Liceul Teoretic Pâncota

LUCRARE DE ATESTAT

DISCIPLINA INFORMATICĂ

Candidat,

Doca Andrei

Coordonator,

Prof. Moş Alina

2023

Investiție automată la bursă

(aplicație în C++ și Python)

**Cuprins**

1. Argument …………………………………………………………… 4

2. Partea teoretică ……………………………………………………… 5

3. Descrierea aplicaţiei ………………………………………………… 9

4. Concluzii …………………………………………………………… 17

5. Webografie ………………………………………………………… 18

**Argument**

Am ales această temă, deoarece, în urma vizionării unei reclame pe Youtube despre eToro, o platformă de investiții, am decis să mă educ din punct de vedere financiar. Am căutat să învăț cât mai multe despre investiții și am îmbinat elemente de matematică, programare și economie pentru a crea atestatul.

Am implementat în Python cele mai eficiente și cele mai simple strategii de investit: Simple Moving Average și Relative Strength Indicator + Bollinger Bands. Pentru datele istorice ale companiei Apple, aceste strategii au adus profit, însă, se spune că o strategie nu poate fi considerată complet eficientă dacă nu e testată și pe date în timp real. Totodată, acest mod de a investi (Close Day Trading) este mult mai greu de înfăptuit pentru un om față de un robot.

Domeniul investițiilor automate este foarte vast și, ca atare, dificil. Este necesar un pachet de cunoștințe în matematică și în programare. Cunoscând limbajul de programare C++, am reușit să învăț foarte ușor librăriile NumPy, Pandas și Matplotlib care, deși sunt exclusive pentru Python, ele au fost scrise în C++. Anumite secvențe de cod în Python au un echivalent în C++.

Interfața Jupyter Notebook a fost foarte utilă pentru scrierea și depanarea programului. Ea poate chiar să aloce RAM și CPU în Cloud pentru viteză și putere mai mare de procesare. Cu ajutorul editorului de text Visual Studio Code și al comenzilor rapide din VIM, am avut parte de o experiență plăcută în scrierea programului.

**Partea teoretică**

Visual Studio Code este un editor de cod extensibil dezvoltat de Microsoft pentru Windows, Linux și macOS. Funcțiile includ suport pentru depanare, evidențierea sintaxei, completarea codului, etc. Utilizatorii pot modifica tema, comenzile rapide de la tastatură, preferințele și pot instala extensii care adaugă funcții suplimentare.

Python este un limbaj de programare dinamic, multifuncțional folosit de exemplu de către companii ca Google sau Yahoo! Pentru programarea aplicațiilor web, însă există și o serie de aplicații științifice sau de divertisment. Popularitatea în creștere, dar și puterea limbajului de programare Python au dus la adoptarea sa ca limbaj principal de dezvoltare de către programatori specializați și chiar și la predarea limbajului în unele medii universitare.

Python pune accentul pe curățenia și simplitatea codului, iar sintaxa sa le permite dezvoltatorilor să exprime unele idei programatice într-o manieră mai clară și mai concisă decât în alte limbaje de programare ca C/C++.

Python este un limbaj de programare interpretat, nu compilat. Tipul variabilelor nu este fixat la momentul declarării acestora (de fapt, declararea variabilelor nu este necesară ca în C/C++), ci este determinat de interpretator după conținutul lor sau după operațiile efectuate. Într-un asemenea caz se vorbește de tipizare dinamică.

Prin urmare, codul produs este mai clar și mai intuitiv:

a = 10 *# a va fi considerat un număr întreg (integer)*

b = 11.5 *# b va fi considerat un număr rațional (double sau float)*

c = „un rând de text” *# c va fi considerat un șir (string)*

Există totuși unele excepții logice, cum ar fi în cazul numerelor întregi și celor raționale (float), între care sunt permise operații ca adunarea sau împărțirea, tipul rezultatului ținând cont de natura operației:

d = 10 + 11.5 *# rezultatul va fi un număr rațional, 21.5*

e = „o brioșă” + 4 *# această comandă va genera o eroare.*

Alt concept important în Python este cel al tipurilor mutabile și nemutabile. Așa cum implică numele, datele cu un tip mutabil pot fi modificate după inițializare, în vreme ce la date nemutabile lucrul acesta este imposibil. Într-o listă se pot modifica sau adăuga elemente, în timp ce într-un șir de caractere este imposibil:

lista\_mea = [„măr”, „pară”]

lista\_mea.append(„strugure”) *# adaugă la listă*

print(lista\_mea[2]) *# afișează: strugure*

șirul\_meu = „varză”

