

Dokumentacija koda

Uvod:

Da bi razumjeli moj kod i moju inspiraciju za napraviti ga, prvo moramo razumjeti osnove dionica i financija.

Financije u kratko

Svakoga dana događa se nešto nama nepoznato, velike kompanije poput Facebooka i Googla još prije mnogo godina počeli su prodavati dionice na njujorškoj burzi. Što to nama znači ? U suštini znači da mi obični ljudi možemo biti vlasnici maloga udjela neke velike kompanije. Od ponedjeljka do petka, od 9 do 16 sata(po američkom istočnom vremenu) investitori kupuju i prodaju dionice. To radi na vrlo jednostavnom principu. Kompanija podjeli svoju vrijednost na puno malih djelova(dionica) te im onda zada početnu cijenu. Ako izađe neka dobra vijest o kompaniji cijena će porasti i oni koji su kupili dionice po manjoj cijeni mogu ih sada prodati po većoj. Tako danas ukratko kompanije i investitori zarađuju novac. Investitori već godinama pokušavaju naći dugotrajni sistem predikcije cijene dionica kako bi zaradili novac. Tako je došla moja inspiracija, odlučio sam napraviti model pomoću strojnog učenja koji bi predvidio cijene dionica. Moj cilj i ideja jest da korisnik upiše ime kompanije za koje ga zanima cijena dionice u budućnosti. Ono što se dogodi sljedeće jest da izađe cijena do sada i predikcija u budućnosti.

Detaljan opis rada:

Kod sam podjelio u 3 djela, tako da sam na svakom radio zasebno i nakraju ih sve spojio u jedan veliki.

1.Dio-GUI

U ovom djelu isprogramirao sam GUI(to jest grafičko sučelje) u kojem korisnik upisuje ime dionice i klikče na gumb koji ga vodi na graf i predikciju. Koristio sam biblioteku tkinter.

```

from tkinter import *
import tkinter.font as tkFont
root = Tk()
font = tkFont.Font (family= "Helvetica", size=12, weight= "bold" )
font2 = tkFont.Font (family= "Helvetica", size=9, weight= "normal" )
root.title("Prediktor cijene dionica")
label1=Label(root, text= "Puno ime kompanije :",borderwidth=5, relief="groove",f
label2= Label(root, text = "Primjeri: Microsoft Corporation, Intel Corporation,
e = Entry(root, width=35,borderwidth=5, bg= "white",fg= "blue",)
e.grid(row=0, column=1, padx=10, pady=10)
a= ""
def klik ():
    global a
    a= e.get()
    root.destroy()

gumb= Button(root,text="Kliknite za predikciju", padx=100, pady= 25,command=klik
if __name__=="__klik__":
    klik()

label1.grid(row=0,column=0)
label2.grid(row=3,columnspan=3)
gumb.grid(row=2,sticky=N+S+E+W,columnspan=2)

root.mainloop()

