.tail(5))


                # Handle Interest Rate
                interest_rate['year'] = interest_rate['index'].dt.year  #
Extract year from interest rate dates
                interest_rate['month'] = interest_rate['index'].dt.month  #
Extract month from interest rate dates

                # Create a 'year_month' column in the format YYYY-MM from
year and month
                interest_rate['year_month'] =
pd.to_datetime(interest_rate['year'].astype(str) + '-' +
interest_rate['month'].astype(str), format='%Y-%m')

                # Merge interest rate with stock data on the 'year_month'
column
                df = pd.merge(df, interest_rate[['year_month',
'interest_rate']], on='year_month', how='left')

                print('df po merge z interest rate', df)


                # Forward-fill missing interest rate data for the current
month
                df['interest_rate'] = df['interest_rate'].ffill()

                print('df po merge z interest rate + ffil', df)

                #Set the 'date' column as the index before trying the model
                df.set_index('date', inplace=True)

                try:
                    X_train, X_test, y_train, y_test = prepare_data(df)
```

```
                    if selected_model == 'lstm':
                        # Normalizacija vhodnih podatkov
                        scaler_X = MinMaxScaler(feature_range=(0, 1))
                        scaler_y = MinMaxScaler(feature_range=(0, 1))

                        # Normaliziraj X_train, X_test in y_train
                        X_scaled = scaler_X.fit_transform(X_train)  # Fit na
trening podatke
                        X_test_scaled = scaler_X.transform(X_test)  #
Transformiraj testne podatke
                        y_scaled =
scaler_y.fit_transform(y_train.values.reshape(-1, 1))  # Fit na y_train
                        y_test_scaled =
scaler_y.transform(y_test.values.reshape(-1, 1))  # Transformiraj y_test

                        # Ustvari časovne korake za LSTM model
                        time_steps = 20
                        X_train_lstm, y_train_lstm = [], []
                        for i in range(time_steps, len(X_scaled)):
                            X_train_lstm.append(X_scaled[i-time_steps:i])
                            y_train_lstm.append(y_scaled[i])

                        X_train_lstm, y_train_lstm = np.array(X_train_lstm),
np.array(y_train_lstm)

                        # Pripravi testne podatke
                        X_test_lstm, y_test_lstm = [], []
                        for i in range(time_steps, len(X_test_scaled)):
                            X_test_lstm.append(X_test_scaled[i-
time_steps:i])
                            y_test_lstm.append(y_test_scaled[i])

                        X_test_lstm, y_test_lstm = np.array(X_test_lstm),
np.array(y_test_lstm)

                        # Oblikovanje LSTM modela
                        model = models.Sequential([
                            layers.LSTM(128, return_sequences=True,
input_shape=(X_train_lstm.shape[1], X_train_lstm.shape[2])),
                            layers.Dropout(0.4),
                            layers.LSTM(128, return_sequences=False),
                            layers.Dropout(0.4),
                            layers.Dense(64, activation='relu'),
                            layers.Dense(1)
                        ])
                        model.compile(optimizer='adam',
loss='mean_squared_error')

                        # Treniranje modela
```

```
                        model.fit(X_train_lstm, y_train_lstm, epochs=100,
batch_size=16, verbose=1)

                        # Napoved na testnih podatkih
                        y_pred_scaled = model.predict(X_test_lstm)
                        y_pred = scaler_y.inverse_transform(y_pred_scaled)
                        y_test_trimmed =
scaler_y.inverse_transform(y_test_lstm)

                        # Stabilno napovedovanje prihodnjih dni
                        future_predictions = []
                        last_known_data = X_test_scaled[-time_steps:]
                        last_known_data = np.reshape(last_known_data, (1,
last_known_data.shape[0], last_known_data.shape[1]))

                        # Eksponentno glajenje za stabilnost
                        alpha = 0.5  # Faktor glajenja
                        for _ in range(prediction_days):
                            # Napoved za naslednji dan
                            next_day_pred_scaled =
model.predict(last_known_data)[0][0]
                            next_day_pred =
scaler_y.inverse_transform([[next_day_pred_scaled]])[0][0]

                            # Eksponentno glajenje
                            if len(future_predictions) > 0:
                                smoothed_pred = alpha * next_day_pred + (1 -
alpha) * future_predictions[-1]
                            else:
                                smoothed_pred = next_day_pred

                            # Shrani in posodobi
                            future_predictions.append(smoothed_pred)

                            # Posodobi last_known_data za naslednji korak
                            # Uporabimo celoten vektor funkcij z ustreznim
številom lastnosti
                            new_row = last_known_data[:, -1, :].copy()  #
Uporabimo zadnjo vrstico
                            new_row[-1] =
scaler_y.transform([[smoothed_pred]])[0][0]  # Nadomestimo ciljno
spremenljivko
                            last_known_data =
np.concatenate((last_known_data[:, 1:, :], new_row[np.newaxis, :, :]),
axis=1)


