Toolwindow Usage Data Analysis

By Dheer Maheshwari

In this analysis, I investigate whether the duration a toolwindow is open differs depending on whether it was opened manually (by user action) or automatically (triggered by IDE events). The dataset is an event log with user_id, timestamp, open_type (manual/auto) and event (opened/closed).

Some assumptions I made:

- open_type is valid only for open events, close events may have empty open_type.
- orphaned close events (no open before the closed event) are probably mistakes in data entry, and no conclusion can be made out of them, so they are removed
- open events with no matching close before the next open or dataset end are removed as unmatched
- Very short durations (bottom X percentile) and very long durations (top Y percentile) may reflect accidental popups/clicks, or idle sessions

Based on these assumptions and other observations, there are some data cleaning and modification steps taken in order to "clean" the messy data and make inferences based on the clean data.

Converting timestamps to numeric/datetime and sort by user_id, timestamp:

Code block which does that:

```
# convert timestamps

df['timestamp'] = pd.to_numeric(df['timestamp'],
errors='coerce')

df['dt'] = pd.to_datetime(df['timestamp'], unit='ms',
origin='unix', utc=True)

df = df.sort_values(['user_id',
'timestamp']).reset_index(drop=True)
```

Creating open-close pairs and handling messy data:

The dataset contains some messy data such as unmatched opened and closed events (example, missing closes, multiple opens without closes, or closes appearing before any open). Hence it was made sure to get rid of these inconsistencies, while also creating valid open-closed pairs.

Matching open/close, handling subsequent opened event without closes and open without closes:

```
while j < len(user_events):

    next_event_type = user_events[j]['_normalized_event']

    if next_event_type == 'closed':

        close_index = j

        break

    if next_event_type == 'opened':

        close_index = None

        break
```

For every "opened" event, this code searches forward in the same user’s timeline for the next "closed" event. If an open-close pair is visible, the index of the closed event is saved in close_index variable, used later to pair. If another "opened" appeared before a "closed", or there’s no “closed” for the “opened” event, the earlier open was treated as incomplete and added to removed_records (block of code below).

```
else:

    # opened event with no close event found so delete

    removed_records.append({

        'original_index': orig_idx,

        'user_id': user_id,

        'event': current_event[event_column],

        'timestamp': current_event[timestamp_column],

        'reason': 'unmatched_open_no_later_close'

    })

    i += 1
```

Creation of tool window usage episodes:

Each valid open-close pair was converted into an episode, which represents one complete tool window session. The variable close_index (explained above) is used here. Cases are handled when time is 0 or negative, or null. The iterator i is updated so that it tracks new opened or closed instances.

```
open_event = user_events[i]

close_event = user_events[close_index]

if duration_seconds is not None and duration_seconds > 0:

    # only add if neither row is already marked for deletion

    deleted_indices = [rec['original_index'] for rec in
removed_records]

    if orig_idx not in deleted_indices and
int(close_event['original_index']) not in deleted_indices:

        paired_sessions.append({

            'user_id': user_id,

            'open_index': int(open_event['original_index']),

            'close_index': int(close_event['original_index']),

            'open_timestamp': open_event[timestamp_column],

            'close_timestamp': close_event[timestamp_column],

            'open_type': open_type_val if open_type_val else
'unknown',

            'duration_seconds': duration_seconds

        })

        kept_indices.add(int(open_event['original_index']))

        kept_indices.add(int(close_event['original_index']))

i = close_index + 1
```

Creating the final episodes DataFrame:

After iterating through all users, all valid episodes were combined into a single DataFrame and all further statistical analysis was conducted on that.

```
episodes_df = pd.DataFrame(paired_sessions)

print(f"Paired episodes dataframe length: {len(episodes_df)}")
```

Each row in episodes_df represented one complete session. The data it includes is:

- The user ID
- The open and close timestamps
- The open type ("manual" or "auto")
- The duration in seconds

Handling other cases:

The block of code below handles edge cases such as an orphaned event (closed without any open):

```
elif event_type == 'closed':

                # no open for the closed event so delete

                removed_records.append({

                    'original_index': orig_idx,

                    'user_id': user_id,

                    'event': current_event[event_column],

                    'timestamp':
current_event[timestamp_column],

                    'reason': 'close_with_no_open'

                })

                i += 1
```

The block of code removes rows with NaN:

```
# remove any rows with null (not for cleaned because every
closed event type has open_type null)

episodes_df = episodes_df.dropna().reset_index(drop=True)
```

