```
import pandas as pd
import io

import matplotlib.pyplot as plt
import seaborn as sns

sns.set(style="whitegrid", palette="bright", font_scale=1.1)

!git clone
https://personalaccesstoken@github.com/Aaaanastasia/Payoneer_test_task
.git
```

```
Cloning into 'Payoneer_test_task'...
remote: Enumerating objects: 26, done.ote: Counting objects: 100%
(26/26), done.ote: Compressing objects: 100% (24/24), done.ote: Total
26 (delta 5), reused 0 (delta 0), pack-reused 0 (from 0)
```

```
#load all datasets
df_bots =
pd.read_excel('/content/Payoneer_test_task/task/Payoneer_DA_Compliance
_Test.xlsx', index_col=0, sheet_name='Q1_Bots')
#df_bots_info =
pd.read_excel('/content/Payoneer_test_task/task/Payoneer_DA_Compliance
_Test.xlsx', index_col=0, sheet_name='Q2_Assisting_Information')
df_payments =
pd.read_excel('/content/Payoneer_test_task/task/Payoneer_DA_Compliance
_Test.xlsx', index_col=0, sheet_name='Q3_Incoming_Payments')
df_edd =
pd.read_excel('/content/Payoneer_test_task/task/Payoneer_DA_Compliance
_Test.xlsx', index_col=0, sheet_name='Q3_EDD_Reviews')
```

#1. ***Question 1:***

> To identify the bots you sampled a list of Customers registering and their start-time for each step. Please write a query that will print out users and a column with some measure you created that will help you identify if users are bots (The query does not need to compile seamlessly, though you're welcome to have it so)

#2. ***Question 2:***

```
df_bots.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 116 entries, 1798518 to 27287015
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   StepId     116 non-null    int64
 1   StartTime  116 non-null    object
```

```
dtypes: int64(1), object(1)
memory usage: 2.7+ KB
```

```
# Correct datatypes timestamps shoul be in datatime type
df_bots['StartTime'] = pd.to_datetime(df_bots['StartTime'])

#Examine data
df_bots = df_bots.sort_values(['CustomerId', 'StepId', 'StartTime'])
df_bots.head()
```


```
{"summary":"{\n  \"name\": \"df_bots\",\n  \"rows\": 116,\n
\"fields\": [\n    {\n      \"column\": \"CustomerId\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
5370756,\n        \"min\": 1798518,\n        \"max\": 27287015,\n
\"num_unique_values\": 29,\n        \"samples\": [\n
27131046,\n          22523443,\n          19905257\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"StepId\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 1,\n
\"min\": 1,\n        \"max\": 4,\n        \"num_unique_values\": 4,\n
\"samples\": [\n          2,\n          4,\n          1\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"StartTime\",\n
\"properties\": {\n        \"dtype\": \"date\",\n        \"min\":
\"2024-12-02 01:20:20.423000\",\n        \"max\": \"2024-12-30
12:02:58.113000\",\n        \"num_unique_values\": 116,\n
\"samples\": [\n          \"2024-12-23 05:21:41.270000\",\n
\"2024-12-03 08:14:46.800000\",\n          \"2024-12-09
21:28:17.080000\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df_bots"}
```


```
# Now to get the anomaly we need to calculate time between steps per
user
df_bots['Next_step_time'] = df_bots.groupby('CustomerId')
['StartTime'].shift(-1)
df_bots['Time_diff_seconds'] = (df_bots['Next_step_time'] -
df_bots['StartTime']).dt.total_seconds()

# Calculate total time for all steps per user
df_bots['Total_time'] = (df_bots.groupby('CustomerId')
['StartTime'].max() - df_bots.groupby('CustomerId')
['StartTime'].min()).dt.total_seconds()

"""
Lets dive into basic data statistical indicators and features
We already can see that are a large range between the minimum and
maximum time difference between steps and total tipe for steps per
user,
as well as a large difference between the average and the median.
"""
```

```
desc = df_bots.describe().T
desc['empty'] =  len(df_bots) - desc['count']
desc['zero'] = df_bots.isin([0]).sum()
display(desc)
```


```
{"summary":"{\n  \"name\": \"desc\",\n  \"rows\": 5,\n  \"fields\": [\n    {\n      \"column\": \"count\",\n      \"properties\": {\n        \"dtype\": \"date\",\n        \"min\": \"87\",\n        \"max\": 116.0,\n        \"num_unique_values\": 2,\n        \"samples\": [\n          \"87\",\n          116.0\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"mean\",\n      \"properties\": {\n        \"dtype\": \"date\",\n        \"min\": \"1970-01-01 00:00:00.000000002\",\n        \"max\": \"2024-12-15 19:50:58.869229824\",\n        \"num_unique_values\": 5,\n        \"samples\": [\n          \"2024-12-15 19:50:33.914473984\",\n          151.89344827586203\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"min\",\n      \"properties\": {\n        \"dtype\": \"date\",\n        \"min\": \"1970-01-01 00:00:00\",\n        \"max\": \"2024-12-02 01:21:33.110000\",\n        \"num_unique_values\": 5,\n        \"samples\": [\n          \"2024-12-02 01:20:20.423000\",\n          2.3\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"25%\",\n      \"properties\": {\n        \"dtype\": \"date\",\n        \"min\": \"1970-01-01 00:00:00.000000001\",\n        \"max\": \"2024-12-06 16:35:19.388499968\",\n        \"num_unique_values\": 5,\n        \"samples\": [\n          \"2024-12-06 16:34:49.276000\",\n          143.6\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"50%\",\n      \"properties\": {\n        \"dtype\": \"date\",\n        \"min\": \"1970-01-01 00:00:00.000000002\",\n        \"max\": \"2024-12-17 15:08:36.312999936\",\n        \"num_unique_values\": 5,\n        \"samples\": [\n          \"2024-12-17 15:08:20.046499840\",\n          166.307\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"75%\",\n      \"properties\": {\n        \"dtype\": \"date\",\n        \"min\": \"1970-01-01 00:00:00.000000003\",\n        \"max\": \"2024-12-23 05:21:40.820000256\",\n        \"num_unique_values\": 5,\n        \"samples\": [\n          \"2024-12-23 05:21:40.594999808\",\n          195.083\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"max\",\n      \"properties\": {\n        \"dtype\": \"date\",\n        \"min\": \"1970-01-01 00:00:00.000000004\",\n        \"max\": \"2024-12-30 12:02:58.113000\",\n        \"num_unique_values\": 4,\n        \"samples\": [\n          \"2024-12-30 12:02:58.113000\",\n          237.524\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"std\",\n      \"properties\": {\n        \"dtype\": \"date\",\n        \"min\": 1.122884484360431,\n        \"max\": 65.34075903557537,\n
```