șirul\_meu[0] = „b” *# schimbă cuvântul în „barză” -> Eroare*

Python oferă tipuri tradiționale de date, cum ar fi numărul întreg (integer, int) sau cel rațional (float), dar introduce totodată și concepte noi. De exemplu, un grad mare de flexibilitate îl oferă listele (tablourile) în Python. Acestea nu sunt statice, ci pot conține orice tipuri de date (în cadrul aceleiași liste) și pot fi modificate pe loc adăugând și eliminând elemente fără a declara sau utiliza funcții de manipulare a memoriei:

lista\_mea = [10, „șir de caractere”, variabilă, [„altă”, „listă”], chiar\_și\_o\_funcție]

Alte structuri de date sunt tuplurile/perechile (tuples) și dicționarele (dictionaries, mappings). Tuplurile sunt liste care au un număr prestabilit de elemente, și nu pot fi modificate parțial. Tuplurile pot fi utilizate în cazuri în care este nevoie de o anumită structură de date specializată, de exemplu coordonate în spațiul cartezian. Dicționarele sunt liste neordonate în care fiecare element are asociat o cheie, care poate fi număr sau șir de caractere. Dicționarele au foarte multe aplicații, inclusiv crearea structurilor de tip hash-tables.

Din punctul de vedere al sintaxei, Python are un număr de construcții și cuvinte cheie cunoscute oricărui programator, dar prezintă și un concept unic: nivelul de indentare are semnificație sintactică. Blocurile de cod sunt delimitate prin simplă indentare. În C/C++ astfel de blocuri sunt deseori desemnate prin acolade, {<cod>}, dar în Python nu este nevoie de astfel de construcții. Nivelele de indentare îndeplinesc această funcție.

*Int* main(){

    for(*int* i = 0; i < 10; i++){

        cout<< i << „\n”;

}

    return 0;

}

*def* main():

    for i in range(0, 10):

        print(i)

if \_\_name\_\_ == „\_\_main\_\_”:

    main()

C/C++ este un limbaj de programare standardizat. Este implementat pe majoritatea platformelor de calcul existente azi, și este cel mai popular limbaj de programare pentru scrierea de software de sistem. Este apreciat pentru eficiența codului obiect generat de compilatoarele C/C++, și pentru portabilitatea sa.

C/C++ este prezentat uneori ca „asamblor portabil”, făcându-se astfel diferențele principale față de limbajele de asamblare: codul unui program C poate fi compilat și rulat pe aproape orice tip de mașină (calculator), asemănător altor limbaje de programare, în timp ce limbajele de asamblare sunt specifice unui anumit model de mașină.

C/C++ a fost creat având drept scop important de a face ca programele mari să poată fi scrise mai ușor și cu mai puține erori în paradigma programării procedurale și are următoarele caracteristici importante:

* Este un limbaj de bază simplu, care include funcțiile matematice sau cele de manipulare ale fișierelor
* Este focalizat pe paradigma programării procedurale, care facilitează programarea într-un mod structurat
* Utilizează un set simplu de tipuri de date ce împiedică multe operații neintenționate
* Folosește un limbaj preprocesor, preprocesorul C, pentru sarcini cum ar fi definirea de macrouri și includerea mai multor fișiere sursă
* Permite accesarea la nivel scăzut a memoriei calculatorului prin utilizarea pointerilor
* Permite folosirea parametrilor, care sunt comunicați funcțiilor prin valoare și nu prin referință
* Pointeri la funcții, ce permit forme rudimentare de închidere (engleză closure) și polimorfism
* Declararea variabilelor
* Structuri de date sau tipuri de date agregate, definite de utilizator prin (struct), ce permit ca date înrudite să fie combinate și manipulate ca un întreg

