Tool Window Open Duration Analysis

A Comparison of Manual vs. Automatic Activation

Author: Michał Jaskulski

Date: 4.11.2025

1. Objective and introduction:

The primary objective of this analysis was to determine whether there's a meaningful difference in how long the tool window stays open depending on how it was opened. The analysis involved processing event logs using data science tools such as Jupyter Notebook setted up in PyCharm with Pandas library.

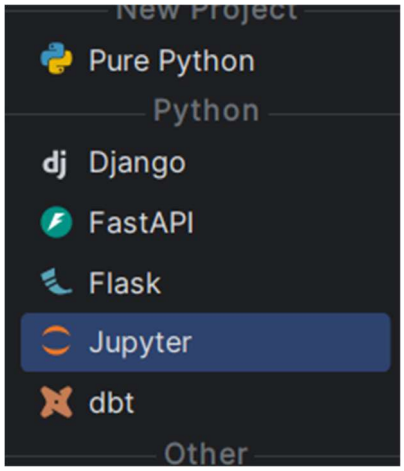
2. Setup and Execution Instructions

Prerequisites:

- Python 3.8 or newer
- PyCharm IDE is recommended, though any environment capable of running Jupyter Notebook will work.
- Data: file „toolwindow_data.csv” is included in „Files” directory.

Project setup:

- **Download all the files from the Google Drive Folder.**
- Open your IDE and create a new project, in PyCharm choose the option „Jupyter”, name it as you wish, e.g. „JupyterProject”.

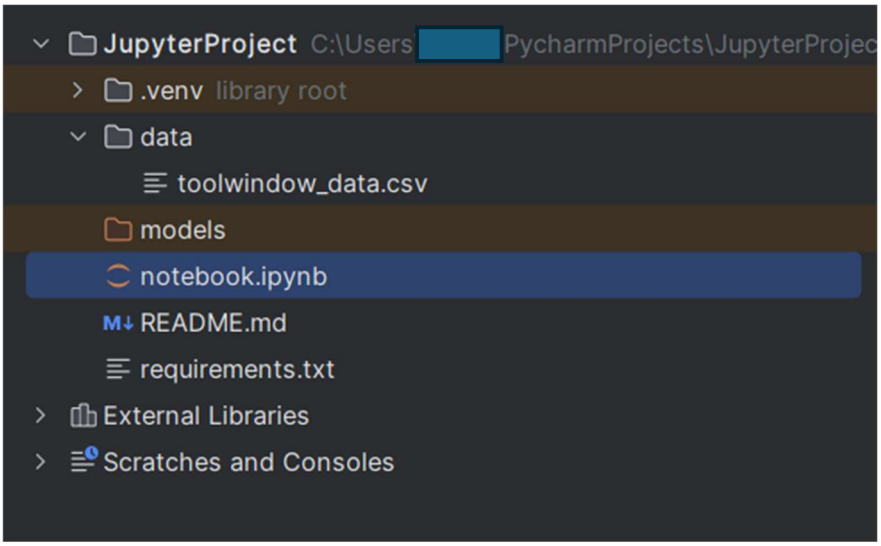


(Note: from now on, we will call your project directory **root directory** to simplify next steps)

- In the root directory, include the notebook.ipynb file, after that create there directory named „data” (if not creaded automatically) and place in it file „toolwindow_data.csv”.
- Open your terminal in the root directory and copy a command:

„pip install jupyter pandas scipy”

- Your project directory should look like this:



3. CODE OVERWIEV

First of all, let's import nessecary library and csv file

```
#imports
import pandas as pd
from scipy import stats
[185]
```

```
df = pd.read_csv("data/toolwindow_data.csv")
#let's change our timestamp to a datetime
df['timestamp'] = pd.to_datetime(df['timestamp'], unit='ms')
df.sort_values(by=['user_id', 'timestamp'],ascending=[True, True], inplace=True)
[114]
```

We sort values by user_id and timestamp. This way, we will have correctly placed events (as timestamp is in epoch milliseconds, closure follows an opening FOR A SPECIFIC USER, because we cannot take an opening from user1 and closure for user2 and call it a day.)