\"num_unique_values\": 3,\n   \"samples\": [\n
1.122884484360431,\n   31.145079408489767\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"empty\",\n   \"properties\": {\
n   \"dtype\": \"date\",\n   \"min\": 0.0,\n   \"max\":
\"29\",\n   \"num_unique_values\": 2,\n   \"samples\": [\n
\"29\",\n   0.0\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"zero\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 0,\n   \"min\": 0,\n   \"max\": 0,\n
\"num_unique_values\": 1,\n   \"samples\": [\n   0\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   }\n]\n}","type":"dataframe","variable_name":"desc"}

```
#Lets research the difference per steps per users more
user_stats = df_bots.groupby('CustomerId', as_index=False)
[['Time_diff_seconds']].agg([
    'mean', 'min', 'max', 'std', 'count'
]).droplevel(axis=1, level=0).rename(columns={
    '' : 'customerId',
    'mean': 'avg_time',
    'min': 'min_time',
    'max': 'max_time',
    'std': 'std_time',
    'count': 'steps_count'
})

user_stats.head()
```

{"summary":"{\n \"name\": \"user_stats\",\n \"rows\": 29,\n
\"fields\": [\n   {\n   \"column\": \"customerId\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
5442210,\n   \"min\": 1798518,\n   \"max\": 27287015,\n
\"num_unique_values\": 29,\n   \"samples\": [\n
27131046,\n   22523443,\n   19905257\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"avg_time\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\":
22.07002521297785,\n   \"min\": 0.7666666666666666,\n
\"max\": 79.17466666666667,\n   \"num_unique_values\": 29,\n
\"samples\": [\n   59.25333333333333,\n
46.42666666666667,\n   65.02766666666666\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"min_time\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\":
17.135443733195167,\n   \"min\": 0.5,\n   \"max\": 56.02,\n
\"num_unique_values\": 29,\n   \"samples\": [\n   5.157,\n
20.96,\n   54.403\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"max_time\",\n   \"properties\": {\n   \"dtype\":

```
\"number\",\n        \"std\": 35.68497050690693,\n        \"min\":
0.9,\n        \"max\": 136.956,\n        \"num_unique_values\": 28,\n
\"samples\": [\n          84.213,\n          68.436,\n
79.567\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"std_time\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 16.550862960367557,\n        \"min\":
0.051316014394468805,\n        \"max\": 68.99774148719169,\n
\"num_unique_values\": 29,\n        \"samples\": [\n
68.99774148719169,\n          25.065143792392917,\n
10.410550241621877\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"steps_count\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 3,\n
\"max\": 3,\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          3\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"user_stats"}
```

```
# Lets calculate the threshold for time difference disribution
statiscically
Q1 = user_stats['avg_time'].quantile(0.25)
Q3 = user_stats['avg_time'].quantile(0.75)
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
print(f"IQR Threshold (lower bound): {lower_bound:.2f} seconds")
```

```
IQR Threshold (lower bound): 22.13 seconds
```

