

VNUHCM-UNIVERSITY OF SCIENCE
FACULTY OF INFORMATION TECHNOLOGY



21KHDL - Intelligent Data Analysis

PROJECT REPORT

Instructors

Mr. Nguyễn Tiến Huy
Mr. Nguyễn Trần Duy Minh
Mr. Lê Thanh Tùng

Student

21127038 – Võ Phú Hân

TABLE OF CONTENTS

I - INTRODUCTION.....	3
II - DATA DISCOVERY PROCESS	3
1. Data basic overview	3
1.1. Traing and test data	3
1.2. Labels data	6
2. Exploring data.....	8
2.1. session_id	9
2.2. index	10
2.3. elapsed_time.....	10
2.4. event properties: name, type, id	11
2.5. game room.....	13
2.6. level and level_group	13
2.7. click geo-location	16
2.8. text.....	20
2.9. page	21
2.10. hover.....	22
2.11. game properties: fullscreen, hq, music	23
III - DATA PREPROCESSING.....	23
1. Train and labels data.....	23
2. Feature selection data.....	24
IV - FEATURE ENGINEERING PROCESS.....	25
V - MODEL BUILDING	26
1. Main idea	26
2. About XGBoost.....	26
2.1. XGBoost Introduction	26
2.2. How it work	26
2.3. Pros	26
2.4. Cons.....	27
VI - EXPERIMENT	27
1. Training	27
2. Testing.....	29
3. Results achieved, comments and analysis.....	30
4. Conclusion and development direction.....	30
5. References.....	31

I - INTRODUCTION

This project explores a dataset from a [Kaggle competition](#) focused on predicting student performance in game-based learning environments. The dataset, provided by Field Day Lab, contains detailed game logs and aims to advance research in knowledge-tracing methods. Using this data, I developed models to predict student performance, with the goal of contributing to the creation of more effective educational games. This report summarizes my approach, results, and key insights.

II - DATA DISCOVERY PROCESS

1. Data basic overview

The dataset for this [Kaggle competition](#) focuses on predicting whether players will answer questions correctly in an online educational game. The data is delivered incrementally through a time series API, grouped by three checkpoints: Level 4, Level 12, and Level 22, with a total of 18 questions across all sessions. Training data includes past player interactions and labels indicating whether each question was answered correctly, while test data provides access to previous events for each checkpoint. Predictions are made iteratively, and the results are submitted as a `submission.csv` file. For more information about the dataset and competition, please refer to the [official data description](#).

The input data include:

- training set with game sessions data `train.csv`;
- correct answers for all questions for each session `train_labels.csv`;
- test set with game sessions data `test.csv`;
- sample submission file `sample_submission.csv`.

1.1. *Training and test data*

First, let have a quick look about data's features:

Column	Meaning
session_id	the ID of the session the event took place in
index	the index of the event for the session
elapsed_time	how much time has passed (in milliseconds) between the start of the session and when the event was recorded
event_name	the name of the event type
name	the event name (e.g. identifies whether a notebook_click is opening or closing the notebook)
level	what level of the game the event occurred in (0 to 22)
page	the page number of the event (only for notebook-related events)

room_coor_x	the coordinates of the click in reference to the in-game room (only for click events)
room_coor_y	the coordinates of the click in reference to the in-game room (only for click events)
screen_coor_x	the coordinates of the click in reference to the player's screen (only for click events)
screen_coor_y	the coordinates of the click in reference to the player's screen (only for click events)
hover_duration	how long (in milliseconds) the hover happened for (only for hover events)
text	the text the player sees during this event
fqid	the fully qualified ID of the event
room_fqid	the fully qualified ID of the room the event took place in
text_fqid	the fully qualified ID of the
fullscreen	whether the player is in fullscreen mode
hq	whether the game is in high-quality
music	whether the game music is on or off
level_group	which group of levels - and group of questions - this row belongs to (0-4, 5-12, 13-22)

About the data shapes:

Train data shape: (26296946, 20)

Test data shape: (3728, 21)

Because train.csv is pretty big, ~5 GB large with 26 million records, if we use the default pandas scheme, the memory for storing the dataframe would be enormous. Configure the scheme as below to load data into memory with smaller footprint.

```
dtypes = { 'session_id': 'category',
           'elapsed_time': np.int32,
           'event_name': 'category',
           'name': 'category',
           'level': np.uint8,
           'page': 'category',
           'room_coor_x': np.float32,
           'room_coor_y': np.float32,
           'screen_coor_x': np.float32,
           'screen_coor_y': np.float32,
           'hover_duration': np.float32,
           'text': 'category',
           'fqid': 'category',
           'room_fqid': 'category',
           'text_fqid': 'category',
```

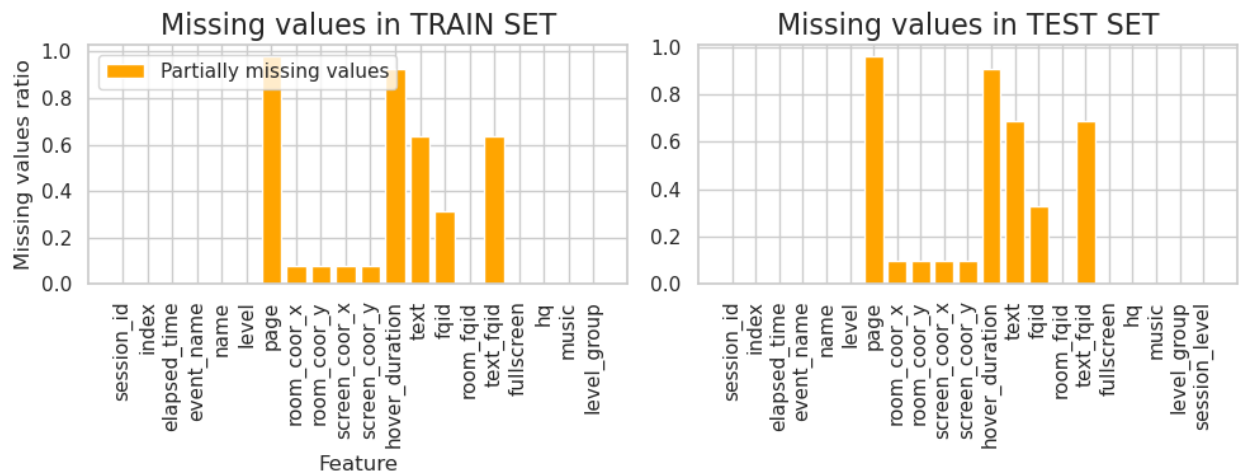
```
'fullscreen': np.int8,
'hq': np.int8,
'music': np.int8,
'level_group': 'category'}
```

As we can see, there are 10 numerical features and 10 categorical features.

