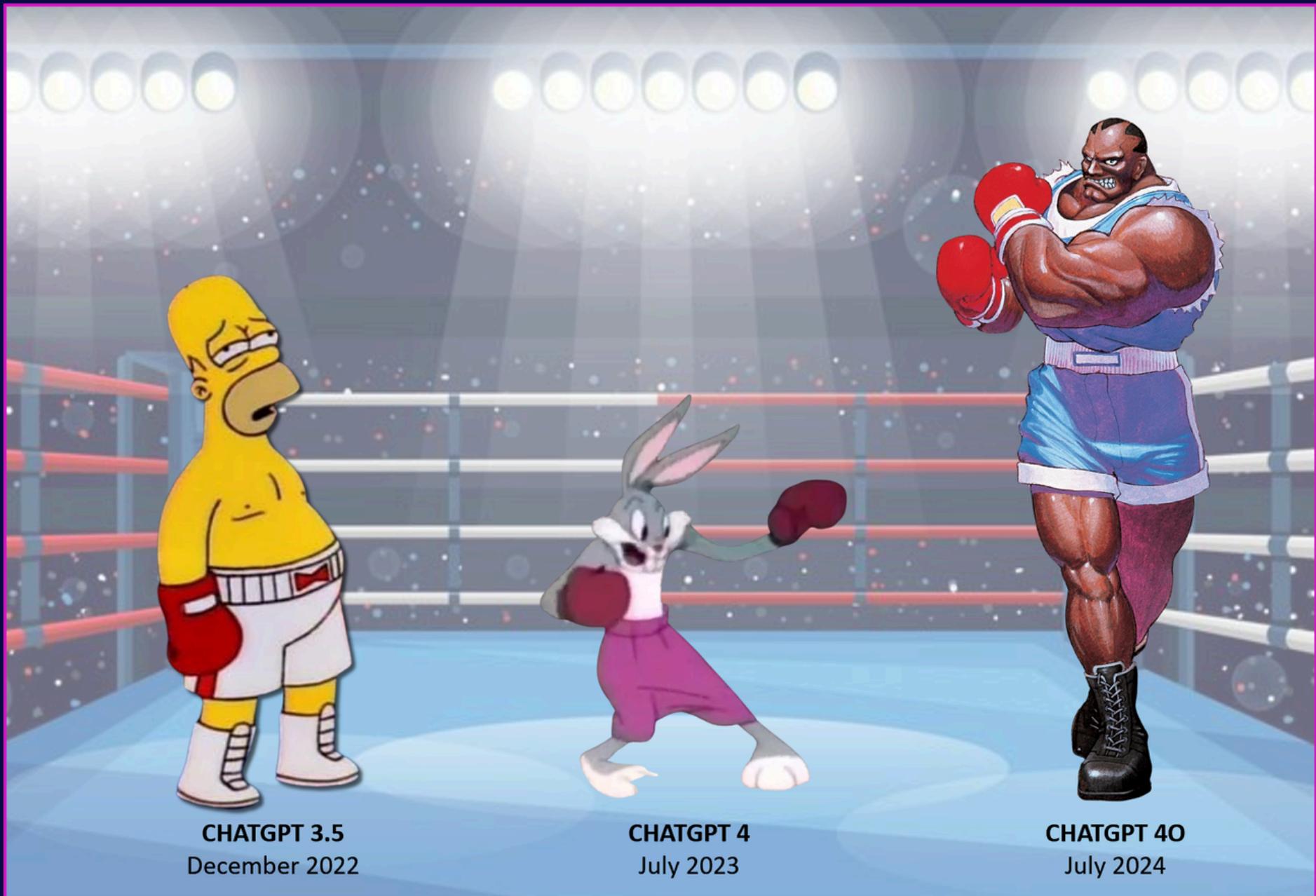


HOW GOOD IS GPT4O AT POWER QUERY/M CODING RELATIVE TO PREVIOUS MODELS?



SPOILER ALERT: Pretty #@&^ing good...



Brian Julius

INTRODUCTION

ChatGPT was initially released at the end of Nov 2022. Since early Dec 2022, I have been regularly testing its coding capabilities in a number of the languages most frequently used by Data Analysts and Data Scientists for data cleaning, transformation, and analysis - DAX, M, R, and Python

Over the next weeks, I will be performing in-depth assessments of current AI frontier model capabilities in these four languages (I'll leave SQL and Excel to others, since I don't use them extensively in my own work)

I wanted to revisit M first since I had done two previous assessments of ChatGPT's abilities with M, and also because I hypothesize AI will be the least capable with M for a number of reasons:

- M is a fairly complex language with nearly 800 different defined functions, as well as the ability to create custom user-defined functions
- Of the languages I'll be assessing, by far the least has been written about M
- Much of the highest-quality M codebase is behind logins or paywalls, and thus is not part of the standard AI training corpus
- In my prior testing, GPT 4 performed markedly better on Python and R, compared to M



VS



PREVIOUS ASSESSMENTS

Here are summaries of the two previous assessments of ChatGPT's ability to write M code that I conducted 17 and 12 months ago respectively, following the releases of ChatGPT 3.5 and Advanced Data Analysis (ADA)

HOW GOOD IS CHATGPT AT WRITING POWER QUERY/M CODE?



