AI Assistant Report

Group Members:

- Krylova Alena
- Dudic Mateja
- Saavedra Triana Erwin Omar
- Maringer Kelvin

Python Versions used:

- 3.13
- 3.14

Contributions:

- Krylova Alena:
  - Dataset overview
  - Data quality check
  - Additional insights (4-5)
- Dudic Mateja:
  - Data analysis **I** (until (excl.) “Visualizations”)
- Saavedra Triana Erwin Omar:
  - Data preprocessing
  - Additional insights (1-3)
- Maringer Kelvin:
  - Data analysis **II** (starting from (incl.) “Visualizations”)

Foreword

Throughout the entire document, the data will be described as though it were a real (not synthetic / generated) dataset, and the findings are actually representative for the usage of AI.

In reality, this dataset would be pretty meaningless as the limited number of rows would not grant a big enough sample-size to accurately determine real-world usage.

It is, therefore, also assumed that the data is directly proportional to real usage. This is done as to avoid, talking about the limited number of entries and other factors of this dataset again and again. (It would get pretty stale to repeat that for every analysis)

ALSO: within the **assistant_model** column, there is a value called “mini”. There is no indication of what model this refers to. Within this analysis, we are going to assume that it means “GPT-4o mini”

Dataset Overview

The daily AI assistant usage behaviour dataset captures real-world patterns of how users interact with AI assistants throughout their daily activities. It provides insights into when, how, and for what purposes people used AI tools, as well as session characteristics and user satisfaction.

The dataset is published on the Kaggle platform and is intended for researchers, developers, and data science practitioners interested in user behaviour analysis, personalization systems, recommendation engines, and conversational AI. It covers a wide range of AI usage scenarios, including learning, productivity, rese-arch, and routine daily tasks.

Task:

Get the number of columns and rows, provide the list of features with their meaning and data type.

Solution:

The function `data.info()` was used to get the information about the dataset – number of columns and rows, data types

Result:

The dataset contains 300 rows and 8 columns.

Features (their meaning and data types):

- 1st column: **timestamp** - date and time when the interaction with an AI tool started, data type - string
- 2nd column: **device** - type of device which was used to access an AI tool (desktop, mobile, smart speaker), data type - categorical (string)
- 3rd column: **usage_category** - for what purpose the user used an AI tool (education, daily tasks, research and etc), data type - categorical (string)
- 4th column: **prompt_length** - lenght of the user`s prompt (measured in characters), data type - integer
- 5th column: **session_length_minutes** - duration of the session in minutes, data type - float

- 6th column: **satisfaction_rating** - user satisfaction score from 1 to 5, data type - integer
- 7th column: **assistant_model** - which AI assistant model was used during the session, data type - categorical(string)
- 8th column: **tokens_used** - number of tokens used during the session, data type – integer

Most features from the dataset are categorical, making the dataset suitable analysing patterns and user behaviour segmentation (for example, feature **'timestamp'** allows to see if people use AI tools more often on weekdays or weekends, in the mornings or in the evenings).

To obtain a statistical summary of the numerical features, the `describe()` method was used. It provided key statistics such as mean, standard deviation, minimum and maximum values, as well as quartiles. It allows to better understand distribution of data.

Some observations from the `describe()` *function:*
The average prompt length is 129 characters; it indicates that users often submit detailed prompts.
The average session duration is about 7.7 minutes, indicating that most interactions with the AI assistant are relatively short.
The average satisfaction rating is close to 3 (on a scale from 1 to 5), which shows users` experience in general is neutral (or positive).
Token usage varies significantly, showing the differences in query complexity.

Data quality check

Task:

Investigate the dataset for missing values.

Solution:

`data.isnull().sum()` was used to calculate the number of missing values.

Result:

in the result, the dataset contains no missing values.

Task:

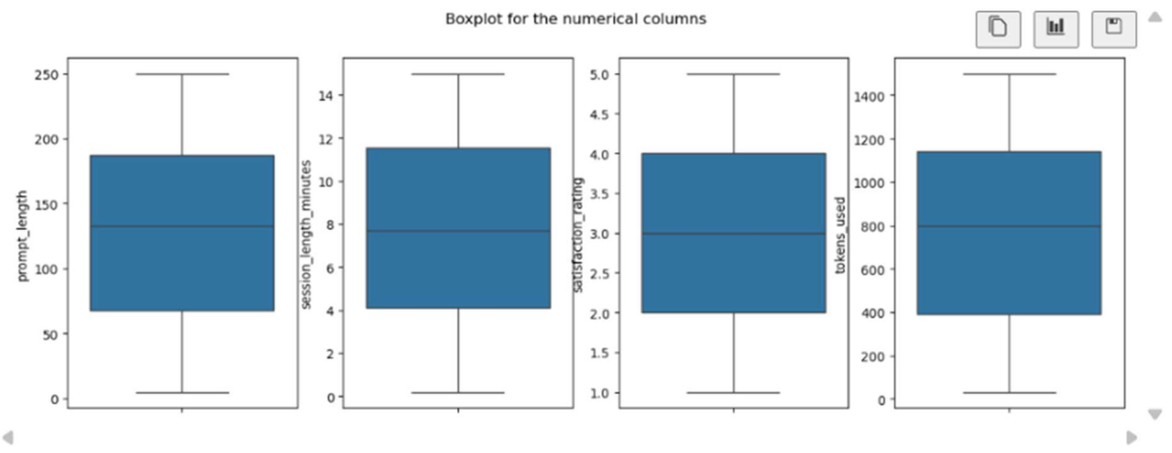
Investigate the dataset for outliers.

Solution:

boxplots were used for detection and visualization of numerical values in the dataset. We can`t use boxplots for categorical values, so to check whether categorical values contain outliers, the function `data['name_of_column'].value_counts()` was used.

Result:

From the boxplots we can see that there are no outliers in the numerical values of this dataset. From the result of function `value_counts()` we can see how many times unique values appeared in each categorical column. As no unusual or extremely rare entries were detected, we can make a conclusion, that the dataset contains no outliers in categorical values too.



Overall, the dataset is complete, consistent, and contains no apparent outliers or missing values, making it suitable for further analysis

Data preprocessing

Task:

Handle missing values and outliers appropriately

Solution:

For this task I did nothing

Result:

There were no outliers in this dataset so there was nothing to be handled.

Visualization:

Nothing to see

Task:

Create a new column **timeOfDay**, which bins the timestamp into morning, afternoon, evening, and night

Solution:

For this task I create a function to help me determine the moment of the day based on a number , Also I decided to take advantage of the fact that the datatype of **timestamp** was string and It was the whole in a correct form to slice the string and take specifically the numbers for hours , convert it from string to integer and then put it in the function to determine the moment of the day , I used the function of `apply()` in the **timestamp** column with the function I created to create this new column with the proper time of the day

Result:

I created a new column with values related to the time of the day in the timestamp

Visualization:

I add the visualization in the next task to show it together

Task:

Create a new column **year** which comes from **timestamp**.

Solution:

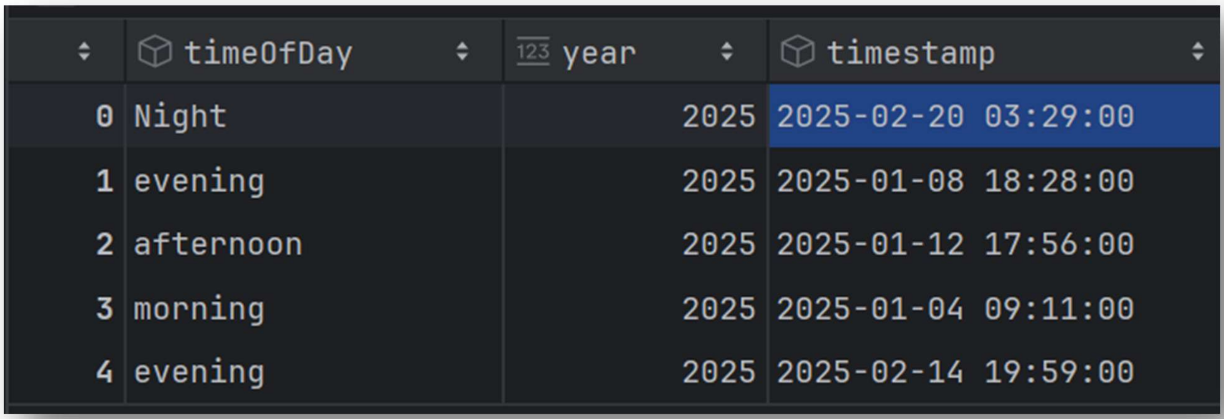
I repeat something very similar to the last task , taking advantage of the string nature of the **timestamp** column and that is clean and kind of ready to convert to a timestamp type , I slice the first elements of the string , the 20** part of the string that they all share(and is 2025 in all of them btw) and set it to be the year value.

