**A close up of a logo

Description automatically generatedSUPERVISION CERTIFICATE**

A PROJECT REPORT SUBMITTED TO CENTRAL BOARD OF SECONDARY EDUCATION FOR THE PARTIAL FULFILLMENT OF INFORMATICS PRACTICES PROJECT WORK.

DEPARTMENT OF COMPUTER SCIENCE

ARMY PUBLIC SCHOOL, AMRITSAR

PUNJAB-143107

(2024-2025)

Kanithi Karthik

Tarun

This Is to Certify That Work Presented in The Project Entitled "StockSIM" Submitted for The Partial Fulfilment of CBSE INFORMATICS PRACTICES Exam To The CBSE New Delhi Is A Bona fide Project Work Carried Out By KARTHIK, TARUN Students of Class 12, Army Public School, Amritsar, Punjab Under My Guidance And Supervision.

MR. IQBAL SINGH

PGT COMPUTER SCIENCE

ARMY PUBLIC SCHOOL AMRITSAR

**ACKNOWLEDGEMENT**

I Take This Opportunity to Express My Deep Sense of Gratitude to Mr. Iqbal Singh, PGT (Computer Science), Army Public School, Amritsar for His Guidance, Advice and Support from The Beginning Of My Project Work To The Completion Of The Same. His Keen Interest in MY Project Work, Devotion of Time, And Inspiration During This Project Tenure Has Made This Work to Come in This Form.

I Also Pay Gratitude Towards Our Principal Mrs. Rajdeep Jain, Army Public School Amritsar Permitting Me to Avail Necessary Facilities of The Lab as Well as School for My Work. The Support Received from Friends During My Project Work Cannot Be Acknowledged in Words.

I Express My Deepest Affection, Appreciation and Thank to My Parents, Who Encouraged and Supported me In This Project. All Of Them Have Made Immense Contribution to My Studies and Life by Their Support, Guidance, Understanding and Sacrifices.

…………………………………

Army Public School Amritsar

***PROJECT SYNOPSIS***

**PROJECT DEFINITION**

To Design a project to allow user to simulate trading in the stock market. The User can view price history of any Indian stock, simulate buying and selling stocks.

**TEAM MEMBERS:**

* Kanithi Karthik
* Tarun

**REASON FOR CHOOSING THE TOPIC:**

It provides a risk-free environment for students and novices to learn how the stock market works, understand trading strategies, and develop financial literacy without the risk of losing real money.

**OBJECTIVE:**

* To let user simulate trading Indian stocks over historical data.
* To graphically show the financial data of stocks.
* To graphically show the historical prices of stocks.
* To let user have a portfolio.
* To let user be able to see their transaction history and profit/loss for each transaction.

**HARDWARE REQUIREMENT:**

* A computer/laptop
* Operating System – Windows 10 or above
* 4 GB RAM
* 512 MB of free disk space

**LIMITATIONS:**

* It is not a web-based project
* UI can be more responsive
* More functionality can be added as per requirements

**REFERENCES:**

* Online Python Documentation
* Textbook – Class XI and XII – INFORMATICS PRACTICES as per NCERT
* Stackoverflow
* GitHub

**TreeMap of Project:**

.

|-- main.py

`-- bin/

|-- UI.py

`-- plot/

|-- data.py

`-- data/

|-- datelist.csv

|-- ledger.csv

|-- tickers.csv

|-- userdata.csv

|-- <tradable\_data>.csv

`-- images/

|-- add.png

|-- home.png

`-- pf.png

**main.py:**

.

|-- main.py 🡨- You are here

`-- bin/

|-- UI.py

`-- plot/

|-- data.py

`-- data/

|-- datelist.csv

|-- ledger.csv

|-- tickers.csv

|-- userdata.csv

|-- <tradable\_data>.csv

`-- images/

|-- add.png

|-- home.png

`-- pf.png

**#****DRIVER CODE**

from bin.UI import \*

if \_\_name\_\_ == "\_\_main\_\_":

    app = UI()

    app.protocol("WM\_DELETE\_WINDOW", save)

    app.mainloop()

**UI.py:**

**#IMPORTING LIBRARIES, INITIALISING GLOBAL VARIABLES AND EXIT FUNCTION**

from bin.plot.data import \*

import mplfinance as mpf

from functools import partial

import customtkinter as ctk

from PIL import Image

import tkinter.messagebox as mb

import matplotlib.pyplot as plt

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

from datetime import datetime, timedelta

import sys

datelist = pd.read\_csv("bin/plot/data/datelist.csv",header=None, parse\_dates=[0])

datelist[0] = pd.to\_datetime(datelist[0], format="%d/%m/%y").dt.date

datelist = datelist[0].tolist()

.

|-- main.py

`-- bin/

|-- UI.py 🡨- You are here

`-- plot/

|-- data.py

`-- data/

|-- datelist.csv

|-- ledger.csv

|-- tickers.csv

|-- userdata.csv

|-- <tradable\_data>.csv

`-- images/

|-- add.png

|-- home.png

`-- pf.png

xledger = ledger("bin/plot/data/ledger.csv")

TRDX = None

xcode = None

ddays = 21

sdate = datetime(2000, 1, 3).date()

edate = None

itertime = 5000

bw = 110

tw = (bw\*2)+48

vw = 1180-2\*tw

rfw = tw+40

usr = None

pwd = None

liq = None

play = True

tasv = 0

tval = None

loopid = None

images = {"home": ctk.CTkImage(dark\_image=Image.open("bin/plot/data/images/home.png"), size=(25, 25)),

"pf": ctk.CTkImage(dark\_image=Image.open("bin/plot/data/images/pf.png"), size=(25, 25)),

"add": ctk.CTkImage(dark\_image=Image.open("bin/plot/data/images/add.png"), size=(25, 25))}

**# defining an exit function**

def save():

    if usr is not None:

        xledger.save\_to\_csv()

        with open("bin/plot/data/userdata.csv", "a") as file:

            file.write("{},{},{},{},{},{},{}\n".format(

                usr, pwd, sdate, edate, ddays, itertime, liq))

        sys.exit()

**# creating a class for candlestick chart which supports on-demand update:**

class customcandlestick(ctk.CTkFrame):

    def \_\_init\_\_(self, parent):

        super().\_\_init\_\_(parent)

        global xcode, ddays, edate, sdate, bw, tw

        self.ndays = ddays

        self.trd = tradable(xcode, sdate, ddays)

        self.configure(width=1280, height=720)

        sdate = self.trd.index[0].date()

        edate = self.trd.index[-1].date()

        self.fig, self.ax = mpf.plot(self.trd, title=TRDX.loc[xcode, "name"], type="candle",

datetime\_format='%d/%m/%y',style=binance\_dark, volume=True,ylabel="Price", ylabel\_lower="Shares Traded", returnfig=True, show\_nontrading=False, figscale=0.8, panel\_ratios=(3, 1), tight\_layout=False)

        self.canvas = FigureCanvasTkAgg(self.fig, master=self)

        self.canvas.draw()

        self.canvas.get\_tk\_widget().grid(row=0, column=0)

    def upd(self, ndate=sdate, ndays=ddays):

        global xcode, ddays, edate, sdate, bw, tw

        plt.close()

        del self.fig, self.ax, self.canvas, self.trd

        self.code = xcode

        self.trd = tradable(self.code, ndate, ndays, False)

        sdate = self.trd.index[0].date()

        edate = self.trd.index[-1].date()

        self.fig, self.ax = mpf.plot(self.trd, title=TRDX.loc[xcode, "name"], type="candle",

datetime\_format='%d/%m/%y',style=binance\_dark, volume=True,ylabel="Price", ylabel\_lower="Shares Traded", returnfig=True, show\_nontrading=False, figscale=0.8, panel\_ratios=(3, 1), tight\_layout=False)

        self.canvas = FigureCanvasTkAgg(self.fig, master=self)

        self.canvas.draw()

        self.canvas.get\_tk\_widget().grid(row=0, column=0)

    def show\_history(self, UI):

        global play

        def on\_exit():

            global play

            play = True

            UI.movedays(date=sdate)

            UI.histwin.destroy()

        play = False

        UI.histwin = ctk.CTkToplevel(UI)

        UI.histwin.title("History")

        UI.histwin.geometry("1280x720")

        data = loadhistory(xcode, edate)

        \_fig, ax = mpf.plot(data, type="line", title=TRDX.loc[xcode, "name"], datetime\_format='%d/%m/%y',

style=binance\_dark,volume=True,ylabel="Price", ylabel\_lower="Shares Traded", returnfig=True, show\_nontrading=False, figscale=2, panel\_ratios=(3, 1), tight\_layout=False)

        canvas = FigureCanvasTkAgg(\_fig, master=UI.histwin)

        canvas.draw()

        canvas.get\_tk\_widget().pack()