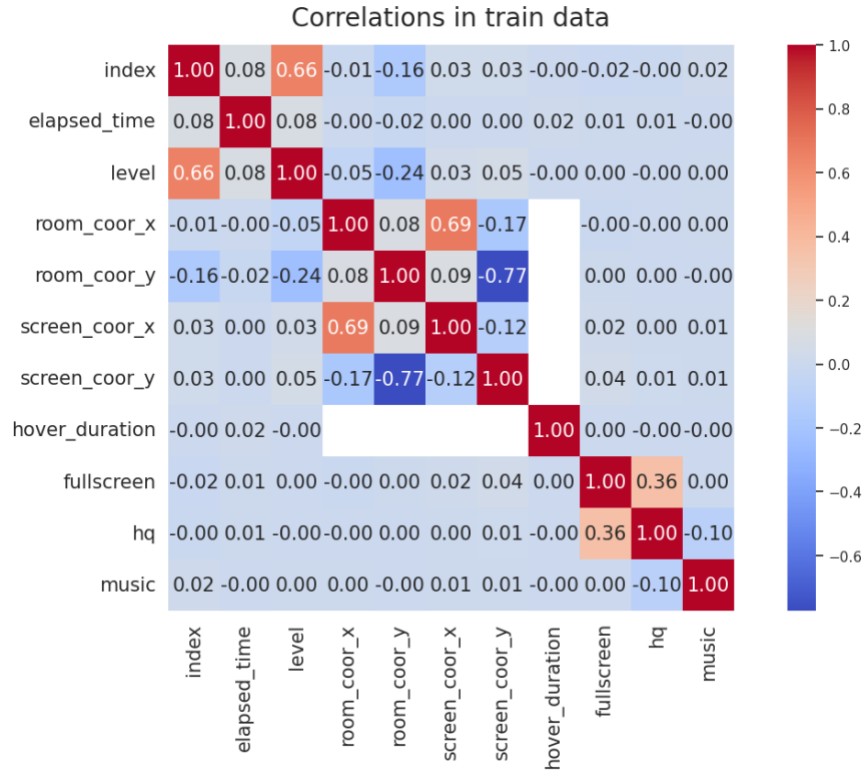
This is some records of the train data.

	session_id	index	elapsed_time	event_name	name	level	page	room_coor_x	room_coor_y	screen_coor_x	screen_coor_y	hover_duration	text	fqid
0	20090312431273200	0	0	cutscene_click	basic	0	NaN	-413.991394	-159.314682	380.0	494.0	NaN	undefined	intro
1	20090312431273200	1	1323	person_click	basic	0	NaN	-413.991394	-159.314682	380.0	494.0	NaN	Whatcha doing over there, Jo?	gramps
2	20090312431273200	2	831	person_click	basic	0	NaN	-413.991394	-159.314682	380.0	494.0	NaN	Just talking to Teddy.	gramps
3	20090312431273200	3	1147	person_click	basic	0	NaN	-413.991394	-159.314682	380.0	494.0	NaN	I gotta run to my meeting!	gramps
4	20090312431273200	4	1863	person_click	basic	0	NaN	-412.991394	-159.314682	381.0	494.0	NaN	Can I come, Gramps?	gramps

Let see some statistic and correlation among features. The charts below show missing ratios in train and test dataset:



There are a significant number of missing values in some columns. However, it's worth noting that both the training and test dataframes exhibit similar ratios of missing values, which is a positive aspect. We may consider dropping columns with more than 80% missing data. The decision to retain or discard columns with a high missing value ratio will depend on evaluating their relevance to the predictive modeling process.



The coordinate features have some correlations because screen_coor is part of room_coor. Level and index also show correlation, likely because at higher levels, a user needs to complete more events. I will discuss this further in the following sections.

1.2. Labels data

	session_id	correct
0	20090312431273200_q1	1
1	20090312433251036_q1	0
2	20090312455206810_q1	1
3	20090313091715820_q1	0
4	20090313571836404_q1	1

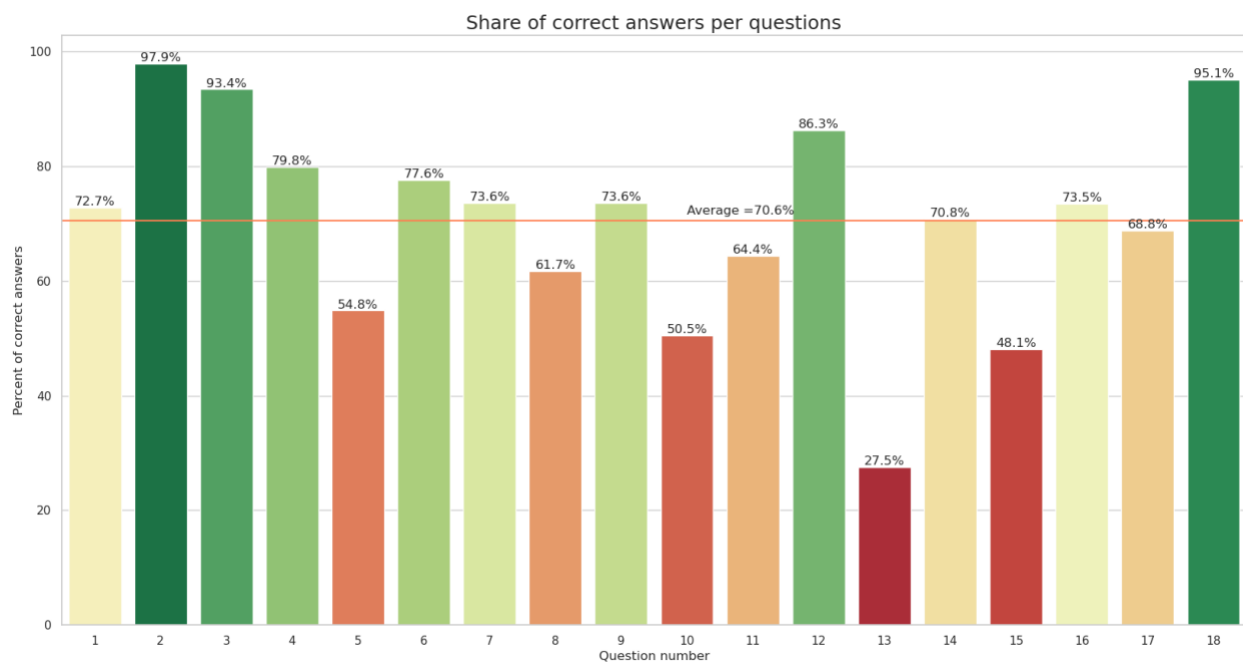
Labels data shape: (424116, 2)

`train_labels.csv` includes two values:

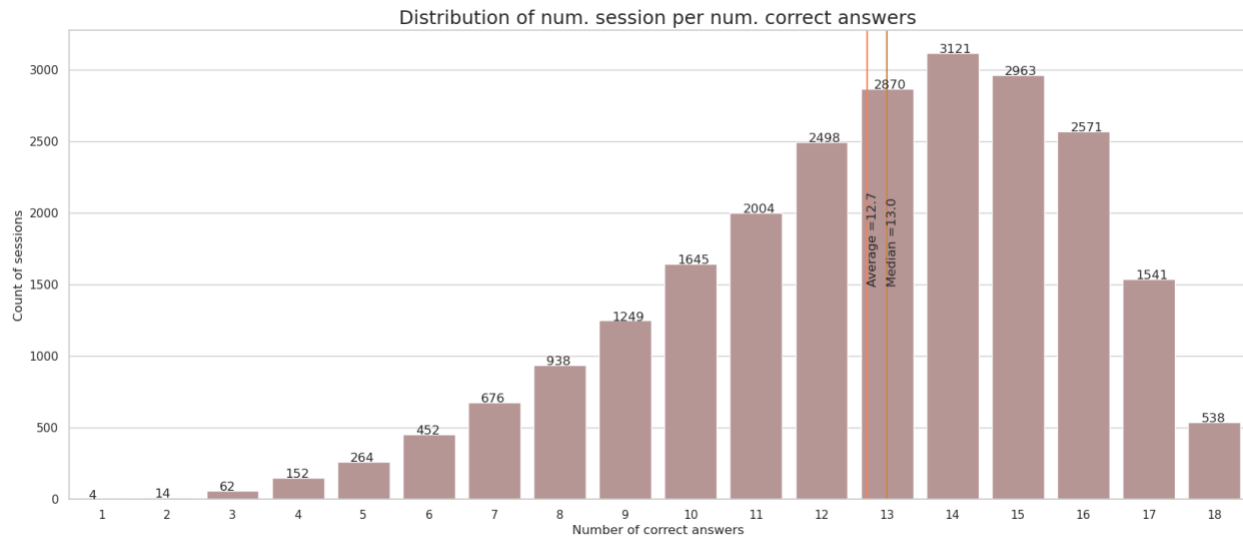
- session_id: does not equal to session_id from the training set, this is combination <session_id>_<question #>;
- correct: flag for correct (1) or incorrect (0) answer.