Result:

A new column was created and everything worked as intended.

Visualization:

Here I also add the visualization for the last task too, they can be shown in one image



	timeOfDay	year	timestamp
0	Night	2025	2025-02-20 03:29:00
1	evening	2025	2025-01-08 18:28:00
2	afternoon	2025	2025-01-12 17:56:00
3	morning	2025	2025-01-04 09:11:00
4	evening	2025	2025-02-14 19:59:00

Task:

Convert data types where needed

Solution:

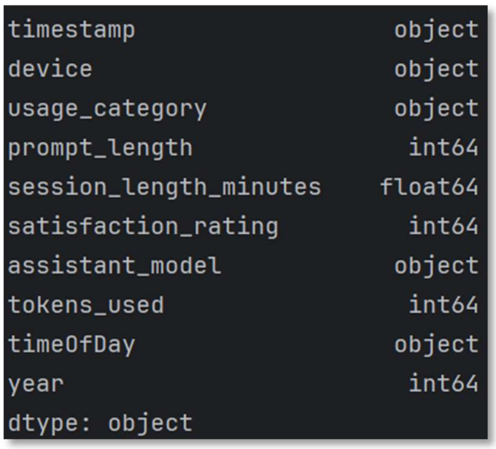
I execute the `astype()` function for most of the columns to convert to the proper data type for them. The non-numeric values mostly became categorical and the numerical values kind of stayed the same. However, for **timestamp** I used `pd.toDatetime()` function to convert the string to the Datetime data type.

Result:

All the columns were converted to their proper data types without any problem.

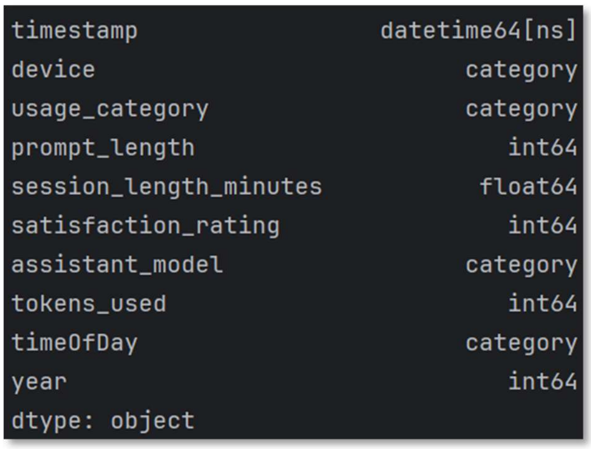
Visualization:

Before:



```
timestamp                  object
device                     object
usage_category             object
prompt_length               int64
session_length_minutes    float64
satisfaction_rating         int64
assistant_model            object
tokens_used                 int64
timeOfDay                  object
year                        int64
dtype: object
```

After:



```
timestamp                 datetime64[ns]
device                          category
usage_category                  category
prompt_length                      int64
session_length_minutes           float64
satisfaction_rating                int64
assistant_model                 category
tokens_used                        int64
timeOfDay                       category
year                               int64
dtype: object
```

Data Analysis I

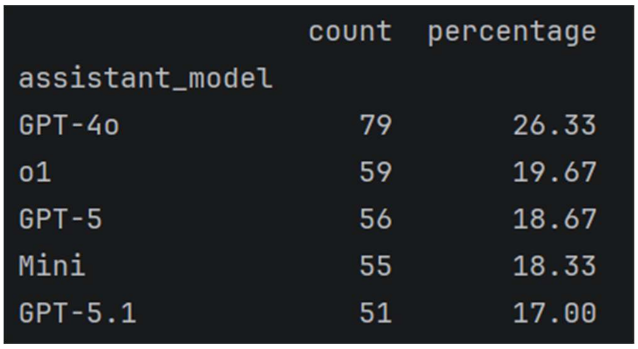
Task:

How many different AI assistants are in the dataset? Show counts and percentages.

Solution:

1. All the assistant models were selected with the variable assistant by selecting the column **assistant_model**.
2. With the variable count we selected how many times each model was found in the dataset using `value_counts()` because this function counts the number of occurrences of each model separately.
3. With the variable percentage we calculated the percentage by dividing the variable count with the total number of assistants in the dataset calculated with `.count()` and multiplying all of that by 100 (percentages) and rounding the result.
4. The table was made using `pd.DataFrame` and the two columns are count and percentage variables while the rows are the assistants

Result: There are 5 different AI models in the dataset, and the table shows their count and percentages



```
                 count  percentage
assistant_model
GPT-4o              79       26.33
o1                  59       19.67
GPT-5               56       18.67
Mini                55       18.33
GPT-5.1             51       17.00
```

Task:

What is the average session length per assistant? Are there noticeable differences?

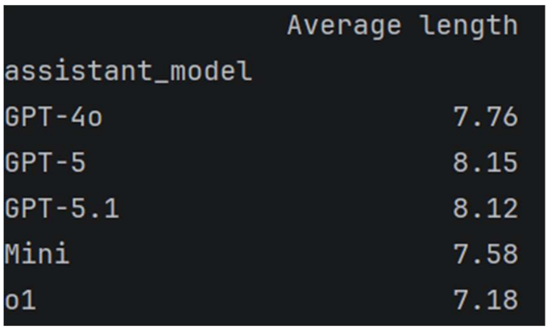
Solution:

1. With **average_session_length** we grouped the data by each assistant model with `groupby('assistant model')` and calculated the average session length for each model by selecting the column **session_length_minutes** and using `.mean()` and also we rounded the result.

2. With **average_session_length_output** we made a `DataFrame` with the output column being **average_session_length** and rows are the assistants.

Result:

The output is a table with rows being the assistant models and the column being the average length of a session. Average session length is relatively similar for all AI models, with the highest being GPT-5 at 8.15 and the lowest being o1 at 7.18 minutes.



```
                Average length
assistant_model
GPT-4o                    7.76
GPT-5                     8.15
GPT-5.1                   8.12
Mini                      7.58
o1                        7.18
```

Task:

Are specific assistants used more often for certain tasks (e.g., education)? If so, why do you think specific devices are used more often for these tasks?

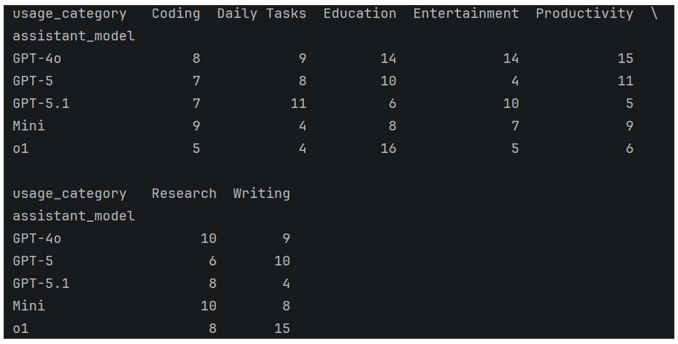
Solution:

To see which tasks are used per each assistant we will use a pivot table because it is the best way to lay out such data for multiple datapoints from a single column

1. with **usage_per_assistant** we created a pivot table using `pd.pivot_table`, with `index='`**assistant_model**`'` we made the rows the assistants, `columns='`**usage_category**`'` made the columns each datapoint in the **usage_category** column, with `aggfunc='count'` we made the function of the pivot table counting and `values='`**timestamp**`'` is the column we are counting (basically any column just used for counting)

Result:

A table where each row is one assistant and the columns are each task from **usage_category**. An interesting observation is that o1 is used the most for Writing and Education compared to other categories, most likely due to better reasoning than other models. In general, the most used models are the three GPT models, with GPT-4 being the most consistently used of the three



```
usage_category   Coding  Daily Tasks  Education  Entertainment  Productivity  \
assistant_model
GPT-4o                8            9         14             14            15
GPT-5                 7            8         10              4            11
GPT-5.1               7           11          6             10             5
Mini                  9            4          8              7             9
o1                    5            4         16              5             6

usage_category   Research  Writing
assistant_model
GPT-4o                 10        9
GPT-5                   6       10
GPT-5.1                 8        4
Mini                   10        8
o1                      8       15
```

Task:

Which type of tasks have the longest average prompt size and use time?