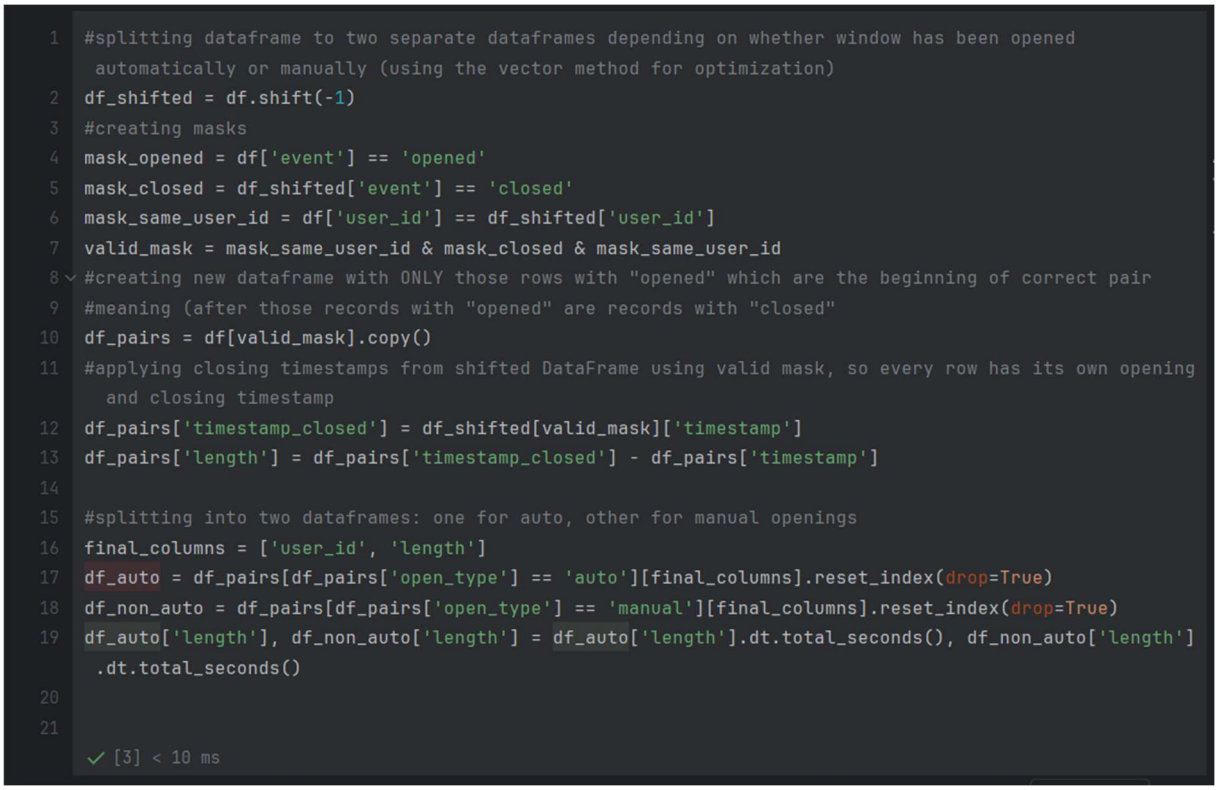
Next step is creating "perfect pairs". A perfect pair is one in which closure follows an opening in the sorted Dataframe, as long as user_id remains the same. For comparison, we will create two Dataframes: one for manually opened windows, and one for those opened automatically. There are two ways to do that:

1. First one **(not recommended):** Iterate through Dataframe using „for” loop.

```
df_auto, df_non_auto = pd.DataFrame(), pd.DataFrame()     df_auto
for i in range(len(df)-1):
    if df.iloc[i]['event'] == 'opened':
        if df.iloc[i+1]['event'] == 'closed' and df.iloc[i+1]['user_id'] == df.iloc[i]['user_id']:
            is_opened_automatically = (df.iloc[i]['open_type'] == "auto")
            length = df.iloc[i+1]['timestamp'] - df.iloc[i]['timestamp']
            data = {
                'user_id': [df.iloc[i]['user_id']],
                'length': [length.total_seconds()],
            }
            new_df = pd.DataFrame(data)
            if is_opened_automatically:
                df_auto = pd.concat([df_auto, new_df ], ignore_index=True)
            else:
                df_non_auto = pd.concat([df_non_auto, new_df ], ignore_index=True)
[26] 638ms
```

It is possible to use this iteration, but this method is very slow (It took 638miliseconds to finish, look at the lower bottom corner of the screenshot).

2. Second one **(recommended)**: Vector method.

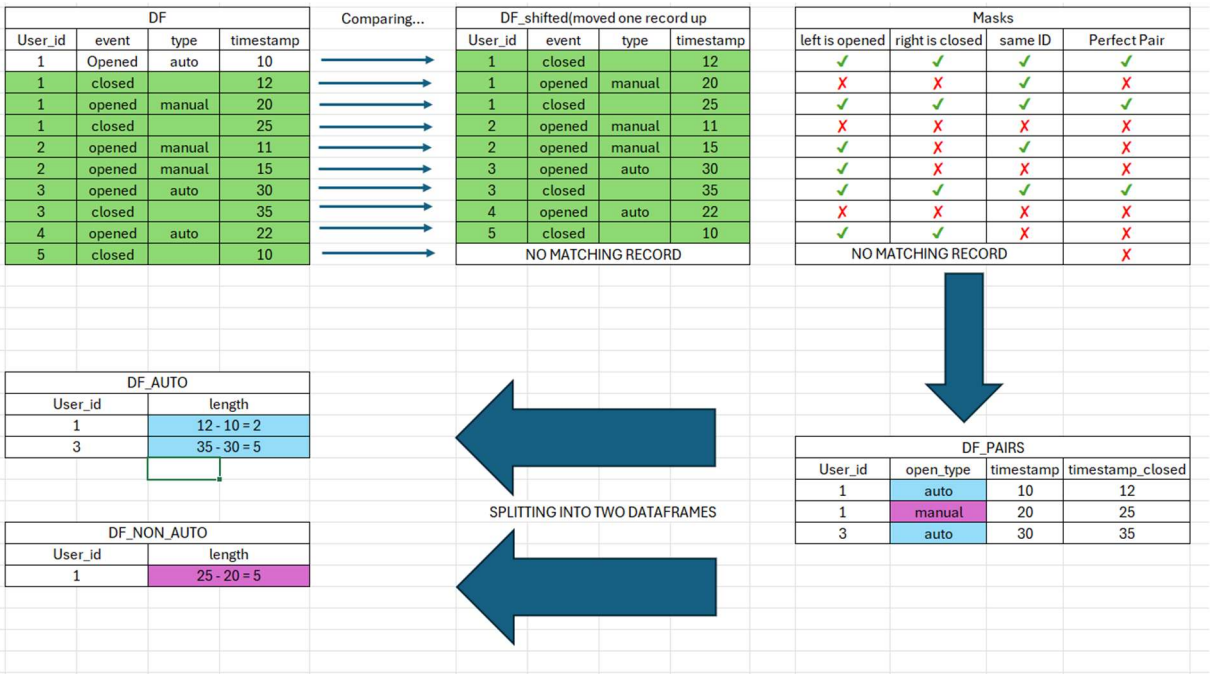


```
#splitting dataframe to two separate dataframes depending on whether window has been opened
 automatically or manually (using the vector method for optimization)
df_shifted = df.shift(-1)
#creating masks
mask_opened = df['event'] == 'opened'
mask_closed = df_shifted['event'] == 'closed'
mask_same_user_id = df['user_id'] == df_shifted['user_id']
valid_mask = mask_same_user_id & mask_closed & mask_same_user_id
#creating new dataframe with ONLY those rows with "opened" which are the beginning of correct pair
#meaning (after those records with "opened" are records with "closed"
df_pairs = df[valid_mask].copy()
#applying closing timestamps from shifted DataFrame using valid mask, so every row has its own opening
  and closing timestamp
df_pairs['timestamp_closed'] = df_shifted[valid_mask]['timestamp']
df_pairs['length'] = df_pairs['timestamp_closed'] - df_pairs['timestamp']

#splitting into two dataframes: one for auto, other for manual openings
final_columns = ['user_id', 'length']
df_auto = df_pairs[df_pairs['open_type'] == 'auto'][final_columns].reset_index(drop=True)
df_non_auto = df_pairs[df_pairs['open_type'] == 'manual'][final_columns].reset_index(drop=True)
df_auto['length'], df_non_auto['length'] = df_auto['length'].dt.total_seconds(), df_non_auto['length']
 .dt.total_seconds()


[3] < 10 ms
```