        UI.histwin.protocol("WM\_DELETE\_WINDOW", on\_exit)

**# creating a tab class for selecting which stock to choose which supports on-demand update:**

class tab(ctk.CTkFrame):

    def \_\_init\_\_(self, parent, Code, Curr, Dperc):

        global xcode, ddays, edate, sdate, bw, tw

        super().\_\_init\_\_(parent)

        labelc = "#000"

        self.Tcode = Code

        self.text = TRDX.loc[Code, "name"]

        if len(self.text) > 15:

            self.text = self.text[:15]+".."

        self.code = ctk.CTkLabel(self, text=Code, font=("Arial", 12, "bold"), padx=5, width=bw, height=24,

anchor="w",bg\_color=labelc)

        self.name = ctk.CTkLabel(self, text=self.text, width=bw, font=('Helevtica', 10, "italic"),

text\_color="grey", padx=5, height=24, anchor="w", bg\_color=labelc)

        self.curr = ctk.CTkLabel(self, text=str(Curr)+" INR", width=bw, height=24, anchor="e", bg\_color=labelc)

        self.Dperc = ctk.CTkLabel(self, text=str(Dperc)+"%", width=bw, height=24, anchor="e", bg\_color=labelc)

        self.tradbutton = ctk.CTkButton(self, text="View", width=48, height=48, corner\_radius=10,

fg\_color="#202020",anchor="c")

        if Dperc > 0:

            self.Dperc.configure(text\_color="#3dc985")

        else:

            self.Dperc.configure(text\_color="#ef4f60")

        self.code.grid(row=0, column=0)

        self.name.grid(row=1, column=0)

        self.curr.grid(row=0, column=1)

        self.Dperc.grid(row=1, column=1)

        self.tradbutton.grid(row=0, column=2, rowspan=2)

    def upd(self, Curr, Dperc):

        global xcode, ddays, edate, sdate, bw, tw

        self.curr.configure(text=str(Curr)+" INR")

        self.Dperc.configure(text=str(Dperc)+"%")

        if Dperc > 0:

            self.Dperc.configure(text\_color="#3dc985")

        else:

            self.Dperc.configure(text\_color="#ef4f60")

**# GUI:**

class UI(ctk.CTk):

    def \_\_init\_\_(self):

        global xcode, ddays, edate, sdate, bw, tw, tasv

        super().\_\_init\_\_()  # self refers to the GUI window

        self.title("StockSim")  # set title to StockSim

        self.configure(fg\_color="#0b1015")  # set background color to #0b1015

        # calculate midpoint of screen width

        x = (self.winfo\_screenwidth()-1280)//2

        # calculate midpoint of screen height

        y = (self.winfo\_screenheight()-720)//2

        self.resizable(False, False)  # disable window resize

        # set window size and position

        self.geometry('%dx%d+%d+%d' % (1280, 730, x, y))

        self.wm\_attributes("-alpha", "0.9")  # set window transparency

        self.login()  # show login screen

        # create frame(holder) for icons

        self.iconframe = ctk.CTkFrame(self, fg\_color="#151928", width=60, height=720, corner\_radius=0)

        # create home button inside frame

        self.homeicon = ctk.CTkButton(self.iconframe, text=None, image=images["home"], height=60,

width=60,command=self.home)

        # create porfolio button inside frame

        self.portficon = ctk.CTkButton(self.iconframe, text=None, image=images["pf"], height=60,

width=60,command=self.portf)

**# defining a function that updates the transaction history for a specific ticker**

    def botrightfill(self):

        rbw = tw+20  # calculate width for each cell

        for widget in self.botrightscrollable.winfo\_children():  # clear previous table incase of updation

            widget.destroy()

# label for if no trade has taken place yet

        none = ctk.CTkLabel(self.botrightscrollable, text="Initiate a trade", fg\_color="#151928",

width=rbw, height=15)

        row1 = []

        for i in ["Date", "Code", "Price", "Units", "Amt", "Action"]:  # create column labels

            row1.append(ctk.CTkLabel(self.botrightscrollable, text=i,fg\_color="#151928",

width=rbw//6, height=15))

        for i in row1:  # display column labels

            i.grid(row=0, column=row1.index(i))

        tokenledgerlist = {}

**# load transaction history of current token line by line**

        for i in list(self.tokenledger.index):

            date\_label = ctk.CTkLabel(self.botrightscrollable, text=self.tokenledger.loc[i, 'date'],

font=("font77", 10),width=rbw//6, padx=5, fg\_color="#000022")

            token\_label = ctk.CTkLabel(self.botrightscrollable, text=self.tokenledger.loc[i, 'token'],

font=("font77", 7),width=rbw//6, fg\_color="#000033")

            price\_label = ctk.CTkLabel(self.botrightscrollable,text=self.tokenledger.loc[i,'price'].round(2),

font=("font77",10),width=rbw//6, fg\_color="#000044")

            qty\_label = ctk.CTkLabel(self.botrightscrollable,text=abs(self.tokenledger.loc[i,'qty'].round(2)),

font=("font77",10),width=rbw//6, fg\_color="#000055")

            amt\_label = ctk.CTkLabel(self.botrightscrollable, text=self.tokenledger.loc[i, 'amt'].round(2),

font=("font77", 10),width=rbw//6, fg\_color="#000066")

            buy\_sell\_label = ctk.CTkLabel(self.botrightscrollable, text="₹buysell", font=("font77", 11),

width=rbw//6, fg\_color="#000077")

            if self.tokenledger.loc[i, 'qty'] > 0:

                buy\_sell\_label.configure(text="Buy", text\_color="#3dc985")

            else:

                buy\_sell\_label.configure(text="Sell", text\_color="#ef4f60")

            rown = [date\_label, token\_label, price\_label,

                    qty\_label, amt\_label, buy\_sell\_label]

            for j in rown:

                if self.tokenledger.loc[i, 'qty'] > 0:

                    j.configure(text\_color="#3dc985")

                else:

                    j.configure(text\_color="#ef4f60")

            tokenledgerlist[i] = rown

        # if no trade has taken place, then show text "Initiate a trade"

        if self.tokenledger.empty:

            none.grid(row=1, column=0, rowspan=3, columnspan=6)

            return

        # insert values into the table in descending order of transaction date

        else:

            revkeys = sorted(list(tokenledgerlist.keys()), reverse=True)

            none.grid\_forget()

            for c, i in enumerate(revkeys):

                for j in range(6):

                    tokenledgerlist[i][j].grid(row=c+1, column=j)

**# defining a function to update the top right frame with the current token's data when user changes what token they are trading**

    def toprightfill(self):

        # update name being shown

        self.trf\_name.configure(text=TRDX.loc[xcode, "name"])

        self.trf\_code.configure(text=xcode)

        self.trf\_curr.configure(text="₹ "+str(self.lddict[xcode]["Close"].iloc[0].round(3)))

        self.trf\_d.configure(text=str(self.lddict[xcode]["D"].iloc[0].round(3))+" INR")

        self.trf\_dperc.configure(text=str(self.lddict[xcode]["D%"].iloc[0].round(3))+"%")

        self.shares.configure(text=self.tokenledger["qty"].sum().round(2))

        self.netval.configure(text=str(round(self.lddict[xcode]["Close"].iloc[0]

\*self.userledger[self.userledger["token"] == xcode]["qty"].sum(), 2))+"INR")

        self.lopenv.configure(text=self.lddict[xcode]["Open"].iloc[0].round(3))

        self.lhighv.configure(text=self.lddict[xcode]["High"].iloc[0].round(3))

        self.llowv.configure(text=self.lddict[xcode]["Low"].iloc[0].round(3))

        self.lclosev.configure(text=self.lddict[xcode]["Close"].iloc[0].round(3))

        # change color of text according to profit or loss

        if self.lddict[xcode]["D"].iloc[0] > 0:

            self.trf\_dperc.configure(text\_color="#1f9358")

            self.trf\_d.configure(text\_color="#1f9358")

        else:

            self.trf\_dperc.configure(text\_color="#e04d5c")

            self.trf\_d.configure(text\_color="#e04d5c")

**# function to move the data to certain date and/or change the number of days displayed**

    def movedays(self, date: datetime.date, ndays=21):

        global xcode, ddays, edate, sdate, bw, tw, play, loopid

        # if date is out of range, show error message

        if date > datelist[-1] or date < datelist[0]-timedelta(5):

            mb.showerror(title="Date out of range",

message="Date Must be Greater than 2010-01-01 and Less than Today")

            return

        if play:  # check if in hometab

            ddays = ndays  # set new number of days

            sdate = date  # set new sdate

            self.graphspace.upd(ndate=sdate, ndays=ddays)  # update graph

            self.lddict = {i: tradable(i, sdate, ddays, True) for i in TRDX.index}  # update ticker data

            tasv = 0

            for i in self.lddict:

                # recalculate total assets value

                tasv += self.lddict[i]["Close"].iloc[0]\*

self.userledger[self.userledger["token"] == i]["qty"].sum()

            tasv = round(tasv, 2)

            tval = tasv + liq

            tval = round(tval, 2)

            self.topbar\_children\_dynamic["Liquid Assets"].configure(text=liq)  # update topbar

            self.topbar\_children\_dynamic["Assets Value"].configure(text=tasv)

            self.topbar\_children\_dynamic["Total Value"].configure(text=tval)

            for i in self.btndict:

                self.btndict[i].upd(self.lddict[i]["Close"].iloc[0].round(1),

self.lddict[i]["D%"].iloc[0].round(1))

            self.toprightfill()  # update stock data

            self.currd.configure(text="Curently Displaying: {} thru {}".format(sdate, edate))

            try:

                self.after\_cancel(loopid)

            except:

                pass

**# This is to keep updating the interface by 1 day after set time(default 5s)**

            loopid = self.after(itertime, partial(self.movedays, datelist[(datelist.index(sdate))+1]))

**#login screen backend**

def login(self):

        global xcode, ddays, edate, sdate, bw, tw, usr, pwd, liq

        # frame containing login interface

        self.login\_form = ctk.CTkFrame(self, width=300)

        self.usren = ctk.CTkEntry(self.login\_form, width=150, height=32,placeholder\_text="Username")

        self.pwden = ctk.CTkEntry(self.login\_form, width=150, height=32,placeholder\_text="Password",show="\*")

        self.title1 = ctk.CTkLabel(self.login\_form, text="Welcome To", text\_color="#4c4c4c",

font=("Roboto",10))

        self.title2 = ctk.CTkLabel(self.login\_form, text="STOCKSIM", font=("Roboto", 30))

        self.title3 = ctk.CTkLabel(self.login\_form, text="Login to continue", font=("Roboto", 15))

        self.loginbtn = ctk.CTkButton(self.login\_form, text="Login", width=100, height=32)

        self.title1.grid(row=0, column=0)  # placing all elements

        self.title2.grid(row=1, column=0, padx=10, pady=(0, 20))

        self.title3.grid(row=2, column=0)

        self.usren.grid(row=3, column=0)

        self.pwden.grid(row=4, column=0)

        self.loginbtn.grid(row=5, column=0)

**# login function**

        def loginx(event):

            global xcode, ddays, edate, sdate, bw, tw, usr, pwd, liq, tasv, tval, itertime, TRDX

            usr = self.usren.get()  # get user entry

            pwd = self.pwden.get()

            config = get\_config(usr, pwd)  # get user information from database

            match config[0]:  # fancy if-elif

                case 200:  # if user is in database

                    # load the tickers user had previously traded thru this app

                    TRDX = get\_tickers(usr)

                    sdate = config[1].date()  # get last start date on graph

                    edate = config[2].date()  # get last end date on graph

                    # get number of days being displayed in previous session

                    ddays = config[3]

                    # get last ticker displayed in previous session

                    xcode = get\_tickers(usr).index.tolist()[-1]

                    itertime = config[4]

                    # get total credits before leaving previous session

                    liq = round(config[5], 2)

                    self.home()

                    self.iconframe.place(x=0, y=0)  # begin loading homescreen

                    self.homeicon.place(x=0, y=0)

                    self.portficon.place(x=0, y=60)

                case 401:

                    mb.showerror(title="Error", message="Check your password".format(usr),

icon="info", type=mb.OK)  # if password is wrong but user exists show error

                case 400:

                    msg = mb.showerror(title="Error",

message="No user named {}, Do you want to create one?".format(usr),icon="info", type=mb.YESNO)  # if user doesn't exist ask to create

                    # if user wants to create a session and password is longer than 5 characters

                    if msg == "yes" and len(pwd) > 5:

                        with open("bin/plot/data/userdata.csv", "a") as file:  # add user to database

                            file.write("{},{},{},{},{},{},{}\n"

.format(usr,pwd,"2010-01-04",None,21,5000,10000))

                        with open("bin/plot/data/tickers.csv", "a") as file:

                            file.write("{},{},{}".format(usr, "SBIN.NS", "State Bank of India"))

                        self.login()  # revert back to login screen

                    else:

                        mb.showerror(title="Error", message="Create a stronger password!".format(

                            usr), icon="info", type=mb.OK)  # if weak password show error

        # set enter key to submit login form

        self.pwden.bind("<Return>", command=loginx)

        # set login button to submit login form

        self.loginbtn.configure(command=partial(loginx, None))

        # place login form in center of screen

        self.login\_form.place(relx=0.5, rely=0.5, anchor="center")

**# homescreen**

    def home(self):

        global xcode, ddays, edate, sdate, bw, tw, tasv, play

        try:  # try clearing screen

            for i in self.login\_form.winfo\_children():

                i.destroy()

            self.login\_form.destroy()

        except:

            pass

        try:

            self.portfupperframe.destroy()

            self.portflowerframe.destroy()

        except:

            pass

        self.homeicon.configure(state="disabled")

        self.portficon.configure(state="normal")

        play = True

        i = None

        self.btndict = {}

        self.txnlist = []

        self.lddict = {i: tradable(i, sdate, ddays, True) for i in TRDX.index}  # load all tradable tokens

        self.tokenledger = xledger.fetch\_token\_data(usr, xcode)  # load data of token currently on screen

        self.userledger = xledger.fetch\_user\_data(usr)  # load user transaction data

        tasv = 0  # load user's total asset value

        for i in self.lddict:

            tasv += self.lddict[i]["Close"].iloc[0] \*

self.userledger[self.userledger["token"] == i]["qty"].sum()

        tasv = round(tasv, 2)

        tval = round(tasv + liq, 2)

        def Trade(Tcode):

            global xcode, ddays, edate, sdate, bw, tw

            del xcode

            xcode = Tcode

            self.tokenledger = xledger.fetch\_token\_data(usr, xcode)

            self.userledger = xledger.fetch\_user\_data(usr)

            self.toprightfill()

            self.botrightfill()

            for i in self.btndict:

                if i == Tcode:

                    self.btndict[i].tradbutton.configure(state="disabled")

                else:

                    self.btndict[i].tradbutton.configure(state="normal")

            self.graphspace.upd(ndate=sdate, ndays=ddays)

            self.currd.configure(

                text="Curently Displaying: {} thru {}".format(sdate, edate))

        def addx():

            self.add\_btn.grid\_forget()

            self.addentry.grid(row=0, column=0, padx=5, columnspan=2)

            self.addcnf.grid(row=1, column=0, padx=5)

            self.addcanc.grid(row=1, column=1, padx=5)

        def add\_cnf():

            global TRDX

            self.add\_btn.grid(row=0, column=0, padx=5, columnspan=2)

            tname = self.addentry.get()

            txr = yf.Ticker(tname)

            if tname in TRDX.keys():

                mb.showinfo(title="Already added", message="Ticker is already added")

            elif txr.info["quoteType"] == "NONE":

                mb.showerror(title="Invalid Trade Symbol",

message="Please check the symbol or refer to https://finance.yahoo.com/lookup/ for valid BSE/NSE symbols")

            elif txr.info["financialCurrency"] != "INR":

                mb.showerror(title="Invalid Currency",

message="Please check the symbol or refer to https://finance.yahoo.com/lookup/ for valid BSE/NSE symbols trading in INR")

            else:

                add\_tickers(usr, sdate, ddays, tname)

                self.addentry.delete(0, 200)