Solution:

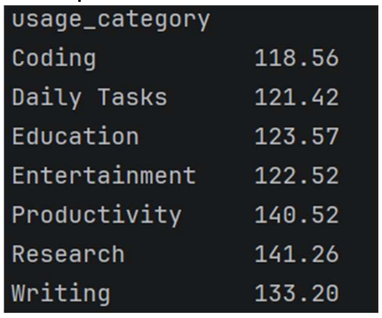
1. With **longest_avg_prompt** we grouped the data by **'usage_category'** (the tasks) and calculated the average prompt length per each task by selecting the column **'prompt_length'** and using `.mean()` and rounding the result with `.round(2)`.

2. With **longest_avg_time** we grouped the data by **'usage_category'** (the tasks) and calculated the average time per each task by selecting the column **'session_length_minutes'** and using `.mean()` and rounding the result with `.round(2)`.

Result: The output is two tables each has one column, one **prompt_length** and one **session_length_minutes**, with the rows being individual tasks.

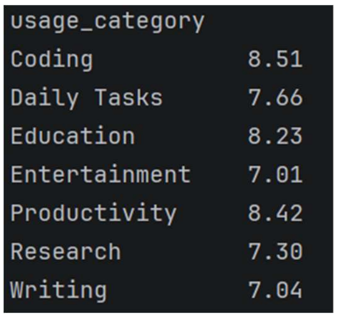
The longest average prompt length is in Research with 141.26 characters on average, while the longest average session length is for coding with 8.51 minutes

Prompt:



```
usage_category
Coding           118.56
Daily Tasks      121.42
Education        123.57
Entertainment    122.52
Productivity     140.52
Research         141.26
Writing          133.20
```

Time:



```
usage_category
Coding           8.51
Daily Tasks      7.66
Education        8.23
Entertainment    7.01
Productivity     8.42
Research         7.30
Writing          7.04
```

Task:

Is there a pattern between the task type and the time of day it was used?

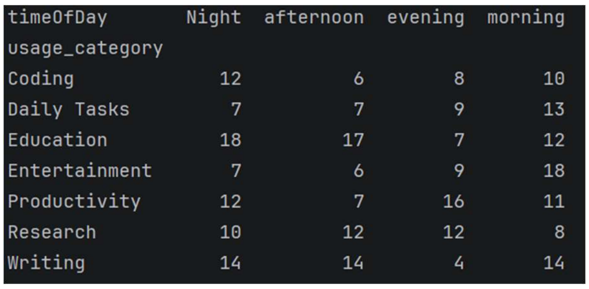
Solution:

1. Here we will again make a pivot table to be able to see for each datapoint (time of day) which task was used most often.

usage_category_per_timeOfDay here we make a pivot table using `pd.pivot_table`, with `index='usage_category'` we made the rows the tasks, with `columns='timeOfDay'` we make the columns the different times of day, with `aggfunc='count'` we make the function counting and we count the **'timestamp'** column (`values='timestamp'`, any other not selected column could be used).

Result:

The output is a table with rows being the tasks and columns the times of day with the values of the table being the number of uses of each task per time of day. Most categories have a specific time of day during which they are the least used. For example, education is least used at night, while writing is the least used during the evening.



```
timeOfDay       Night  afternoon  evening  morning
usage_category
Coding             12          6        8       10
Daily Tasks         7          7        9       13
Education          18         17        7       12
Entertainment       7          6        9       18
Productivity       12          7       16       11
Research           10         12       12        8
Writing            14         14        4       14
```

Task:

Have some AI assistants become more popular over time? Analyze their occurrence for each year. Plot their average number of usages per year as a scatter plot.

Solution:

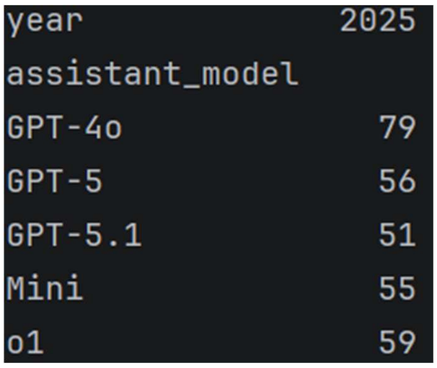
To analyze occurrence per year we will create a pivot table to show each year and then we will make a graph of each year and the assistants.

1. With the **assistant_model_per_year** variable we make a pivot table using `pd.pivot_table`, index (row) are the assistant models (column '**assistant_model**'), columns are the years (column '**years**'), `aggfunc='count'` the function is counting, and we are counting the timestamp column (`value =`'**timestamp**').
2. Since we cannot use a pivot table to make a graph we will make a new table **usage_per_year**. Here we will group the data by '**assistant_model**' and '**year**' and we will count the number of rows with a specific '**assistant_model**' and '**year**' value using `size()` (we can use `count()` too the only difference is that `count()` counts a single columns non-null values and `size()` counts the number of rows). `reset_index(name='count')` makes the table ready for graphing by converting the indexes ('**assistant_model**' and '**year**' columns) into normal columns. `name=`'**count**' gives a name to the new column created.
3. We make the canvas for the scatterplot with `plt.figure`. In this example `figsize=(10,7)` means the canvas is 10*7 units in size
4. We use the scatterplot function of seaborn to make a plot. `data=`**usage_per_year** tells us which table the data is based on, x axis is year, y axis is the number of uses, `hue=`'**assistant_model**' gives colors to **assistant_models**.

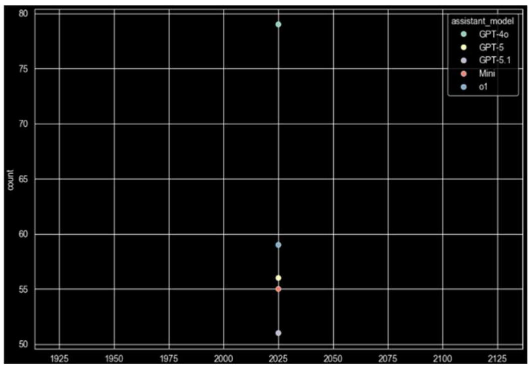
Result:

The result is a pivot table with the columns being each year and the rows being the models, and the data is the number of uses. The other output is the scatter plot of that table.

Not much we can read from the graph since there is only one year given, which is 2025. The most used model is GPT-4.



```
year             2025
assistant_model
GPT-4o             79
GPT-5              56
GPT-5.1            51
Mini               55
o1                 59
```



Data Analysis II

Task:

Visualizations:

- Plot distributions of key features using histograms, KDE plots, and boxplots.
- Use Color to distinguish individual assistants.

Solution:

To create Visualizations, the features to be visualized must be selected first.

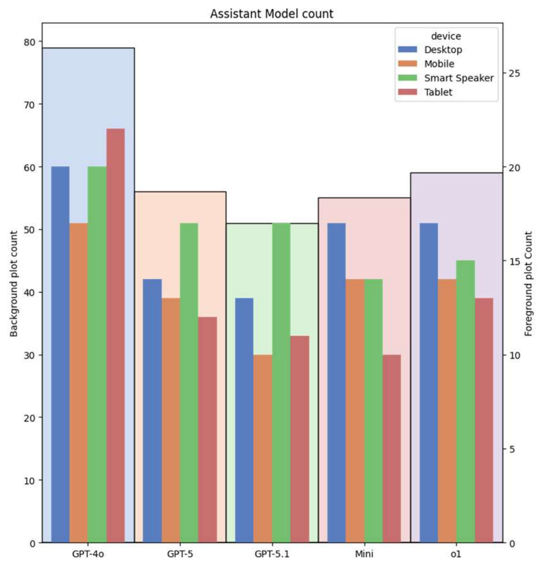
This was done through plotting a lot of different feature combinations and selecting the most interesting / insightful ones.