Optimized in the c++ vector method is way faster than the previous one and just as effective. First of all, we create shifted (moved one row up)

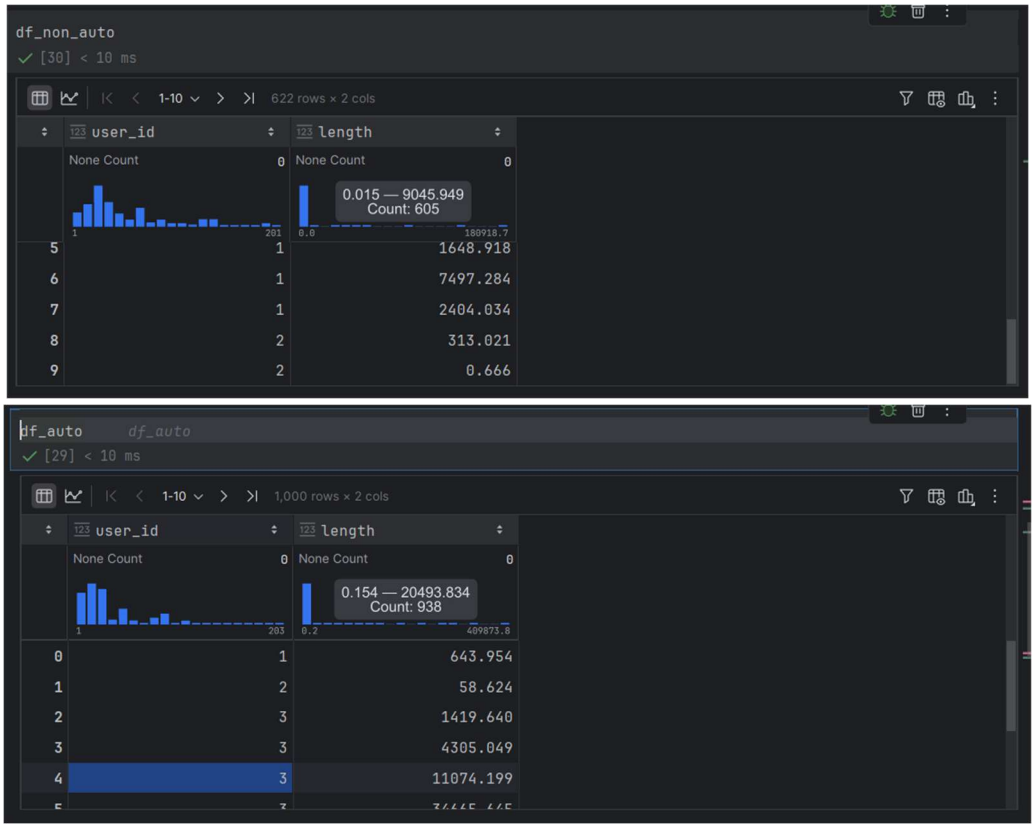
dataframe to compare them side by side. After that, we create masks that works like “checkboxes”. They validate if after event == ‘opened’ appears event==’closed’ and if user_id remains the same. When all the “beginnings of the perfect pairs” are found, they are moved to another dataframe called df_pairs. Then we add column ‘timestamp_closed’ which is filled with timestamps from df_shifted. At the end, df_pairs is splitted to two Dataframes, depending on auto or manual opening, those dataframes have only two columns: user_id and length (difference between opening and closure time).