C/C++ utilizează foarte mult pointerii, un tip de referință foarte simplu, care păstrează adresa unui obiect din memorie. Adresa poate fi manipulată cu ajutorul aritmeticii pointerilor. În momentul compilării, un pointer este un tip de dată complex, ce reprezintă atât adresa de memorie cât și tipul de dată. Acest lucru permite expresiilor ce utilizează pointeri să fie evaluate după tipul de dată. Pointerii au mai multe utilizări în C/C++. De exemplu, șirurile de caractere sunt adesea reprezentate printr-un pointer la un vector de caractere. Alocarea dinamică a memoriei este realizată tot cu ajutorul pointerilor.

Char \* p, s[31] = „pbinfo”;

Variabila p este un pointer la char, adică o variabilă a cărei valoare este adresa unei date de tip char. Ea nu a fost inițializată, iar valoarea ei este o adresă aleatorie.

S este un șir de caractere, dar practic este tot un pointer. Valoarea sa este adresa primului element din șir, adică adresa lui s[0]. Observăm că de fapt, variabilele p și s sunt de același tip, pointer la char. Diferența dintre cele două variabile este că s memorează o adresa de memorie unde începe un șir de caractere (la acea adresă există o dată de tip char ) în timp ce p memorează o adresă aleatorie.

În C/C++ nu există liste care să conțină mai multe tipuri de date ca în Python, iar tipul unei date odată declarat, nu mai poate fi modificat pe toată durata execuției programului.

*Int* v[10] = {0}; // un vector este o listă cu același tip de dată

*int* a = 4; // a este de tip număr întreg

    a = „sir\_de\_caractere”; // Transformare din număr în șir -> Eroare

Jupyter Notebook este o platforma web interactivă open-source care permite crearea, rularea și distribuirea documentelor de tip notebook. Un astfel de document integrează secvențe de cod, vizualizări, descrieri în mod text, ecuații, rezultate, resurse media, și reprezintă un mediu puternic atât pentru dezvoltarea codului, cât și pentru comunicarea rezultatelor.

Jupyter Notebook poate fi utilizat cu succes în diverse activități din ingineria și știința datelor (Data Engineering and Science), precum: curățarea și transformarea datelor, analiza exploratorie a datelor, vizualizarea datelor, modelarea statistică, învățarea automată (machine learning), învățarea profundă (deep learning).

**Descrierea aplicației**

Investiție la bursa istorică (Backtesting) folosind SMA și RSI + Bollinger Bands

Includerea (importarea) de librării necesare executării programului

import numpy as np

#Pentru structuri de date de tip tablou si operatii matematice complexe

import pandas as pd

#Pentru manipularea datelor organizate sub forma de tabel precum datele despre o companie

import matplotlib.pyplot as plt

#Pentru reprezentarea grafica a informatiilor

import yfinance as yf

#Obtinerea datelor despre actiuni folosind Yahoo Finance

Downloadarea datelor despre o companie

Ca exemplu, am folosit datele despre compania AAPL (Apple Inc.)

n = 10 # Pe cati ani se face investitia

n = n \* 365 # Transformat in zile

start\_date = pd.to\_datetime(„today”) – pd.Timedelta(n, *unit*=’D’) # Din prezent scad n ani

start\_date = start\_date.strftime(„%Y-%d-%m”) # Formatul necesar pentru yfinance : an/zi/luna

end\_date = pd.to\_datetime(„today”).strftime(„%Y-%d-%m”)

#Nu se vor lua in calcul ultimele 2-3 zile pentru ca nu am Yahoo premium

data = yf.download(‚AAPL’,*start* = start\_date,*end* = end\_date)

#Functia download primeste mai multi parametri printre care simbolul companiei si datele de inceput

#si sfarsit pentru care se realizeaza cautarea informatiilor

data

#Afisarea variabilei

Implementarea și reprezentarea grafică a strategiei Moving Average Crossover

Funcție pentru calcularea SMA (medie aritmetică mobilă)

Formula pentru calcularea mediei aritmetice mobile este următoarea:

SMA =

Unde **k** reprezintă perioada aleasă, **i** reprezintă ziua curentă, iar prețul de vânzare / cumpărare al unei acțiuni (p=).

