“Stock Analyzer” README

Summary:

“Stock Analyzer” is a Microsoft Visual Basic for Applications macro designed to perform basic analysis of large volumes of individual stock price data.

The macro will, for each sheet in a workbook, create two tables.

The first table will summarize data for all stocks on the table, indicating the absolute price change (with cells containing absolute price changes color-coded with a green background for positive, red background or negative, or gray for zero), percentage price change, and total volume traded for all entries on a sheet.

The second table will identify the stocks with the greatest percentage increase, greatest percentage decrease, and maximum trading volume, among all those found on the sheet.

Detailed Information:

Purpose: This macro is designed to be a prototype for more general analysis of instrument-generated data sets. These data sets tend to come in the form of data files with a header row, labeled key column, and data in several numeric columns. These data are often imported into MS Excel from text files. Stock price tables are an excellent example. Although for stock tables the functionality is more sophisticated than needed, the macro is written for easy adaptability to the more general instrument data sets.

Primary Use Case:

The primary use case is to analyze MS Excel workbooks in which stock price data for each calendar year of interest is provided as a separate worksheet in the workbook. The data is sorted by stock ticker label and includes, for each ticker symbol, one entry per trading date (including holidays). Each entry includes at least a date (in a numeric format), opening price, closing price, and trading volume. Intra-day high and low prices (along with any other information) is also allowed in the data table.

Expected Input:

The data table must contain a header row. The header row has column labels, corresponding to each column of data. The header row must be the row in the sheet which is just above the first row containing four or more numeric entries. (These entries are assumed to be the required numeric data, opening price, closing price, and volume information, although in practice any four or more numeric cells in the same row will mark the start of the table.) The macro will automatically identify the header row based on these criteria. If no header row can be identified, the user will be notified and analysis of that particular sheet will be skipped.

The macro will parse the column headings in the header row to determine which columns contain the data of interest. The macro will identify the four left-most columns (need not be contiguous), in which the first sequence of letters (a to z only, case-insensitive, ignoring anything else prior to the first letter) are “date”, “open”, “clos”, and “vol”. Thus, “<VoL283>” will be understood as volume, “@ Open Value” will be understood as the opening value, and “DATE in yyyymmdd format” will be understood as the date. “Price at open”, however, would not be understood, and “clo!!!s” would not be understood. The macro will also identify the left-most column in which the first sequence of letters (a-z only, case insensitive, ignoring anything else) is either “sym” or “tick” to identify the ticker symbol. The macro will expect this column to be sorted, such that all valid data rows for a given ticker symbol lies in a series of contiguous rows (invalid rows within the listing for a given ticker symbol are OK). If more than one block of data for a given ticker symbol exists on the sheet, the macro will treat the data as representing two or more separate ticker symbols. The macro does not check for repeated symbols. If the macro cannot identify the needed column headings, the user will be notified and the sheet will be skipped. Other columns will be ignored, so comments on lines are OK if they do not replace data in the columns of interest.

The macro will check each row below the header row for validity before analyzing it. To be valid, the ticker column must not be empty, the date must be a number, and the prices and volume must be numbers of zero or greater. Zero volumes and zero prices are accepted as valid because they typically indicate trading holidays. The opening price on a holiday can differ from the opening price on the subsequent trading day. For annual stock data in particular, the “annual change” is understood to be the difference between the closing price on 12/31 and the opening price on 1/1, which is a holiday. To correctly compute this quantity, the price data for holidays must be included. The validity check allows the user to enter a comment on any line of the table, which will be ignored if no data is present. If no valid rows are found on a sheet, the user will not be notified. The creation of the summary tables will simply not take place and the macro will move on to the next sheet.