All plots / graphs are styled in a relatively similar fashion, using functions like “`[plot.Axes].set_y/xlabel() | .set_title()`”

A helper function “`conv_axis_txt()`” (originally: “`nicer_dicer()`”) was created to remove underscores (_), Capitalization of first words and generally formatting the axis titles (from “**assistant_model**” to “Assistant Model”).

Histograms:

After the features were selected, the histograms were made. First, a group of 3 plots are defined via the `subplots()` function, which are then filled with the histograms of the selected features (via the `sea.histplot()` function). These histograms have then been expanded via the `[plot.Axes].twinx()` function, to overlay a second plot onto the first (`countplot()`). This was done to show the distribution of e.g. devices or AI models within a category. The example below shows this more clearly:



The Plot in the background shows the total assistant model count, to get a feel for what people are using more (or rather what is most represented within the dataset). The foreground plot shows the breakdown of these counts into device types for each category.
This foreground / background plot style of presentation was chosen as it looks relatively cool and compresses much information into an easy-to-read graph.

A function "`auto_overlay_scaler()`" has been created to dynamically scale the foreground plot to fit within the background plot. This is just a stylistic choice.

Furthermore, the function "`cts_histplot()`" has been created to generate these plots.

KDE plots:

For the KDE plots, a helper function "`cts_kdeplot()`" was created to give all KDE plots a similar style and to cut down on unnecessary repetition.

The Y labels of these plots have been removed, because they do not really show anything of value, and the point is to show the rough distribution of the features.

Apart from that, there is really nothing else going on, that is worth mentioning.

Boxplots:

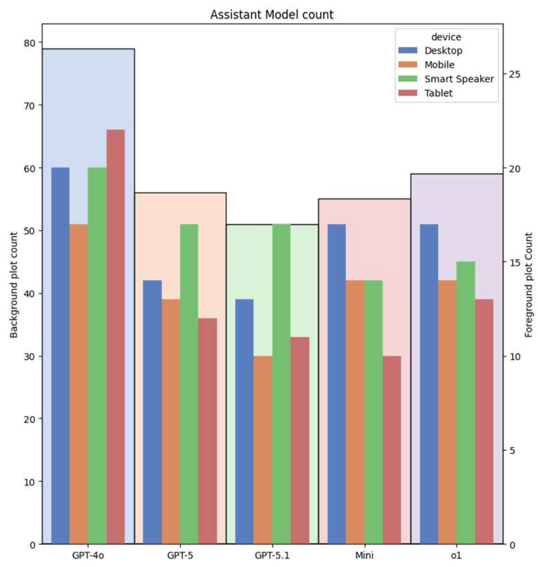
The boxplots are relatively similar, using the "`sea.boxplot()`" command with a few tweaks to make them look nicer.

Nothing else to say really.

Result + Visualization:

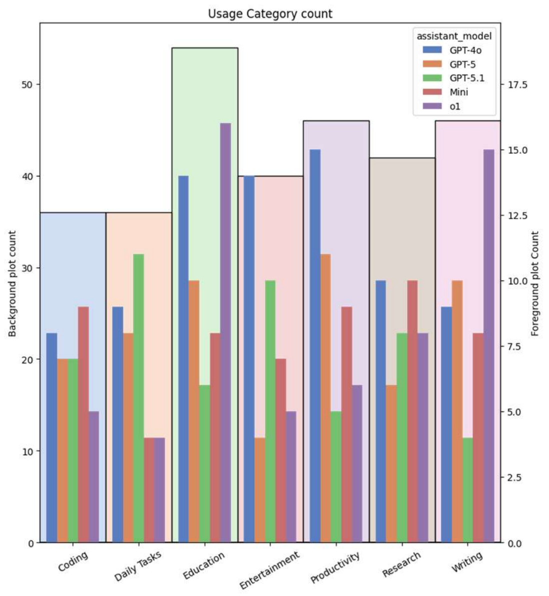
(it is assumed that the occurrence of feature classes in the dataset is directly proportional to the actual usage -> see "Foreword")

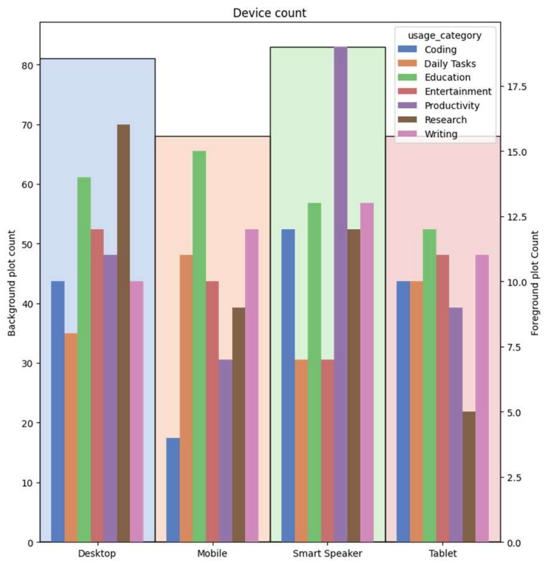
Boxplots:



Most people seem to use GPT-4o, which is interesting as it is not exactly the cheapest model available.
GPT-5.1 is used a lot on Smart Speakers apparently, whilst Mini is more used on desktop.
This could point towards smart-speaker manufacturers shipping their product with an OpenAI subscription.

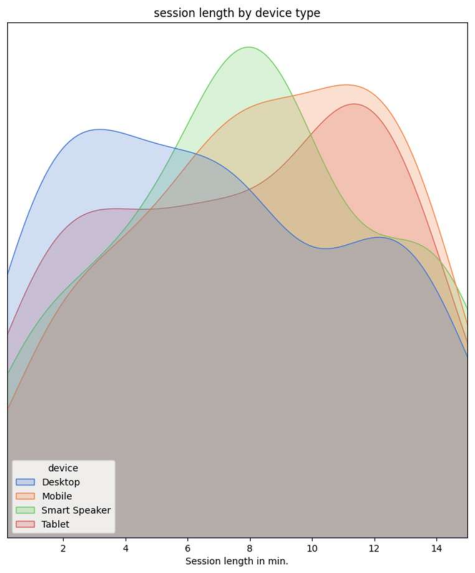
The most common usage for AI seems to be education. Depending on the definition of this category, this could mean that a lot of people are using these assistants for homework and study-help.
This is relatively interesting as I would expect a LOT more people to use AI in a more professional setting (work, coding, writing etc.) rather than for education.
Coding is one of the least common, which is very surprising to me, considering that AI coding assistants are one of the more actually useful applications of AI right now.





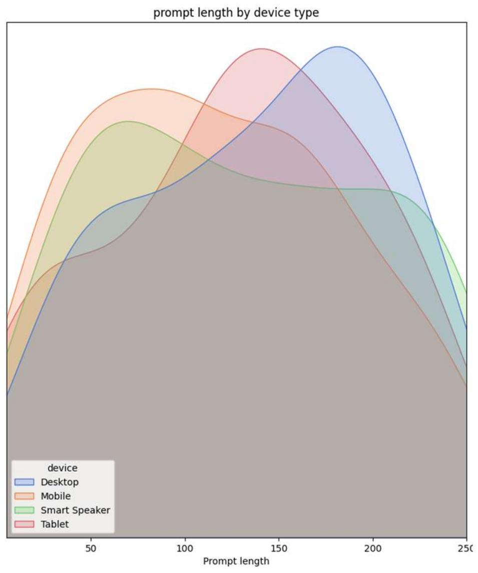
The biggest platform is relatively surprising: smart speakers.
I assume that this means devices like Amazon Alexa and Google Home, which is interesting since these devices are not really known for their AI capabilities.
Perhaps AI enabled smart speakers are more of a thing in the USA compared to the EU?
Also, interesting is the huge share of coders on smart speakers. This is VERY strange, as I honestly can't imagine how that would even remotely work. This might be an indication that the labeling process might be biased, or the share of users sampled very narrow and specific. Perhaps it also means that this data is generated.
Who's to say?

