Stock transaction processor

This code reads stock prices since the beginning of the experiment and transactions and shows each player's total.

The Function

We will create a function that takes as arguments file names of players, stocks, and transactions and returns a table with the players sorted in order of their portfolio performance.

We start by calling the function and displaying the resulting table.

Test Data

While the function will eventually support 101 stocks, dozens of players, and hundreds of transactions, we start with a small list of players, stocks and transactions.

`TestPlayers.csv`

This file contains three players. Normally we use email addresses for player names:

```
"email"

"Player1"

"Player2"

"Player3"
```

`TestStocks.csv`

This file contains the list of stocks that we'll use for all transactions.

```
AMZN

CSCO

META

NVDA
```

`TestTransactions.csv`

The transactions have four field in a CSV file

Timestamp,Email Address,Ticker,Type,Shares

```
3/4/2024 18:42:51,Player1,NVDA,Buy,11

3/6/2024 18:42:51,Player1,NVDA,Sell,11

3/7/2024 18:42:51,Player1,NVDA,Sell,11

3/8/2024 18:42:51,Player1,NVDA,Buy,50

3/5/2024 00:00:00,Player3,CSCO,Buy, 2

3/9/2024 18:42:51,Player1,NVDA,Buy,8
```

```
3/9/2024 18:42:51,Player2,NVDA,Buy,10

3/10/2024 18:42:51,Player3,NVDA,Buy,10

3/9/2024 18:42:51,Player1,AMZN,Buy,7

3/9/2024 18:42:51,Player2,AMZN,Buy,2

3/10/2024 18:42:51,Player3,AMZN, Buy,7

3/11/2024 00:00:00, notAPlayer,AMZN, Sell, 10

3/11/2024 00:00:00, Player3,AMZN, Sell, 5

3/19/2024 18:42:51,Player2,AMZN,Sell,1

3/19/2024 00:00:00, Player2, META, Sell, 7

3/20/2024 00:00:00, Player3, AMZN, Sell, 10
```

We call `processTransactions()` using the three files.

The `processTransactions()` function prints out the transactions and error messages. Then it returns the table with the players sorted in descending order of their portfolio value.

```
finalTable = processTransactions("TestPlayers.csv", "TestStocks.txt",
"TestTransactions.csv");
```

```
TRAN: Player1 wants to Buy 11.000 shares of NVDA for $9375.63 total
TRAN: Player1 wants to Sell 11.000 shares of NVDA for $9757.00 total
TRAN: Player1 wants to Sell 11.000 shares of NVDA for $10193.59 total
Player1 has 0.00 shares of NVDA and wants to sell 11.00
TRAN: Player1 wants to Buy 50.000 shares of NVDA for $43764.00 total
Player1 has 10381.37 dollars and needs 43764.00 to buy 50.00 shares of NVDA
TRAN: Player3 wants to Buy 2.000 shares of CSCO for $97.11 total
TRAN: Player1 wants to Buy 8.000 shares of NVDA for $7002.24 total
TRAN: Player2 wants to Buy 10.000 shares of NVDA for $8752.80 total
TRAN: Player3 wants to Buy 10.000 shares of NVDA for $8752.80 total
TRAN: Player1 wants to Buy 7.000 shares of AMZN for $1227.45 total
TRAN: Player2 wants to Buy 2.000 shares of AMZN for $350.70 total
TRAN: Player3 wants to Buy 7.000 shares of AMZN for $1227.45 total
Player3 has 1150.09 dollars and needs 1227.45 to buy 7.00 shares of AMZN
TRAN: notAPlayer wants to Sell 10.000 shares of AMZN for $1719.60 total
notAPlayer is not in the player list
TRAN: Player3 wants to Sell 5.000 shares of AMZN for $859.80 total
Player3 has 0.00 shares of AMZN and wants to sell 5.00
TRAN: Player2 wants to Sell 1.000 shares of AMZN for $175.90 total
TRAN: Player2 wants to Sell 7.000 shares of META for $3473.68 total
Player2 has 0.00 shares of META and wants to sell 7.00
TRAN: Player3 wants to Sell 10.000 shares of AMZN for $1781.50 total
Player3 has 0.00 shares of AMZN and wants to sell 10.00
```

```
format bank
finalTable
```

`finalTable = 3x2 table`

	players	CashValue
1	Player1	10509.47

	players	CashValue
2	Player2	10077.13
3	Player3	10065.67

The processTransactions() function

Now we define the function. It takes three file names as arguments and returns a table.

```
function finalTable = processTransactions(playerFile, stockFile,
transactionFile)
```

Read in the data

We store data that remains the same across many transactions as catagorical. This includes player names and stock ticker symbols.

```
    % Read the list of player emails
    data = readtable(playerFile,"TextType","string");
    players = categorical(data.email);

    % Read the list of stocks
    stocks = categorical(string(importdata(stockFile)));
```

Set up player data

As players buy shares of stock, we store their stock totals in a matrix. One row per player and one column per stock.

```
    % Zero out all player holdings of all stocks.
    holdings = zeros(numel(players), numel(stocks));
```

Everyone gets $10,000 to start.

```
    % Give each player 10,000 to start.
    cash = ones(numel(players),1) * 10000;
```

Read the stock history

We read all the stock price history for the stocks in the stock list from the beginning of the year to today. We use this data to determine profit, loss, and final value of each stock.

```
    % Get the prices from Yahoo if prices doesn't exist.  No sense rereading
    % them.
    prices = hist_stock_data('01012024',now,stockFile);
```

```
    % Read the list of transactions
    % Set up the Import Options and import the data
```