GRAPHICAL PRESENTATION OF CODE:

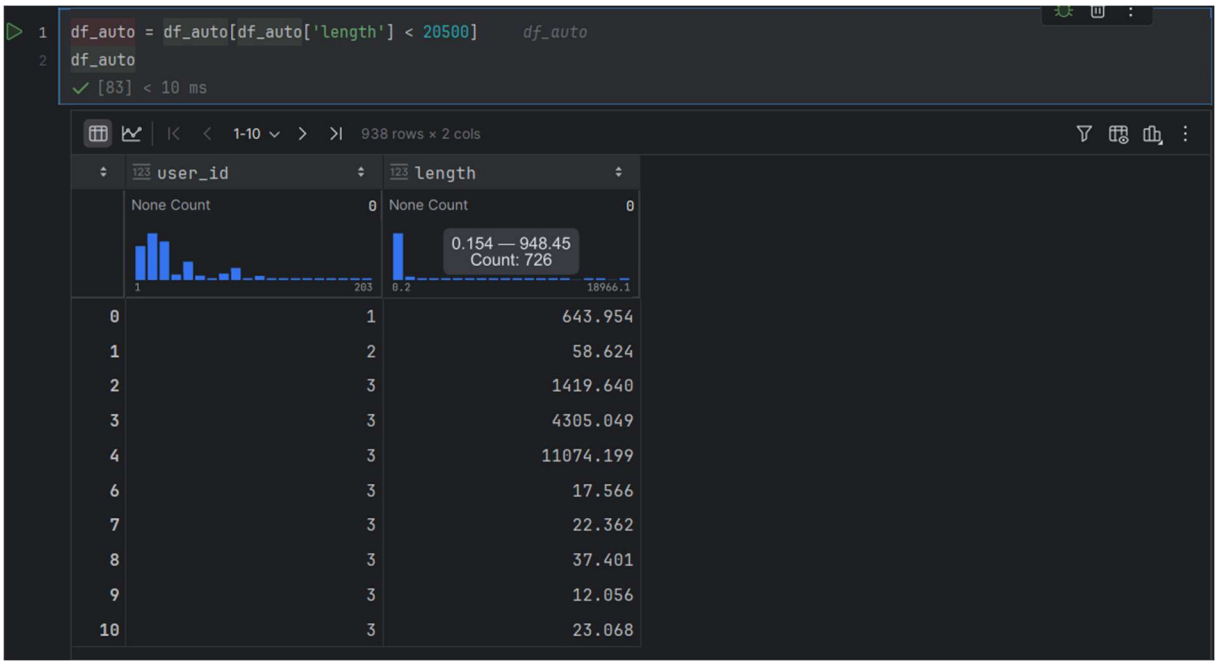


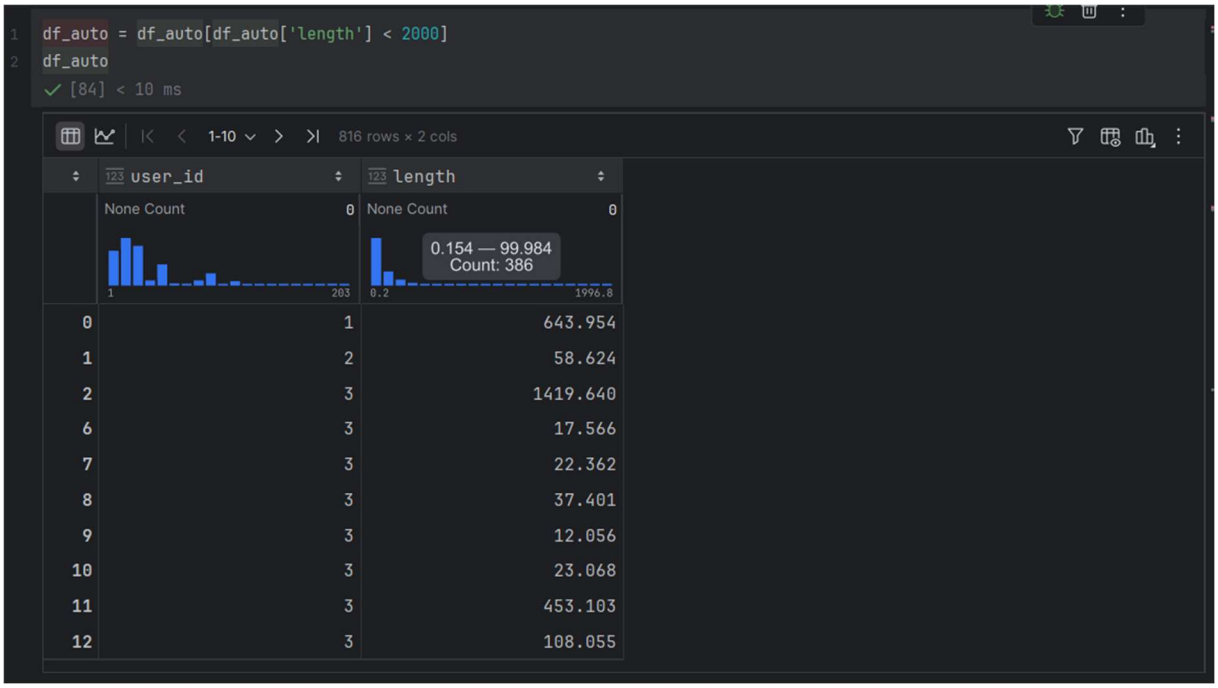
DATA ANALYSIS

To analyze the program’s output, we will be using PyCharm built-in tools for data analysis.