                TRDX = get\_tickers(usr)

                self.lddict[tname] = tradable(tname, sdate, ddays, True)

                self.btndict[tname] = tab(self.leftscroller, tname,

self.lddict[tname]["Close"].iloc[0].round(3),self.lddict[tname]["D%"].iloc[0].round(1))

                self.btndict[tname].tradbutton.configure(command=partial(Trade, tname))

                self.btndict[tname].pack(anchor="w", pady=(0, 1))

            self.addentry.grid\_forget()

            self.addcnf.grid\_forget()

            self.addcanc.grid\_forget()

        def add\_canc():

            self.add\_btn.grid(row=0, column=0, padx=5, columnspan=2)

            self.addentry.grid\_forget()

            self.addcnf.grid\_forget()

            self.addcanc.grid\_forget()

        def buy():

            self.buy.configure(height=24, text="Confirm?",

font=("Helvetica", 15, "bold"), command=partial(confirm, "buy"))

            self.entry.grid(row=4, column=0, rowspan=1, columnspan=1)

            self.buy.grid\_configure(row=5, column=0, rowspan=1, columnspan=1)

            self.sell.configure(text="Cancel", command=cancel)

        def sell():

            self.sell.configure(height=24, text="Confirm?",

font=("Helvetica", 15, "bold"), command=partial(confirm, "sell"))

            self.entry.grid(row=4, column=1, rowspan=1, columnspan=1)

            self.sell.grid\_configure(row=5, column=1, rowspan=1, columnspan=1)

            self.buy.configure(text="Cancel", command=cancel)

        def confirm(what):

            self.buy.configure(text="Buy", font=("Helvetica", 25, "bold"), command=buy, height=48)

            self.sell.configure(text="Sell", font=("Helvetica", 25, "bold"), command=sell, height=48)

            self.entry.grid\_forget()

            self.buy.grid\_configure(row=4, column=0, rowspan=2, columnspan=1)

            self.sell.grid\_configure(row=4, column=1, rowspan=2, columnspan=1)

            nstock = self.entry.get()

            try:

                float(nstock)

            except:

                mb.showerror(

                    title="Error", message="Enter a valid quantity", icon="info", type=mb.OK)

                return

            if float(nstock) > 0:

                if what == "buy":

                    if float(nstock)\*self.lddict[xcode]["Open"].iloc[0] > float(liq):

                        mb.showerror(title="Error", message="Not enough Liquidity", icon="info", type=mb.OK)

                    else:

                        sell\_buy\_update(

                            self.lddict[xcode]["Open"].iloc[0], float(nstock))

                elif what == "sell":

                    if float(nstock) > float(self.tokenledger["qty"].sum()):

                        mb.showerror(title="Error",

message="Not enough Shares to sell", icon="info",type=mb.OK)

                    elif sdate < datetime.strptime(self.userledger[self.userledger["token"] ==

xcode]["date"].iloc[0],"%Y-%m-%d").date():

                        mb.showerror(title="Error",

message="You can't sell before buying", icon="info",type=mb.OK)

                    else:

                        sell\_buy\_update(

                            self.lddict[xcode]["Close"].iloc[0], -float(nstock))

            elif float(nstock) == 0:

                mb.showerror(title="Good thinking!!",message="Try that in real world", icon="info", type=mb.OK)

            else:

                mb.showerror(

                    title="Error", message="Enter a valid quantity", icon="info", type=mb.OK)

            del nstock

        def cancel():

            self.buy.configure(text="Buy", font=(

                "Helvetica", 25, "bold"), command=buy, height=48)

            self.sell.configure(text="Sell", font=(

                "Helvetica", 25, "bold"), command=sell, height=48)

            self.entry.grid\_forget()

            self.buy.grid\_configure(row=4, column=0, rowspan=2, columnspan=1)

            self.sell.grid\_configure(row=4, column=1, rowspan=2, columnspan=1)

        def sell\_buy\_update(price, units):

            global liq, tasv, tval

            xledger.txn(edate, usr, xcode, price, units)

            self.tokenledger = xledger.fetch\_token\_data(usr, xcode)

            self.userledger = xledger.fetch\_user\_data(usr)

            liq -= units\*price

            liq = round(liq, 2)

            tasv += units\*price

            tasv = round(tasv, 2)

            tval = liq + tasv

            tval = round(tval, 2)

            self.topbar\_children\_dynamic["Liquid Assets"].configure(text=liq)

            self.topbar\_children\_dynamic["Assets Value"].configure(text=tasv)

            self.topbar\_children\_dynamic["Total Value"].configure(text=tval)

            self.entry.delete(0, len(self.entry.get()))

            self.configure(text=self.tokenledger["qty"].sum())

            self.configure(text=self.tokenledger["amt"].sum())

            self.shares.configure(text=self.tokenledger["qty"].sum().round(2))

            self.netval.configure(text=str(round(self.lddict[xcode]["Close"].iloc[0]\*

self.userledger[self.userledger["token"] == xcode]["qty"].sum(), 2))+"INR")

            self.botrightfill()

        self.leftframe = ctk.CTkFrame(self, width=tw, height=720, corner\_radius=0, fg\_color="#2d303e")

        self.leftscroller = ctk.CTkScrollableFrame(self.leftframe, width=tw,

height=720, corner\_radius=0,fg\_color="#2d303e")

        self.leftscroller.pack()

        self.tlable = ctk.CTkLabel(self.leftscroller, text="Tradables",

height=48,width=tw+2, font=("Arial", 20),bg\_color="#2b2b2b").pack()

        self.add\_frame = ctk.CTkFrame(

            master=self.leftscroller, width=3\*bw, height=48)

        self.add\_btn = ctk.CTkButton(self.add\_frame, text=None, image=images["add"],

width=2\*bw+48, height=48, corner\_radius=10, fg\_color="#242424",command=addx)

        self.addentry = ctk.CTkEntry(self.add\_frame, width=2\*bw+48-1, height=24,

placeholder\_text="Enter <STOCK>.NS or <STOCK>.BO",fg\_color="#202020")

        self.addcnf = ctk.CTkButton(self.add\_frame, text="Add", width=bw, height=24, command=add\_cnf)

        self.addcanc = ctk.CTkButton(self.add\_frame, text="Cancel", width=bw, height=24, command=add\_canc)

        self.add\_btn.grid(row=0, column=0, padx=5, columnspan=2)

        self.add\_frame.pack()

        for i in TRDX.index:

            self.btndict[i] = tab(self.leftscroller, i, self.lddict[i]["Close"].iloc[0].round(3),

self.lddict[i]["D%"].iloc[0].round(1))

        for i in self.btndict:

            self.btndict[i].tradbutton.configure(command=partial(Trade, i))

            self.btndict[i].pack(anchor="w", pady=(0, 1))

        self.graphspace = customcandlestick(self)

        self.topbar = ctk.CTkFrame(self, width=vw, height=48, fg\_color="#000")

        self.topbar\_children\_static=

{"Total Value": ctk.CTkLabel(self.topbar,text="Total Value",width=vw//2,height=10, fg\_color="#000"),

"Liquid Assets": ctk.CTkLabel(self.topbar, text="Liquid Assets",width=vw//4,height=10, fg\_color="#000"),

"Assets Value": ctk.CTkLabel(self.topbar,text="Assets Value", width=vw//4, height=10,  fg\_color="#000")}

        self.topbar\_children\_dynamic =

{"Total Value": ctk.CTkLabel(self.topbar, text=tval, width=vw//2, height=36, fg\_color="#000"),

"Liquid Assets": ctk.CTkLabel(self.topbar, text=liq, width=vw//4, height=36, fg\_color="#000"),

"Assets Value": ctk.CTkLabel(self.topbar, text=tasv, width=vw//4, height=36,  fg\_color="#000")}

        for count, i in enumerate(self.topbar\_children\_static):

            self.topbar\_children\_static[i].grid(row=0, column=count)

            self.topbar\_children\_dynamic[i].grid(row=1, column=count)

        self.currd = ctk.CTkLabel(self, text="Curently Displaying: {} thru {}".format(sdate, edate),

fg\_color="#000",width=rfw//2+2, height=6, anchor="w", padx=5)

        self.toprightframe = ctk.CTkFrame(self, width=rfw, height=360, corner\_radius=0, fg\_color="#2d303e")

        self.trf\_name = ctk.CTkLabel(self.toprightframe, text=TRDX.loc[xcode, "name"],

width=rfw, height=24, anchor="w", bg\_color="#0d1016", pady=10, padx=5)

        self.trf\_code = ctk.CTkLabel(self.toprightframe, font=("Helvetica", 15), text=xcode,

width=rfw, height=24, anchor="w", bg\_color="#151928", pady=10, padx=5)

self.trf\_curr = ctk.CTkLabel(self.toprightframe,

text="₹"+str(self.lddict[xcode]["Close"].iloc[0].round(3)), font=("Helvetica", 30,"bold"), width=rfw//2, height=48, anchor="w", bg\_color="#212533", pady=18, padx=5)

self.trf\_d = ctk.CTkLabel(self.toprightframe,text=str(self.lddict[xcode]["D"].iloc[0].round(3))+"INR",

width=rfw//2, height=24,anchor="e", bg\_color="#212533", pady=10, padx=5)

        self.trf\_dperc = ctk.CTkLabel(self.toprightframe,

text=str(self.lddict[xcode]["D%"].iloc[0].round(3))+"%", width=rfw//2, height=24,anchor="e", bg\_color="#212533", pady=10, padx=5)