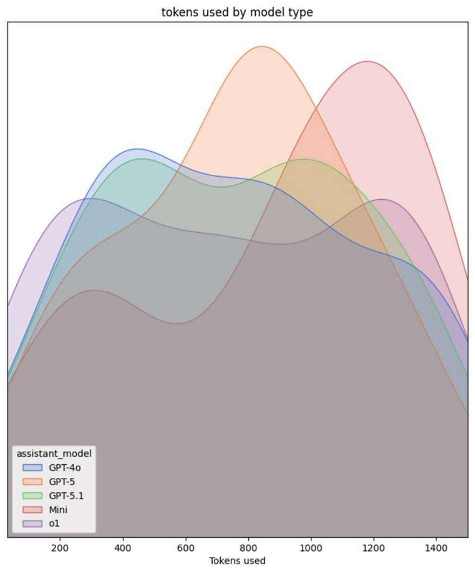
KDE Plots:



The session length seems to be shorter on desktop devices compared to mobile devices, which is weird. (one would think that desktop users would spend more time because of work/study etc.)
Similarly, tablets have the longest session lengths on average. This divide between mobile and desktop could stem from the different use cases for each device type. (or the time it takes to enter prompts)
This points towards desktop users using the AI assistant for quick queries, while mobile/tablet users might be engaging in longer interactions.

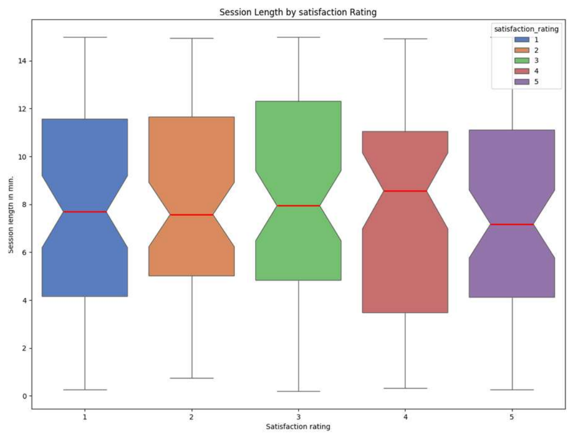
Looking at the prompt lengths, we can see that desktop users tend to have longer prompts on average compared to mobile and tablet users.
This could be due to the ease of typing on a physical keyboard, allowing for longer and more complex prompts. This is pretty strange though, since the session lengths were shorter on desktop.





As expected, more advanced models like GPT-5 and GPT-4o tend to use more tokens on average compared to older models like o1.
However, mini seems to use the most tokens on average, which could either stem from its use (longer prompts on average) or its complexity. (which is weird since mini is supposed to be a smaller model).

Boxplot:



session_length_minutes seems relatively unaffected by satisfaction rating, which is counter intuitive. One would expect that longer sessions would directly correlate with a higher **satisfaction_score**, however this does not seem to be the case here for whatever reason.

Task:

Are there any features that clearly differentiate device types?

Solution:

Looking at the previous assignment (Visualization) and the resulting plots (and plots that are not shown)

Result:

usage_category seems to be a good indicator as every device has a different, relatively high occurrence in one specific category.

- Desktop -> Research
- Mobile -> uniquely low coding
- Smart Speaker -> Productivity (for whatever reason)
- Tablet -> low research

It's not perfect, but this feature could be used to a reasonable degree to differentiate device types. (especially if all sub-categories are taken into account at once)

session_length_minutes and **prompt_length** are semi-useful in this case, because there are some slight differences in the distribution

- e.g. Desktop users tend to have shorter session lengths on avg. compared to tablet users
- Smart Speaker users tend to have a very narrow session length distribution (around 8 minutes)

Combining these together could be used to ***estimate*** the device someone is using, but calling this a “clear differentiation” is too much.

So, **NO**. There is no (good) feature that clearly differentiates device types.

Task:

Correlation Analysis:

- Create a correlation heatmap for all features.
- Which features are strongly correlated with each other?

- Are any features highly correlated with **assistent_model** and **session_length_minutes**?

Solution:

For this task, a few conversions were made:

- Converting **timestamp** to a UNIX timestamp. This is mostly redundant, as the correlation heatmap of seaborn apparently does this automatically.
- Converting **assistant_model** to a numerical feature by ranking them according to their release date. (`o1=1, ..., GPT-5.1=5`).
- Every other categorical column was also converted to a numerical feature. This, however, makes almost no sense and is only done, because the task specifically mentions "all features".

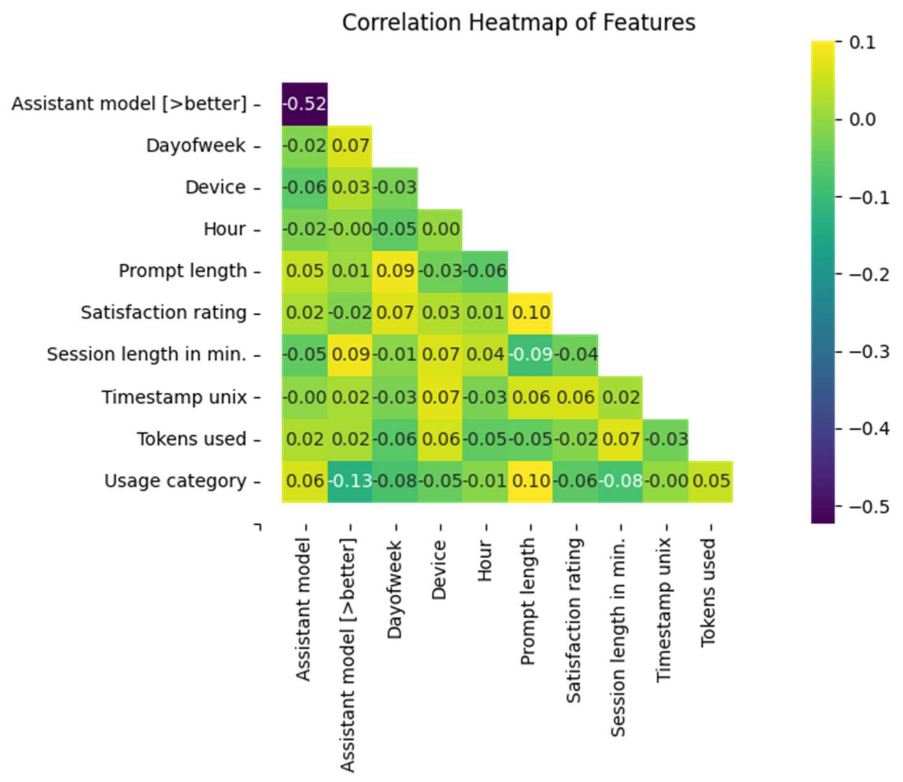
The conversions were made on a copy of the original Data-frame to allow for the dropping of values.

Because we are working in a jupyter notebook, the kernel remembers the Data-frame between cell executions. This means that upon re-execution of the cell, the columns can no longer be dropped because they no longer exist, which leads to an error.

This is very annoying, which is why a helper function was introduced: "`only_once()`". This function is actually a wrapper, which remembers if a function has already been executed and stops the function from running if so.

After everything has been completed, a heatmap is created via `sea.heatmap()` . For visual reasons, a triangular mask is also created to cut the correlation heatmap in two, as everything above (incl.) the diagonal is redundant information.

Result + Visualizations:



NOTE: categorical features were included, because of the task's wording. (see "Solution" above)

There are no really strong correlations between ANY features. The strongest correlations are between [**satisfaction_rating**, **prompt_length**] and [**usage_category**, **prompt_length**]. both hover around .1 which is quite weak.

This is relatively surprising, as one would imagine that at least some features would have some correlation (like **tokens_used** and **prompt_length**). This is another indicator that this dataset might be synthetic. (like the coding on smart speakers...)

the most associated feature with **assistant_model** is **usage_category** at .06. This is, again, quite weak. It makes sense though, other than the fact that it is the strongest correlation.

The most associated feature with **session_length_minutes** is **tokens_used** at .07. Again, no real correlation. This one really doesn't make sense though.

Task:

Use pair plots to explore feature relationships, color-coded by `assistant_model`

Solution:

This task is relatively straightforward.