About 70% of the answers are correct, so the dataset is slightly imbalanced:

	count	mean	std	min	25%	50%	75%	max
correct	424116.0	0.705635	0.455757	0.0	0.0	1.0	1.0	1.0



Questions Q2 and Q18 have over 95% of their answers marked as correct, so during model training, we may safely skip these two questions and set their correct flag to true. On the other hand, Q13 has the lowest percentage of correct answers, with only 27.5% of answers correct.



Each session completes all 18 questions. No session has failed to make at least one correct answer on the first attempt. In total, 50% of the sessions achieve at least 13 correct answers on their first attempt.

2. Exploring data

Jo Wilder game is an example of the point-and-click genre. To progress through a game, the player must find hidden objects and/or answer quizzes.

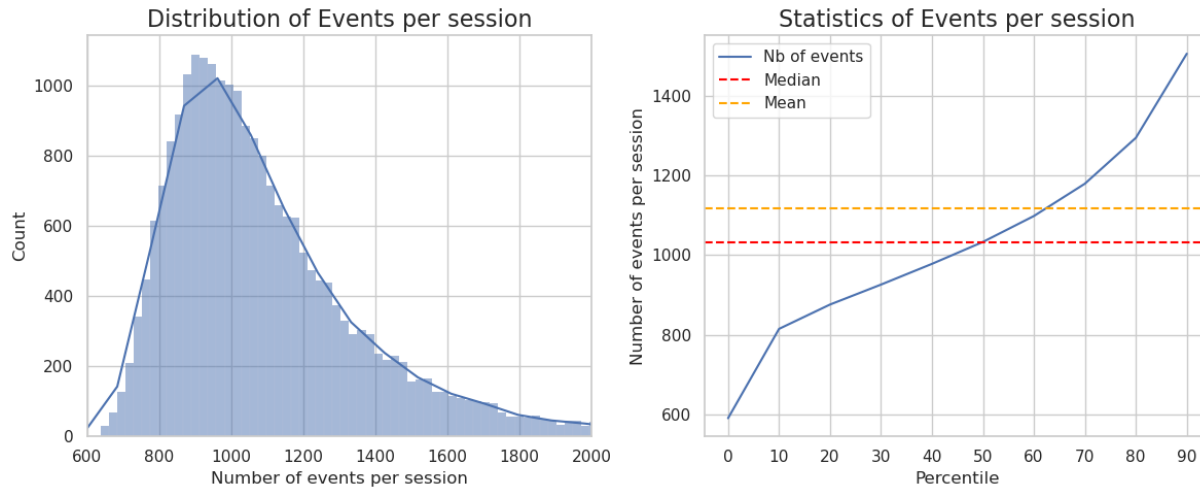
Data matched to the game setup:

- Each game is a session defined by `session_id`
- Each row in the data is a game event, defined by name, type (`event_name`), unique ID (`fqid`), and game progress index (`index`)
- The player progresses by moving from one game room (`room_fqid`) and level (`level`) to another
- Remaining columns refer to specific events:
 - For click events, the coordinates of the click are defined in `room_coor_x` and `room_coor_y` (in reference to the in-game room) or in `screen_coor_x` and `screen_coor_y` (in reference to the player's screen).
 - For notebook-related events `page` identifies the page number;
 - For hover events `hover_duration` shows how long was the hover.

You can play the game here: <https://pbswisconsineducation.org/jowilder/play-the-game/>. The game code source is public: https://github.com/fielddaylab/jo_wilder

2.1. *session_id*

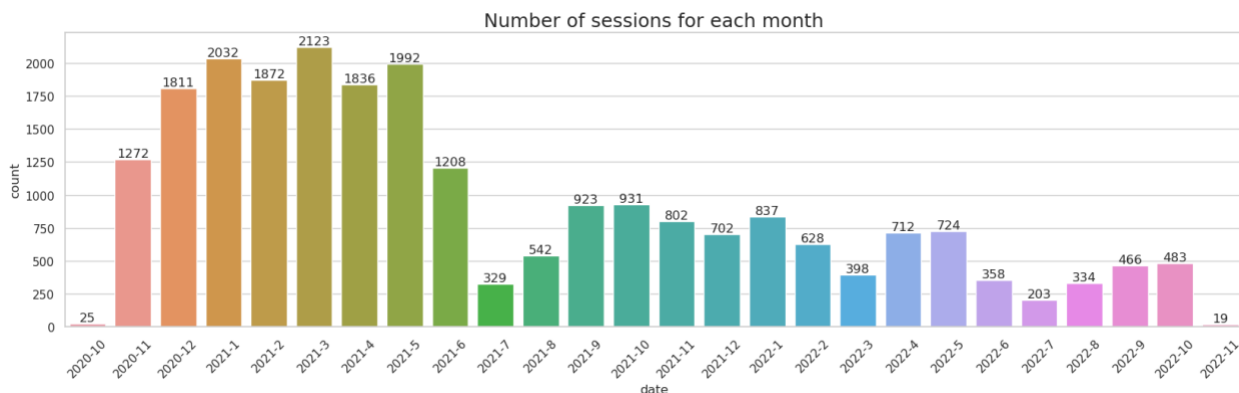
We have 23562 unique sessions in the train, but only 3 in the test.



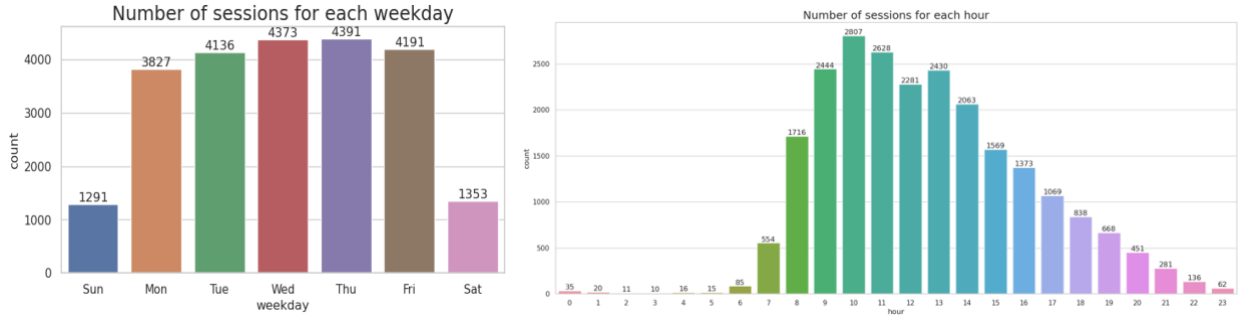
A session typically includes several hundred to a few thousand individual events. Both the median and the mean are slightly above 1000. The distribution appears to be normal with a slight positive skew.

Next, as noticed by [@pdnartreb](#) in their [notebook](#), *session_id* likely contains date and time information, however, the significance of the last two digits remains unclear. Therefore, I have split the *session_id* to analyse and found some insights:

The data were collected from Oct 2020 to Nov 2022



The game is mostly played during the school days (Mon to Fri) and working hours (8 am. to 18 pm.)