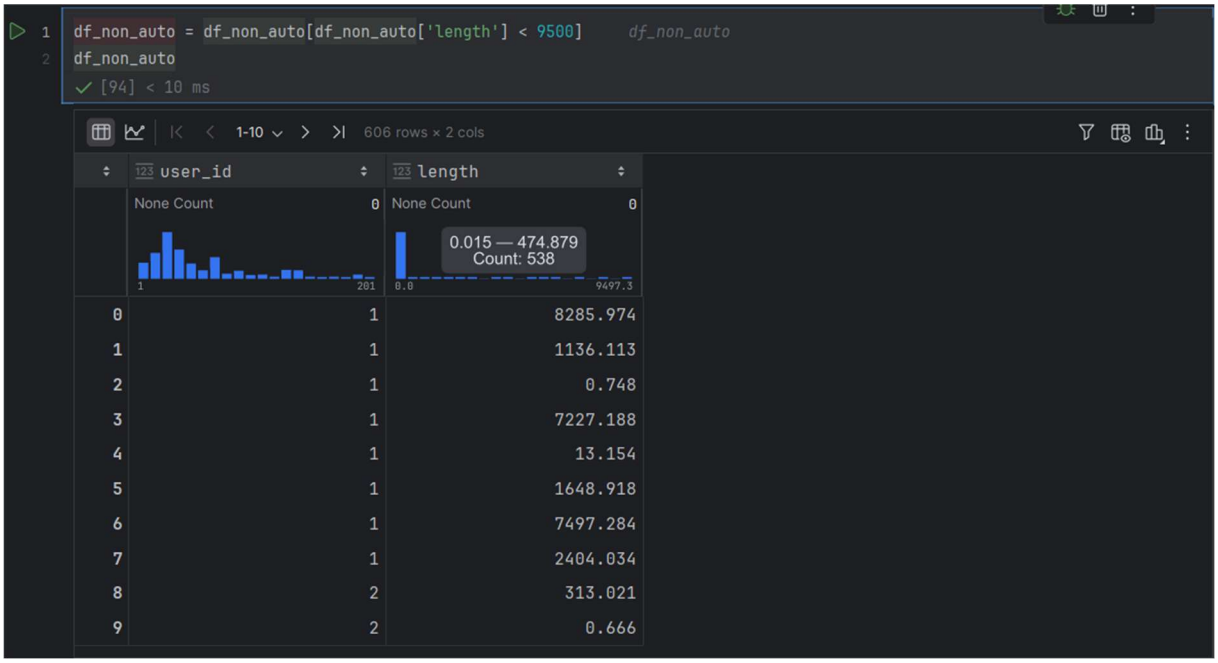


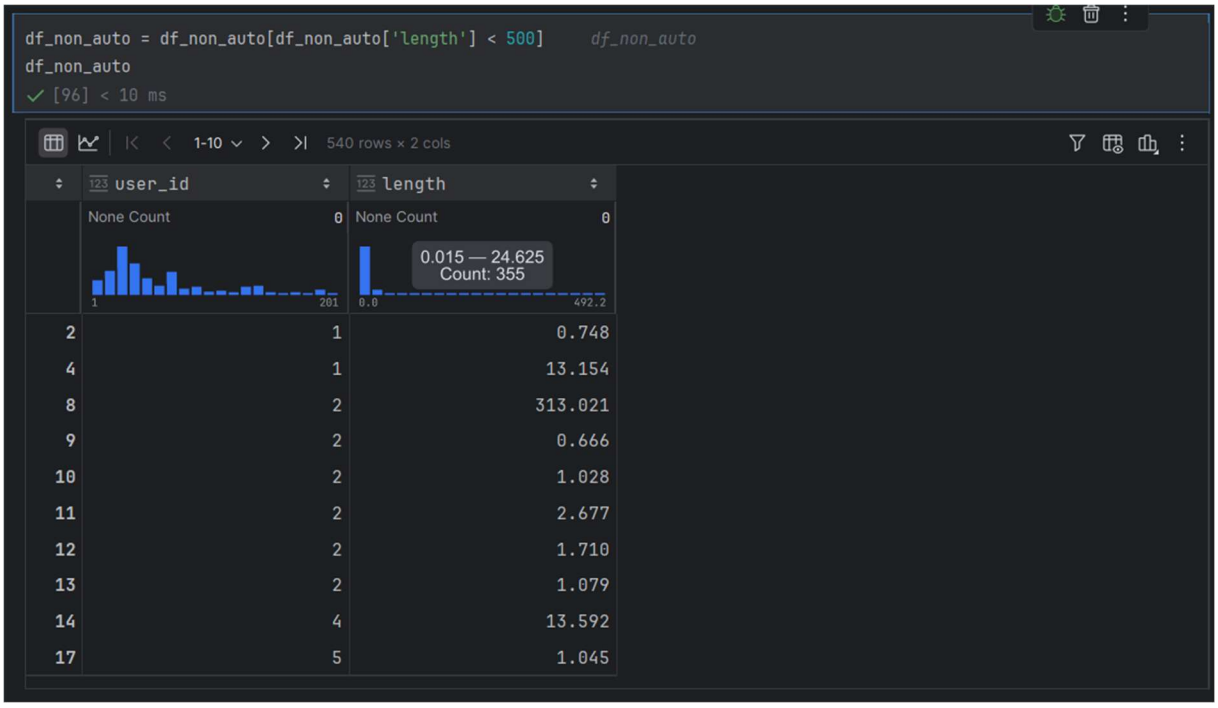
At first glance you can see that majority of manual opened windows have almost three times shorter „lifespan” than majority of auto opened windows. Let’s try to zoom a little bit more those compartments.