```

Ln: 2 Col: 29

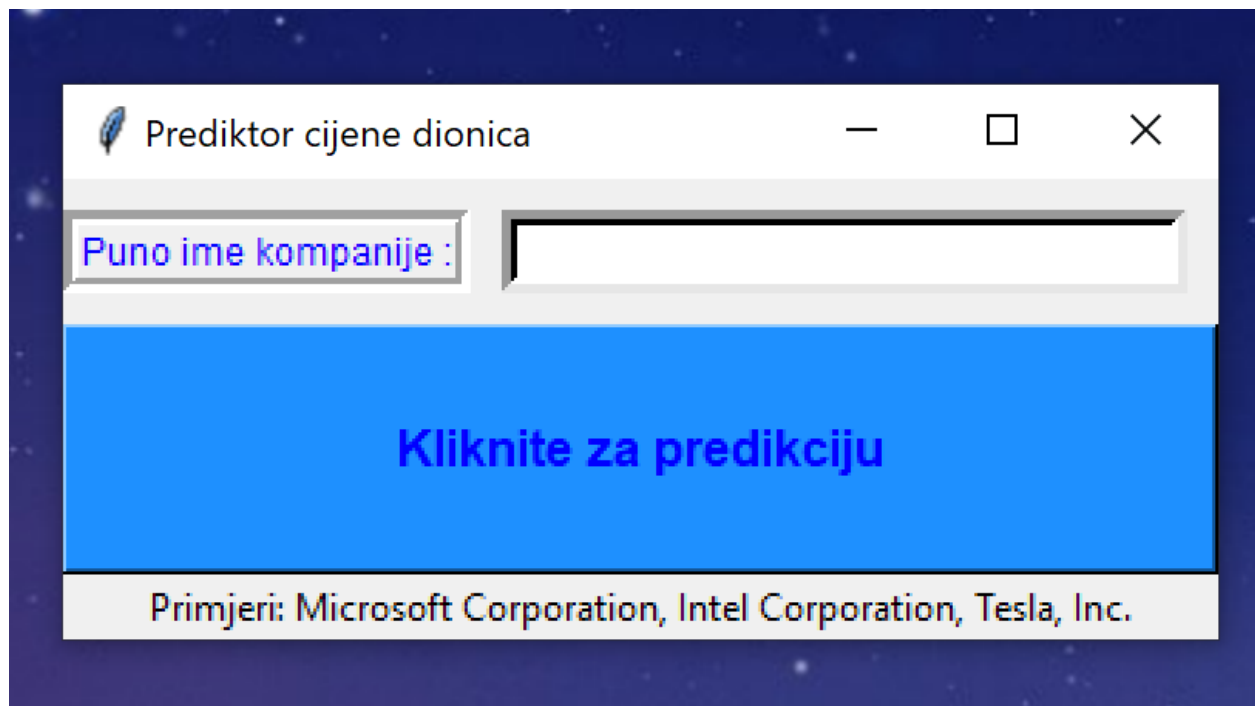
Pomoću `root = TK()` nisam morao cijelo vrijeme pisati `tk.argument` već samo `root` usvakoj zagradi. Kao i u CSS-u napravio konfiguraciju teksta(oblik,font,..) napočetku i samo je kasnije umetnuo u labele(sve djelove koji imaju tekst). `Label1` i `label2` predstavljaju dijelove na GUI ju na kojima će pisati tekst, dok `e(Entry)` predstavlja dio u kojemu će korisnik upisati informacije. `Klik` je samo dio(funkcija) u kojem se daje komanda što će se dogoditi kada korisnik klikne `gumb`. Ono što se dogodi jest da se sačuva input(`e.get`) i makne GUI(`root.destroy`). `Gumb` mora imati komandu(komanda je `klik`) i onda sam ga ja još oblikovao i obojao.

Ovaj dio na slici dolje je bitan jer on zapravo nalaže da je klik glavna funkcija pa onda se taj dio koda prvi izvršava i na taj način nemamo kasnije problema s varijablama koje nisu definirane (jer u kliku dajemo varijabli a string). Ovaj problem sam dugo rješavao jer kada bi pokrenuo kod a ne bi bio definiran jer se cijeli kod pokrenuo “istovremeno”.

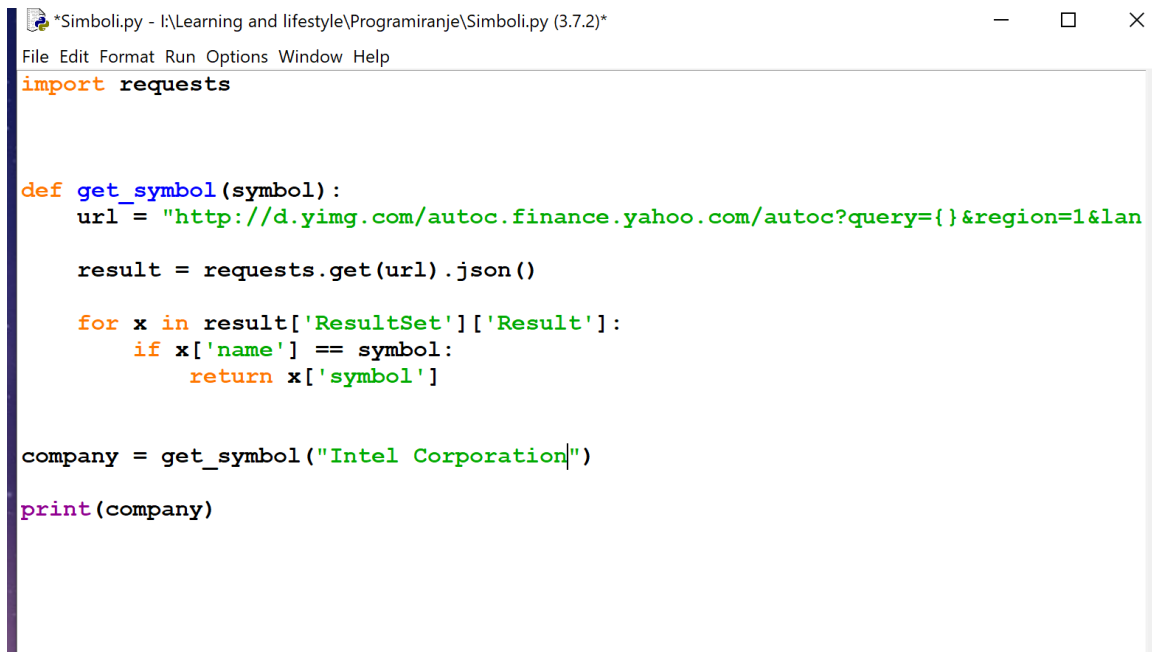
```
gumb= Button(root,text='  
if __name__=="__klik__"  
    klik()
```

```
label1.grid(row=0,column=  
label2.grid(row=3,column=
```