This code removes of extreme data (for example episodes being too long or too short). Also tested 5% trims and raw data to examine any visible differences between raw and cleaned data:

```
# remove data below 5 percentile and above 95 percentile

def remove_lower_percentile_outliers(data: np.ndarray, low,
high):

    lower_threshhold = np.percentile(data, low)

    higher_threshhold = np.percentile(data, high)

    filtered = data[(data >= lower_threshhold) & (data <=
higher_threshhold)]



    return filtered



manual_filtered = remove_lower_percentile_outliers(manual,
low=5, high=95)

auto_filtered = remove_lower_percentile_outliers(auto, low=5,
high=95)
```

Handling of messy data was effective, as the length of original dataset and cleaned dataset has noticeable difference, and paired episodes are exactly half of the cleaned dataframe (which is expected as 1 open and 1 close event merges into a single episode, halving the dataframe):

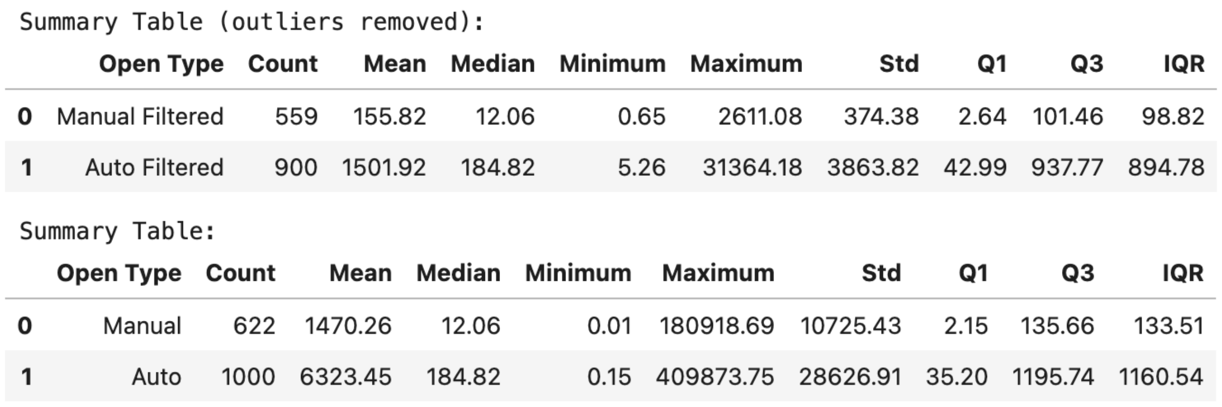
```
Original dataframe length: 3503

Cleaned dataframe length: 3244

Paired episodes dataframe length: 1622
```

<u>Analysis:</u>

After cleaning the dataset and constructing valid open close pairs, I analysed the duration of tool window sessions based on how they were opened (manual or auto).



```
Summary Table (outliers removed):
```

	Open Type	Count	Mean	Median	Minimum	Maximum	Std	Q1	Q3	IQR
0	Manual Filtered	559	155.82	12.06	0.65	2611.08	374.38	2.64	101.46	98.82
1	Auto Filtered	900	1501.92	184.82	5.26	31364.18	3863.82	42.99	937.77	894.78