2.2. *index*

Index is the index of the event in a session. Despite naive expectation, I have analysed and found out that there are many problems with index:

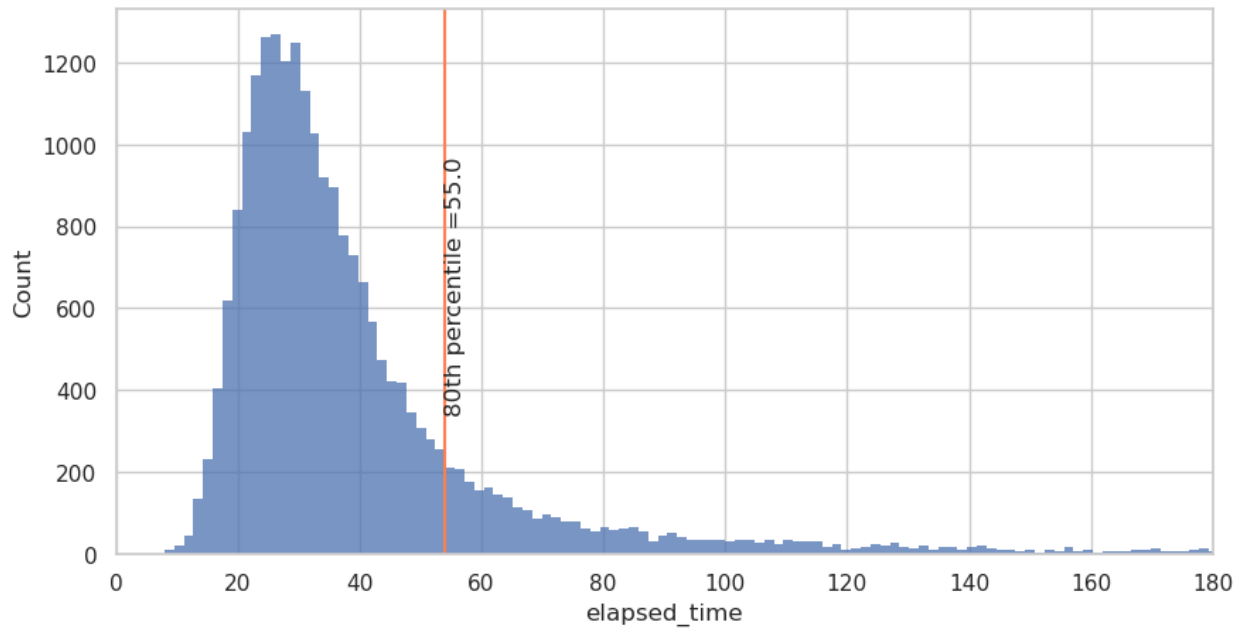
- Combination of session_id and index is not unique, there are duplicates.
- Some sessions have no index = 0, which is the start event of the game.
- Some sessions have the "reversed index" phenomenon which is the index in that session do not monotonic increase.

The competition host [states](#) that all of this might be due to data errors

2.3. *elapsed_time*

elapsed_time shows how much time has passed (in milliseconds) between the start of the session and when the event was recorded. Same as index, elapsed_time also has reversed phenomenon. This bug with negative time change seems to [be present](#) in the hidden test data, used for evaluation.

Assumpt last elapsed_time in a session is the time the session finish, I have found that 80% of session finish before 1 hour, however, there are still some sessions last many days:



2.4. *event properties: name, type, id*

Let's now look at event properties:

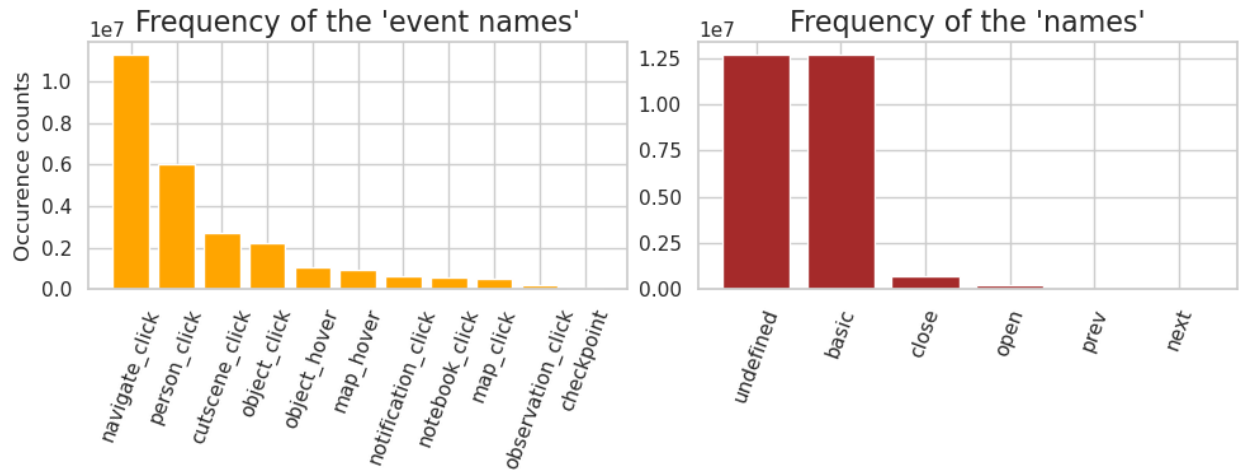
- `event_name` is the event type (e.g. person click, cutscene click, object hover, etc)
- `name` is the event name (e.g. identifies whether a `notebook_click` is opening or closing the notebook)

Meaning of the `event_name`:

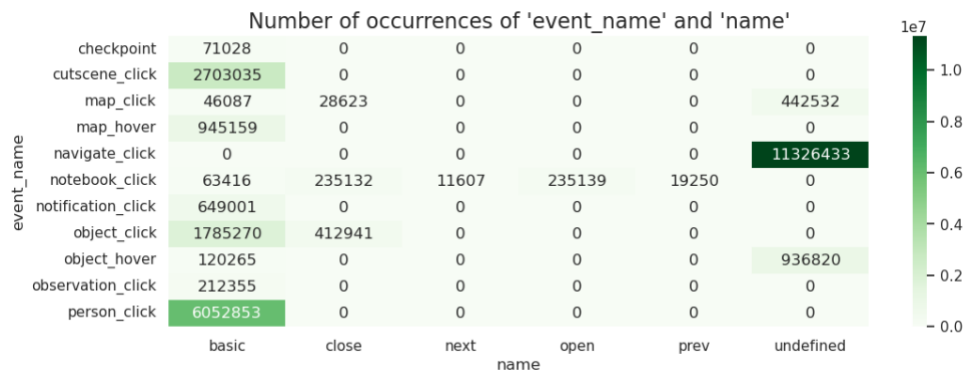
- *navigate_click*: click on the place we want move protagonist (Jo) to move;
- *observation_click*: click on the black object that appears when you take your mouse pointer close to them;
- *notification_click*: after navigate click on an object like the notebook, the retirement letter - the click to hide or to continue after the text appears;
- *object_click*: click anywhere on the object pop up (only after *notification_click*)
- *object_hover*: the student takes the mouse pointer above an object;
- *map_hover*: the student takes the pointer above any place in the map;
- *map_click*: click on a place on the map to go there;
- *notebook_click*: click on notebook to look for notes made by Jo;
- *checkpoint*: the last event of a level-group/chapter in the data.

For additional details see [this notebook](#).