Svi ovi djelovi s gridom definiraju veličinu i koliko će prostora svaki segment uzimati na GUI – u(gumb,label...). Kada pokrenemo kod dogodi se sljedeće:



2.Dio-Simboli



```
*Simboli.py - I:\Learning and lifestyle\Programiranje\Simboli.py (3.7.2)*
File Edit Format Run Options Window Help

import requests

def get_symbol(symbol):
    url = "http://d.yimg.com/autoc.finance.yahoo.com/autoc?query={}&region=1&lan"

    result = requests.get(url).json()

    for x in result['ResultSet']['Result']:
        if x['name'] == symbol:
            return x['symbol']

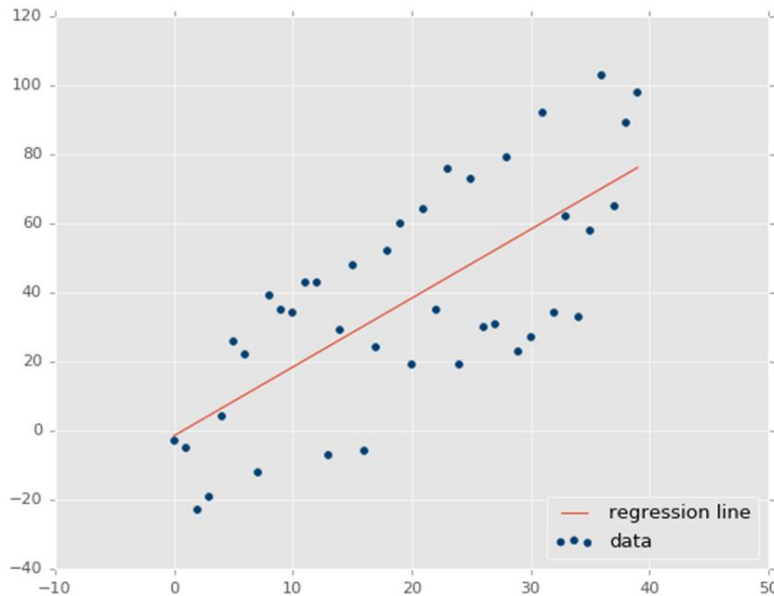
company = get_symbol("Intel Corporation")

print(company)
```

U ovom djelu korisnikov input se uspoređuje s listom kompanija na API -u (to jest na stranici). Kada se nađe isto ime kompanije „Kod“ uzima simbol te kompanije s te stranice i sačuva ga kao string. Dakle pomoću biblioteke requests smo se spojili na API(yahoo finance). Ova for petlja uspoređuje input s ostalim kompanija na toj stranici i kada se poistovjete returna njezin simbol. Kasnije se samo pozovemo na funkciju za neku kompaniju(ja sam ovdje stavio Intel) i ona nam da njezin simbol.

3.Dio-model

Ovaj dio je najkompleksniji jer ovdje se događa predikcija cijene i cijeloukupni proces izdavanja grafa. Da bi razumjeli 3. dio moramo prvo razumjeti regresiju.



Regresija je oblik takozvanog nadziranog učenja u kojem mi stroju pokažemo podatke i njihove ishode kako bi ono naučilo te s obzirom na to pretpostavilo kretanje u budućnosti. Ovakav algoritam može se koristiti na svakakvim poljima znanosti, no ni on nije u potpunosti točno jer uvijek postoji odstupanje od pravila.