CHATGPT VS EXCEL BI CHALLENGE

Brian Julius, 2022

December 2022

ChatGPT 3.5

<https://bit.ly/GPT35vsExcelBI>

HOW GOOD IS CODE INTERPRETER AT DATA CLEANING?



CODE
INTERPRETER

VS



DATA
CLEANING
NINJAS



Brian Julius

July 2023

ChatGPT 4/ADA

<https://bit.ly/DataCleanNinjas>

Bottom Line:

“Despite going 0 for 3, I think it shows tremendous promise for the future. The AI in each case clearly understood the framing of the problem, and provided logic that with some additional training in the choice and use of M functions would be sufficient to produce an accurate solution.”

So, don't get rid of your M Reference Guide and cheatsheets yet, but that day doesn't seem too far off... “

Bottom Line:

“Given that we each logged successful solutions on both challenges, and split on speed vs. accuracy, I'm going to call this battle a draw. But despite my 1-0-1 record vs. ChatGPT on data cleaning, I think it will be the favorite going into our third fight”



Brian Julius

EVALUATION APPROACH

My approach for testing GPT4O's ability to code in M was to revisit each of the five problems I had used for prior testing and examine whether GPT4O produced materially better solutions

Because each of these problems was behind a login, I am confident that these were not included in the training dataset for GPT4O, even though they were created before the training cutoff

In prompting GPT 4O, I provided only the text of the challenge problem, the dataset and a request for GPT 4O to provide the M code that would accurately produce the desired result

Each problem had a max of 10 points to be awarded if GPT 4O provided code that ran without error and accurately produced the entire desired result

One point was subtracted for each round of reprompting needed to produce the desired result

For reprompting, I decided to limit my reprompts only to observational statements that could be made by a Power Query user with no M expertise (e.g., the code provided produced the following error message..., the results for row 1 and 3 are incorrect, etc)

I have developed a [series of specialized custom GPTs](#), including one for M code. To test whether the GPT made a material difference in the result, after achieving the base GPT 4O result, I restarted ChatGPT (so there would be no memory of the prior run), initiated the custom GPT and re-ran the exact same prompt that base GPT 4O was given

All of the verbatim chat transcripts and results from the 10 runs (5 problems, each run twice (with and without custom GPT) are available in [my Github repo](#)



PROBLEM 1

EXCEL BI CHALLENGE #107

The image shows a woman with dark hair and glasses, wearing a black blazer over a white top. She is holding a pen and a small notepad. To her left is a table with data and a large blue cloud containing instructions. The table has columns: From, To, Count, Min, and Max. The data is as follows:

From	To	Count	Min	Max
1	100	9	11	99
28	178	15	33	171
78	499	42	88	494
234	655	41	242	646
1000	20000	190	1001	19991

Palindrome numbers are those which are same even when read from backwards. We need not consider single digit numbers for this purpose. Hence, 1 to 9 will not be considered as Palindromes.

Example - 44, 121, 5665.

Find Count, Min and Max Palindromes for the given range of

Excel BI ANALYTICS
IN PARTNERSHIP WITH
ENTERPRISE DNA
EMPOWERING DATA. ENABLING TO CHANGE THE WORLD

Analysis of Problem

I would classify [this challenge](#) as an intermediate level problem. In a four star difficulty rating system where ★ represents a problem solvable by a beginner and ★★★★ represents a problem that can only likely be solved by an expert, I would rate this as ★★

It cannot be solved solely from the Power Query UI - custom M code is required to create a list of numbers between “From” and “To”, and then some simple code has to be written to convert each number to a text list and test whether that list reads the same forwards and backwards

The user then needs to perform a GroupBy on the results of that code to derive the required Count, Min and Max figures



Brian Julius

PROBLEM 1

RESULTS AND ANALYSIS

GPT 4O Result - 9 points

GPT 4O provided code that fully ran on its first attempt. The only problem was that GPT 4O failed to take note that single digit numbers should not be included as palindrome numbers. This was easily correct on the first reprompt (below)

Did custom GPT improve the result?