For auto opened windows, vast majority have a very short lifespan, more than 70% of windows were open for maximum of 16 minutes, more than half of them – 2 to 3 minures.





while for manually opened windows the lifespan is even shorter. Out of 622 rows, 538 (approximately 87%) were open for a less than 9 minutes, 355 (57%) - not even half a minute.

Open_type/how much	Vast majority	Half of them
Manual	No more than 9 minutes	No more than 16 minutes
Auto	Less than half a minute	Less than 3 minutes

A simple conclusion already arises: There is a significant difference between auto and manual opened windows, numbers don't lie. However, we know nothing about our group of subjects e.g. user4 lowers the average lifespan of auto opened windows, because he/she doesn't need it but didn't turn it off in the settings and closes it as soon as it appears out of habit. Luckily, we have ID's for every log, so let's put it to use.

```
df_auto = df_auto.groupby('user_id')['length'].mean().reset_index()     df_auto
df_auto
```

[98] 13ms

1-10 166 rows × 2 cols

	user_id	length
	None Count 0	None Count 0
	1 — 203	3.396 — 10310.007 Count: 147 / 3.4 — 206135.6
0	1	643.954000
1	2	58.624000
2	3	3140.815105
3	4	27777.630778
4	5	8841.726000
5	6	7413.007778
6	7	39479.047000
7	8	4340.351944
8	9	495.970000
9	10	5087.560408

```
df_auto = df_auto[df_auto['length'] < 10320]     df_auto
df_auto
```

[101] < 10 ms

1-10 147 rows × 2 cols

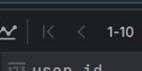
	user_id	length
	None Count 0	None Count 0
	1 — 203	3.396 — 484.054 Count: 88 / 3.4 — 9616.6
0	1	643.954000
1	2	58.624000
2	3	3140.815105
4	5	8841.726000
5	6	7413.007778
7	8	4340.351944
8	9	495.970000
9	10	5087.560408
11	12	6021.673442
12	13	9.581000

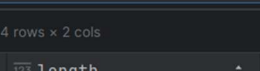
```
df_non_auto = df_non_auto.groupby('user_id')['length'].mean().reset_index()
df_non_auto
```

[102] < 10 ms

1-10 | 144 rows × 2 cols

	user_id	length
	None Count 0	None Count 0
0	1	3526.676625
1	2	53.363500
2	4	13.592000
3	5	25525.585400
4	6	295.383667
5	7	6.448000
6	8	148.362500
7	9	10.880500
8	10	171.381500
9	11	1407.875500



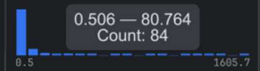



```
df_non_auto = df_non_auto[df_non_auto['length'] < 2263]
df_non_auto
```