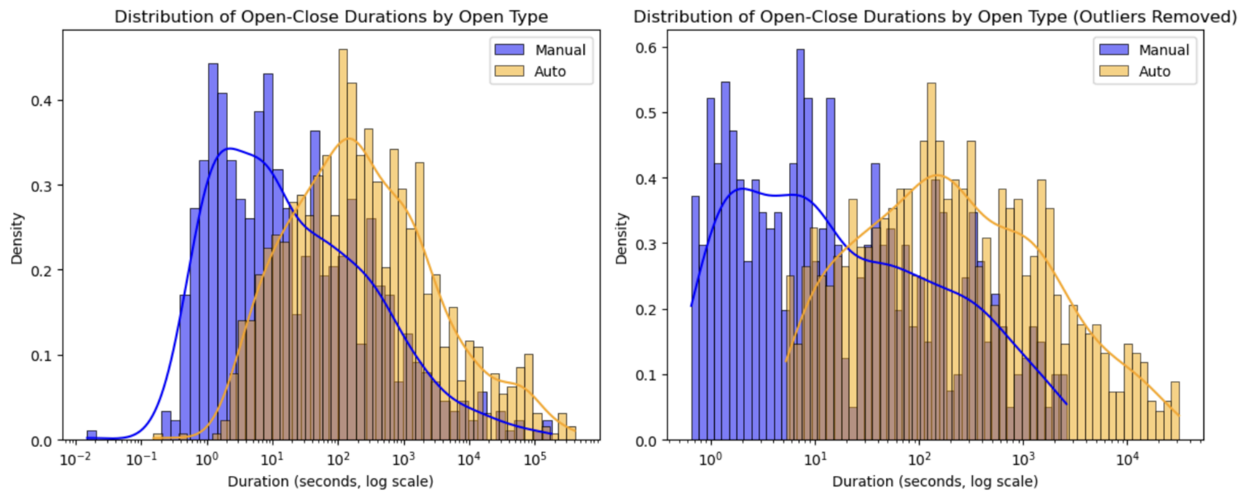
```
Summary Table:
```

	Open Type	Count	Mean	Median	Minimum	Maximum	Std	Q1	Q3	IQR
0	Manual	622	1470.26	12.06	0.01	180918.69	10725.43	2.15	135.66	133.51
1	Auto	1000	6323.45	184.82	0.15	409873.75	28626.91	35.20	1195.74	1160.54

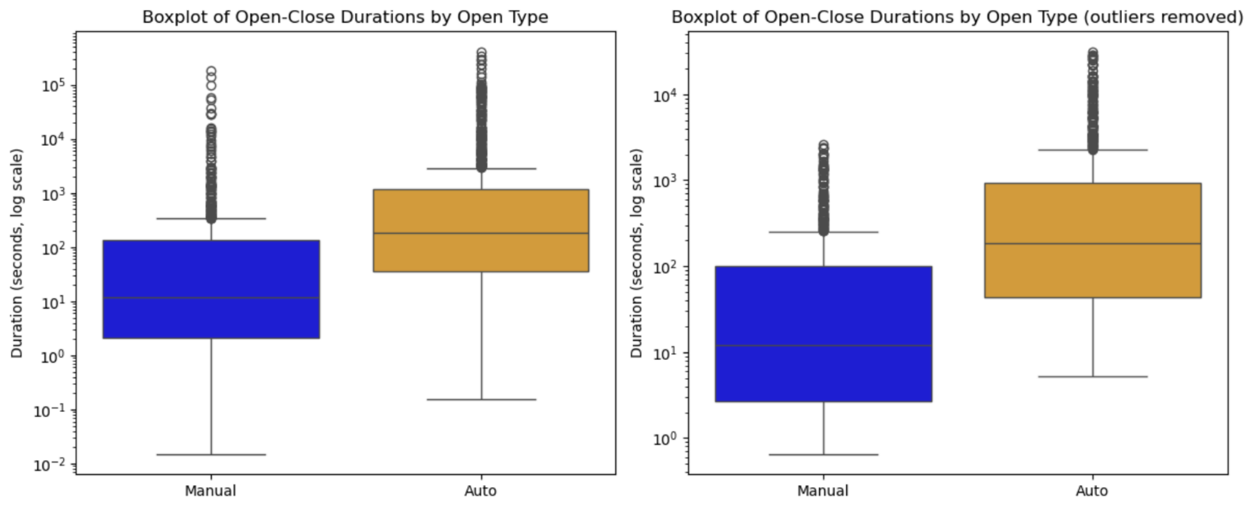
1. **Overall pattern:** For this tool window, automatically triggered openings stay open much longer than those opened manually. Mean duration is almost 10 times greater for auto, after outliers are removed. The median implies that a typical manual open lasts only about 12 seconds, while auto ones last around 3 minutes.
2. **Spread and variability:** The standard deviation for auto opens is larger than manual, therefore, high variance is evident. This means that while some auto sessions close quickly, others remain open for very long periods.
3. **Distribution shape:** The mean > median pattern for both types shows right-skewed distributions. Extremely long sessions are also seen when looking at the maximum values for both manual and auto, confirming that a few unusually long auto events inflate the mean, even when lowest and highest 5% of data is removed.

Visualisations:

Using matplotlib and seaborn, I plotted data without filtering, and with filtering (removal of top and bottom 5%), which resulted in the following graphs:



Histogram of duration (seconds, log) vs density



Box-plot of duration (seconds, log) vs category (manual/auto)

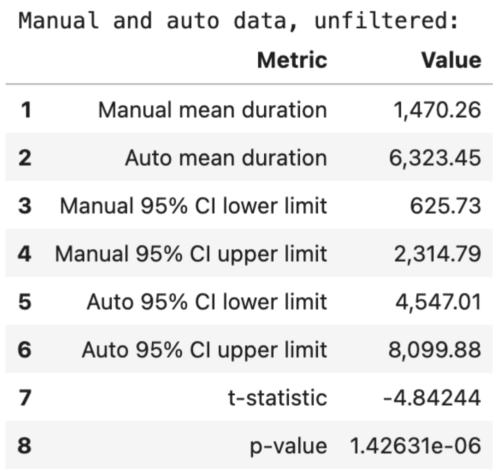
Interpretation of Plots:

- **Histogram:** The manual distribution is tightly concentrated near 10^0 and 10^1 (1 to 10). The auto distribution has a long right tail, confirming occasional very long auto sessions.