No. It also provided a strong solution, but made the same error on the single digits initially, so was unable to improve on the base model's performance

```
1 let
2   Source = Excel.CurrentWorkbook(){[Name = "Table1"]}[Content],
3   // Function to check if a number is a palindrome and exclude single digits
4   IsPalindrome = (n as number) as logical =>
5     let
6       str    = Text.From(n),
7       revStr = Text.Reverse(str)
8     in
9       str = revStr and Text.Length(str) > 1,
10  // Function to get palindromes within a range
11  GetPalindromes = (from as number, to as number) as record =>
12    let
13      Numbers    = List.Numbers(from, to - from + 1),
14      Palindromes = List.Select(Numbers, each IsPalindrome(_)),
15      Count      = List.Count(Palindromes),
16      Min        = if Count > 0 then List.Min(Palindromes) else null,
17      Max        = if Count > 0 then List.Max(Palindromes) else null
18    in
19      [Count = Count, Min = Min, Max = Max],
20  // Apply the GetPalindromes function to each range
21  AddPalindromes = Table.AddColumn(Source, "PalindromeData", each GetPalindromes([From], [To]))
22  ExpandPalindromes = Table.ExpandRecordColumn(
23    AddPalindromes,
24    "PalindromeData",
25    {"Count", "Min", "Max"})
26
27 in
28 ExpandPalindromes
```

Analysis of Results

Strong start for GPT 4O. Solution is well-coded, clearly documented. Mildly surprising that both the base model and custom GPT failed to pick up initial instructions to exclude single digit numbers, but quickly corrected

ROUND 1	GPT 3.5	GPT 4O
Points	0	9
Total Points	0	9
% of Max Pts	0%	90%



Brian Julius

PROBLEM 2

EXCEL BI CHALLENGE #105

The image shows a woman with dark hair, smiling, standing behind a whiteboard. On the whiteboard, there is a screenshot of an Excel spreadsheet. The spreadsheet has a header row with columns labeled 'DAY' and 'COUNT'. Below this, there are nine rows of data showing counts for Sunday through Saturday. Row 1 contains the value '100' in cell A1. The data is as follows:

DAY	COUNT
Sun	15
Mon	13
Tue	15
Wed	14
Thu	14
Fri	15
Sat	14

To the right of the whiteboard, there is a text box containing the following instructions:

Find the counts of weekday names of Christmas during last n years mentioned in A1. I have shown the output for last 100 years. A1 can vary.

In the bottom right corner of the whiteboard, there is a logo for "Excel BI ANALYTICS" and "ENTERPRISE DNA".

Analysis of Problem

Another [intermediate level problem](#), not dissimilar to Problem 1, with the main difference being that instead of receiving a dataset it has to generate one on its own based on the context and logic of the problem. Difficulty: ★★

This one also cannot be solved entirely from within the Power Query UI. Much of it can be, but custom M code is required to create the year list and the custom sort order



[in](#) Brian Julius

PROBLEM 2

RESULTS AND ANALYSIS

GPT 4O Result - 9 points

GPT 4O provided code that produced accurate results on its first try. The only problem was that GPT 4O had a minor syntax error in the implementation of the custom day sort order. This was easily correct on the first reprompt

Did custom GPT improve the result?

Yes. It provided a perfect solution on the first attempt (below), including correct implemetnion of the custom sort order, garnering the full 10 points

```
1 let
2 // Generate a list of years from 1923 to 2022
3 Years = {1923 .. 2022},
4 // Create list of Christmas dates for each year
5 ChristmasDates = List.Transform(Years, each #date(_, 12, 25)),
6 // Convert the list of dates to a table
7 ChristmasTable = Table.FromList(ChristmasDates, Splitter.SplitByNothing(), {"Date"}),
8 // Add a column for the day of the week for each date
9 AddDayOfWeek = Table.AddColumn(
10    ChristmasTable,
11    "DayOfWeek",
12    each Date.DayOfWeekName([Date], "en-US")
13 ),
14 // Group by the day of the week and count occurrences
15 GroupedByDay = Table.Group(
16    AddDayOfWeek,
17    {"DayOfWeek"},
18    {"Count", each Table.RowCount(_, Int64.Type)}
19 ),
20 // Sort the results by the order of days in a week starting with Sunday
21 DaysOfWeekOrder = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"},
22 AddSortOrder = Table.AddColumn(
23    GroupedByDay,
24    "SortOrder",
25    each List.PositionOf(DaysOfWeekOrder, [DayOfWeek])
26 ),
27 SortedTable = Table.Sort(AddSortOrder, {"SortOrder", Order.Ascending}),
28 // Keep only the necessary columns and rename them
29 Result = Table.SelectColumns(SortedTable, {"DayOfWeek", "Count"}),
30 RenamedResult = Table.RenameColumns(Result, {"DayOfWeek", "Day"}),
31 // Extract the first three letters of each day
32 FinalResult = Table.TransformColumns(RenamedResult, {"Day", each Text.Start(_, 3), type text})
33 in
34 FinalResult
```

Analysis of Results

GPT 4O had no trouble at all with this problem - not surprising given its similarity to Problem 1 and how well GPT 4O performed on that one. Major improvement over GPT 3.5, which hallucinated a non-existent List.Grouup function and never got back on track