`sea.pairplot()` is used to create the pair plot. (wow)

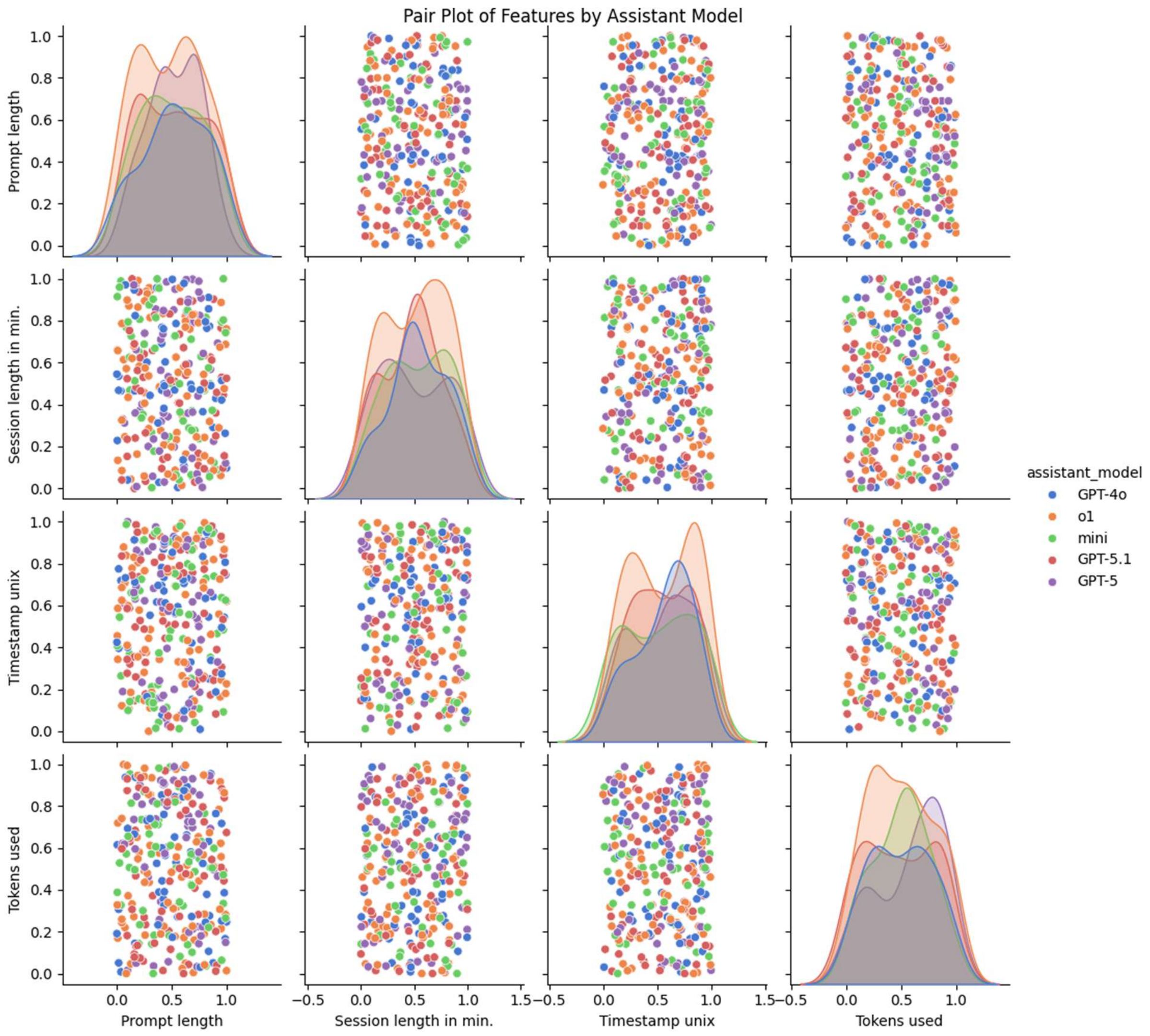
Categorical columns are excluded from this pair plot, since it doesn't really make sense to. Also, it clutters up the otherwise relatively nice pair plot

The most coding went into the subsequent generation of "paired pair plots" [PPP's] (always only 2 AI assistants at a time). This was done, because the general pair-plot did not provide satisfactory results and was too messy to get any information from. (spoiler)

For the PPP's a palette was created from the built-in seaborn "muted" color palette, as it would not keep a uniform color scheme between the plots otherwise.

Other than the loop that creates these plots, there is nothing else going on.

Result + Visualization:



To be quite honest, I don't really know what to make of this. There are no clear patterns or relationships that stand out. This aligns pretty well with the correlation heatmap from before and with the assumption that this dataset might possibly maybe be synthetic.

Paired Pair Plots [PPP's] (see "Solutions" above) have also been used but, sadly, lead towards no satisfactory result either.

The correlations look like random noise, with no real patterns or relationships that can be found. Not even a little bit of a trend.

Task:

Grouped by **assistant_model**, calculate mean, median, and standard deviation for each feature.

- What patterns do you observe?

Solution:

Relatively straightforward task again.

First, all numerical features were extracted via `select_dtypes(include=["number”])` and then grouped via `groupby(“assistant_model”)`.

Result:

assistant_model [MEAN]	**prompt_length**	**session_length_minutes**	**satisfaction_rating**	**tokens_used**
o1	131.13	7.18	3.10	740.76
Mini	132.52	7.58	2.87	882.65
GPT-4o	123.48	7.75	2.96	762.11
GPT-5	127.30	8.15	2.94	765.66
GPT-5.1	133.86	8.12	3.05	761.29

All means are relatively close to each other across all models with only a few exceptions.

o1 has a notably low mean **session_length_minutes** (7.18) compared to other models which hover around 7.5-8.1 minutes.

Mini has a notably high mean **tokens_used** (882) which is way above other models which are all around 740-765. This effect can also be observed in the boxplot from before.

assistant_model [MEDIAN]	**prompt_length**	**session_length_minutes**	**satisfaction_rating**	**tokens_used**
o1	125.0	7.630	3.0	731.0
Mini	132.0	6.720	3.0	985.0
GPT-4o	135.0	8.380	3.0	742.0
GPT-5	124.5	7.865	3.0	775.5

GPT-5.1	134.0	7.540	3.0	804.0

The medians are also relatively close to each other (in some cases, aka **satisfaction_rating**, they are identical).

The only real exception is in **tokens_used** where the spread is quite high. o1, GPT-4o and GPT-5 are relatively low (around 730-775), whilst GPT-5.1 is quite high (805) and mini is, again, very high at a whopping 985 tokens.

assistant_model [Std.dev]	**prompt_length**	**session_length_minutes**	**satisfaction_rating**	**tokens_used**
o1	74.32	4.09	1.39	461.38
Mini	58.53	4.80	1.26	440.66
GPT-4o	73.41	4.29	1.43	433.20
GPT-5	70.04	4.40	1.54	383.20
GPT-5.1	73.04	4.07	1.43	413.99

The standard deviations differ more between models compared to the means and medians.

prompt_length has a notably low Std.dev for Mini (58.5) compared to other models which are all around 70-75.

The conclusion I draw from this is that the models are quite similar in most cases, with o1 and Mini being the most notable outliers.

Task:

Time-Based Analysis:

- How does session length vary by day or time of day?
- Are there peak usage times for certain assistants?
- Create a line plot or a heatmap to show usage trends over time.

Solution:

For this task, a few new columns were quickly introduced:

- **hour**: extracts the hour from **timestamp** via `.dt.hour`
- **dayofweek**: extracts the day of the week via `.dt.dayofweek`
- **dayofweek_name**: gets the name of the week via a list and `.apply()`
- **timeofday**: maps extracted hour to Morning (5-11), Midday (11-17), Evening (17-23) and Night (23-5) via custom function `_map_timeofday_def()`

After introduction and a nice tea, the plots were created.

Nothing out of the ordinary really.