Strojno učenje grana je umjetne inteligencije koja se bave oblikovanjem algoritama koji svoju učinkovitost poboljšavaju na temelju empirijskih podataka. Strojno učenje jedno je od danas najaktivnijih i najuzbudljivijih područja računarske znanosti, ponajviše zbog brojnih mogućnosti primjene koje se protežu od raspoznavanja uzoraka i dubinske analize podataka do robotike, računalnog vida, bioinformatike i računalne lingvistike. Kolegij obuhvaća dva osnovna pristupa strojnom učenju: nadzirano učenje (klasifikacija i regresija) i nenadzirano učenje (grupiranje i smanjenje dimenzionalnosti). U mom slučaju je problem što je model zaonovan na prošlim cijenama što nije nužno točan prikaz budućih cijena.

```

style.use('ggplot') # stil grafa.

df = web.DataReader('TSLA', 'yahoo')
df = df[['Open', 'High', 'Low', 'Close', 'Volume']] # Ovim djelom smo definirali promjenu u postotku. Podatke smo uzeli s yahoo financa te smo ih pohranili
df['HL_PCT'] = (df['High'] - df['Low']) / df['Close'] * 100.0
df['PCT_change'] = (df['Close'] - df['Open']) / df['Open'] * 100.0
print(df.tail())

df = df[['Close', 'HL_PCT', 'PCT_change', 'Volume']]
forecast_col = 'Close' # ovdje smanjujemo količinu podataka te nepotrebene podatke mičemo tako da su veliki negativan broj te ih onda st
df.fillna(value=-99999, inplace=True)
forecast_out = int(math.ceil(0.01 * len(df)))
df['label'] = df[forecast_col].shift(-forecast_out)

X = np.array(df.drop(['label'], 1)) # ovdje smo label koristili pretvorili u array.
X = preprocessing.scale(X)
X_lately = X[-forecast_out:]
X = X[:-forecast_out]

df.dropna(inplace=True)

y = np.array(df['label'])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2) # ovdje je kod kojim računalo trenira s prijašnjim podacima. Ovdje počinje koristiti regresijski
clf = LinearRegression(n_jobs=-1)
clf.fit(X_train, y_train)
confidence = clf.score(X_test, y_test)

forecast_set = clf.predict(X_lately)
df['Forecast'] = np.nan

last_date = df.iloc[-1].name
last_unix = last_date.timestamp()
one_day = 86400 # cijelim ovim djelom smo s obzirom na prijašnji dan procijenili sljedeći i tako dalje.
# tako smo dobili više dana u budućnosti koji su svi pretpostavljeni s obzirom na prijašnje.

next_unix = last_unix + one_day

for i in forecast_set:
    next_date = datetime.datetime.fromtimestamp(next_unix)
    next_unix += 86400
    df.loc[next_date] = [np.nan for _ in range(len(df.columns)-1)]+[i]

df['Close'].plot()
df['Forecast'].plot()
plt.legend(loc=4) # ovime smo namjestili graf.
plt.xlabel('Vrijeme')
plt.ylabel('Cijena')
plt.show()

```

U ovom kodu se pojavljuje riječ *df*. Dakle *dataframe(df)* je nešto kao tablica i pomoću njega možemo spakirati puno informacija na jedno mjesto.

```
style.use('ggplot') # stil grafa.
```

```
df = web.DataReader('TSLA', 'yahoo')
df = df[['Open', 'High', 'Low', 'Close', 'Volume']]
df['HL_PCT'] = (df['High'] - df['Low']) / df['Close'] * 100.0
df['PCT_change'] = (df['Close'] - df['Open']) / df['Open'] * 100.0
print(df.tail())
```

```
df = df[['Close', 'HL_PCT', 'PCT_change', 'Volume']]
forecast_col = 'Close' # ovdje smo
df.fillna(value=-99999, inplace=True)
forecast_out = int(math.ceil(0.01 * len(df)))
df['label'] = df[forecast_col].shift(-forecast_out)
```

```
X = np.array(df.drop(['label'], 1)) # ovdje smo
X = preprocessing.scale(X)
X_lately = X[-forecast_out:]
X = X[:-forecast_out]
```

U ovom djelu smo definirali promjenu u postotku. Podatke smo uzeli s yahoo financa te smo ih pohranili za koristiti.

U drugom djelu smanjujemo količinu podataka te nepotrebene podatke mičemo tako da su veliki negativan broj te ih onda stroj smatra kao nevažnim. To je važno jer određuje vrijeme koje je prihvatljivo za ovaj kod. Jer kada bi uzeli svaku sekundu cijene bilo bi nemoguće napraviti predikciju (barem ne s mojim računalom).