The only non-data entries that may “confuse” the macro are numbers entered into columns on the header row or numbers in table rows not meant for data. Thi should only happen if there are four or more such cells with numbers on the same row. Before the header, the macro will interpret these as the start of the table, then fail to find the column headers needed, and so skip the sheet. Below the header row, only if four cells containing numbers happen to fall in the data columns of interest will the macro function incorrectly. Otherwise, the macro will see these numbers as part of an invalid row and convert the format of these numbers to non-numeric, in order to avoid miscalculating the quantities of interest. For example, a table row anywhere with the comment “1, 2, 3, 4, I declare thumb war” is fine if it does not span more than four cells. Writing the comment out as “1” in a cell, next to “2”, and so forth, will confuse the macro. The non-numeric forcing of extraneous data in the table should only matter if other calculations are being carried out on the sheet. For that reason, if you intend to do other calculations with data not meant to be included in the table, please don’t stick these numbers in and among the table entries.

Output:

There will be two output tables. The first, summarizing individual labeled data, will appear at row 1, two columns to the right of the last used column in each sheet. It will list the labels (ticker names), the absolute change, computed as the difference between the closing value on the date with the largest number, and the opening value on the date with the smallest value, the percentage change using the same difference divided by the opening value on the date with the smallest value (“n/a” is returned if this value is zero), and the sum of the volume entry for all dates in the block. The second, absolute change column, will have a color coded background – green for positive values, red for negative, and gray for zero.

The second output table will consist of only three rows below the headers. The first row will contain the ticker symbol (column 1) and value (column 2) of the stock (or label) wit the greatest positive percentage change. If no entries have a positive change, this will be noted. The second row will indicate the symbol and value of the greatest negative change, with the same formats and notes as the above row. The final row will indicate the symbol and value corresponding to the greatest total volume. This second table will begin at row 1, three columns to the right of the first table. In case of ties, the first stock in the output list having the desired value will be chosen. Because an actual tie is very unlikely, the macro will not check for ties and will not notify the user.

Procedural Outline:

For the purposes of understanding the code, the procedural outline is included here. The code pieces are identified in this outline (but not in the code itself) with letters and numbers (e.g. W1, F2a, and so on).

The script will (W) run a loop for every page in the workbook.

The script needs four pieces of information per stock: 1) ticker symbol, 2) opening price, 3) closing price, 4) volume, 5) date. To find (F) these it needs to F1) locate the header row, and F2) identify in the header row which columns have these four items.

Once done, it will L) loop through every row from the row just below the header to row past the end of the data on the sheet, keeping a record of the “current symbol” (meaning the identity of the ticker it is gathering data for), and which rows contain data for that ticker (the “active range”).

For each row, it will L1) validate the row by checking that the five expected pieces of data are valid.

L2) If all data are not valid, any numeric data occupying the cells containing expected data will be converted to text format, without altering cell contents (this is a safeguard against extraneous values messing up calculations).

For valid data, the loop will check the symbol key value in the current row and compare it the current ticker.

L3) When the loop encounters a symbol that matches the current ticcker, it will simply extend its active range downward.

L4) When it encounter a different symbol, it will assume it has come to the end of a block. It will L4a) process the current block by retrieving the appropriate information, L4b) perform calculations as needed, and L4c) output a row on the results table. It will then L4d) reset the active range to start at the current row and update the current ticker.

The loop will have special behavior for two cases: For the first valid row found, it will L5) set the current ticker, start the active range at the current row, and write the table headings. For the last row, it will behave as if it has found a new ticker symbol (a “blank” one), and process accordingly, one last time.

Once all rows are done (R), the script will R1) set the ranges in the output table where it needs to look for data R2) for each criterion in the summary table, the script R2a) will use a built-in method to search the appropriate range for the appropriate maximum or minimum value and its index, and R2b) output these to the summary table. These will be coded as separate events since there are only three.

Detailed steps:

W) main loop – use a “for each” sheet in workbook main loop

W.1 Call F1) locate header row and store value (separate macro function)

W.2 If F1 returns 0 then msg an error (sheetname has no numeric data!) and move to the next sheet by exiting the main for loop, we’ll need a “GoTo” for this. Although we could avoid it by clever loop design, the “for … each” loop + use of a good variable name (“SkipSheet”) will make the code much easier to follow than if we used a clever loop.

W.3 Call F2) locate columns and store values, this will be a separate macro function.

W.4 If any of the five needed columns has an index of 0, then msg an error (could not find all the needed data on sheetname) and GoTo SkipSheet.