*Def* SMA(*data*, *period*, *column* = ‚Close’):

    return *data*[*column*].rolling(*period*).mean()

Adăugarea de noi coloane în structura de date și ștergerea celor utilizate

data[‚Short\_SMA’] = SMA(data,50)

#Short SMA, adica pe o perioada mai scurta de zile

data[‚Long\_SMA’] = SMA(data,100)

#Long SMA, adica pe o perioada mai lunga de zile

data.drop(*columns* = [„Open”,”High”,”Low”,”Adj Close”, „Volume”], *inplace* = True)

data

Strategia propriu zisă

# Descriere:

# Cand media aritmetica mobila pe o perioada mai scurta este sub media aritmetica mobila pe o perioada mai lunga atunci se pot cumpara actiuni, altfel se pot vinde actiuni.

*Def* strategy(*data*):

    #Listele sunt initial vide.

    Buy = []

    sell = []

    sum = []

    s = 1000 # investitia initiala

    nr\_actiuni = 0 # numar de actiuni per companie

    for i in range(0, len(*data*)):

        if *data*[‚Short\_SMA’][i] < *data*[‚Long\_SMA’][i] and s > *data*[‚Close’][i]:

            # semnal de cumparare

            s = s – *data*[‚Close’][i]

            nr\_actiuni = nr\_actiuni + 1

            buy.append(*data*[‚Close’][i])

            sell.append(np.nan)

        elif *data*[‚Short\_SMA’][i] > *data*[‚Long\_SMA’][i] and nr\_actiuni != 0:

            # semnal de vanzare

            s = s + *data*[‚Close’][i]

            nr\_actiuni = nr\_actiuni – 1

            buy.append(np.nan)

            sell.append(*data*[‚Close’][i])

        else:

            buy.append(np.nan)

            sell.append(np.nan)

        sum.append(s + *data*[‚Close’][i] \* nr\_actiuni)

    return (buy, sell, sum)

Adăugarea de informații în structura de date pentru a putea fi reprezentate grafic

S-au adăugat atât prețul de cumpărare și de vânzare, cât și suma pe parcursul investițiilor și s-au afișat pe ecran.

Strat = strategy(data)

data[‚Buy’] = strat[0]

data[‚Sell’] = strat[1]

data[‚Sum’] = strat[2]

data

Reprezentarea grafică

plt.figure(*figsize* = (16, 8))

plt.plot(data[‚Close’],*alpha* = 0.5,*label* = ‚Close’)

plt.plot(data[‚Short\_SMA’],*alpha* = 0.5,*label* = ‚Short\_SMA’)

plt.plot(data[‚Long\_SMA’],*alpha* = 0.5,*label* = ‚Long\_SMA’)

plt.scatter(data.index,data[‚Buy’],*color* = ‚green’,*label* = ‚Buy signal’, *marker* = ‚^’,*alpha* = 1)

plt.scatter(data.index,data[‚Sell’],*color* = ‚red’,*label* = ‚Sell signal’,*marker* = ‚v’,*alpha* = 1)

plt.title(„Semnalele de vanzare si cumparare”,*fontsize* = 18)

plt.xlabel(„Data curenta”,*fontsize* = 18)

plt.ylabel(„Pret actiune in USD ($)”,*fontsize* = 18)

plt.legend(*loc* = „upper left”,*fontsize* = 12)

plt.show()

Afișarea rezultatelor strategiei SMA Crossover

plt.figure(*figsize* = (16, 8))

plt.plot(data[‚Sum’],*alpha* = 0.5, *label* = ‚Sum over time’)

plt.title(„Suma dupa investitii”,*fontsize* = 18)

plt.xlabel(„Data curenta”,*fontsize* = 18)

plt.ylabel(„Suma in USD ($)”,*fontsize* = 18)

plt.legend(*loc* = „upper left”,*fontsize* = 12)

plt.show()

maxim\_sma = data[‚Sum’][0]

minim\_sma = data[‚Sum’][0]

for i in range(0,len(data)):

    if(maxim\_sma > data[‚Sum’][i]):

        maxim\_sma = data[‚Sum’][i]

    if(minim\_sma < data[‚Sum’][i]):

        minim\_sma = data[‚Sum’][i]

total\_sma = data[‚Sum’][len(data)-1]