Threshold Calculation for Bot Detection

To identify users with suspiciously fast registration times, we used the **Interquartile Range (IQR)** method, becouse unlike standard deviation, IQR is not affected by extreme values and is great for behavioral data that may not be normally distributed:

- **IQR (Interquartile Range)**: `IQR = Q3 - Q1` — represents the middle 50% spread of the data
- **Lower bound** for outlier detection: `Q1 - 1.5 * IQR`

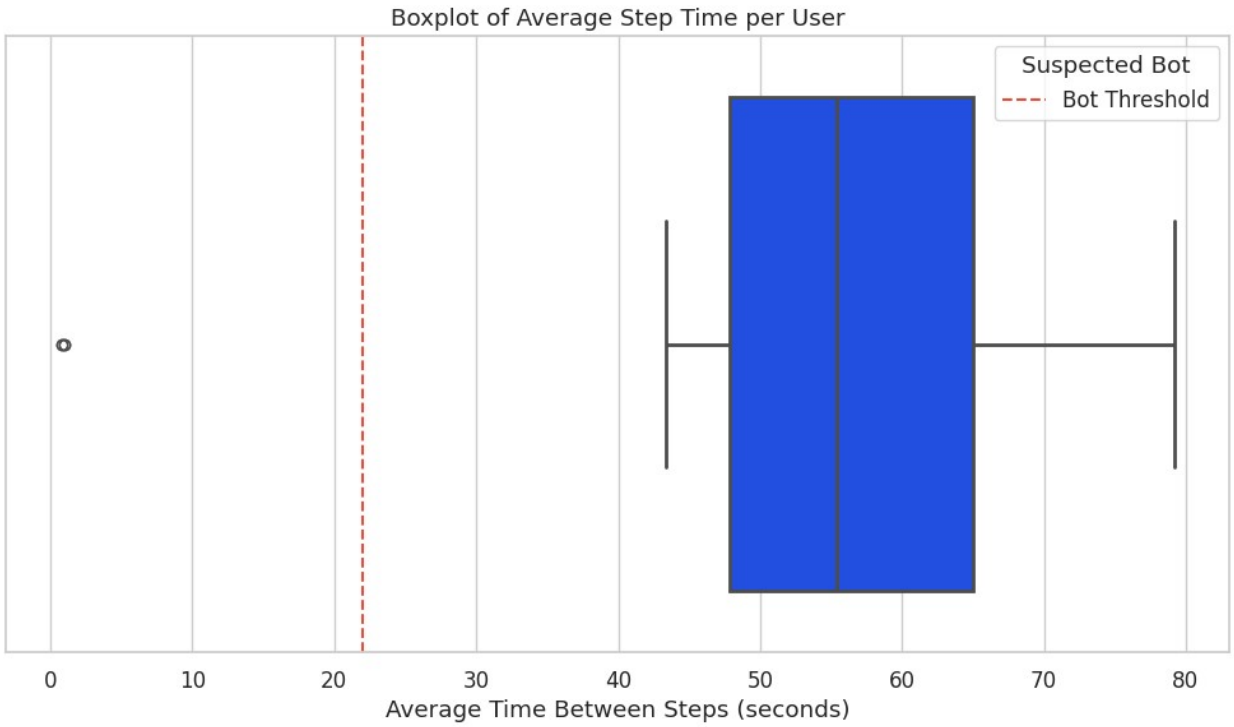
Users whose **average step time is below this lower bound** are flagged as potential bots due to their unusually fast progression through the registration steps.

```
# Sort out the style
#sns.set(style="whitegrid", palette="pastel", font_scale=1.1)
sns.set(style="whitegrid", palette="bright", font_scale=1.1)

# Create the figure
plt.figure(figsize=(10, 6))
```

```
# Boxplot
sns.boxplot(
    x='avg_time',
    data=user_stats,
    #color='#AED6F1',
    linewidth=2
)

# Aesthetics
plt.axvline(22, color='#E74C3C', linestyle='--', label='Bot
Threshold')
plt.title('Boxplot of Average Step Time per User')
plt.xlabel('Average Time Between Steps (seconds)')
plt.legend(title='Suspected Bot', loc='upper right')
plt.tight_layout()
plt.show()
```



Average Step Time Per User

The boxplot above shows the distribution of **average time between steps** during user registration.

The **red line** at 22 seconds marks a threshold below which user behavior is considered suspicious.

Users flagged as **suspected bots** (in red) tend to have:

- Extremely fast completion times between steps (often < 1 second),
- Very **low variance**, suggesting automated behavior.