W.5 Initialize key values. Use a counter for the output row as a loop controller also. Use header row and “UsedRange” command to set for loop with limits. It appears calling “.UsedRange” takes a lot of time, so the row and column attributes of “.UsedRange” will be determined at the start of the loop and used in variables, rather than queried repeatedly.

W5.1 Within the for loop, set up if .. then … else type branch with two possibilities. First branch, if the validity check (as separate macro function L1) is true, or the last row has been reached (do L3, L4, or L5), else do L2.

L2: (else branch – invalid row) Check the columns containing the date, open, close, and volume in the current row. If these contain numbers, set the format type to text to keep them from injecting spurious data into the calculations.

L3-L5: (if-true branch – valid row): This will be a three part if-elseif-else branch to cover the different cases.

L5: Special case of first valid row. If the output-row counter is still at its initial value (1), then this is the first time we’ve found a valid row. (If-true branch)

L5.1. Set current-ticker to value found in this row.

L5.2 Set block-start and block-end to the current row.

L5.3 Write the column headers for the output table.

L5.4 Set output-row to 2 (this will keep this code from running more than once, and update the pointer for where to write data)

L4. (else-if branch): If the ticker value in this row differs from current-ticker (we do it this way to make it more default-like behavior for odd cases)

L4.1 Output results table row (see sub-procedure in this outline) – in code, this will be inline as it is only used once, relatively simple, and would need a lot of parameter passing if made a sub-procedure

L4. 2 Initialize current-ticker with value from this row, reset block-start and block-end with current row, and increment output-row

L3. (else-branch) In this case we’re still in the same block, so

L3.1 Update block-end to current row (not a simple increment because we may have skipped rows due to invalid data).

W6. Use the output-row pointer and output-column pointers left over from writing output table to figure out each range, then write data

W6.1 -- If output-row > 1 (execute the rest of W6)

W6.2 – set ticker-range to rows 2 to output-row, column output-column, percent-change range to rows 2 to output-row, column output-column + 2, vol-range to rows 2 to output-row, column output-column + 3

W6.3 – set summary-column to index of first completely empty column + 2 (this should reflect the fact that the results table is no longer empty). Summary-column shows where to put the data (we know by default the summary table goes in rows 1-4, so no pointer is needed).

W6.4 – write column headers and row labels for the summary table

W6.5 -- fill in each row of the summary table – these will be inline code blocks because they are simple, combining R2a (compute results) and R2b (write results) into a single, integrated block R2.

-- Functions and sub-procedures. Functions will be coded as such, sub-procedures will be inline.

F1) locate header row – function – needs sheetname, returns header row index –use “with sheetname” loop for F1.1, F1.2, and F1.3

F1.1 – initialize: set default return to 0

F1.2 – Initialize # of used-rows and used-columns in sheet using .UsedRange

F1.3 – do a while loop through each used row and column

F1.3.1 – initialize row-index and col-index at 1, initialize a counter of columns that contain numbers (cols-with-nums) at 0

F1.3.2 – while cols-with-nums < 4 and row index < used-rows do

F1.3.2.1 if the cell at row-index, col-index is numeric, increment cols-with-nums

F1.3.2.2 increment col-index

F1.3.2.3 if col-index > used-columns, then

F1.3.2.3.1 Reset cols-with-nums to 0

F1.3.2.3.2 Reset col-index to 0

F1.3.2.3.3 Increment row-index

F1.4 – after while loop exit, if cols-with-nums > 3, then return row-index - 1

F2) locate columns with data – function – needs sheetname and header row, returns five column indices, use with-sheetname loop for F2.2

F2.1 – initialize: set the five column index return values (in a temp array) to 0

F2.2 – initialize: get the number of columns in sheet with .UsedRange

F2.3 – do a for loop through each column in reverse order

F2.3.1 -- if the cell at header-row, col-index is not empty then

F2.3.1.1 – put contents into a temp label-holder string

F2.3.1.2 – convert string to all lowercase

F2.3.1.3 – for loop trough all string characters 1 at a time

F2.3.1.3.1 – if the char is not like a-z, replace it with a space

F2.3.1.4 – use trim to remove all leading spaces from temp-holder

F2.3.2 – If the trimmed temp-holder contains text that starts with “open” set the index for open to the column value