        self.buy = ctk.CTkButton(self.toprightframe, text="Buy", font=("Helvetica", 35, "bold"), width=rfw//2,

fg\_color="#1f9358", height=48, command=buy)

self.sell = ctk.CTkButton(self.toprightframe, text="Sell", font=("Helvetica", 35, "bold"),

width=rfw//2,fg\_color="#e04d5c",height=48, command=sell)

        self.entry = ctk.CTkEntry(self.toprightframe, width=rfw//2, height=24, placeholder\_text="Enter Units")

        if self.lddict[xcode]["D"].iloc[0] > 0:

            self.trf\_dperc.configure(text\_color="#1f9358")

            self.trf\_d.configure(text\_color="#1f9358")

        else:

            self.trf\_dperc.configure(text\_color="#e04d5c")

            self.trf\_d.configure(text\_color="#e04d5c")

        self.aag = ctk.CTkLabel(self.toprightframe, text="At A Glance {}".format(edate),

fg\_color="#0d1017", width=rfw, height=24)

        self.lopen = ctk.CTkLabel(self.toprightframe, text="Open",

fg\_color="#1d2950", width=rfw//2, height=24, padx=5)

        self.lhigh = ctk.CTkLabel(self.toprightframe, text="High",

fg\_color="#161929", width=rfw//2, height=24, padx=5)

        self.llow = ctk.CTkLabel(self.toprightframe, text="Low",

fg\_color="#1d2950", width=rfw//2, height=24, padx=5)

        self.lclose = ctk.CTkLabel(self.toprightframe, text="Close",

fg\_color="#161929", width=rfw//2, height=24, padx=5)

        self.lshares = ctk.CTkLabel(self.toprightframe, text="Shares",justify="center",

fg\_color="#1c2951", width=rfw//2, height=24)

        self.lnetval = ctk.CTkLabel(self.toprightframe, text="Net Value",justify="center",

fg\_color="#1c2951", width=rfw//2, height=24)

self.lopenv = ctk.CTkLabel(self.toprightframe, text=self.lddict[xcode]["Open"].iloc[0].round(3),

fg\_color="#000",width=rfw//2, height=24)

        self.lhighv = ctk.CTkLabel(self.toprightframe,text=self.lddict[xcode]["High"].iloc[0].round(3),

fg\_color="#000", width=rfw//2, height=24)

self.llowv = ctk.CTkLabel(self.toprightframe, text=self.lddict[xcode]["Low"].iloc[0].round(3),

fg\_color="#000", width=rfw//2, height=24)

        self.lclosev = ctk.CTkLabel(self.toprightframe, text=self.lddict[xcode]["Close"].iloc[0].round(3),

fg\_color="#000", width=rfw//2, height=24)

        self.shares = ctk.CTkLabel(self.toprightframe, text="₹Shares",

                                   fg\_color="#1c2971", justify="center", width=rfw//2, height=48)

        self.netval = ctk.CTkLabel(self.toprightframe, text="₹Net Value",

                                   fg\_color="#1c2971", justify="center", width=rfw//2, height=48)

        self.shares.configure(text=str(self.tokenledger["qty"].sum()))

        self.netval.configure(text=str(round(self.lddict[xcode]["Close"].iloc[0]\*

self.userledger[self.userledger["token"] == xcode]["qty"].sum(), 2))+"INR")

        self.toprightframe.place(x=1240-tw, y=0)

        rbw = tw+20

        self.botrightframe = ctk.CTkFrame(self, width=rbw, height=300, corner\_radius=0, fg\_color="#151928",

border\_color="#fff", border\_width=1)

        self.botrightscrollable = ctk.CTkScrollableFrame(self.botrightframe, width=rbw, height=360,

corner\_radius=0, fg\_color="#151928")

        self.botrightfill()

        self.botrightscrollable.pack()

        self.botrightframe.place(x=1280-tw-40, y=400)

        self.timecontrolframe = ctk.CTkFrame(self, width=tw+96, height=48, corner\_radius=0,

fg\_color="#151928",border\_color="#fff")

        self.timecontrolable = ctk.CTkLabel(self.timecontrolframe, width=tw+96, height=6, text="Time Controls",

fg\_color="#151928", font=("Helvetica", 20, "bold"))

        self.calentry = ctk.CTkEntry(self.timecontrolframe, width=tw//2, height=6,

placeholder\_text="Enter Date DD/MM/YYYY")

        self.ndaysentry = ctk.CTkEntry(self.timecontrolframe, width=tw//2, height=6,

placeholder\_text="Enter Number of Days")

        self.ndaysentry.insert(0, str(ddays)+" Days")

        self.go = ctk.CTkButton(self.timecontrolframe, text="Go", height=48, width=48,

command=lambda: self.movedays(datetime.strptime(self.calentry.get(),"%d/%m/%Y").date(), self.ndaysentry.get().split(" ")[0]))

        self.histbtn = ctk.CTkButton(self.timecontrolframe, text="Show History", width=tw//2,height=48,

fg\_color="#1c2951", command=partial(self.graphspace.show\_history, self))

        self.timecontrolable.grid(row=0, column=0, columnspan=3, rowspan=1)

        self.calentry.grid(row=1, column=0, columnspan=1, rowspan=1)

        self.ndaysentry.grid(row=2, column=0, columnspan=1, rowspan=1)

        self.go.grid(row=1, column=1, columnspan=1, rowspan=2)

        self.histbtn.grid(row=1, column=2, columnspan=1,rowspan=3, padx=(20, 0))

        self.timecontrolframe.place(x=500, y=650)

        self.btndict[xcode].tradbutton.configure(state="disabled")

        self.trf\_name.grid(row=0, column=0, columnspan=2, rowspan=1)

        self.trf\_code.grid(row=1, column=0, columnspan=2, rowspan=1)

        self.trf\_curr.grid(row=2, column=0, columnspan=1, rowspan=2, padx=(0, 1))

        self.trf\_d.grid(row=2, column=1, columnspan=1, rowspan=1, pady=(0, 1))

        self.trf\_dperc.grid(row=3, column=1, columnspan=1, rowspan=1)

        self.buy.grid(row=4, column=0, columnspan=1, rowspan=2, pady=(5, 0))

        self.sell.grid(row=4, column=1, columnspan=1, rowspan=2, pady=(5, 0))

        self.aag.grid(row=6, column=0, columnspan=2, rowspan=1, pady=(10, 1))

        self.lopen.grid(row=7, column=0, columnspan=1, rowspan=1, pady=(0, 1))

        self.lopenv.grid(row=7, column=1, columnspan=1, rowspan=1, pady=(0, 1))

        self.lhigh.grid(row=8, column=0, columnspan=1, rowspan=1, pady=(0, 1))

        self.lhighv.grid(row=8, column=1, columnspan=1, rowspan=1, pady=(0, 1))

        self.llow.grid(row=9, column=0, columnspan=1, rowspan=1, pady=(0, 1))

        self.llowv.grid(row=9, column=1, columnspan=1, rowspan=1, pady=(0, 1))

        self.lclose.grid(row=10, column=0, columnspan=1,rowspan=1, pady=(0, 1))

        self.lclosev.grid(row=10, column=1, columnspan=1,rowspan=1, pady=(0, 1))

        self.lshares.grid(row=11, column=0, columnspan=1,rowspan=1, pady=(5, 1))

        self.lnetval.grid(row=11, column=1, columnspan=1,rowspan=1, pady=(5, 1))

        self.shares.grid(row=12, column=0, columnspan=1, rowspan=2)

        self.netval.grid(row=12, column=1, columnspan=1, rowspan=2)

        self.graphspace.place(x=60+tw, y=48+48)

        self.currd.place(x=20+tw, y=48)

        self.leftframe.place(x=60, y=0)

        self.topbar.place(x=60+tw, y=0)

        self.movedays(date=sdate)