Read the transactions

We used the import tool to create code that sets up the transaction table. The `opts` variable holds all the import options.

```
    opts = delimitedTextImportOptions("NumVariables", 5);

    % Specify range and delimiter
    opts.DataLines = [2, Inf];
    opts.Delimiter = ",";

    % Specify column names and types
    opts.VariableNames = ["Timestamp", "EmailAddress", "Ticker", "Type",
"Shares"];
    opts.VariableTypes = ["datetime", "categorical", "categorical",
"categorical", "double"];

    % Specify file level properties
    opts.ExtraColumnsRule = "ignore";
    opts.EmptyLineRule = "read";

    % Specify variable properties
    opts = setvaropts(opts, ["EmailAddress", "Ticker", "Type"],
"EmptyFieldRule", "auto");
    opts = setvaropts(opts, "Timestamp", "InputFormat", "MM/dd/yyyy
HH:mm:ss", "DatetimeFormat", "preserveinput");
```

The readtimetable() function reads transactions. The function assumes that we have date/time data in the first column, which we do.

```
    % Import the data
    transactions = readtimetable(transactionFile, opts, "RowTimes",
"Timestamp");

    % Clear temporary variables
    clear opts
```

Clear the time of day data.

We don't care what time of day the transctions happen as we only look at stock prices at the end of day. The dateshift() function moves all the datetimes to the start of the day (00:00:00)

```
    transactions.Timestamp = dateshift(transactions.Timestamp,"start","day");
```

Processing Transactions

Now we'll loop through the transactions in the table row-by-row and read their data.

```
    for tn = 1:numel(transactions.Timestamp)
        tran = transactions(tn,:);         % get a table row.
        % Extract the transaction information
        % The x<variables> are the data for this xaction.
        xtype = tran.Type;
        xdate = tran.Timestamp;
        xplayer= tran.EmailAddress;
        xstock = tran.Ticker;
        xshares = tran.Shares;
        % find the cash associated with this transaction
        % The try/catch system tries to do an operation and then executes
        % the `catch` statements if the operation fails.  In this case, we
        % catch the case where there was no ticker that matched xstock.

        try
            stockStruct = prices({prices.Ticker} == xstock); % Get the table
row for the stock
        catch
            fprintf("%s is not in the price list\n", xstock )
            continue
        end
        datetest = stockStruct.Date == xdate;  % Read the vector of stock
dates from the table.
        % If the transaction date does not match any date in the stock
        % dates then it happened on a weekend or holiday.  Roll the date
        % back until you find data.
        while ~datetest
            xdate = xdate - days(1);
            datetest = stockStruct.Date == xdate;  % we will use datetest to
index prices
        end

        xprice = stockStruct.AdjClose(datetest); % Get the stock price for
the date using datetest
        xcash = xprice * xshares; % Get the cash for the transaction. # of
shares * price for the day.
        % Tell the user what the transaction says.
        fprintf("TRAN: %s wants to %s %.3f shares of %s for $%.2f total\n",
xplayer, xtype, xshares, string(xstock), xcash);
```

Execute the transaction

Now we have all the transaction data. We change the stock holdings and cash to match the transaction instructions.

First find the player's row in the cash and holdings matrices.

```
        playerRow = xplayer == players; % Find the index for the player in
the holdings and cash matrices

        if ~playerRow % catch a missing-player error
            fprintf ("%s is not in the player list\n", xplayer);
            continue
        end

        playercash = cash(playerRow);
```

Now find the column for the stock in the stocks array.

```
        stockCol = xstock == categorical(stocks);
        % Catch the missing stock error.
        if ~stockCol
            fprintf("%s does not exist in stock list\n", xstock)
            continue
        end
```

Are we buying or selling

If we are buying, check that we have enough cash to make the purchase. Then subtract the cash from the `cash` vector and add the stock holdings to the `holdings` array.

```
        if xtype == "Buy"
            if playercash < xcash
                fprintf("%s has %.2f dollars and needs %.2f to buy %.2f
shares of %s\n", string(xplayer), playercash, xcash, xshares, xstock);
                continue;
            end

            cash(playerRow) =cash(playerRow) - xcash;
            holdings(playerRow,stockCol) = holdings(playerRow,stockCol) +
xshares;
        else
```

If we are selling, check that the player has enough shares in the holdings array to make the sale. Then subtract the sale from `holdings` and add the player's cash to `cash`.

```
            holding = holdings(playerRow, stockCol);
            if xshares > holding
                fprintf("%s has %.2f shares of %s and wants to sell %.2f\n",
...
                    string(xplayer), holding, xstock, xshares);
                continue;
            end
```

```
            holdings(playerRow, stockCol) = holding - xshares;
            cash(playerRow) = cash(playerRow) + xcash;
        end
    end
```

Output each player's final value

We have processed all the transactions. It is time to find the cash value of the holdings using the most recent `adjClose` prices.

```
    stockVal = zeros(1, numel(stocks));
    % Loop through the AdjClose array and put the most recent price
    % into stockVal.
    for ii = 1:numel(stocks)
        stockVal(1,ii) = prices(ii).AdjClose(end);
    end
    finalTable = array2table(players);
    % Add a CashValue variable to finalTable
    finalTable.CashValue = zeros(numel(players),1);
    % Loop through the players.  Multiply the stock holdings by the latest
    % stoc value and store the sum of the stock values plus cash in the
    % CashValue variable for each player.
    for ii = 1:numel(players)
        playerStockVals = holdings(ii,:) .* stockVal;
        finalTable.CashValue(ii) = round(sum(playerStockVals)+cash(ii),2);
    end
    sortrows(finalTable,"CashValue", "descend");  % Sort the table before
returning it.
end
```