                        # Metrike
                        mae = mean_absolute_error(y_test_trimmed, y_pred)
```

```
                        rmse = root_mean_squared_error(y_test_trimmed,
y_pred)
                        mape = calculate_mape(y_test_trimmed, y_pred)
                        r2 = r2_score(y_test_trimmed, y_pred)

                        predicted_price = future_predictions[-1]

                        # Prikaz rezultatov na grafu
                        fig = go.Figure()  # Initialize the figure here to
avoid the UnboundLocalError

                        # 1. Zgodovinske cene
                        fig.add_trace(go.Scatter(
                            x=df.index,
                            y=df['Close'],
                            mode='lines',
                            name='Historical Stock Price',
                            line=dict(color='blue', width=2)
                        ))

                        # 2. Dejanske in napovedane vrednosti za zadnjih X
dni
                        comparison_start = -len(y_test_trimmed)
                        fig.add_trace(go.Scatter(
                            x=df.index[comparison_start:],
                            y=y_test_trimmed.flatten(),
                            mode='lines',
                            name='Actual Prices (Last X Days)',
                            line=dict(color='orange', width=2, dash='solid')
                        ))
                        fig.add_trace(go.Scatter(
                            x=df.index[comparison_start:],
                            y=y_pred.flatten(),
                            mode='lines',
                            name='Predicted Prices (Last X Days)',
                            line=dict(color='red', dash='dash')
                        ))

                        # 3. Prihodnje napovedi
                        future_dates = pd.date_range(df.index[-1],
periods=prediction_days + 1, freq='B')[1:]
                        fig.add_trace(go.Scatter(
                            x=future_dates,
                            y=future_predictions,
                            mode='lines',
                            name='Future Predictions',
                            line=dict(color='green', dash='dot')
                        ))
```

```
                        # Dodaj metrike na graf
                        fig.add_annotation(
                            text=(f"MAE: {mae:.2f}<br>"
                                f"RMSE: {rmse:.2f}<br>"
                                f"MAPE: {mape:.2f}%<br>"
                                f"R²: {r2:.2f}<br>"
                                f"Predicted Price in {prediction_days} days:
{predicted_price:.2f}"),
                            xref='paper', yref='paper', x=0.05, y=1,
showarrow=False,
                            bordercolor='black', bgcolor='white',
font=dict(size=12),
                            align='left'
                        )

                        fig.update_layout(
                            title=f"{ticker} Stock Price Prediction for
{prediction_days} Days using LSTM with {historical_data}",
                            xaxis_title="Date",
                            yaxis_title="Price",
                            height=600
                        )

                        return fig, fig


                    else:
                            # Select the appropriate model and perform fine-
tuning
                        if selected_model == 'linear':
                            model = LinearRegression()
                            model.fit(X_train, y_train)

                        elif selected_model == 'ridge':
                            model = fine_tune_model('ridge', X_train,
y_train)  # Fine-tune Ridge model

                        elif selected_model == 'lasso':
                            model = fine_tune_model('lasso', X_train,
y_train)  # Fine-tune Lasso model

                        y_pred = model.predict(X_test)

                        # Calculate metrics
                        mae = mean_absolute_error(y_test, y_pred)
                        rmse = root_mean_squared_error(y_test, y_pred)
                        mape = calculate_mape(y_test, y_pred)
                        r2 = r2_score(y_test, y_pred)
```

```
                        # Predict the stock price for the next n days
                        last_known_data = X_test.iloc[-1:].copy()
                        future_predictions = []

                        for _ in range(prediction_days):
                            next_day_pred =
model.predict(last_known_data)[0]
                            future_predictions.append(next_day_pred)
                            last_known_data =
update_features_for_prediction(last_known_data, next_day_pred)

                        predicted_price = future_predictions[-1]

                        # Plot stock price and prediction graph
                        fig = go.Figure()

                        # Plot actual stock prices
                        fig.add_trace(go.Scatter(
                            x=df.index,
                            y=df['Close'],
                            mode='lines',
                            name='Actual Stock Price'
                        ))

                        # Add predicted prices to the graph
                        fig.add_trace(go.Scatter(
                            x=X_test.index,
                            y=y_pred,
                            mode='lines',
                            name='Predicted Prices',
                            line=dict(dash='dash')
                        ))

                        # Add future predictions
                        future_dates = pd.date_range(df.index[-1],
periods=prediction_days + 1)
                        fig.add_trace(go.Scatter(
                            x=future_dates,
                            y=future_predictions,
                            mode='lines',
                            name='Future Predictions',
                            line=dict(color='green', dash='dot')
                        ))

                        # Add annotations with the metrics including MAPE
and predicted price
                        fig.add_annotation(
                            text=(
                                f"MAE: {mae:.2f}<br>"
```