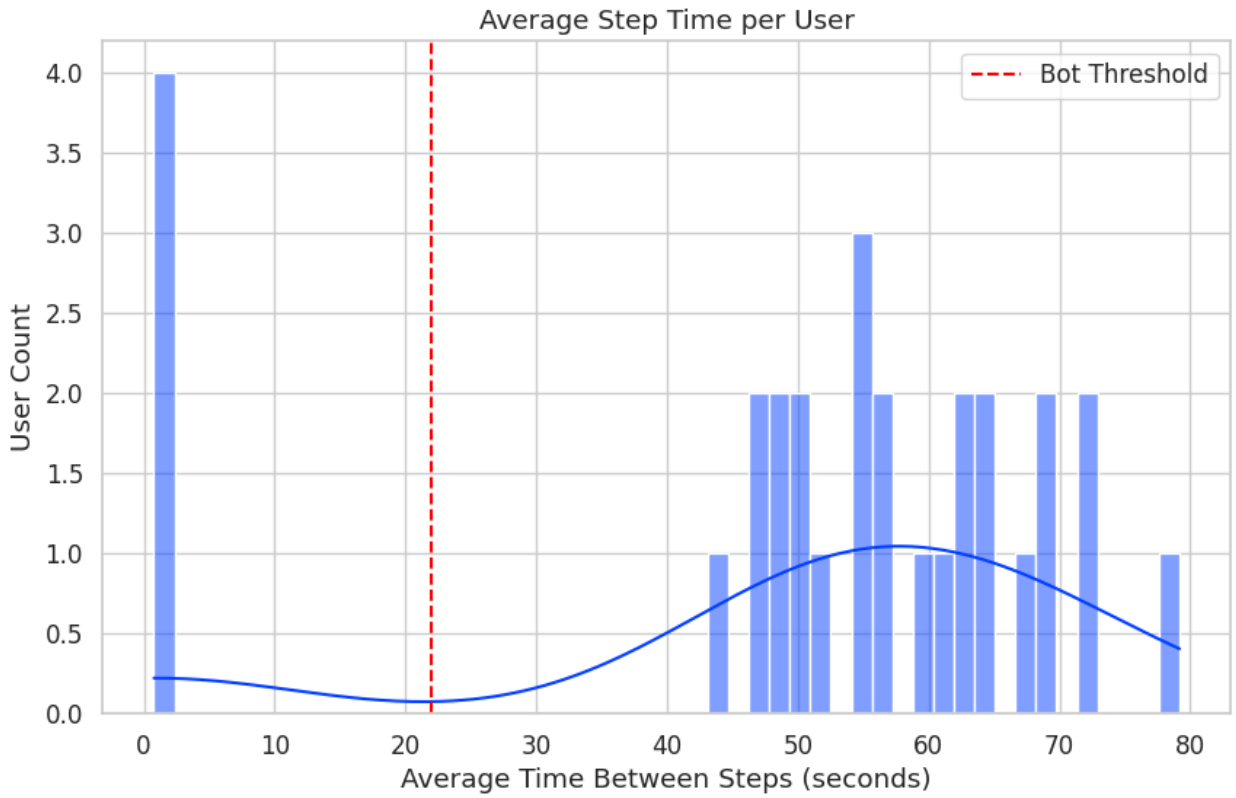
```
# Lets detect suspected bots
user_stats['suspected_bot'] = user_stats['avg_time'] < lower_bound

user_stats.head()
```


```
{"summary":"{\n  \"name\": \"user_stats\",\n  \"rows\": 29,\n
\"fields\": [\n    {\n      \"column\": \"customerId\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
5442210,\n        \"min\": 1798518,\n        \"max\": 27287015,\n
\"num_unique_values\": 29,\n        \"samples\": [\n
27131046,\n          22523443,\n          19905257\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"avg_time\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
22.07002521297785,\n        \"min\": 0.7666666666666666,\n
\"max\": 79.17466666666667,\n        \"num_unique_values\": 29,\n
\"samples\": [\n          59.25333333333333,\n
46.42666666666667,\n          65.02766666666666\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"min_time\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
17.135443733195167,\n        \"min\": 0.5,\n        \"max\": 56.02,\n
\"num_unique_values\": 29,\n        \"samples\": [\n          5.157,\n
20.96,\n          54.403\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"max_time\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 35.68497050690693,\n        \"min\":
0.9,\n        \"max\": 136.956,\n        \"num_unique_values\": 28,\n
\"samples\": [\n          84.213,\n          68.436,\n
79.567\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"std_time\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 16.550862960367557,\n        \"min\":
0.051316014394468805,\n        \"max\": 68.99774148719169,\n
\"num_unique_values\": 29,\n        \"samples\": [\n
68.99774148719169,\n          25.065143792392917,\n
10.410550241621877\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"steps_count\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 3,\n
\"max\": 3,\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          3\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"suspected_bot\",\n      \"properties\": {\n        \"dtype\":
\"boolean\",\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          true\n        ],\n        \"semantic_type\": \"\",\n
```