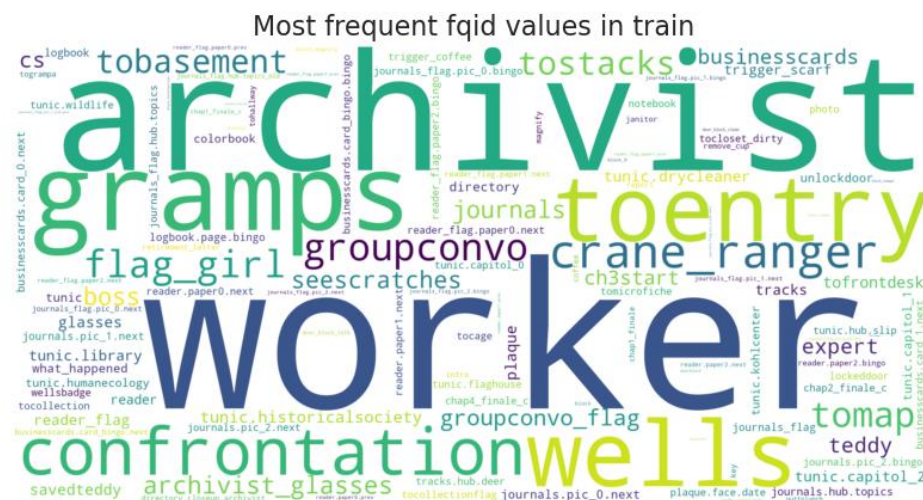
There are 11 unique event_names, 6 names and 128 fqid. The charts below show the frequencies of event_name and name, we can see that the events mainly consist of clicks and occasionally hovers:



And certain names only occur with specific event_names, not all combinations are possible.

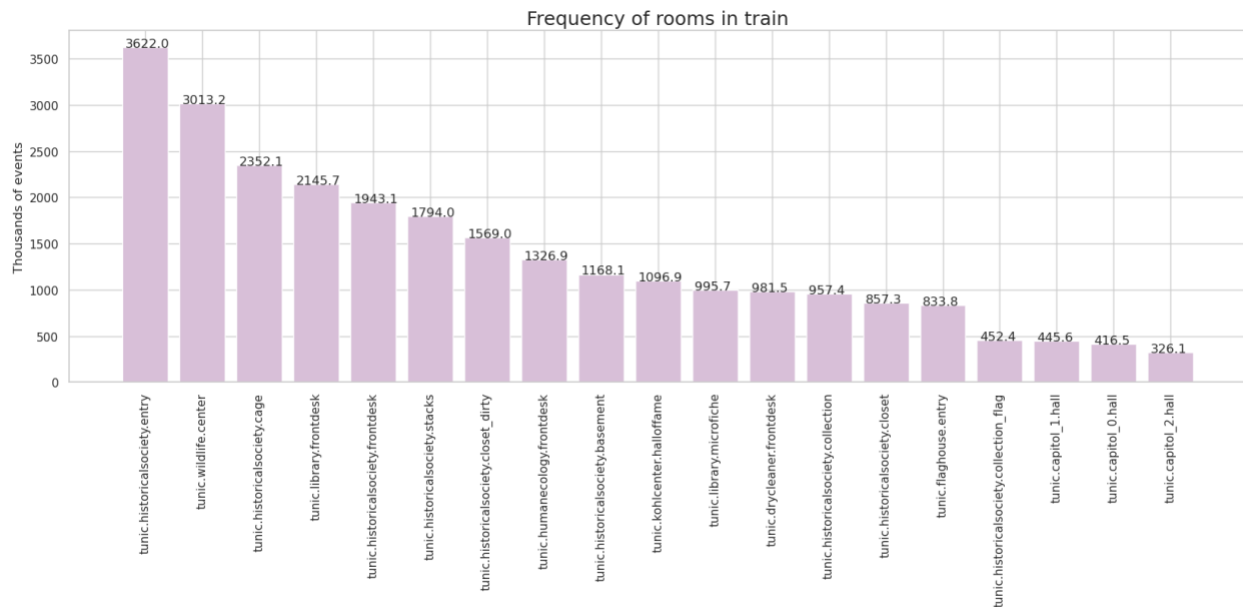


Fqid is the fully qualified ID of the event (it refers to the person/object the player is interacting with). This is the word cloud of fqid.



2.5. *game room*

room_fqid is the ID of the room the event took place in. The player progresses through the game by moving from one room to another. There are 19 game rooms in total. The chart below show the number of events happen in each game room:



2.6. *level and level_group*

level is what level of the game the event occurred in (0 to 22). level_group represents which group of levels - and group of questions - this row belongs to (0-4, 5-12, 13-22)

The number of questions in each level_group is fixed:

level_group	questions	number of questions
0-4	q1 to q3	3
5-12	q4 to q13	10
13-22	q14 to q18	5

At levels 4, 12, 22 the game has *checkpoint* events when questions are being asked. However there are sessions in which one of the 22 levels is not present. Most likely this is due to data record issues:

Number of sessions in which one of the 22 levels is not present: 577

Level 7 is missing in 576 sessions

Examples of sessions with missing 7 level: ['20090316190523732', '20100007445515820', '20100014211946468', '20100018421509572', '20100020122406090']

Level 15 is missing in 17 sessions

Examples of sessions with missing 15 level: ['20100020122406090', '20100110093663916', '20100309472366890', '20100512192183730', '20110410570739092']

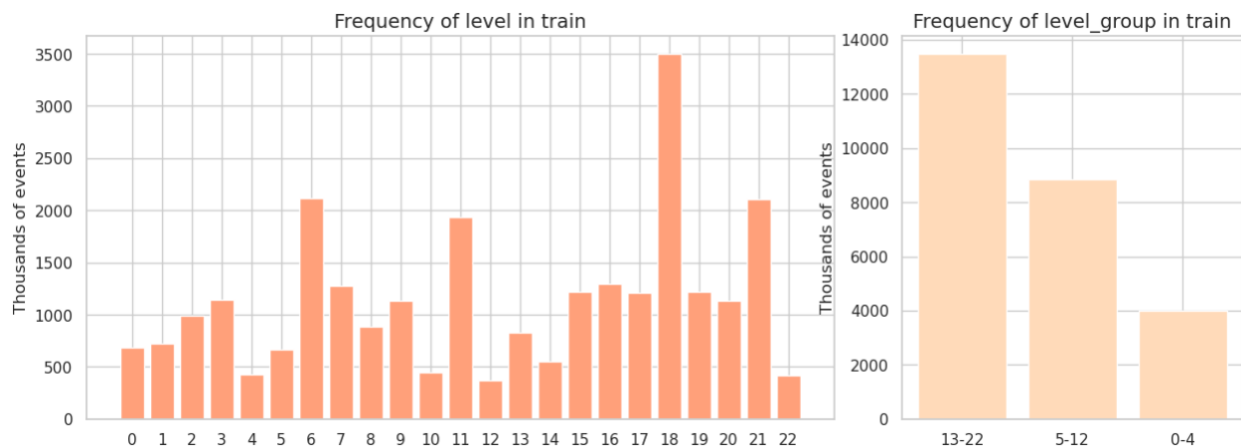
Level 20 is missing in 561 sessions

Examples of sessions with missing 20 level: ['20090316190523732', '20100007445515820', '20100014211946468', '20100018421509572', '20100115005512812']