[106] < 10 ms

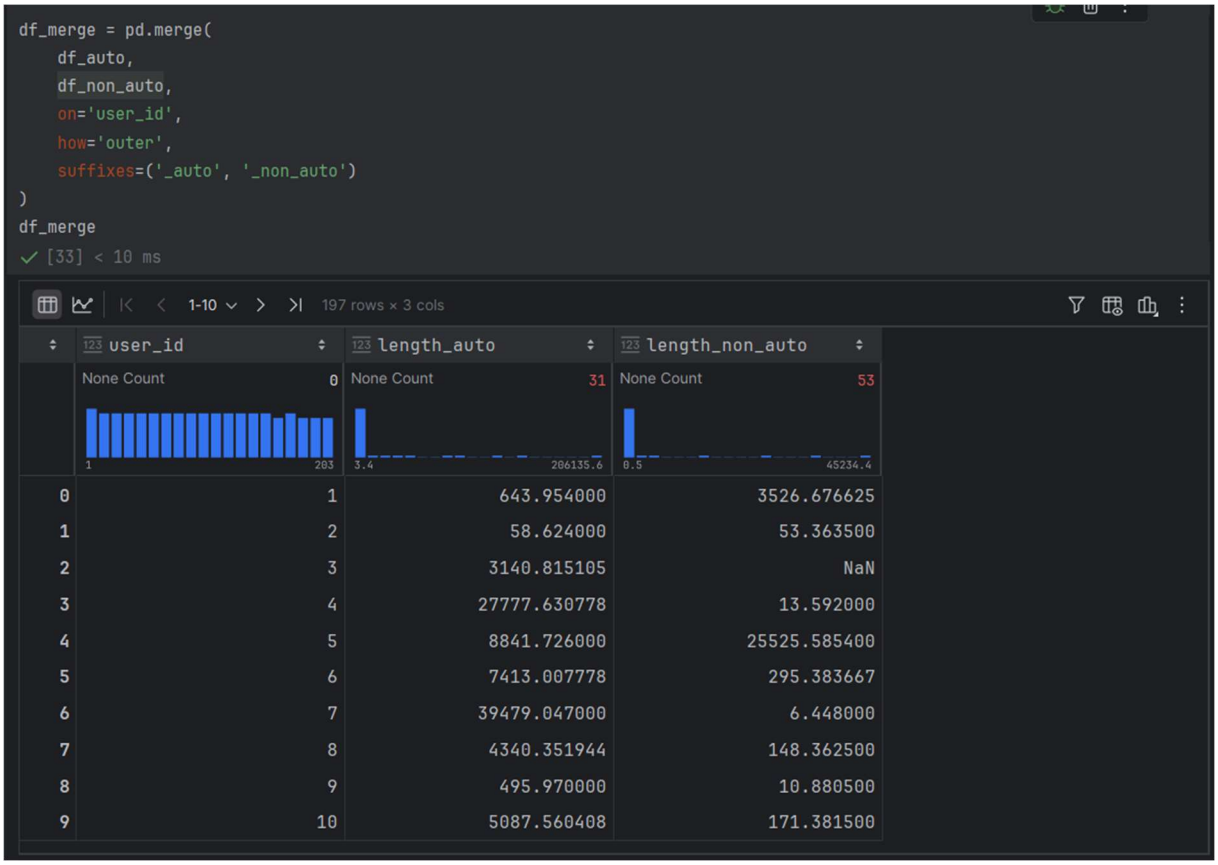
1-10 | 134 rows × 2 cols

	user_id	length
	None Count 0	None Count 0
1	2	53.363500
2	4	13.592000
4	6	295.383667
5	7	6.448000
6	8	148.362500
7	9	10.880500
8	10	171.381500
9	11	1407.875500
10	12	1605.657778
11	14	26.026500



Open_type/How much	Vast majority	Half of them
Manual	93% of users have their manually opened window open for less than an 38 minutes	More than half of users Have their window opened less than 1.5minutes
Auto	Very big span for 89% of users, from few seconds to almost 3 hours	More than half of users Have their window opened less than 10 minutes

After taking into consideration average of every user we can see that our suspicion that some of the users can lower the average is correct, because our previous table showed that manually opened windows lasted longer, however now we see that the roles have reversed. Unfortunately, those numbers are far from perfect: e.g. user1 opens and closes his IDE few times a day, which generate more logs (reason why manually opened windows were winning in table 1), and user2 keeps his IDE opened when he/she is away from computer (reason why automatically opened windows were winning in table 2). So what can we do now? After all, in both dataframes we have (most of the times) the same group of users. Let's compare how much changes lifespan of a window for every user.



```
df_merge = pd.merge(
    df_auto,
    df_non_auto,
    on='user_id',
    how='outer',
    suffixes=('_auto', '_non_auto')
)
df_merge
```

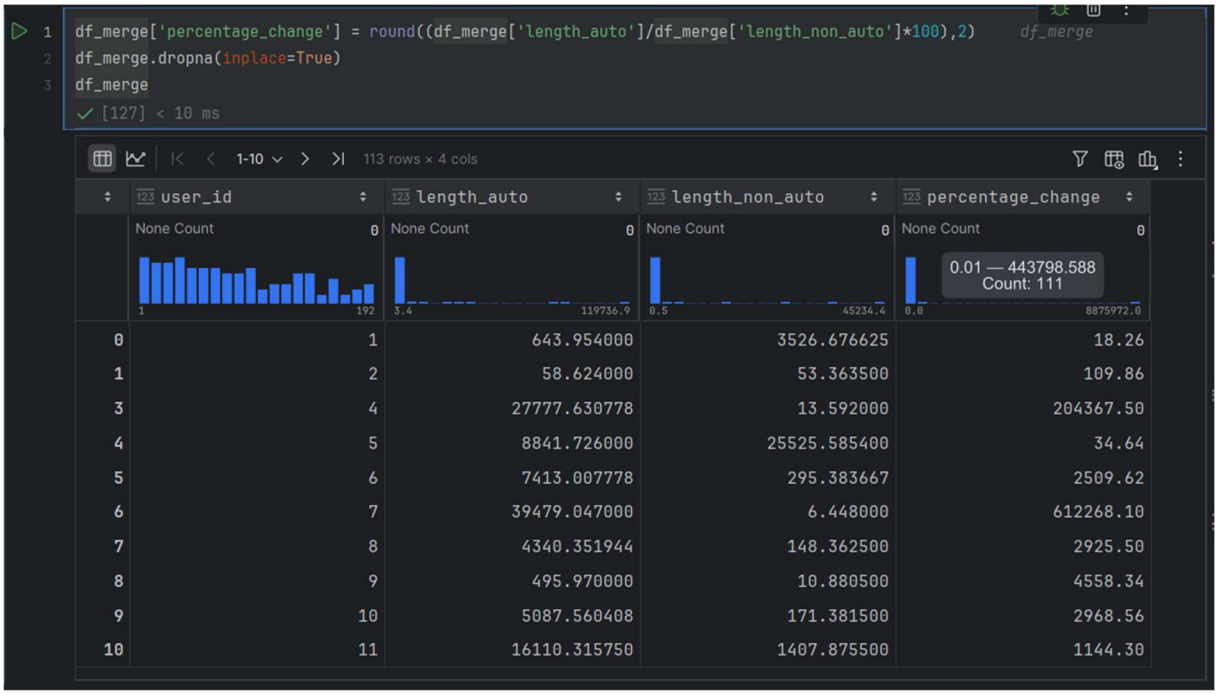
✓ [33] < 10 ms

1-10 197 rows × 3 cols

	user_id	length_auto	length_non_auto
	None Count 0	None Count 31	None Count 53
0	1	643.954000	3526.676625
1	2	58.624000	53.363500
2	3	3140.815105	NaN
3	4	27777.630778	13.592000
4	5	8841.726000	25525.585400
5	6	7413.007778	295.383667
6	7	39479.047000	6.448000
7	8	4340.351944	148.362500
8	9	495.970000	10.880500
9	10	5087.560408	171.381500

This way, we can reduce the group of subjects as some of the users doesn't open their windows manually, and on the other hand – some of the users doesn't open their windows automatically. We don't have to worry about other factors that influence our analysis, such as length of programming sessions, amout of IDE closures etc.