- **Boxplot:** The auto group's entire plot is noticeably higher than manual's, and it goes further into larger durations, meanwhile manual's lowest duration is below 10^0 even when bottom 5% is removed, suggesting that unintentional manual opens are still taken into account.

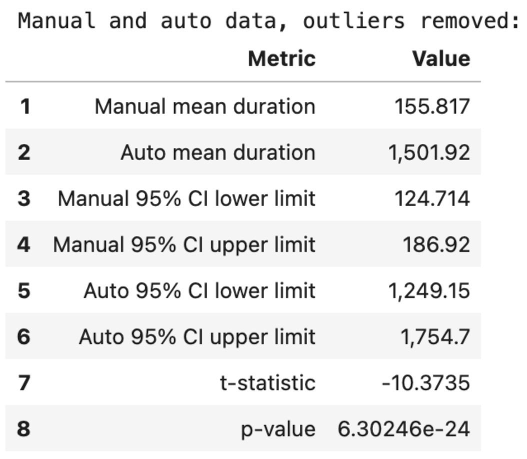
Measures of confidence:

I carried out multiple measures of confidence and measures of statistical significance for both manual and auto sessions, and compiled them in a single dataframe:



```
Manual and auto data, unfiltered:
```

	Metric	Value
1	Manual mean duration	1,470.26
2	Auto mean duration	6,323.45
3	Manual 95% CI lower limit	625.73
4	Manual 95% CI upper limit	2,314.79
5	Auto 95% CI lower limit	4,547.01
6	Auto 95% CI upper limit	8,099.88
7	t-statistic	-4.84244
8	p-value	1.42631e-06



```
Manual and auto data, outliers removed:
```

	Metric	Value
1	Manual mean duration	155.817
2	Auto mean duration	1,501.92
3	Manual 95% CI lower limit	124.714
4	Manual 95% CI upper limit	186.92
5	Auto 95% CI lower limit	1,249.15
6	Auto 95% CI upper limit	1,754.7
7	t-statistic	-10.3735
8	p-value	6.30246e-24

1. **Confidence intervals:** Both groups' 95% confidence intervals are well-separated, meaning the true average duration of auto sessions is very unlikely to overlap with that of manual ones. The narrow interval for manual opens implies more consistency than auto opens, as auto open have larger interval, which is less narrow than manual
2. **Statistical significance:** Both t-statistic and p-value is very low, which suggests that the difference between manual and auto open durations is statistically significant.

<u>**Strengths, limitations, and further work:**</u>

Strengths:

- By considering orphan closes and consecutive opens, messy data is cleaned up and only essential data is used
- Clear difference between manual and auto is seen in both visualisations: histogram and box plots
- Use of multiple confidence and statistical significance measures (t-test, confidence interval, t-statistic, p-value) confirms that there is a significant difference between manual and auto opens

Limitations:

- High variance in auto durations may result from outliers, like sessions left open overnight, which skewes the data by a considerable extent
- It is assumed that all unmatched or orphaned events were noise and removes them, however, some may be legitimate ones, that lacked recorded open/close events
- The data does not distinguish passively opened window (window open but unused) vs actively used, which does not give much room for behavioural analysis

Further work:

- We can also collect further data such as clicks, actions, focus time, etc to better know the reasons why toolwindow was opened and measure user engagement with toolwindow
- Introduce features such as auto-closing inactive tool windows through a/b testing, to investigate whether the durations change

<u>What I can infer:</u>

Through the analysis, it is seen that automatic toolwindow opens by the IDE last longer than manual opens. I can hypothesise this in terms of behaviour: manual opens occur when users intentionally check something briefly (for example: inspecting an output, running a short action) so they close the window quickly. Auto opens often occur when the IDE triggers the window during background operations (for example: builds, debugging sessions). These may stay open much longer because they probably remain unattended, and aren't the user's main focus. This can be detrimental as even when not in use, these toolwindows require memory, which may slow down the tasks users are focusing on. Features such as auto closing the inactive toolwindow may be beneficial for the end users.