```
                                f"RMSE: {rmse:.2f}<br>"
                                f"MAPE: {mape:.2f}%<br>"
                                f"R²: {r2:.2f}<br>"
                                f"Predicted Price in {prediction_days} days:
{predicted_price:.2f}"
                            ),
                            xref='paper', yref='paper', x=0.05, y=1,
showarrow=False,
                            bordercolor='black', bgcolor='white',
font=dict(size=12),
                            align='left'
                        )  # Closing parenthesis added here

                        fig.update_layout(
                            title=f"{ticker} Stock Price Prediction for
{prediction_days} Days using {selected_model.upper()} with
{historical_data}",
                            xaxis_title="Date",
                            yaxis_title="Price",
                            height=600
                        )

                    return fig, fig

                except Exception as e:
                    print("Error encountered:", e)
                    # Return an empty figure in case of error
                    return fig, fig

        else:
            raise PreventUpdate



# stock_news.py
from dash import html, dcc, Input, Output, State
import requests
import datetime
from dash import dash_table

# Define the layout for the stock news tab
layout = html.Div([
    dcc.Store(id='stored-news-data', storage_type='session'),  # Store for
news data and title

    # Input field for stock ticker and button
    html.Div([
        html.Label("Stock Ticker:"),
```

```
        dcc.Input(id='stock-ticker', type='text', placeholder='Enter stock
ticker', style={'width': '200px', 'display': 'inline-block'}),
        html.Button('Get News', id='get-news-btn', style={'margin-left':
'10px', 'vertical-align': 'middle'}),
    ], style={'text-align': 'left', 'width': '30%', 'padding': '20px'}),

    # Dynamic title for the table based on stock ticker
    html.Div(id='news-content', style={'text-align': 'center', 'padding':
'10px'}),
    html.Br(),

    # DataTable to display the news
    html.Div(id='table-container', children=[
        dash_table.DataTable(
            id='news-table',
            columns=[
                {"name": "#", "id": "number"},  # Column for row number
                {"name": "Title", "id": "title"},
                {"name": "Source", "id": "source"},
                {"name": "Date", "id": "date"},
                {"name": "Preview Text", "id": "preview_text"},
                {"name": "Link", "id": "link", "presentation": "markdown"},
            ],
            data=[],  # Initial empty data
            row_deletable=False,
            style_table={'width': '100%', 'margin': '20px auto'},
            style_cell={'text-align': 'left', 'whiteSpace': 'normal',
'height': 'auto'},
            style_header={'fontWeight': 'bold'},
            markdown_options={"html": True}  # Enable markdown for clickable
links
        )
    ], style={'text-align': 'center', 'padding': '20px'})
])

def fetch_stock_news(ticker):
    api_key = "cskj661r01qn1f3vmefgcskj661r01qn1f3vmeg0"  # Replace with
your actual API key

    # Set date range for the last 365 days
    today = datetime.date.today()
    one_year_ago = today - datetime.timedelta(days=365)

    # Build URL with date range
    url = f"https://finnhub.io/api/v1/company-
news?symbol={ticker}&from={one_year_ago}&to={today}&token={api_key}"
    response = requests.get(url)

    if response.status_code == 200:
```

```
        return response.json()
    else:
        print(f"Failed to retrieve the news. Status code:
{response.status_code}")
        return []

def register_callbacks(app):
    @app.callback(
        Output('news-table', 'data', allow_duplicate=True),
        Output('stored-news-data', 'data'),  # Store fetched news data and
title as a dictionary
        Output('news-content', 'children', allow_duplicate=True),  # Update
title with stock ticker
        [Input('get-news-btn', 'n_clicks')],
        [State('stock-ticker', 'value')],
        prevent_initial_call=True
    )
    def fetch_and_store_news(n_clicks, ticker):
        if n_clicks is None or not ticker:
            return [], {}, ""

        news_data = fetch_stock_news(ticker)

        if not news_data:
            return [], {}, f"No news found for {ticker.upper()}."

        # Add row number and format news data for the DataTable
        formatted_news = [
            {
                "number": index + 1,  # Row number
                "title": item['headline'],
                "source": item['source'],
                "date":
datetime.datetime.fromtimestamp(item['datetime']).strftime('%Y-%m-%d
%H:%M:%S'),
                "preview_text": item['summary'],
                "link": f"[Read more]({item['url']})"  # Markdown link for
the URL
            }
            for index, item in enumerate(news_data)
        ]

        # Set the title with the stock ticker
        title = html.H3(f"News for {ticker.upper()}", style={'fontWeight':
'bold'})

        # Store both the formatted news and title in session as a dictionary
        session_data = {
            "news": formatted_news,
```

```
            "title": f"News for {ticker.upper()}"
        }

        return formatted_news, session_data, title

    # Callback to update the DataTable and title from stored data when the
tab is accessed
    @app.callback(
        [Output('news-table', 'data'),
         Output('news-content', 'children')],
        Input('stored-news-data', 'data')
    )
    def update_news_table(stored_data):
        if isinstance(stored_data, dict):
            return stored_data.get('news', []),
html.H3(stored_data.get('title', ""), style={'fontWeight': 'bold'})
        return [], ""
```