**# portfolio screen**

    def portf(self):

        global sdate, edate, play

        play = False

        self.homeicon.configure(state="normal")

        self.portficon.configure(state="disabled")

        try:

            self.leftframe.destroy()

            self.timecontrolframe.destroy()

            self.botrightframe.destroy()

            self.toprightframe.destroy()

            self.graphspace.destroy()

            self.currd.destroy()

            self.topbar.destroy()

        except:

            pass

        upw = 1220//6

        lpw = 1220//8

        self.portfupperframe = ctk.CTkFrame(self, width=1220, height=360, corner\_radius=50, fg\_color="#151928",

border\_color="#fff", border\_width=1)

        self.pfuscroll = ctk.CTkScrollableFrame(self.portfupperframe, width=1220-20, height=360,

corner\_radius=0, fg\_color="#151928")

        self.pfuscroll.pack()

        urow1 = []

        for i in ["Code", "Name", "Units", "Value", "Profitability"]:

            if i == "Name":

                urow1.append(ctk.CTkLabel(self.pfuscroll, text=i, width=upw\*2))

            else:

                urow1.append(ctk.CTkLabel(self.pfuscroll, text=i, width=upw))

        for c, i in enumerate(urow1):

            i.grid(row=0, column=c)

        del urow1

        self.tradabluserdata = []

        for code in TRDX.index:

            code\_label = ctk.CTkLabel(self.pfuscroll, text=code, fg\_color="#000022", width=upw)

            trdx\_label = ctk.CTkLabel(self.pfuscroll, text=TRDX.loc[code, "name"],

fg\_color="#000044", width=upw\*2)

            qty\_label = ctk.CTkLabel(self.pfuscroll,

text=self.userledger[self.userledger["token"] == code]["qty"].sum(),fg\_color="#000066",width=upw)

            value\_label = ctk.CTkLabel(self.pfuscroll,

text=round(self.userledger[self.userledger["token"] == code]["qty"].sum() \* self.lddict[code]["Close"].iloc[0], 3), fg\_color="#000088", width=upw)

            pl\_label = ctk.CTkLabel(self.pfuscroll,

text=round(self.userledger[self.userledger["token"] == code]["qty"].sum() \* self.lddict[code]["Close"].iloc[0] - self.userledger[self.userledger["token"] == code]["amt"].sum(), 3), fg\_color="#0000aa",width=upw)

            self.tradabluserdata.append((code\_label, trdx\_label, qty\_label, value\_label, pl\_label))

        for i in range(len(self.tradabluserdata)):

            for j in range(5):

                self.tradabluserdata[i][j].grid(row=i+1, column=j)

        del code\_label, trdx\_label, qty\_label, value\_label, pl\_label

        self.portflowerframe = ctk.CTkFrame(self, width=1220, height=350, corner\_radius=0, fg\_color="#fff")

        self.pflscroll = ctk.CTkScrollableFrame(self.portflowerframe, width=1220-30, height=360,

corner\_radius=0, fg\_color="#151928")

        self.pflscroll.pack()

        urow2 = []

        for i in ["TXN id.", "Date", "User", "Code", "Units", "Price", "Action", "Liq Change"]:

            urow2.append(ctk.CTkLabel(self.pflscroll, text=i, width=lpw, padx=2))

        for c, i in enumerate(urow2):

            i.grid(row=0, column=c)

        self.txndata = []

        for i in xledger.data[xledger.data["user"] == usr].index:

            txn\_id\_label = ctk.CTkLabel(self.pflscroll, text=i, fg\_color="#000012", width=lpw)

            date\_label = ctk.CTkLabel(self.pflscroll,

text=xledger.data['date'].iloc[i], fg\_color="#000024", width=lpw)

            user\_label = ctk.CTkLabel(self.pflscroll,

text=xledger.data['user'].iloc[i], fg\_color="#000036", width=lpw)

            code\_label = ctk.CTkLabel(self.pflscroll,

text=xledger.data['token'].iloc[i], fg\_color="#000048", width=lpw)

            units\_label = ctk.CTkLabel(self.pflscroll,

text=abs(xledger.data['qty'].iloc[i]), fg\_color="#000060", width=lpw)

            price\_label = ctk.CTkLabel(self.pflscroll,text=xledger.data['price'].iloc[i].round(3),

fg\_color="#000072",width=lpw)

            action\_label = ctk.CTkLabel(self.pflscroll, text="₹BuySell", fg\_color="#000084", width=lpw)

            liqchange\_label = ctk.CTkLabel(self.pflscroll, text=(-xledger.data["amt"].iloc[i].round(3)),

fg\_color="#000096",width=lpw)

            if xledger.data['qty'].iloc[i] > 0:

                action\_label.configure(text="Buy")

            else:

                action\_label.configure(text="Sell")

            rown = [txn\_id\_label, date\_label, user\_label, code\_label,

units\_label, price\_label, action\_label, liqchange\_label]

            for j in rown:

                if xledger.data['qty'].iloc[i] > 0:

                    j.configure(text\_color="#3dc985")

                else:

                    j.configure(text\_color="#ef4f60")

            self.txndata.append(rown)

        for i in range(len(self.txndata)):

            for j in range(8):

                self.txndata[i][j].grid(row=i+1, column=j)

        self.portfupperframe.place(x=60, y=0)

        self.portflowerframe.place(x=60, y=370)

del txn\_id\_label, date\_label, user\_label, code\_label

del units\_label, price\_label, action\_label, liqchange\_label, rown

**data.py**

#IMPORTING LIBRARIES, INITIALISING GLOBAL VARIABLES AND FUNCTIONS

import pandas as pd

import yfinance as yf

from datetime import \*

datelist = pd.read\_csv("bin/plot/data/datelist.csv", header=None)[0].tolist()

binance\_dark = {

    "base\_mpl\_style": "dark\_background",

    "marketcolors": {

        "candle": {"up": "#3dc985", "down": "#ef4f60"},

        "edge": {"up": "#3dc985", "down": "#ef4f60"},

        "wick": {"up": "#3dc985", "down": "#ef4f60"},

        "ohlc": {"up": "green", "down": "red"},

        "volume": {"up": "#247252", "down": "#82333f"},

        "vcedge": {"up": "green", "down": "red"},

        "vcdopcod": False,

        "alpha": 1

    },

    "facecolor": "#1b1f24",

    "gridcolor": "#2c2e31",

    "gridstyle": "--",

    "y\_on\_right": False,

    "rc": {

        "axes.grid": True,

        "axes.grid.axis": "y",

        "axes.edgecolor": "#474d56",

        "axes.titlecolor": "red",

        "figure.facecolor": "#161a1e",

        "figure.titlesize": "x-large",

        "figure.titleweight": "semibold",

    },

    "base\_mpf\_style": "binance-dark",

}

def get\_tickers(usr):

    x = pd.read\_csv("bin/plot/data/tickers.csv", header=None,index\_col=0, names=["ticker", "name"])

    x = x[x.index == usr]

    x.index = x["ticker"]

    del x["ticker"]

    x.index.name = None

    return x

def add\_tickers(usr, sdate, ndays, ticker):

    tickername = str(yf.Ticker(ticker).info["longName"]).replace("Limited", "Ltd.")

    with open("bin/plot/data/tickers.csv", "a") as f:

        f.write("{},{},{}\n".format(usr, ticker, tickername))

def get\_config(usr, pwd):

    x = pd.read\_csv("bin/plot/data/userdata.csv",names=["pwd", "sdate", "edate", "ndays", "itertime", "liq"], index\_col=0)

    x["sdate"] = pd.to\_datetime(x["sdate"], format='%Y-%m-%d')

    x["edate"] = pd.to\_datetime(x["edate"], format='%Y-%m-%d')

    if usr in x.index:

        if pd.Series(x.loc[usr]["pwd"]).iloc[-1] == pwd:

            return (200, pd.Series(x.loc[usr]["sdate"]).iloc[-1], pd.Series(x.loc[usr]["edate"]).iloc[-1],

pd.Series(x.loc[usr]["ndays"]).iloc[-1], pd.Series(x.loc[usr]["itertime"]).iloc[-1], pd.Series(x.loc[usr]["liq"]).iloc[-1])

        else:

            return (401, None, None)

    else:

        return (400, None, None)

def lff(name, sdate, dnrows):

    """Loads Data from file ranging from sdate to sdate+dnrows"""

    with open("bin/plot/data/{}.csv".format(name), "r") as f:

        for count, l in enumerate(f):

            # print(str(l).startswith(str(sdate)))

            if str(l).startswith(str(sdate)):

                df = pd.read\_csv("bin/plot/data/{}.csv".format(name),

                                 header=None, index\_col=0, skiprows=count, nrows=int(dnrows))

                df.index = pd.to\_datetime(df.index, format='%Y-%m-%d')

                df.index.name = None

                df.columns = ["Open", "High", "Low", "Close", "Volume"]

                df["D"] = df["Close"]-df["Open"]

                df["D%"] = df["D"]/df["Open"]\*100

                # df["height"] = df["High"]-df["Low"]

                break

    try:

        return df

    except UnboundLocalError:

        nextday = datetime.strptime(str(sdate), "%Y-%m-%d")+timedelta(days=1)

        nextday = nextday.strftime("%Y-%m-%d")

        return lff(name, nextday, dnrows)

def loadhistory(name, edate):

    with open("bin/plot/data/{}.csv".format(name), "r") as f:

        for count, l in enumerate(f):

            if str(l).startswith(str(edate)):

                df = pd.read\_csv("bin/plot/data/{}.csv".format(name),header=None, index\_col=0, nrows=count+1)

                df.index = pd.to\_datetime(df.index, format='%Y-%m-%d')

                df.index.name = None

                df.columns = ["Open", "High", "Low", "Close", "Volume"]

                return df

def lfw(name):