PROBLEM 3

EXCEL BI CHALLENGE #91

	A	B	C	D	E	F	G	H	I
1	EAN-13 Numbers	Expected Answers							
2	9780141026626	9780141026626							
3	2109876543215	2109876543210							
4	2109876543210	7983102091035							
5	9535208896664	8153309486531							
6	3982698831779								
7	7983102091035								
8	3301801522897								
9	4003736953687								
10	8153309486531								
11	2920020576070								
12									
13									
14									
15									
16									
17									

Provide a formula to list down all valid EAN-13 numbers. EAN-13 numbers are 13 digits and last digit is checksum digit. To calculate checksum digit

1. Take first 12 digits
2. Sum all digits at odd positions (starting from left)
3. Sum all digits at even positions. Multiply by 3 the result of sum at even positions.
4. Sum the results of step 2 and 3.
5. Take the last digit i.e. unit digit of the answer of step 4.
6. If this is 0, then 0 is checksum otherwise 10 minus Result of step 5 is checksum.

This checksum digit has to be equal to 13th digit for a valid EAN-13 number

Analysis of Problem

I like [this problem](#) a lot, since it demonstrates how relatively simple custom M code can be used to powerfully clean and validate dirty data. There is nothing particularly difficult about this problem, but it does require carefully following directions, making it a an excellent test for AI. Difficulty: ★★

This one also cannot be solved entirely from within the Power Query UI. Separating the digits of the string into odd and even postions can be done in a myriad of different ways, but I don't believe there are any approaches that don't require some level of custom M coding

When GPT 3.5 attempted this problem during my initial evaluation, it went off the rails at steps 2 and 3, rendering all of the subsequent calculations moot



PROBLEM 3

RESULTS AND ANALYSIS

GPT 4O Result - 9 points

GPT 4O initially provided code that did not run due to a very simple syntax error. That error was quickly corrected on a simple reprompt that just provided GPT 4O a screenshot of the error message. The resulting solution is provided below

Did custom GPT improve the result?

No. The GPT also initially provided code containing a simple syntax error that was corrected easily on the first reprompt. Thus, while the custom GPT version provided an excellent solution, it did not improve on the result achieved by the base model

```
1 let
2   // List of EAN-13 numbers
3   EANList = [
4     "9780141026626", "2109876543215", "2109876543210", "9535208896664", "3982698831779",
5     "7983102091035", "3301801522897", "4003736953687", "8153309486531", "2920020576070"
6   ],
7   // Function to validate an EAN-13 number
8   ValidateEAN = (ean as text) as logical =>
9     let
10       first12Digits = Text.Start(ean, 12),
11       oddSum = List.Sum(
12         List.Transform(
13           List.Select(List.Positions(Text.ToList(first12Digits)), each Number.Mod(_, 2) = 0),
14           each Number.FromText(Text.Middle(first12Digits, _, 1)))
15         ),
16       evenSum = List.Sum(
17         List.Transform(
18           List.Select(List.Positions(Text.ToList(first12Digits)), each Number.Mod(_, 2) = 1),
19             each Number.FromText(Text.Middle(first12Digits, _, 1)))
20         )
21       )
22       * 3,
23       totalSum = oddSum + evenSum,
24       checksumDigit = Number.FromText(Text.End(ean, 1)),
25       calculatedChecksum = if Number.Mod(totalSum, 10) = 0 then 0 else 10 - Number.Mod(totalSum, 10),
26       isValid = checksumDigit = calculatedChecksum
27     in
28       isValid,
29   // Filter list of EAN-13 numbers to only include valid ones
30   ValidEANS = List.Select(EANList, each ValidateEAN(_))
31 in
32 ValidEANS
```

Analysis of Results

GPT 4O completes the initial assessment problems, going 3 for 3 on a problem set that 3.5 failed to log any correct solutions on, and was just a couple of minor errors (quickly corrected on reprompt) away from a perfect score

ROUND 3	GPT 3.5	GPT 4O
Points	0	9
Total Points	0	28
% of Max Pts	0%	93%



Brian Julius

PROBLEM 4

EXCEL BI CHALLENGE #85

The diagram illustrates a data transformation challenge. On the left is a 'PROBLEM' table showing monthly revenue and cost data for four companies (MSFT, Berkshire, Exxon) across five departments (Sales, Marketing, HR, IT, Purchasing). On the right is a 'RESULT' table summarizing total annual revenue, cost, and profit for each company. A blue thought bubble in the center contains the instruction: 'Generate the result table from problem table.' Red arrows point from the 'PROBLEM' table to the 'RESULT' table.