#Afisarea rezultatelor cu doua zecimale

#Aceste sume includ si valorile actiunilor detinute

print(*f*”Suma maxima pe parcursul investitiilor: {format(minim\_sma,’.2f’)} $”)

print(*f*”Suma minima pe parcursul investitiilor: {format(maxim\_sma,’.2f’)} $”)

print(*f*”Suma totala dupa toate investitiile este: {format(total\_sma,’.2f’)} $”)

Implementarea și reprezentarea grafică a strategiei RSI + BB

Funcție pentru calcularea Bollinger Bands

*def* bollinger\_bands(*data*, *period*):

    rolling\_mean = *data*[‚Close’].rolling(*period*).mean() # SMA

    rolling\_std = *data*[‚Close’].rolling(*period*).std() # Std. Deviation

*data*[‚UpperBand’] = rolling\_mean + (2 \* rolling\_std)

*data*[‚LowerBand’] = rolling\_mean – (2 \* rolling\_std)

    return *data*

Funcție pentru calcularea RSI (Relative Strength Index)

Formula pentru calcularea indicelui de putere relativă este următoarea:

RS =

RSI =100-

Unde **avg\_gain și avg\_loss** reprezintă câștigurile și pierderile medii, **RS** este raportul dintre avg\_gain și avg\_loss, iar indicele de putere relativă.

*Def* RSI(*data*, *period*):

    delta = *data*[‚Close’].diff()

    gain = delta.where(delta > 0, 0)

    loss = delta.where(delta < 0, 0)

    avg\_gain = gain.rolling(*period*).mean()

    avg\_loss = loss.rolling(*period*).mean()

    RS = avg\_gain / avg\_loss

    RSI = 100 – (100 / (1 + RS))

*data*[‚RSI’] = RSI

*data*[‚Overbought’] = 70

*data*[‚Oversold’] = 30

    return *data*

Apelul funcțiilor pentru a crea noi coloane în structura de date

data = bollinger\_bands(data, 30)

data = RSI(data, 13)

Strategia propriu zisă

# Descriere:

# Cand RSI este Oversold si ne aflam sub LowerBand se pot cumpara actiuni, altfel daca RSI este Overbought si ne aflam peste UpperBand se pot vinde actiuni

*def* RSIBBstrategy(*data*):

    #Listele sunt initial vide

    buy = []

    sell = []

    sum = []

    s = 1000 # investitia initiala

    nr\_actiuni = 0 # numar de actiuni per companie

    for i in range(0,len(*data*)):

        if *data*[‚Close’][i] < *data*[‚LowerBand’][i] and *data*[‚RSI’][i] < *data*[‚Oversold’][i] and s > *data*[‚Close’][i]:

            #semnal de cumparare

            s = s – *data*[‚Close’][i]

            nr\_actiuni = nr\_actiuni + 1

            buy.append(*data*[‚Close’][i])

            sell.append(np.nan)

        elif *data*[‚Close’][i] > *data*[‚UpperBand’][i] and *data*[‚RSI’][i] > *data*[‚Overbought’][i] and nr\_actiuni != 0:

            #semnal de vanzare

            s = s + *data*[‚Close’][i]

            nr\_actiuni = nr\_actiuni – 1

            buy.append(np.nan)

            sell.append(*data*[‚Close’][i])

        else:

            buy.append(np.nan)

            sell.append(np.nan)

        sum.append(s + *data*[‚Close’][i] \* nr\_actiuni)

    return (buy, sell, sum)

Adăugarea de informații în structura de date pentru a putea fi reprezentate grafic

S-au adăugat atât prețul de cumpărare și de vânzare, cât și suma pe parcursul investițiilor și s-au afișat pe ecran.