    """Loads Ticker from Web"""

    tk = yf.Ticker(name)  # get ticker (YahooFinance module)

    x = pd.DataFrame(tk.history(period="max"))

    x.index = [d.strftime('%Y-%m-%d') for d in x.index.date]

    x = x.drop(columns=["Dividends", "Stock Splits"]).loc["2010-01-04":]

    if x.index[0] != "2010-01-04":

        finalindex = x.index[0]

        print(datelist.index("2010-01-04"))

        print(datelist[datelist.index("2010-01-04")

              :datelist.index(finalindex)])

        zeroindex = datelist[datelist.index(

            "2010-01-04"):datelist.index(finalindex)]

        zdf = pd.DataFrame(

            0, columns=["Open", "High", "Low", "Close", "Volume"], index=zeroindex)

        x = pd.concat([zdf, x])

    x.to\_csv("bin/plot/data/{}.csv".format(name), header=False)

    del x

def tradable(name, sdate, dnrows, lastday=False):

    try:

        if lastday == False:

            return lff(name, sdate, dnrows)

        else:

            return lff(name, sdate, dnrows).tail(1)

    except FileNotFoundError or pd.errors.EmptyDataError:

        lfw(name)

        if lastday == False:

            return lff(name, sdate, dnrows)

        else:

            return lff(name, sdate, dnrows).tail(1)

class ledger():

    def \_\_init\_\_(self, file):

        self.file = file

        import pandas as pd

        self.data = pd.read\_csv(file, header=None, names=["date", "user", "token", "price", "qty", "amt"],

dtype={"user": str, "token": str, "price": float, "qty": float, "amt": float}, index\_col=0)

        if self.data.empty:

            self.last\_index = -1

        else:

            self.last\_index = self.data.index.max()

    def txn(self, date: datetime.date, user: str, token: str, price: float, qty: float):

        amt = abs(price\*qty)

        txn = pd.DataFrame({"date": date, "user": user, "token": token,

                           "price": price, "qty": qty, "amt": amt}, index=[self.last\_index+1])

        self.data = pd.concat([txn, self.data], axis=0)

        self.last\_index += 1

        return amt

    def fetch\_user\_data(self, user: str):

        return self.data[self.data["user"] == user]

    def fetch\_token\_data(self, user: str, token: str):

        return self.data[(self.data["user"] == user) & (self.data["token"] == token)]

    def fetch\_token\_netval(self, user: str, token: str, cprice: float):

        return self.fetch\_token\_data(user, token).sum()["qty"]\*cprice

    def save\_to\_csv(self):

        self.data.to\_csv(self.file, header=False)

**datelist.csv**

.

|-- main.py

`-- bin/

|-- UI.py

`-- plot/

|-- data.py

`-- data/

|-- datelist.csv 🡨 You are here

|-- ledger.csv

|-- tickers.csv

|-- userdata.csv

|-- <tradable\_data>.csv

`-- images/

|-- add.png

|-- home.png

`-- pf.png

2010-01-04

2010-01-05

2010-01-06

2010-01-07

2010-01-08

2010-01-11

2010-01-12

2010-01-13

2010-01-14

2010-01-15

2010-01-18

2010-01-19

2010-01-20

2010-01-21

2010-01-22

2010-01-25

2010-01-27

2010-01-28

.

.

.

.

.

.

2024-11-21

2024-11-22

2024-11-25

2024-11-26

2024-11-27

**ledger.csv**

10,2010-04-22,admin,AXISBANK.BO,173.48922729492188,-3.0,520.4676818847656

9,2010-04-19,admin,ICICIBANK.BO,97.29234313964844,-56.0,5448.3712158203125

8,2010-04-15,admin,LT.NS,562.6217651367188,-5.0,2813.1088256835938

7,2010-04-09,admin,BPCL.NS,21.66822052001953,-50.0,1083.4110260009766

6,2010-04-05,admin,HDFCBANK.BO,144.1591477943974,5.0,720.795738971987

5,2010-03-19,admin,HDFCBANK.BO,134.41543579101562,-5.0,672.0771789550781

4,2010-03-02,admin,AXISBANK.BO,160.79283559449743,3.0,482.3785067834923

3,2010-02-22,admin,LT.NS,533.2737313607433,5.0,2666.3686568037165

2,2010-02-17,admin,ICICIBANK.BO,88.63509760266254,56.0,4963.565465749102

1,2010-02-16,admin,BPCL.NS,23.98537526223804,50.0,1199.268763111902

0,2010-02-15,admin,HDFCBANK.BO,117.98871927852414,5.0,589.9435963926207

.

|-- main.py

`-- bin/

|-- UI.py

`-- plot/

|-- data.py

`-- data/

|-- datelist.csv

|-- ledger.csv 🡨- You are here

|-- tickers.csv

|-- userdata.csv

|-- <tradable\_data>.csv

`-- images/

|-- add.png

|-- home.png

`-- pf.png

.

|-- main.py

`-- bin/

|-- UI.py

`-- plot/

|-- data.py

`-- data/

|-- datelist.csv

|-- ledger.csv

|-- tickers.csv 🡨- You are here

|-- userdata.csv

|-- <tradable\_data>.csv

`-- images/

|-- add.png

|-- home.png

`-- pf.png

**tickers.csv**

admin,SBIN.NS,State Bank of India

admin,BEL.NS,Bharat Electronics Ltd.

admin,ONGC.NS,Oil and Natural Gas Corporation Ltd.

admin,BPCL.NS,Bharat Petroleum Corporation Ltd.

admin,SHRIRAMFIN.NS,Shriram Finance Ltd.

admin,LT.NS,Larsen & Toubro Ltd.

user,SBIN.NS,State Bank of India

user,BEL.NS,Bharat Electronics Ltd.

user,ONGC.NS,Oil and Natural Gas Corporation Ltd.

user,BPCL.NS,Bharat Petroleum Corporation Ltd.

admin,JSWSTEEL.NS,JSW Steel Ltd.

admin,ZOMATO.BO,Zomato Ltd.

admin,ICICIBANK.BO,ICICI Bank Ltd.

admin,AXISBANK.BO,Axis Bank Ltd.

admin,HDFCBANK.BO,HDFC Bank Ltd.

king,SBIN.NS,State Bank of India

**<tickername>.csv (Sample)**

2010-01-04,141.40143229574824,141.40143229574824,139.84446175132268,140.5592498779297,658270

2010-01-05,141.68451560810345,144.21106289161858,141.11834154915672,142.95840454101562,1105750

2010-01-06,143.89966149230176,144.3738291451,140.43185225410141,140.87771606445312,815055

2010-01-07,141.54294589645139,142.25066062593365,139.98597565136438,141.9604949951172,1123435

2010-01-08,142.81686857472997,144.13321871695155,141.11834634022375,143.4608917236328,1219710

2010-01-11,144.37381579617823,149.46936007626923,144.21104441969965,148.7758026123047,1509610

2010-01-12,148.6201203074971,151.8756129766386,145.860036678478,148.49273681640625,1586180

2010-01-13,145.50616889530318,148.33702812829452,145.50616889530318,147.1268310546875,2224945

2010-01-14,147.0490058213376,152.5550270577802,147.0490058213376,150.3257293701172,2228290

2010-01-15,152.70363968901742,154.6993899960967,150.74326648614726,152.61163330078125,2247170

2010-01-18,152.15870179827368,157.82042093495363,150.0851023479233,156.87208557128906,1796635

2010-01-19,157.25425604707584,157.82043015358977,154.73479261190548,156.29176330566406,865845

2010-01-20,156.82965458078118,158.2450846372905,152.7744396936925,153.2556915283203,803705

2010-01-21,152.86641941396488,154.2676919015753,148.7829013901946,150.04263305664062,699775

2010-01-22,148.62011429997096,149.47645399577416,144.9541566640299,145.9520263671875,1537490

2010-01-25,144.37381873155772,145.78924832696515,142.39221513821315,144.03411865234375,588995

2010-01-27,142.95839600947863,142.95839600947863,136.8932747180935,137.65760803222656,1231550

2010-01-28,141.3730925352366,144.65688849596268,138.71208900117458,143.52455139160156,1014920

.

.

.