Company	Dept	2019 Revenue	2019 Cost	2020 Cost	2020 Rev	2021 Cost	2021 Rev	2022 Rev	2022 Cost
MSFT	Sales	314	965	489	548	455	889	725	336
	Marketing	866	305	476	606	735	628	350	380
	HR	893	942	304	873	318	456	461	538
	IT	758	147	239	458	725	894	254	769
	Purchasing	437	821	634	845	596	415	745	596
Berkshire	Sales	678	444	719	244	557	351	835	222
	Marketing	117	319	949	752	251	862	494	367
	IT	527	696	474	326	202	397	440	283
	Purchasing	678	427	270	426	661	429	980	921
	Exxon	Sales	199	600	695	502	101	552	843
Marketing		664	431	594	761	636	490	199	416
HR		314	629	983	542	532	412	550	162
Supply Chain		668	252	252	743	787			
Logistics		784	270	269	617				

Company	Total Revenue	Total Cost	Total Profit
MSFT	12415	10770	1645
Berkshire	8536	7762	774
Exxon	11121	11210	-89
Grand Total	32072	29742	2330

Generate the result table from problem table.

Analysis of Problem

This is an excellent test problem for a number of reasons:

- It's a practical type of data transformation exercise
- It's got multiple solution "paths" - some specific to this dataset, which will break if additional companies are added in the future, while other solution approaches are fully dynamic and can handle the addition of any additional companies and/or rows
- It's a well-designed problem with some intentional traps incorporated (e.g., null entries that will cause a problem in an unpivot, inconsistent designations for revenue) that make it an interesting test for AI

Difficulty: ★★

When I tested this problem initially, the complexity of the problem combined with the intentional "booby traps" that Vijay Verma built into the problem in total were too much for GPT 4 at the time to overcome, and it was unable to produce a valid solution



PROBLEM 4

RESULTS AND ANALYSIS

GPT 4O Result - 10 points

GPT 4O did not have the problems that GPT 4 did on my initial testing, coming right out of the gate with a solution that ran without error and produced the desired result. The resulting solution is provided below

Did custom GPT improve the result?

Yes and no. It should be noted that while the solution that the base model provided did produce the desired result, that code is not dynamic and will break if more years of data are added to the dataset.

The custom GPT seemed to recognize this, and took a better, potentially fully dynamic route, but got stuck in the mud over the “Revenue” vs “Rev” trap in the problem. In reality, that issue is easy for the prompter to spot and steer the GPT to resolving that, but that option was not within the rules of my testing methodology

```
1 let
2   Source = Excel.CurrentWorkbook(){[Name = "Table116"]}{Content},
3   //# Filled Down" = Table.FillDown(Source, {"Company"}),
4   // Unpivot data
5   //#Unpivoted Other Columns" = Table.UnpivotOtherColumns(
6   "#Filled Down",
7   {"Company", "Dept"}, 
8   "Attribute",
9   "Value"
10 ),
11 // Pivot the unpivoted data
12 "#Pivoted Column" = Table.Pivot(
13   "#Unpivoted Other Columns",
14   List.Distinct("#Unpivoted Other Columns"[Attribute]),
15   "Attribute",
16   "Value",
17   List.Sum
18 ),
19 // Calculate Total Revenue, Total Cost, and Total Profit
20 "#Added Custom" = Table.AddColumn(
21   "#Pivoted Column",
22   "Total Revenue",
23   each [#"2019 Revenue"] + [#"2020 Rev"] + [#"2021 Rev"] + [#"2022 Rev"]
24 ),
25 "#Added Custom1" = Table.AddColumn(
26   "#Added Custom",
27   "Total Cost",
28   each [#"2019 Cost"] + [#"2020 Cost"] + [#"2021 Cost"] + [#"2022 Cost"]
29 ),
30 "#Added Custom2" = Table.AddColumn(
31   "#Added Custom1",
32   "Total Profit",
33   each [Total Revenue] - [Total Cost]
34 ),
35 // Group by Company
36 "#Grouped Rows" = Table.Group(
37   "#Added Custom2",
38   {"Company"},
39   {
40     {"Total Revenue", each List.Sum([Total Revenue]), type number},
41     {"Total Cost", each List.Sum([Total Cost]), type number},
42     {"Total Profit", each List.Sum([Total Profit]), type number}
43   }
44 ),
45 // Add Grand Total
46 "#Added Grand Total" = Table.InsertRows(
47   "#Grouped Rows",
48   Table.RowCount("#Grouped Rows"),
49   {
50     [
51       Company = "Grand Total",
52       Total Revenue = List.Sum("#Grouped Rows"[Total Revenue]),
53       Total Cost = List.Sum("#Grouped Rows"[Total Cost]),
54       Total Profit = List.Sum("#Grouped Rows"[Total Profit])
55     ]
56   }
57 )
58 in
59 "#Added Grand Total"
```

Analysis of Results

While GPT 4O performed well here, I think it exposed some issues in my testing methodology that I will need to revisit in future assessments. I think the perfect score awarded overstates the performance of the base model, and the strict constraints on prompting understate the value of the GPT. However, the assessment does capture the overall improvement of GPT 4O relative to GPT the general impro

ROUND 4 GPT 3.5/4		GPT 4O	
Points	0	10	
Total Points	0	38	
% of Max Pts	0%	95%	



Brian Julius

PROBLEM 5

ENTERPRISE DNA WORKOUT #10

You work on a detailed sales report showing all sales transactions and also returns transactions. As you can see each line of the sales report has a unique Sales ID, so you can not match the returns and their corresponding sales by the ID_Sales value. You can identify the returns by the negative number in the Qty column and the objective is to find the same product with the same positive Qty and eliminate it in the final result. As a basic condition you can return goods only after the sales, so you need to make sure that the date of the sales line you take out is before the date of the return line. If you have multiple possible sales matches try to take the sales line with the nearest date to the return date.