Strat = RSIBBstrategy(data)

data[‚RSIBB\_Buy’] = strat[0]

data[‚RSIBB\_Sell’] = strat[1]

data[‚RSIBB\_Sum’] = strat[2]

data

Reprezentarea grafică

plt.figure(*figsize* = (16, 8))

plt.plot(data[‚Close’],*alpha* = 0.5,*label* = ‚Close’,*color* = ‚blue’)

plt.plot(data[‚LowerBand’],*alpha* = 0.5,*label* = ‚Lower Band’,*color* = ‚purple’)

plt.plot(data[‚UpperBand’],*alpha* = 0.5,*label* = ‚Upper Band’,*color* = ‚orange’)

plt.fill\_between(data.index, data[‚UpperBand’],data[‚LowerBand’],*color* = ‚grey’)

plt.scatter(data.index,data[‚RSIBB\_Buy’],*color* = ‚green’,*label* = ‚Buy signal’, *marker* = ‚^’,*alpha* = 1)

plt.scatter(data.index,data[‚RSIBB\_Sell’],*color* = ‚red’,*label* = ‚Sell signal’,*marker* = ‚v’,*alpha* = 1)

plt.title(„Semnalele de vanzare si cumparare”,*fontsize* = 18)

plt.xlabel(„Data curenta”,*fontsize* = 18)

plt.ylabel(„Pret actiune in USD ($)”, *fontsize* = 18)

plt.legend(*loc* = „upper left”,*fontsize* = 12)

plt.show()

Afișarea rezultatelor strategiei RSI + BB

plt.figure(*figsize* = (16, 8))

plt.plot(data[‚RSIBB\_Sum’],*alpha* = 0.5, *label* = ‚Sum over time’)

plt.title(„Suma dupa investitii”,*fontsize* = 18)

plt.xlabel(„Data curenta”,*fontsize* = 18)

plt.ylabel(„Suma in USD ($)”,*fontsize* = 18)

plt.legend(*loc* = „upper left”,*fontsize* = 12)

plt.show()

maxim\_rsibb = data[‚RSIBB\_Sum’][0]

minim\_rsibb = data[‚RSIBB\_Sum’][0]

for i in range(0,len(data)):

    if(maxim\_rsibb > data[‚RSIBB\_Sum’][i]):

        maxim\_rsibb = data[‚RSIBB\_Sum’][i]

    if(minim\_rsibb < data[‚RSIBB\_Sum’][i]):

        minim\_rsibb = data[‚RSIBB\_Sum’][i]

total\_rsibb = data[‚RSIBB\_Sum’][len(data)-1]

#Afisarea rezultatelor cu doua zecimale

#Aceste sume includ si valorile actiunilor detinute

print(*f*”Suma maxima pe parcursul investitiilor: {format(minim\_rsibb,’.2f’)} $”)

print(*f*”Suma minima pe parcursul investitiilor: {format(maxim\_rsibb,’.2f’)} $”)

print(*f*”Suma totala dupa toate investitiile este: {format(total\_rsibb,’.2f’)} $”)

**Concluzii**

Datorită acestui proiect, am dobândit cunoștințe noi în domeniul matematicii, al programării și al investițiilor. Am ales acest proiect ca urmare a unei curiozități și am reușit să înțeleg concepte care în trecut păreau foarte grele. Nu am știut strategiile de investiții, dar cu pasiune și cu răbdare am descoperit mai multe decât m-aș fi gândit.

În viitor, îmi doresc să pot folosi algoritmul pe date în timp real, adică să investesc bani reali, să îl pot depune în Cloud, adică să pot închide calculatorul și algoritmul să funcționeze în continuare, să implementez o încasare a dividendelor, comision pe tranzacție, abilitatea de a compune un portofoliu de investiții după bunul plac și alte strategii mai eficiente.

**Webografie**

<https://www.python.org/>

<https://numpy.org/>

<https://pypi.org/project/yfinance/>

<https://pandas.pydata.org/>

<https://matplotlib.org/>

<https://www.youtube.com/@ComputerSciencecompsci112358>

<https://www.investopedia.com/>

<https://www.geeksforgeeks.org/>