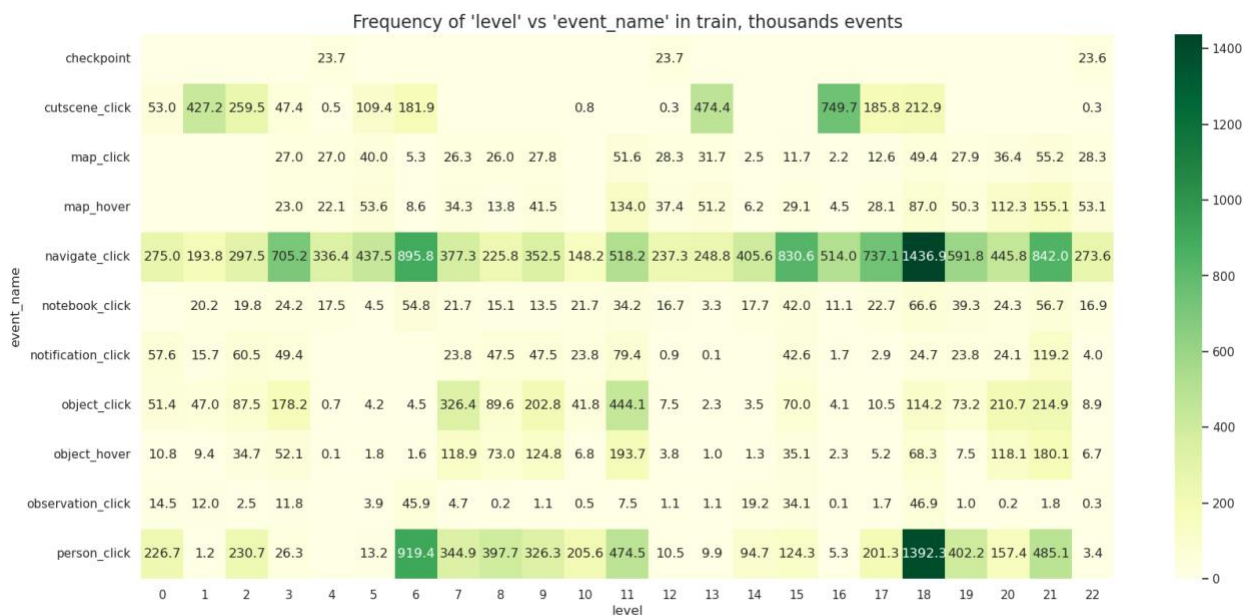
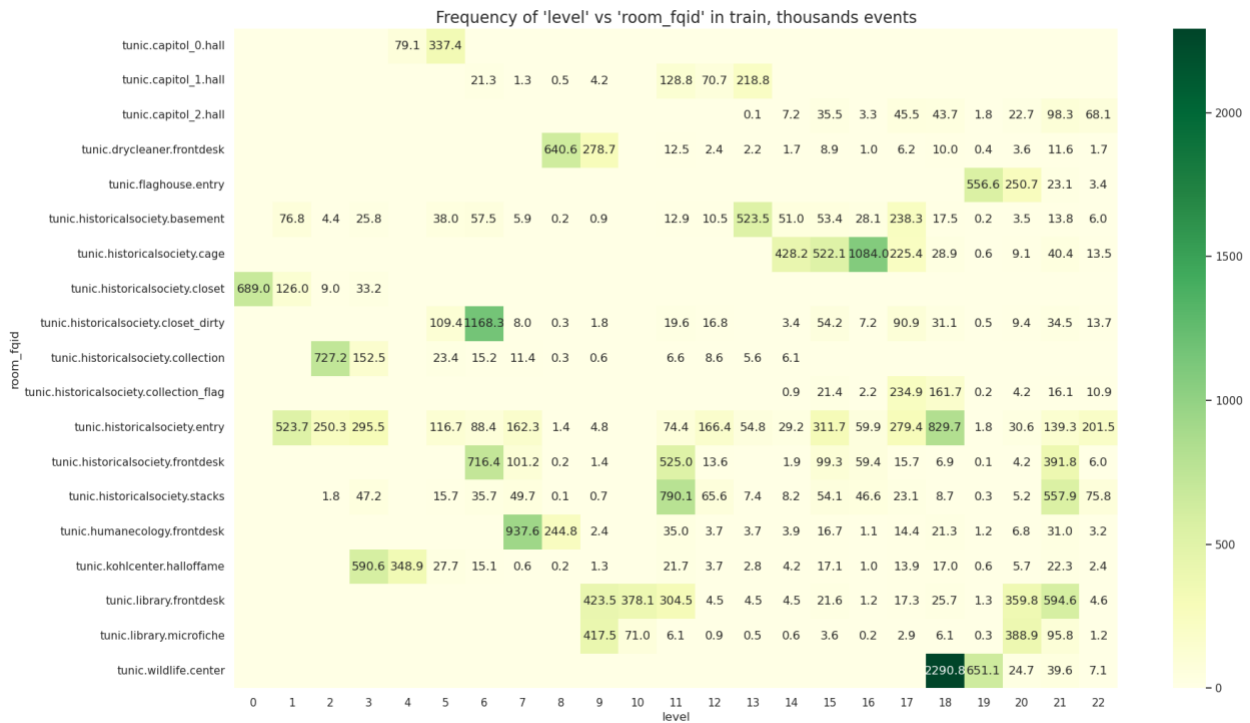
Level 21 is missing in 570 sessions

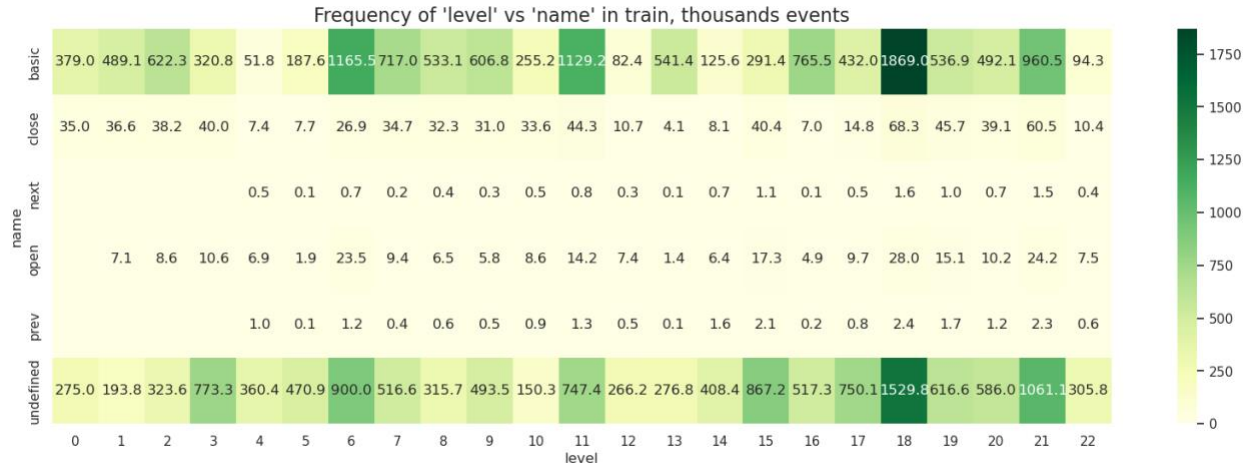
Examples of sessions with missing 21 level: ['20090316190523732', '20100007445515820', '20100014211946468', '20100018421509572', '20100020122406090']

Certain periods experience a higher number of events. For example, there is a significant spike in activity at level 18. While this pattern is seen across various individual cases, each case remains distinct.



Here are the intersections of level with some other variables:



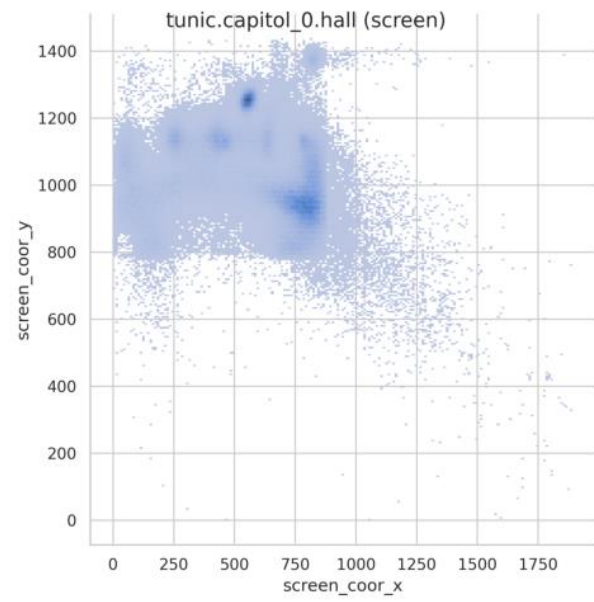
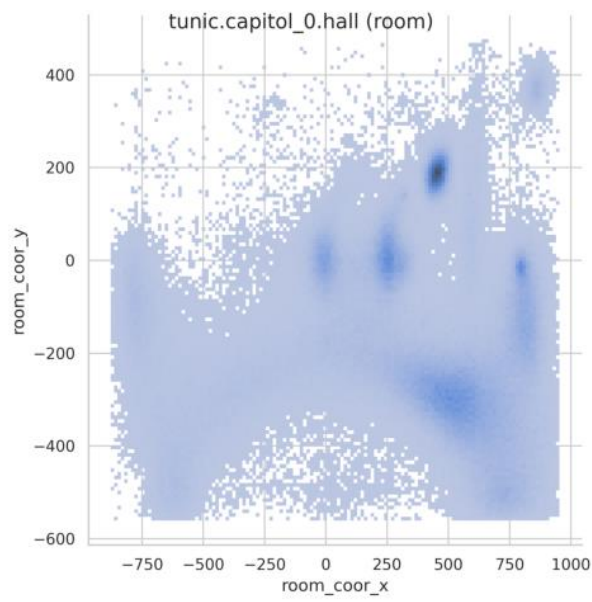
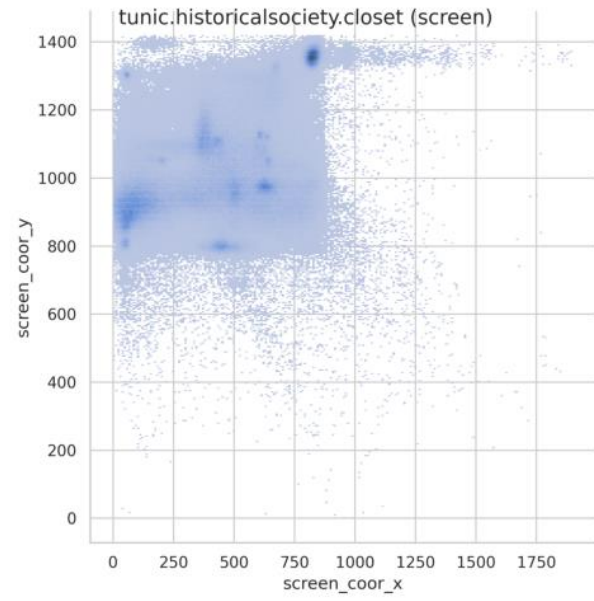
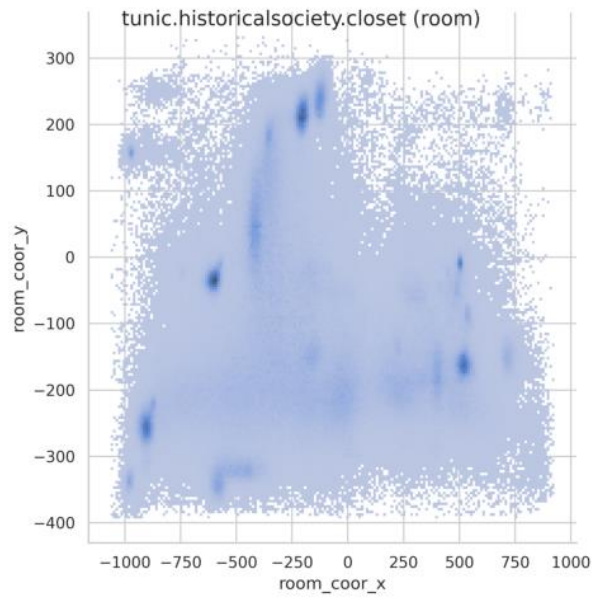


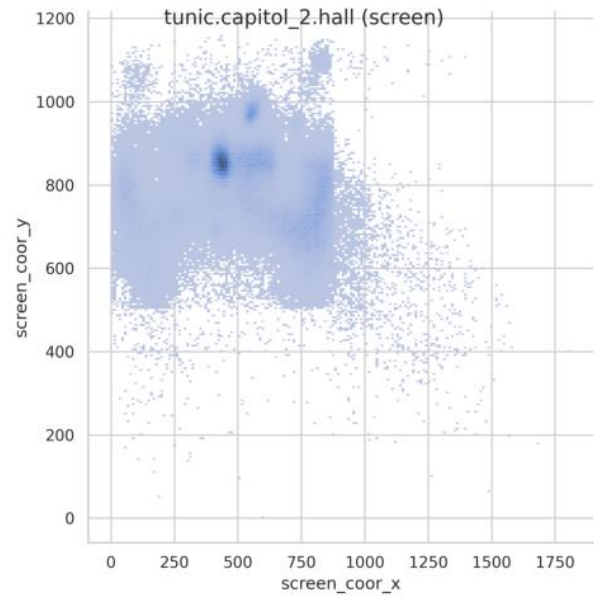
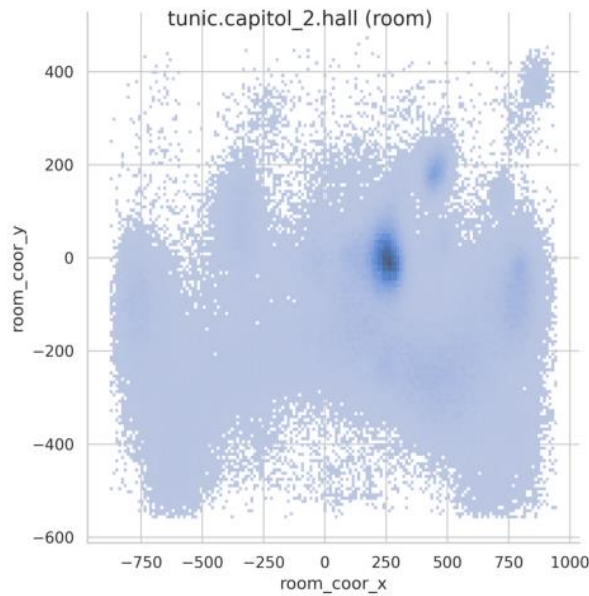
2.7. *click geo-location*

There are four geo-location variables in the data:

- `room_coor_x`: the coordinates of the click in reference to the in-game room (only for click events)
- `room_coor_y`: the coordinates of the click in reference to the in-game room (only for click events)
- `screen_coor_x`: the coordinates of the click in reference to the player's screen (only for click events)
- `screen_coor_y`: the coordinates of the click in reference to the player's screen (only for click events)