Your task: Eliminate all lines with returns and their corresponding sales lines. If possible, eliminate the sales line with the latest date.

ID_Sales	Date	ID_Product	Product	Qty	Unit Value	Value Total
15896	3/1/2023	1	Adidas Superstar	3	500	1,500
15897	3/2/2023	1	Adidas Superstar	5	500	2,500
15898	3/3/2023	3	Vans Old Skool	5	400	2,000
15899	3/4/2023	4	Puma Suede	5	500	2,500
15900	3/5/2023	5	Reebok Classic Leather	4	400	1,600
15901	3/6/2023	1	Adidas Superstar	-5	500	-2,500
15902	3/7/2023	1	Adidas Superstar	5	500	2,500
15903	3/8/2023	5	Reebok Classic Leather	4	400	1,600
15904	3/9/2023	5	Reebok Classic Leather	4	400	1,600
15905	3/10/2023	10	Skechers D'Lites	3	400	1,200
15906	3/11/2023	11	Salomon Speedcross	5	800	4,000
15907	3/12/2023	5	Reebok Classic Leather	2	400	800
15908	3/13/2023	13	Merrell Moab	1	600	600
15909	3/14/2023	10	Skechers D'Lites	3	400	1,200
15910	3/15/2023	5	Reebok Classic Leather	-4	400	-1,600
15911	3/16/2023	16	Saucony Kinvara	1	700	700
15912	3/17/2023	17	On Cloud	1	800	800
15913	3/18/2023	10	Skechers D'Lites	-3	400	-1,200
15914	3/19/2023	19	Dr. Martens 1460	1	1,000	1,000
15915	3/20/2023	1	Adidas Superstar	-5	500	-2,500
15916	3/21/2023	5	Reebok Classic Leather	-4	400	-1,600
15917	3/22/2023	1	Adidas Superstar	5	500	2,500

ID_Sales	Date	ID_Product	Product	Qty	Unit Value	Value Total
15896	44986	1	Adidas Superstar	3	500	1,500
15898	44988	3	Vans Old Skool	5	400	2,000
15899	44989	4	Puma Suede	5	500	2,500
15900	44990	5	Reebok Classic Leather	4	400	1,600
15905	44995	10	Skechers D'Lites	3	400	1,200
15906	44996	11	Salomon Speedcross	5	800	4,000
15907	44997	5	Reebok Classic Leather	2	400	800
15908	44998	13	Merrell Moab	1	600	600
15911	45001	16	Saucony Kinvara	1	700	700
15912	45002	17	On Cloud	1	800	800
15914	45004	19	Dr. Martens 1460	1	1,000	1,000
15917	45007	1	Adidas Superstar	5	500	2,500

Analysis of Problem

This problem was developed by Power Query/M expert Matthias Friedmann, based on a real-world project he had consulted on. He characterized it as an expert-level ★★★★ problem and IMO it is unquestionably the most difficult of the five problems analyzed in this assessment. It took me a few hours to correctly solve it, and to date I think there have only been seven fully correct solutions submitted worldwide

Given the difficulty of the problem and some of the “traps” that Matthias intentionally wired into the formulation of the problem to make it even more difficult, I was very surprised when I initially tested this problem that GPT 4 was able to successfully generate a Python solution to the problem, though it was not successful in accurately translating that solution to M code



in Brian Julius

PROBLEM 5

RESULTS AND ANALYSIS

GPT 4O Result - 9 points

Very surprisingly, given the difficulty level of this problem, GPT 4O had little trouble with it. The initial solution it provided did run, but did not provide the desired results. After a minimal reprompting, consisting of a screenshot and a message that the results were not correct, GPT 4O provided well-documented code that generated exactly the results desired