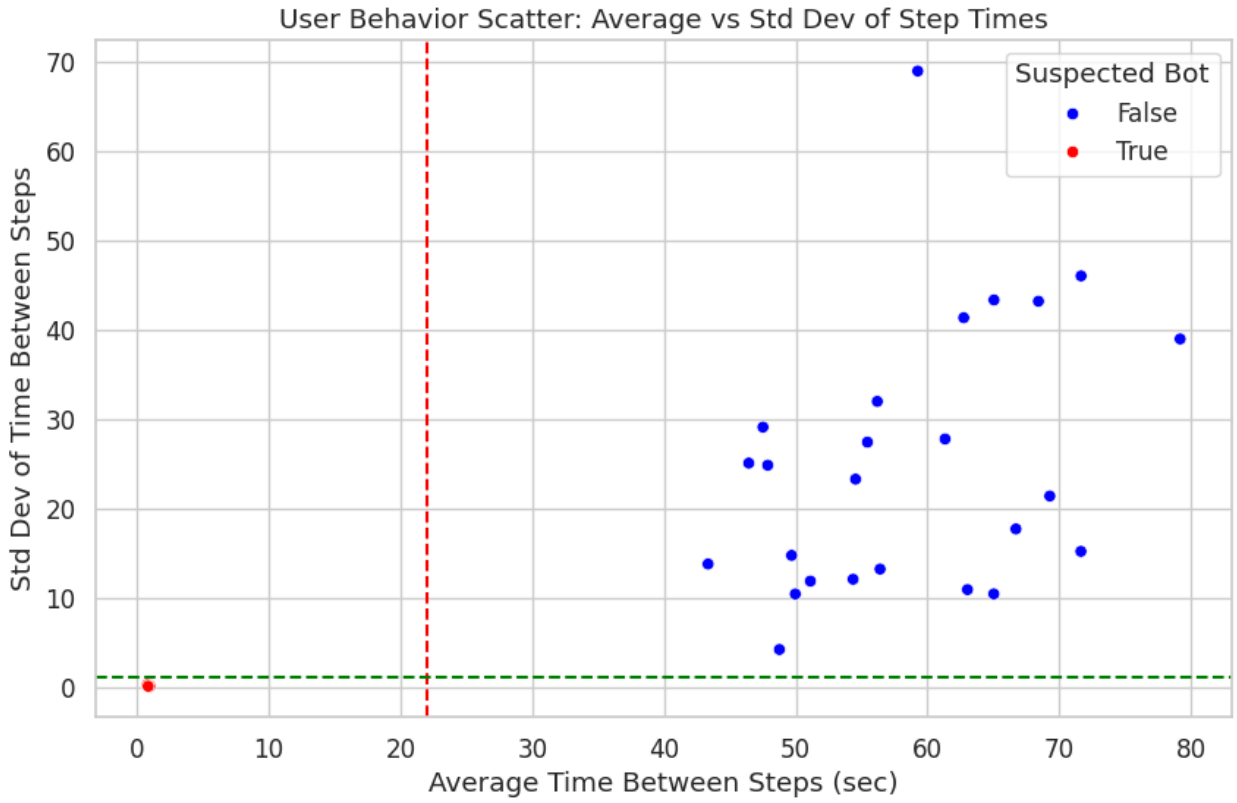
```
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"user_stats"}

# Distribution of average step time
plt.figure(figsize=(10, 6))
sns.histplot(user_stats['avg_time'], bins=50, kde=True)
plt.axvline(22, color='red', linestyle='--', label='Bot Threshold')
plt.title('Average Step Time per User')
plt.xlabel('Average Time Between Steps (seconds)')
plt.ylabel('User Count')
plt.legend()
plt.show()
```




```
# Scatter plot: average vs std dev
plt.figure(figsize=(10, 6))
sns.scatterplot(
    data=user_stats,
    x='avg_time',
    y='std_time',
    hue='suspected_bot',
    palette={True: 'red', False: 'blue'}
)
plt.axvline(22, color='red', linestyle='--')
plt.axhline(1, color='green', linestyle='--')
```

```
plt.title('User Behavior Scatter: Average vs Std Dev of Step Times')
plt.xlabel('Average Time Between Steps (sec)')
plt.ylabel('Std Dev of Time Between Steps')
plt.legend(title='Suspected Bot')
plt.show()
```



User Step Timing – Distribution and Behavioral Patterns

To detect suspected bots, we analyzed the **average time between registration steps** for each user. Two key visualizations support this analysis:

---

1. Histogram: Average Step Time per User

This chart shows the distribution of users' average time between form steps.