In this scenario, automatically opened windows have statistically longer lifespan, so we will check how much longer they are opened (in percent).



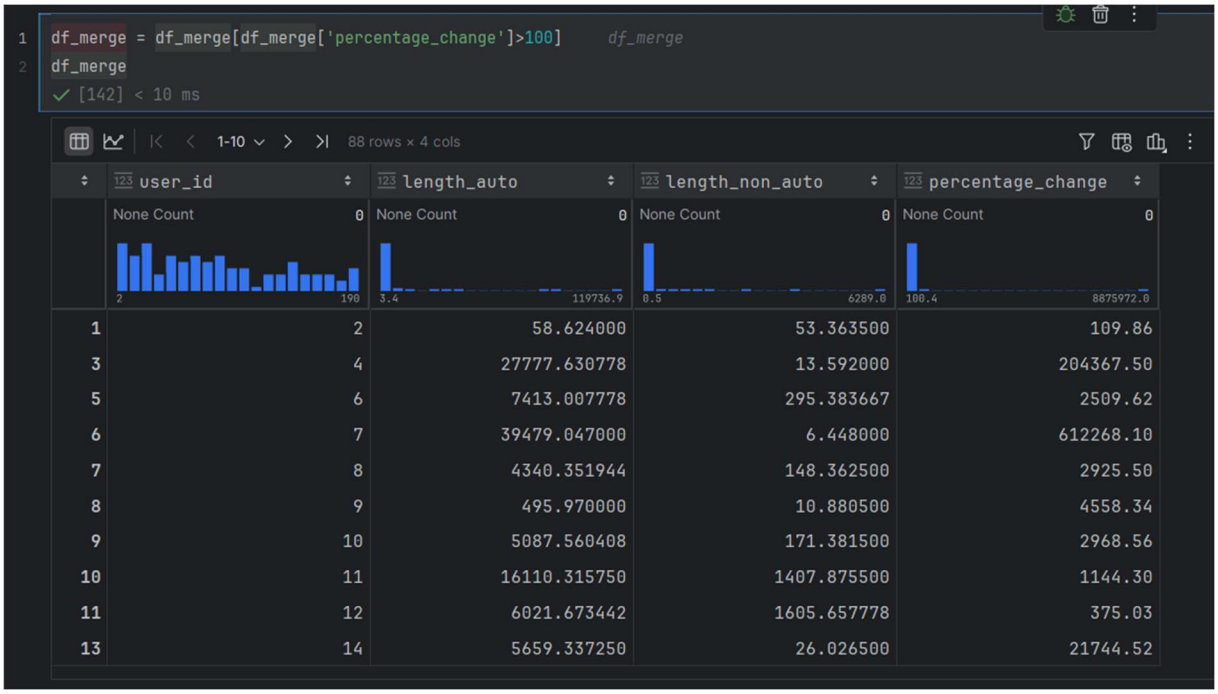
```
df_merge['percentage_change'] = round((df_merge['length_auto']/df_merge['length_non_auto']*100),2)
df_merge.dropna(inplace=True)
df_merge
```

[127] < 10 ms

113 rows × 4 cols

	user_id	length_auto	length_non_auto	percentage_change
	None Count 0	None Count 0	None Count 0	None Count 0
	1 — 192	3.4 — 119736.9	0.5 — 45234.4	0.01 — 443798.588 Count: 111
0	1	643.954000	3526.676625	18.26
1	2	58.624000	53.363500	109.86
3	4	27777.630778	13.592000	204367.50
4	5	8841.726000	25525.585400	34.64
5	6	7413.007778	295.383667	2509.62
6	7	39479.047000	6.448000	612268.10
7	8	4340.351944	148.362500	2925.50
8	9	495.970000	10.880500	4558.34
9	10	5087.560408	171.381500	2968.56
10	11	16110.315750	1407.875500	1144.30

Let’s take into account those records in which percentage_change is higher than 100 (these are the users for whom auto opened windows lasted longer.



```
df_merge = df_merge[df_merge['percentage_change']>100]
df_merge
```

[142] < 10 ms

88 rows × 4 cols

	user_id	length_auto	length_non_auto	percentage_change
	None Count 0	None Count 0	None Count 0	None Count 0
	2 — 190	3.4 — 119736.9	0.5 — 6289.0	100.4 — 8875972.0
1	2	58.624000	53.363500	109.86
3	4	27777.630778	13.592000	204367.50
5	6	7413.007778	295.383667	2509.62
6	7	39479.047000	6.448000	612268.10
7	8	4340.351944	148.362500	2925.50
8	9	495.970000	10.880500	4558.34
9	10	5087.560408	171.381500	2968.56
10	11	16110.315750	1407.875500	1144.30
11	12	6021.673442	1605.657778	375.03
13	14	5659.337250	26.026500	21744.52

We recieved 88 out of 113 records, which means that for majority of users (almost 80%) auto opened windows lasted longer than those opened manually.

To determine wheter there is a meaningful change, we will use Wilcoxon test for dependent samples from scipy library.

We recieved extremely small pvalue - 0.0000000000000003732. It proves, that chances for length_auto being larger than length_non_auto BY ACCIDENT is almost zero.

Due to extreme outliers in the data, the median was used to determine the typical user experience. The median percentage change is +1052.05%. This means that for the typical user, a window opened automatically stays open over 10 times longer than a window opened manually (non-auto).

FINAL CONCLUSION:

Yes, there is a highly significant difference in the tool window's open duration depending on how it was opened. The analysis confirms this difference is both statistically and practically significant.