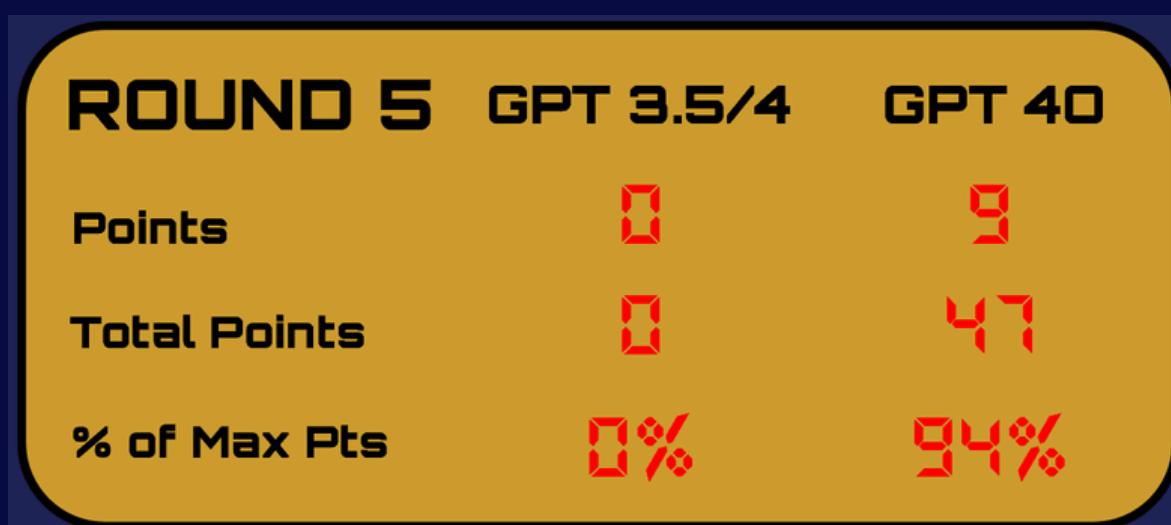
Did custom GPT improve the result?

No. This was an instance where the GPT took a more complex approach based on nested table joins, rather than more straightforward approach the base model applied. Despite seven rounds of reprompting, the custom GPT was unable to resolve the problems with the nested joins and ultimately timed out and crashed when repromted for an eighth time

```
1 let
2 Source = Excel.CurrentWorkbook(){[Name = "Data"]}[Content],
3 ChangedType = Table.TransformColumnTypes(
4 Source,
5 {
6     {"ID_Sales", Int64.Type},
7     {"Date", type date},
8     {"ID_Product", Int64.Type},
9     {"Product", type text},
10    {"Qty", Int64.Type},
11    {"Unit Value", Int64.Type},
12    {"Value Total", Int64.Type}
13 },
14 ),
15 // Separate sales and returns
16 Sales = Table.SelectRows(ChangedType, each [Qty] > 0),
17 Returns = Table.SelectRows(ChangedType, each [Qty] < 0),
18 // Remove the negative sign from Qty in Returns
19 ReturnsAbs = Table.TransformColumns>Returns, {"Qty", Number.Abs}),
20 // Add an Index column to both tables for stable sorting
21 SalesWithIndex = Table.AddIndexColumn(Sales, "SalesIndex", 1, 1, Int64.Type),
22 ReturnsWithIndex = Table.AddIndexColumn>ReturnsAbs, "ReturnsIndex", 1, 1, Int64.Type),
23 // Expand returns with the matching sales rows
24 ExpandedReturns = Table.AddColumn(
25 ReturnsWithIndex,
26 "MatchingSales",
27 each Table.SelectRows(
28 SalesWithIndex,
29 (s) => s[ID_Product] = [ID_Product] and s[Qty] = [Qty] and s[Date] < [Date]
30 ),
31 type table [
32     ID_Sales = Int64.Type,
33     Date = Date.Type,
34     ID_Product = Int64.Type,
35     Product = Text.Type,
36     Qty = Int64.Type,
37     Unit Value = Int64.Type,
38     Value Total = Int64.Type,
39     SalesIndex = Int64.Type
40 ],
41 ),
42 // Add a column with the latest matching sales row
43 LatestMatchingSales = Table.AddColumn(
44 ExpandedReturns,
45 "LatestSales",
46 each
47     if Table.RowCount([MatchingSales]) > 0 then
48         Table.First(Table.Sort([MatchingSales], {[Date], Order.Descending}), 1){0}
49     else
50         null
51 ),
52 // Remove rows where there is no matching sale
53 RemovedEmpty = Table.SelectRows(LatestMatchingSales, each [LatestSales] <> null),
54 // Flatten the table to include the necessary columns from the matching sales
55 Flattened = Table.ExpandRecordColumn(
56 RemovedEmpty,
57 "LatestSales",
58 {"ID_Sales", "Date", "ID_Product", "Product", "Qty", "Unit Value", "Value Total", "SalesIndex"},
59 {
60     "SalesID",
61     "SalesDate",
62     "SalesID.Product",
63     "SalesProduct",
64     "SalesQty",
65     "SalesUnitValue",
66     "SalesValueTotal",
67     "SalesIndex"
68 },
69 ),
70 // Remove return transactions and corresponding sales transactions
71 FilteredSales = Table.SelectRows(
72 SalesWithIndex,
73 each not List.Contains(Flattened{SalesIndex}, [SalesIndex])
74 ),
75 // Remove the Index column from the sales table
76 FinalTable = Table.SelectColumns(
77 FilteredSales,
78 {"ID_Sales", "Date", "ID_Product", "Product", "Qty", "Unit Value", "Value Total"}
79 )
80 in
81 FinalTable
```