- The **KDE curve** (blue) illustrates the density of user behavior.
- The **red dashed line** at 22 seconds marks the **IQR-based lower bound threshold** for detecting anomalies.
- Users to the **left of this threshold** submitted information significantly faster than the majority, a strong indication of automated activity.

---

2. Scatter Plot: Average Time vs Standard Deviation

This plot provides a two-dimensional view of user behavior by comparing:

- **X-axis**: Average time between steps
- **Y-axis**: Standard deviation of those times (how consistent the user was)

Key insights:

- **Red dots** represent users flagged as suspected bots.
- Bots tend to cluster in the **lower-left corner** — showing both low average time and very little variation (i.e., extremely fast and consistent — typical for scripts).
- The **green line** at Std Dev = 1 and **red vertical line** at Avg Time = 22 seconds mark the soft thresholds for natural behavior.

Together, these plots clearly highlight **outliers who behave too quickly and consistently** to be genuine users — strengthening our bot detection model both statistically and visually.

```
print(f"Suspected bots: {user_stats['suspected_bot'].sum()} users")

suspected =
user_stats[user_stats['suspected_bot']].sort_values('avg_time')
display(suspected.head(20))
```

```
Suspected bots: 4 users
```


```
{"summary":"{\n  \"name\": \"display(suspected\",\n  \"rows\": 4,\n
\"fields\": [\n    {\n      \"column\": \"customerId\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
4367369,\n        \"min\": 15578183,\n        \"max\": 25008550,\n
\"num_unique_values\": 4,\n        \"samples\": [\n
25008550,\n          20057177,\n          24332668\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"avg_time\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.08904846786795172,\n        \"min\": 0.7666666666666666,\n
\"max\": 0.9566666666666667,\n        \"num_unique_values\": 4,\n
\"samples\": [\n          0.8300000000000001,\n
0.9566666666666667,\n          0.7666666666666666\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"min_time\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.16938614661968868,\n        \"min\": 0.5,\n        \"max\": 0.9,\n
\"num_unique_values\": 4,\n        \"samples\": [\n          0.79,\n
0.9,\n          0.5\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"max_time\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.09574271077563384,\n        \"min\":
0.9,\n        \"max\": 1.1,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          0.9,\n          1.1,\n          1.0\n
```

```
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"std_time\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.09492236063386823,\n        \"min\": 0.051316014394468805,\n
\"max\": 0.23094010767585035,\n        \"num_unique_values\": 4,\n
\"samples\": [\n          0.06082762530298214,\n
0.051316014394468805,\n          0.23094010767585035\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"steps_count\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0,\n        \"min\": 3,\n        \"max\": 3,\n
\"num_unique_values\": 1,\n        \"samples\": [\n          3\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"suspected_bot\",\n
\"properties\": {\n        \"dtype\": \"boolean\",\n
\"num_unique_values\": 1,\n        \"samples\": [\n          true\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe"}
```


Summary: Bot Detection During Registration

Total users analyzed:

- ~30 (based on visual review)
- Users flagged as bots: 4 users

These users had:

- Average time between 0.76 and 0.95 seconds
- Very low standard deviation in timing (e.g., 0.05 – 0.23s)
- All completed 3 steps in rapid, consistent succession.