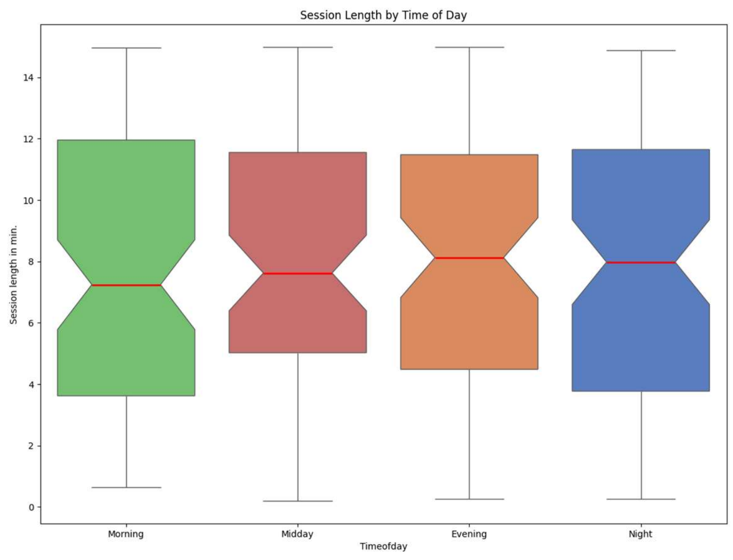
A Boxplot was created to show the variation of **session_length_minutes** by **timeofday**. This seems like a reasonable choice as it should clearly show the changes, whilst smoothing out the roughness that comes from working with such a small dataset (remember: the timespan is only *69* days wide, spread across only 300 rows!).

Additionally, a second boxplot was made that shows **session_length_minutes** by **dayofweek.** This was done, because the results of the first boxplot were relatively underwhelming, and this one actually showed a bit of a trend. (and because it says to do it in the task)

Afterwards KDE plots were made (with the same function as before "`cts_kdeplot()`"). KDE plots were chosen to show the peak usage times of certain **assistant_model**, as this is a very natural way for KDE plots to be interpreted. (because the peaks are actual peaks)

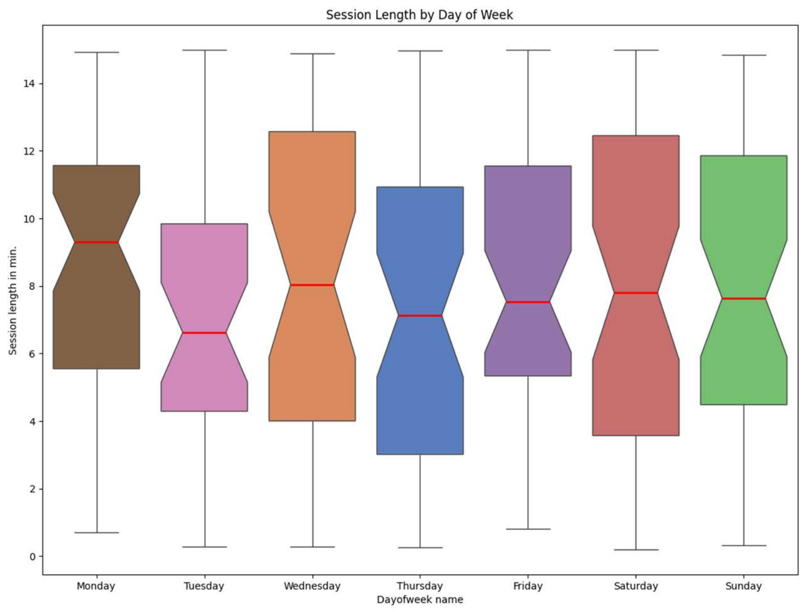
Lastly, line plots (`sea.lineplot()`) in addition to a regression line (`sea.regplot()`) were made to show the change of numerical features over time. (a helper function is also implemented -> `cts_lineplot()`)

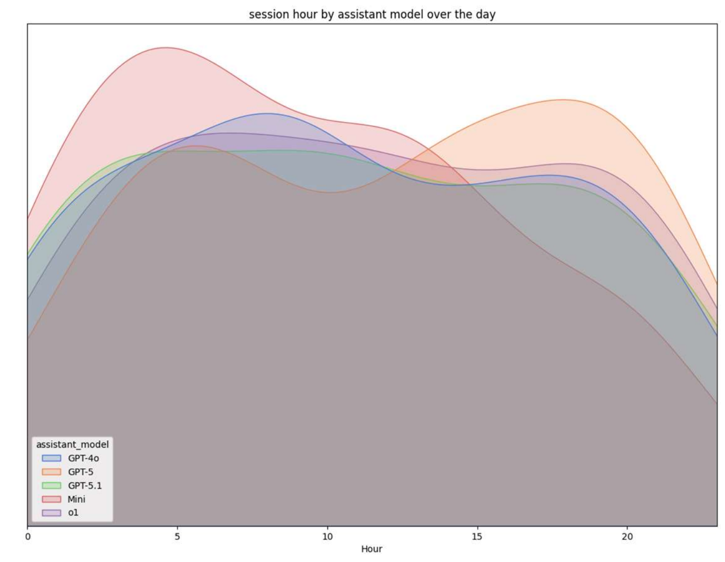
Result + Visualization:



session_length_minutes seems to be quite stable across the day. There is a slight trend towards longer sessions the later it gets, but it's quite small. I guess that people spend the same amount of time no matter when they use an assistant.

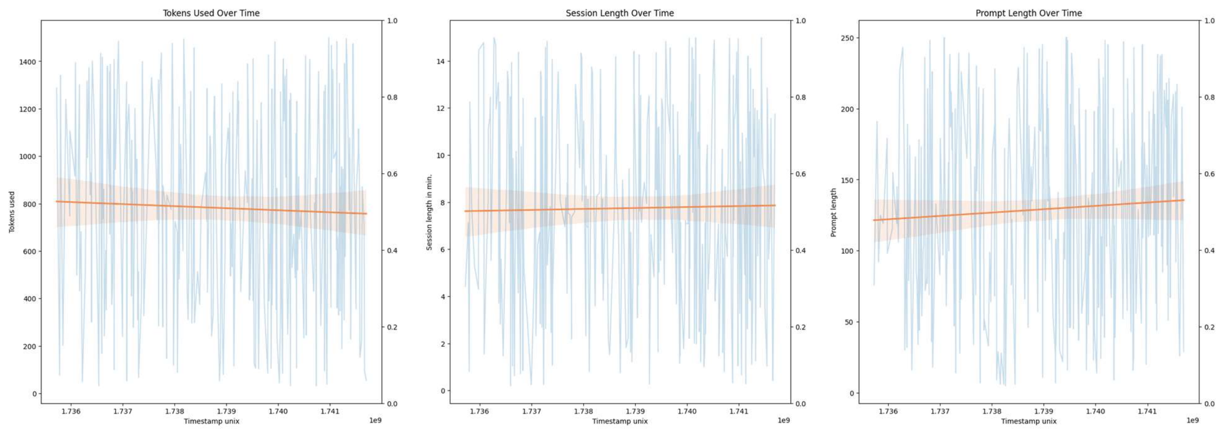
dayofweek varies quite a lot, which is surprising (and a welcome change from the rest of the dataset). Especially Monday seems to have noticeably longer sessions compared to other days. the spread is also quite high across the board.





There seems to be a noticeable falloff in the usage of Mini during later hours, whilst other models remain relatively stable. The general trend seems to be a slight decrease the later it gets, which makes sense.
the only exception to this rule is GPT-5 which slightly increases in usage during the night
Overall, the usage seems to have noticeable peaks for each model (excl. o1 and GPT-5.1).

(Note, that the fall-off at the 2 ends of the KDE-plot are because of its representation, not because of the data)



This plot shows 3 features plotted over time (with a regression line to indicate if the feature went up or down during the 69 days period).

There is not much to be said here, since everything is relatively stable across the ~2 months. The only (slight) exception being **prompt_length**. It has a slight upward trend. Wow.

Additional Insights

Task:

Create 5 custom analyses that you believe provide valuable insights.