Analysis of Results

GPT 4O's quick solution on a complex, difficult problem was very impressive IMO. This problem also provided some useful insights on the custom GPT performance that reinforced my observations re: its performance in the prior problem (see next section for further discussion)



[in](#) Brian Julius

CONCLUSIONS AND NEXT STEPS

GPT 4O has “Made the Leap” in Terms of M Coding Ability

Recently, Wharton Professor and AI expert Ethan Mollick wrote “AI isn't good enough at a task, until it suddenly becomes good enough at a task”.

That pattern is quite evident here, with GPT 3.5/4 going 0 for 5, while GPT 4O went 5 for 5 on the same problem set, and fell just a few minor (and quickly corrected) coding errors short of a perfect score. In addition, even though I manually coded all 5 solutions correctly, GPT 4O coded the solutions more than two orders of magnitude faster than I did (even accounting for reprompting)

A Knowledgeable Human in the Loop is Still Essential

While GPT 4O coded M at a very high level, knowledgeable oversight still is required to evaluate both the accuracy and “durability” of the solutions provided (i.e., are they sufficiently dynamic or will they break and require recoding as more rows and/or columns are added to the dataset?)

Custom GPTs are Helpful, but Not Always Better than the Base Model

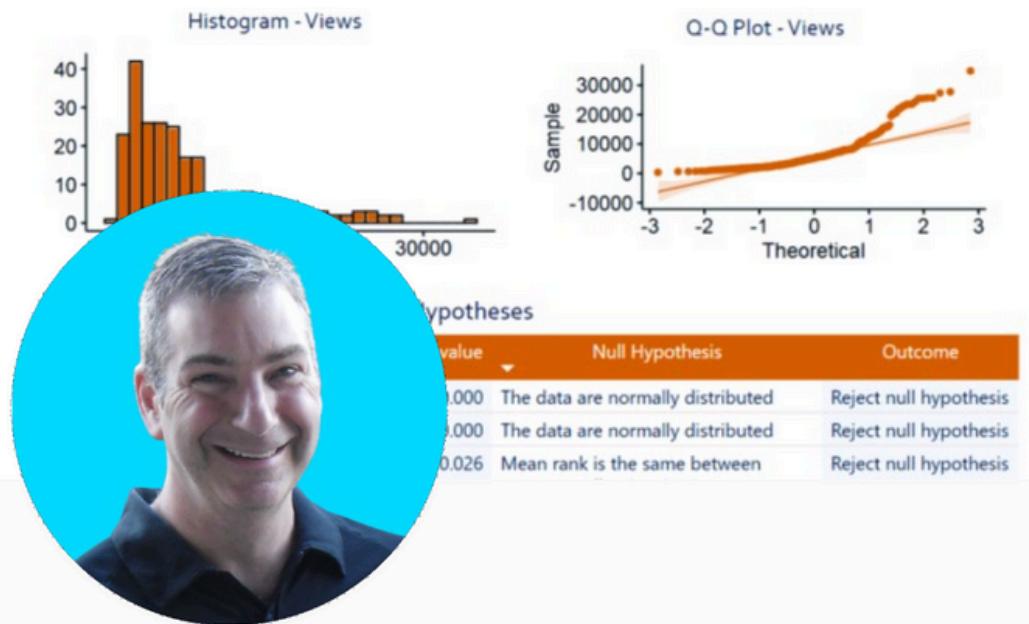
This was an unexpected finding. My custom M GPT contains over 2,000 pages of expert knowledge very unlikely to be available to GPT 4O .

Despite this extensive add'l info, in two cases the use of the custom GPT caused GPT 4O to adopt a more complex solution approach that it was unable to fully execute. Rather than retreating to a simpler approach, in both instances the GPT continued to spin its wheels, ultimately timing out before reaching a valid solution (which the base model was able to do). More testing on this issue is needed to determine if a less restrictive prompting approach would have resulted in better outcomes

These Findings Potentially Have Major Implications re: Other Languages

As discussed in the Introduction, there are numerous reasons to believe that of the most commonly used languages by Data Analysts, AI will be the LEAST competent in M. Though more testing in M (on higher difficulty problems) and in other languages is needed, this analysis provides some evidence that if frontier AI models are not already more capable coders than all but top-tier expert Data Analysts, then they soon will be





“

WITHOUT DATA, YOU'RE JUST ANOTHER PERSON WITH AN OPINION

W. EDWARDS DEMING

I post nightly on LinkedIn on:

- Data Analysis and Data Science
- Power BI, R and Power Query
- AI
- Career Development and Mentoring
- Leadership and Management

Liked this post?

Want to see more?

Ring the on my Profile