#3. ***Question 3:***

> Assuming the payments in sheet "Q3_Incoming_Payments" remain in balance - that is, no funds are leaving the user between payments - Write a query that will output the list of users who had their EDD review after they received a total of 1000 dollars. The output fields are not critical (what columns you include in the output) as long as it contains only the requested list of users. For example, user 48638892 had his EDD review after he had already loaded a total of ~$1,108, so we'd expect to see him in the output.

```
df_payments.info()
df_edd.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 41 entries, 32894092 to 90077471
Data columns (total 2 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
```

```
 0   Transaction_Datetime  41 non-null     object
 1   Volume_Amount_USD     41 non-null     float64
dtypes: float64(1), object(1)
memory usage: 984.0+ bytes
<class 'pandas.core.frame.DataFrame'>
Index: 6 entries, 32894092 to 90077471
Data columns (total 1 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   edd_review  6 non-null      object
dtypes: object(1)
memory usage: 96.0+ bytes
```

```
# Changing the type of date columns into datetime format
df_payments['Transaction_Datetime'] =
pd.to_datetime(df_payments['Transaction_Datetime'], format='ISO8601',
errors='coerce')
df_edd['edd_review'] = pd.to_datetime(df_edd['edd_review'])

df_payments.head()
```


```
{"summary":"{\n  \"name\": \"df_payments\",\n  \"rows\": 41,\n
\"fields\": [\n    {\n      \"column\": \"CustomerID\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
22363580,\n        \"min\": 32894092,\n        \"max\": 90077471,\n
\"num_unique_values\": 6,\n        \"samples\": [\n
32894092,\n          48638892,\n          90077471\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Transaction_Datetime\",\n
\"properties\": {\n        \"dtype\": \"date\",\n        \"min\":
\"2024-12-01 03:24:00.117000\",\n        \"max\": \"2025-01-13
12:08:18.423000\",\n        \"num_unique_values\": 37,\n
\"samples\": [\n          \"2024-12-30 00:32:14.460000\",\n
\"2025-01-01 05:25:39.297000\",\n          \"2024-12-15
21:31:43.243000\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Volume_Amount_USD\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 130.24078760199149,\n        \"min\":
33.5,\n        \"max\": 650.72,\n        \"num_unique_values\": 35,\n
\"samples\": [\n          215.5922,\n          251.6,\n
335.3193\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df_payments"}
```


```
df_edd.head()
```


```
{"summary":"{\n  \"name\": \"df_edd\",\n  \"rows\": 6,\n  \"fields\":
[\n    {\n      \"column\": \"CustomerID\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 23630703,\n        \"min\":
32894092,\n        \"max\": 90077471,\n        \"num_unique_values\":
```

```
6,\n        \"samples\": [\n          32894092,\n          48638892,\n
90077471\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"edd_review\",\n      \"properties\": {\n        \"dtype\":
\"date\",\n        \"min\": \"2024-12-15 21:41:43.243000\",\n
\"max\": \"2025-01-11 12:08:18.423000\",\n
\"num_unique_values\": 6,\n        \"samples\": [\n          \"2024-
12-15 21:41:43.243000\",\n          \"2025-01-03 05:25:39.297000\",\n
\"2025-01-05 11:32:15.330000\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df_edd"}
```


```
# Lets calculate cumulative balance and sort data by transaction date
per user
df_payments['cumulative_usd'] = df_payments.groupby('CustomerID')
['Volume_Amount_USD'].cumsum()
df_payments = df_payments.sort_values(['CustomerID',
'Transaction_Datetime'])

df_payments.head()
```


```
{"summary":"{\n  \"name\": \"df_payments\",\n  \"rows\": 41,\n
\"fields\": [\n    {\n      \"column\": \"CustomerID\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
22363580,\n        \"min\": 32894092,\n        \"max\": 90077471,\n
\"num_unique_values\": 6,\n        \"samples\": [\n
32894092,\n          48638892,\n          90077471\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Transaction_Datetime\",\n
\"properties\": {\n        \"dtype\": \"date\",\n        \"min\":
\"2024-12-01 03:24:00.117000\",\n        \"max\": \"2025-01-13
12:08:18.423000\",\n        \"num_unique_values\": 37,\n
\"samples\": [\n          \"2024-12-30 00:32:14.460000\",\n
\"2025-01-01 05:25:39.297000\",\n          \"2024-12-15
21:31:43.243000\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Volume_Amount_USD\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 130.24078760199149,\n        \"min\":
33.5,\n        \"max\": 650.72,\n        \"num_unique_values\": 35,\n
\"samples\": [\n          215.5922,\n          251.6,\n
335.3193\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"cumulative_usd\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 445.925059734096,\n        \"min\":
35.2,\n        \"max\": 1577.3873,\n        \"num_unique_values\":
41,\n        \"samples\": [\n          808.0124000000001,\n
1108.6,\n          250.0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df_payments"}
```

```
# Now lets calculate the first time each user crossed $1000
cross_time = (
    df_payments[df_payments['cumulative_usd'] >= 1000]
    .groupby('CustomerID')['Transaction_Datetime']
    .min()
    .reset_index()
    .rename(columns={'Transaction_Datetime': 'time_crossed_1000'})
)

# Now lets merge (left join) two datasets and check for violations
combined = cross_time.merge(df_edd, on='CustomerID', how='left')
violations = combined[combined['edd_review'] >
combined['time_crossed_1000']]

# Show results
print("Customers who passed $1000 before EDD review:")
print(violations['CustomerID'].tolist())
```

```
Customers who passed $1000 before EDD review:
[48638892, 84242950]
```

```
# Classify customers
combined['late_edd'] = combined['edd_review'] >
combined['time_crossed_1000']
summary = combined['late_edd'].value_counts().rename({True: 'Late
EDD', False: 'On Time EDD'})

plt.figure(figsize=(6, 4))
sns.barplot(x=summary.index, y=summary.values, palette='Set2')
plt.title('EDD Review')
plt.ylabel('Number of Customers')
plt.show()
```

```
<ipython-input-99-b75e73c50ceb>:6: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.barplot(x=summary.index, y=summary.values, palette='Set2')
```

EDD Compliance Summary

The bar chart above summarizes whether customers passed the **Enhanced Due Diligence (EDD)** review **before** or **after** exceeding the $1,000 threshold in incoming payments.