Solution:

We created the following analysis:

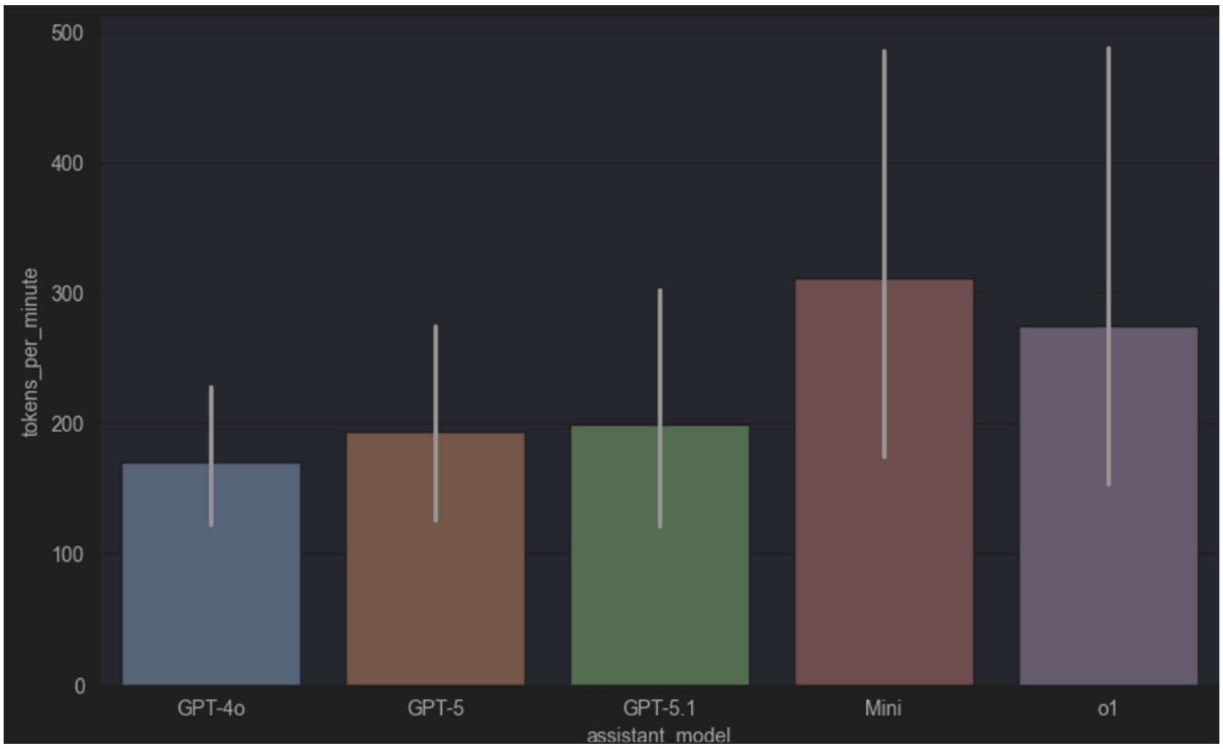
1. Analysis of the time-period of the dataset:

   Solution: By using the `max()` timestamp – the `min()` timestamp we find the time-period, then we print.

   Result: We have found that the whole time-period of the dataset was only 69 days
2. Anaysis of the tokens used per minute per model

   Solution: I created a new table with the values of **model** , **tokens_used** and **session_length_minutes** and then I created a new column dividing the **tokens_used** by the **session_lenght_minutes** and then I print it with seaborn as a bar graph , only the tokens used as y and model as x, also by `sea.barplot()` is going to use by default the mean of the values so we don't have to worry about that

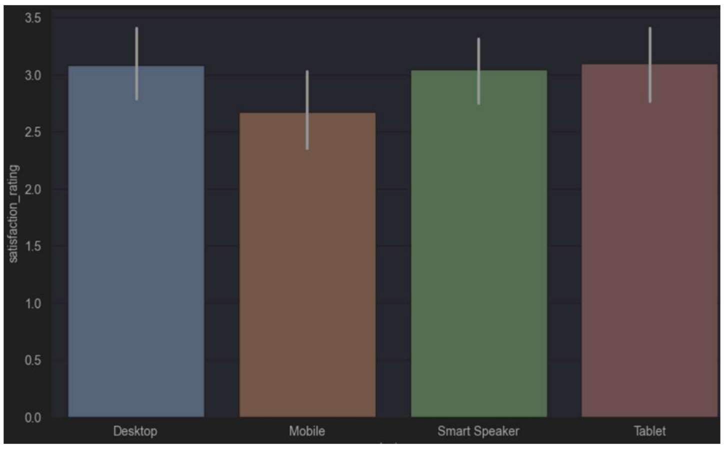
   Result: we got how many tokens per minute was the average for each of the models The one with the lowest tokens per minute was GPT-o4
   Vizualization:



3. Analysis of the relation between Device and Satisfaction rating

   Solution: In this case, I took the values of **device** and **satisfaction_rating**, and I wrote a basic seaborn Barplot , the Barplot by default takes the mean of the satisfaction rating.

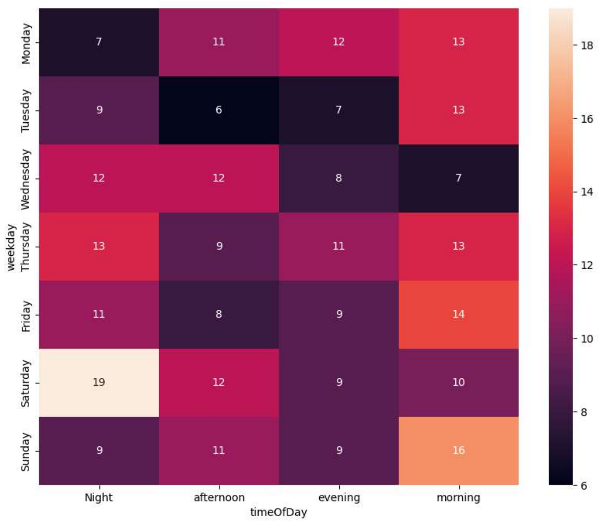
   Result: We got a graph that allows us to visualize the relationship between **device** and **satisfaction_rating**, which also includes the standard deviation. We can notice that the satisfaction rating is more or less the same in each device with the exception of mobile where clearly, it’s 0.5 points lower than the others.
   Visualization:



4. Analysis of the relation between weekday and **timeOftheday**

   Solution: First, a new column with the weekday name was created from the timestamp column with function `dt.day_name()`. Then, the data was grouped by weekday and time of day of sessions with a function `groupby()` and results were visualized with a heatmap to identify usage patterns.

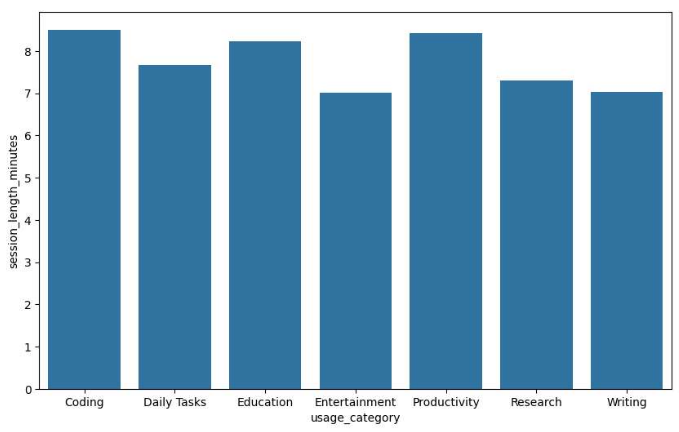
   Result: The heatmap shows AI assistant usage by day of the week and time of day. The number of sessions is the highest on Saturday night (19 sessions) and Sunday morning (16 sessions), while weekday afternoons, especially Tuesday and Wednesday, are less active. Overall, morning and night are the most active periods, indicating that users prefer to interact with AI assistants during these times.
   Vizualization:



5. Analysis of the Average session length per usage category

   Solution: The average session length was calculated for each usage category. Then the results were sorted in descending order and visualized using a bar chart.

   Result: In the result we can see that coding tasks have the longest average session duration, showing that they are more time-consuming and require longer interactions with AI assistants.
   Productivity and education tasks also show relatively long sessions, while daily tasks, research, writing, and entertainment sessions are shorter, indicating quicker interactions for these tasks.


session_length_minutes
0
1
2
3
4
5
6
7
8
Coding
Daily Tasks
Education
Entertainment
Productivity
Research
Writing
usage_category