F2.3.3 – if the trimmed temp-holder contains text that starts with “clos, set the index for close to the column value

F2.3.4 – if the trimmed temp-holder contains text that starts with “tick” or “symb, set the index for the ticker label to the column value

F2.3.5 – if the trimmed temp-holder contains text that starts with “vol”, set the index for volume to the column value

F2.3.6 – if the trimmed temp-holder contains text that starts with “date”, set the index for date to the column value

F2.4 – return the five column index values

L1) validate row -- Function, row is valid, needs sheetname, rownumber, five column indices, returns Boolean flag

L1.1 -- if ticker column is not empty, then ticker is valid

L1.2 – if open column contains a zero or positive number, then open is valid

L1.3 – if close column contains a zero or positive number, then close is valid

L1.4 – if volume column contains a zero or positive number, then volume is valid – should be integer but will take decimals

L1.5 – if date column contains a number, then date is valid – will not check range matching

L1.6 – return value = true if all five components are true

L4.1) Results table builder, procedure, needs sheetname, output-row, output-column (start), current-ticker symbol, block-start, block-end, and column indices – will not build as separate procedure, just an inline block

L4.1.1 – Write current-ticker to cell at output-row, output-col

L4.1.2 – define date-range for analysis, for rows use block-start and block-end, for cols, use the column index of date

L4.1.3 – use max worksheet function to find max date value in date-range

L4.1.4 – use match worksheet function to find index of row with max date value (max-index)

L4.1.5 – use min worksheet function to find min date value in date-range

L4.1.6 -- use match worksheet function to find index of row with min date value (min-index)

L4.1.7 – set start-val to cell value at row block-start + min-index – 1, col open-col-index

L4.1.8 – set end-val to cell value at row bllock-start + max-index – 1, col close-col-index

L4.1.9 – set change to end-val minus start-val

L4.1.10 – set value of cell at output-row-index, output-col- + 1 to change

L4.1.11 – If Change > 0 then output-row cell for change . color = green

L4.1.12 – elseif Change < 0 then output-row cell for change . color = red

L4.1.13 – else output-row cell for change . color = gray

L4.1.14 – if start-val > 0

L4.1.14.1 – percent-change = change / start val, format as %

L4.1.14.2 – set value of cell at output-row, output-col + 2 to percent-change

L4.1.14.3 – else set value of cell at output-row, output-col + 2 to “n/a”

L4.1.15 – set vol-range to rows block-start to block-end, column index for volume

L4.1.16 – total-volume = sum of volume cells in vol-range

L4.1.17 – set value of cell at output-row, output-column + 3 to total-volume

R2 – fill in the summary table – needs ticker-range, percent-change-range, volume-range (where to search), summary-column (where to write)

R2.1 -- max-increase = max of percent-change-range

R2.2 -- if max-increase > 0 then

R2.2.1 -- max-increase-index = index of max-increase in percent-change-range

R2.2.2 -- set cells(2, summary-column + 1) to value of cell at index max-increase-index in ticker-range (write the corresponding ticker symbol)

R2.2.3 -- set cells(2, summary-column + 2) to value of max-increase, formatted as percent

R2.3 – else write that no increases were found at cells(2, summary-column + 1)

R2.4 -- max-decrease = min of percent-change-range

R2.5 -- if max-decrease < 0 then

R2.5.1 --max-decrease-index = index of max-decrease in percent-change-range

R2.5.2 -- set cells(2, summary-column + 1) to value of cell at index max-decrease-index in ticker-range (write the corresponding ticker symbol)

R2.5.3 -- set cells(2, summary-column + 2) to value of max-decrease, formatted as percent

R2.6 – else write that no decreases were found at cells(2, summary-column + 1)

R2.7 -- max-volume = max of volume-range

R2.8 -- max-volume-index = index of max-volume in volume-range

R2.9 -- set cells(2, summary-column + 1) to value of cell at index max-volume-index in ticker-range (write the corresponding ticker symbol)

R2.10 -- set cells(2, summary-column + 2) to value of max-volume