- **"On Time EDD"**: Customers who were reviewed before loading $1,000, in line with policy.
- **"Late EDD"**: Customers who crossed the threshold before their EDD review, indicating a compliance issue.

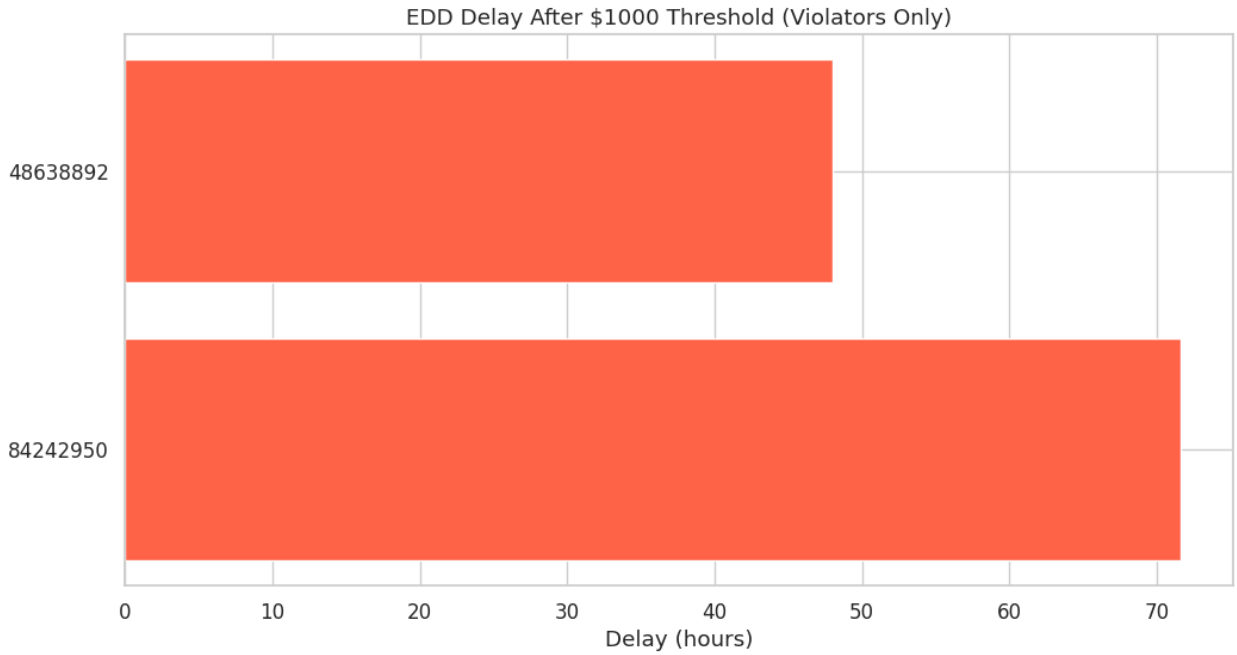
This classification was done by comparing each customer's cumulative payment timestamp with their EDD review time. The chart helps Compliance teams quickly assess whether procedural timing is being followed and highlights areas where policy enforcement may need to be strengthened.

```
plot_df = violations.copy()
plot_df['edd_review'] = pd.to_datetime(plot_df['edd_review'])
plot_df['time_crossed_1000'] =
pd.to_datetime(plot_df['time_crossed_1000'])

plt.figure(figsize=(12, 6))
plot_df = plot_df.sort_values('edd_review')

plt.barh(plot_df['CustomerID'].astype(str), (plot_df['edd_review'] -
plot_df['time_crossed_1000']).dt.total_seconds()/3600,
         color='tomato')
```

```
plt.xlabel('Delay (hours)')
plt.title('EDD Delay After $1000 Threshold (Violators Only)')
plt.show()
```



EDD Delay Analysis for Policy Violators

This horizontal bar chart displays the **delay (in hours)** between the time customers **crossed the $1,000 threshold** and when their **EDD review** was completed — for users who **violated** the compliance policy.

- Each bar represents a customer who received their EDD review **after** they had already received more than $1,000 in incoming payments.
- The **length of the bar** corresponds to the delay in hours between the two events.

This visualization provides a **clear measurement of compliance lag** per customer and helps prioritize reviews where delays were significant. It is particularly useful for identifying recurring issues or systemic slowdowns in the EDD process.

Summary: EDD Timing Compliance

1. Users Who Violated the EDD Timing Policy

- 2 customers were flagged as having their EDD review after crossing the $1000 threshold. (ID: 48638892, 84242950). They loaded over 1000 into their account before the EDD review was completed, which violates the policy.

2. Compliance Breakdown

- 4 users had their EDD review on time
- 2 users had late EDD reviews

3. EDD Delay Insight

- The delay between crossing $1000 and EDD review ranged from a few hours to over a day, depending on the customer.