U trećem djelu smo samo podatke pretvorili u array kako bi ih lakše mogli koristiti za predikciju.

```

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2) # ovdje je kod
clf = LinearRegression(n_jobs=-1)
clf.fit(X_train, y_train)
confidence = clf.score(X_test, y_test)

forecast_set = clf.predict(X_lately)
df['Forecast'] = np.nan

last_date = df.iloc[-1].name
last_unix = last_date.timestamp()
one_day = 86400 # cijelim
# tako smo

next_unix = last_unix + one_day

for i in forecast_set:
    next_date = datetime.datetime.fromtimestamp(next_unix)
    next_unix += 86400
    df.loc[next_date] = [np.nan for _ in range(len(df.columns)-1)]+[i]

df['Close'].plot()
df['Forecast'].plot()
plt.legend(loc=4) # ovime smo namjestili graf.
plt.xlabel('Vrijeme')
plt.ylabel('Cijena')
plt.show()

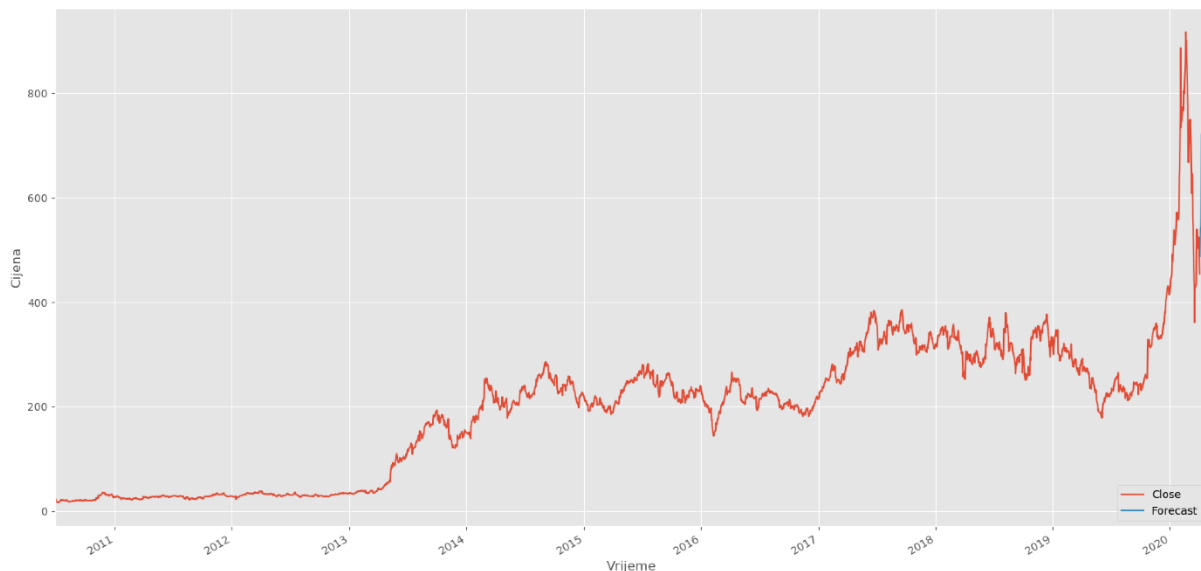
```

U prvom djelu model počinje trenirati s podacima iz prošlosti koju su mu dane u onom labelu(arrayu). Korisiti regresijski algoritam(uz pomoć biblioteke) i onda testira i traži povezanost. Nadalje stvaramo *df Forecast* koji će nam biti cijene u budućnosti.

U drugom i trećem djelu definiramo vrijeme i kako će predvidjeti cijenu. Dajemo danu vrijednost u sekundama i onda pomoću for petlje radimo definiramo koliko će vremena u budućnost predviđati.

U četvrtom djelu stavramo graf i dva djela(close(cijene do sada) i forecast(cijene u budućnosti)).

Rezultat tog koda je sljedeći (ovdje je primjer cijena dionice Tesle):



Crveni dio prikazuje cijenu do sada, a plavi pokazuje cijenu u budućnosti.

Tehničke informacije:

Zadatak sam ispogramirao u Pythonu. Ovdje je lista svih biblioteka koje sam koristio.

Objasniti ću njihove mogućnosti:

pandas - tablično slaganje podataka (data frame).

datetime - vremenski periodi, u našem slučaju možemo točno odrediti period od kojega gledamo cijenu dionice.

matplotlib.pyplot - grafovi.

numpy - modul koji nam je pomogao okom matmatike i pohranjivanja podataka u obliku arraya.

sklearn - najpoznatiji modul za strojno učenje, on nam je dopustio napraviti model na osnovu učenja.

Tkinter - biblioteka za GUI.