.

|-- main.py

`-- bin/

|-- UI.py

`-- plot/

|-- data.py

`-- data/

|-- datelist.csv

|-- ledger.csv

|-- tickers.csv

|-- userdata.csv

|-- <tickername>.csv 🡨- You are here

`-- images/

|-- add.png

|-- home.png

`-- pf.png

2024-11-18,1128.4000244140625,1136.300048828125,1121.550048828125,1126.1500244140625,177580

2024-11-19,1130.0,1138.8499755859375,1122.0999755859375,1135.0,127145

2024-11-21,1136.300048828125,1152.0,1115.699951171875,1139.0999755859375,201912

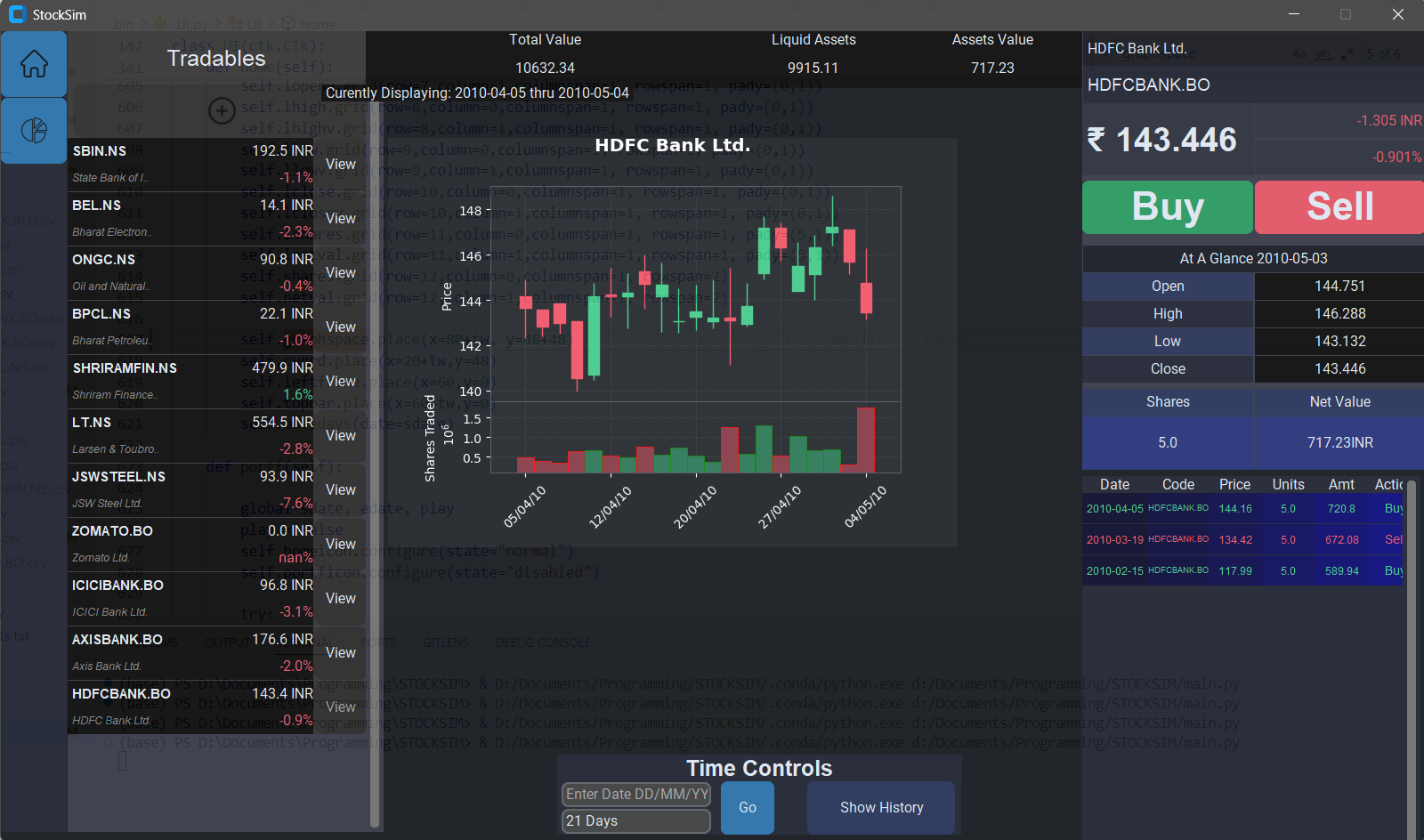
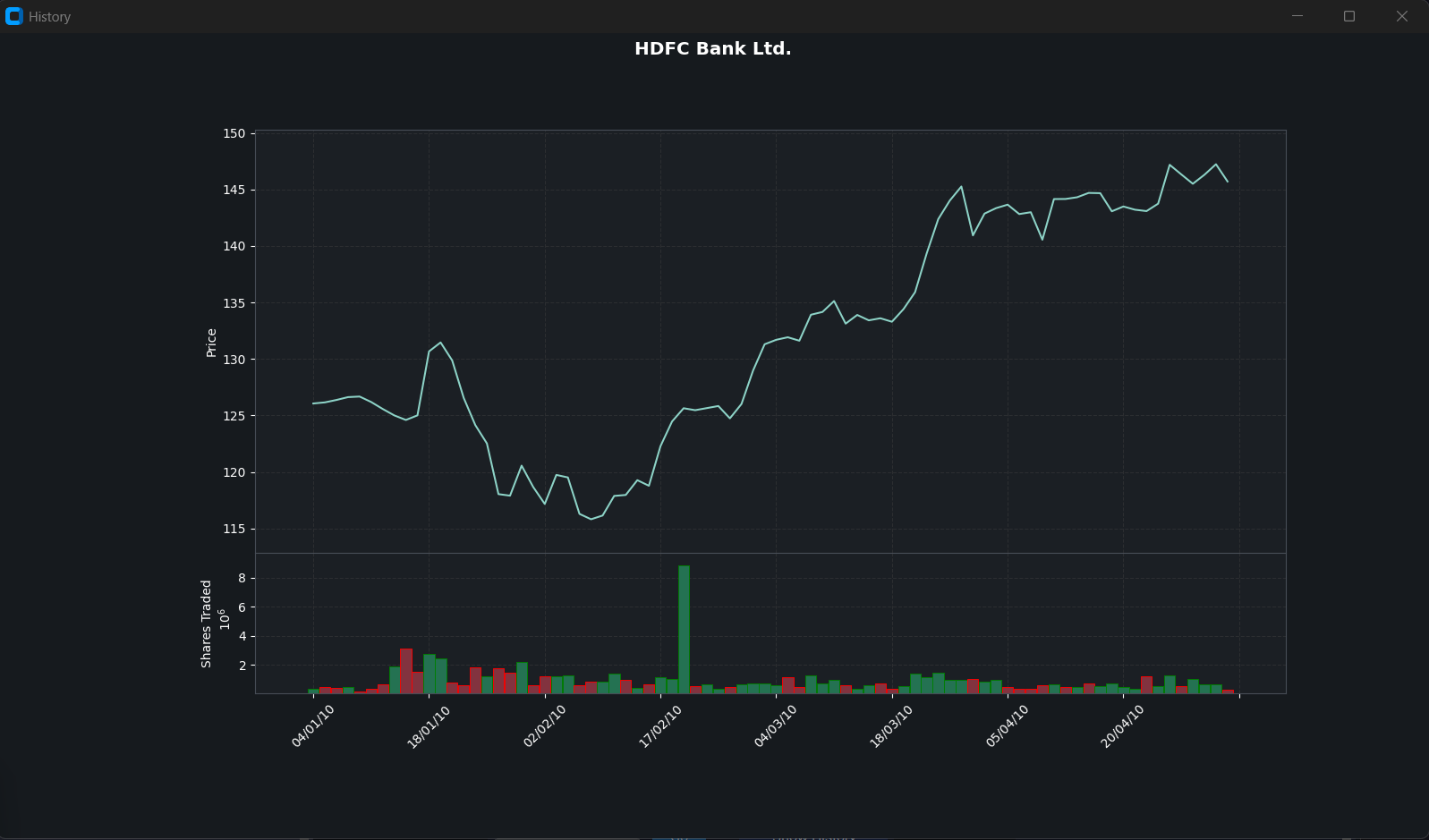
2024-11-22,1136.949951171875,1148.0,1128.25,1143.0,319597

2024-11-25,1155.300048828125,1164.5,1142.8499755859375,1155.5,944028

2024-11-26,1161.300048828125,1161.300048828125,1142.1500244140625,1143.9000244140625,32614

2024-11-27,1143.550048828125,1155.3499755859375,1134.0,1150.300048828125,56600

A screenshot of a computer

Description automatically generated 

A screenshot of a computer

Description automatically generated