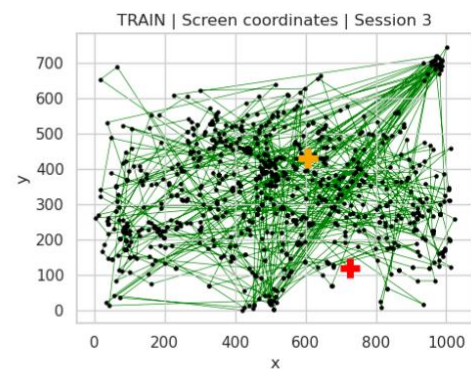
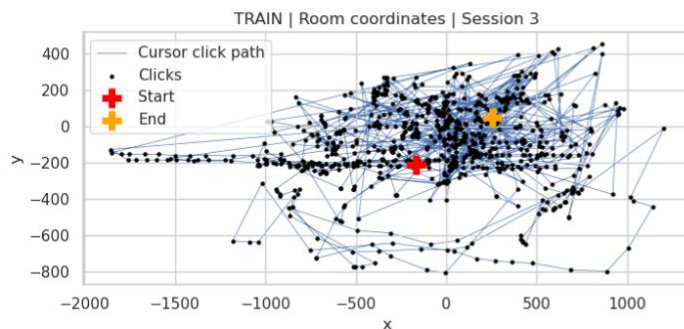
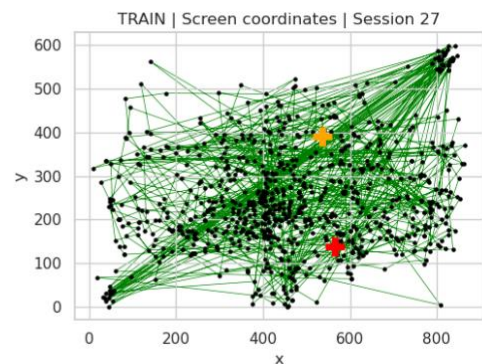
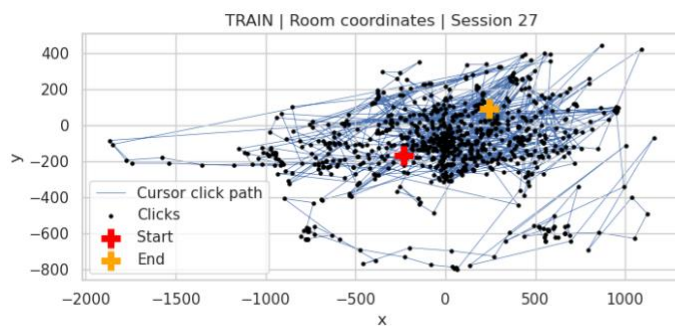
About 90-92% of all events are the click events. Let see the visualizations of the click density of some rooms:

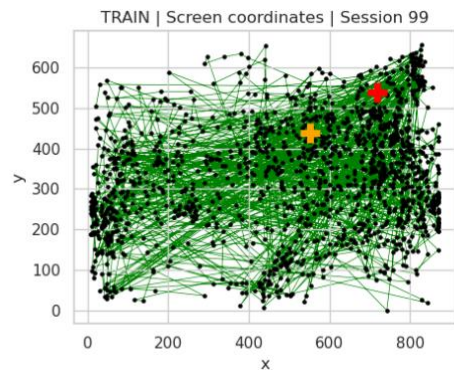
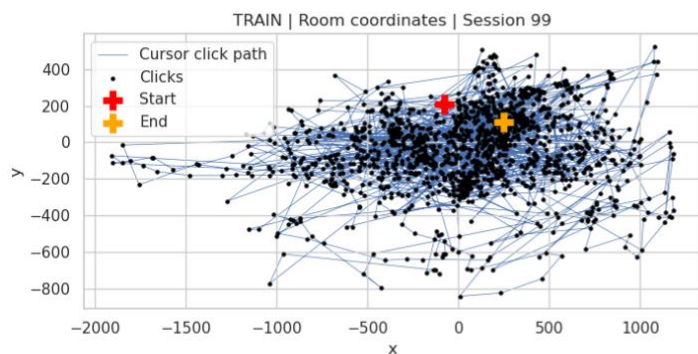




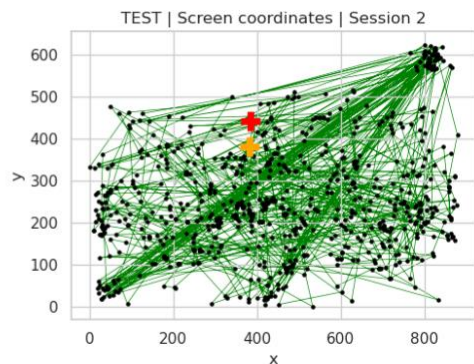
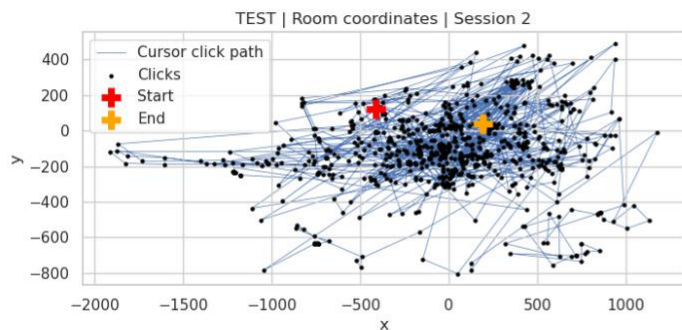
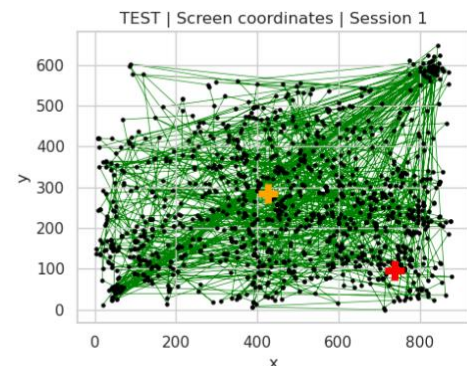
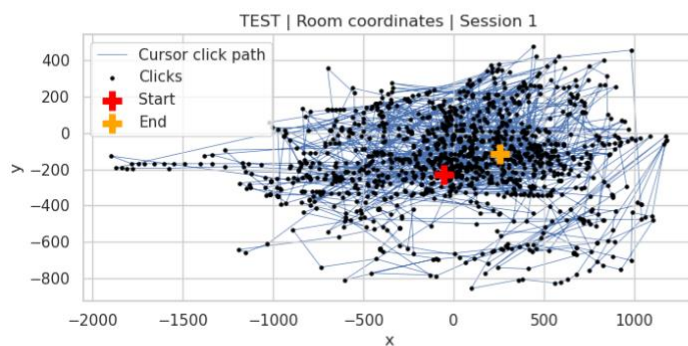
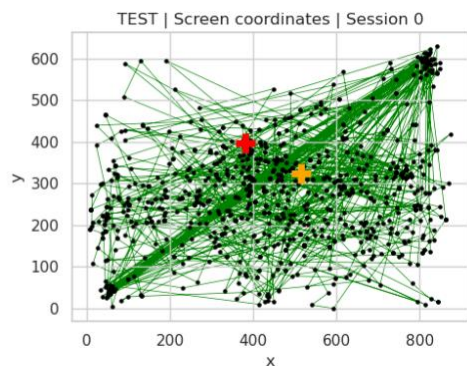
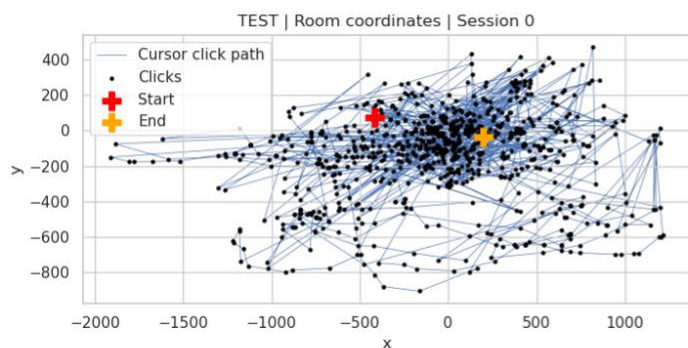
A high density of clicks indicates the locations of important objects or characters. We can observe that the patterns in the room and on the user's screen are quite similar. Now, let's review the geo-location path for a randomly selected session

In train set:





In test set:

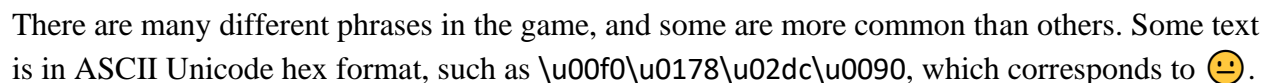


The clicking patterns across various sessions are quite similar, with noticeable clusters of clicks in specific regions. An analysis of the screen coordinates reveals at least 3 potential button clusters,

For additional details on click flow see insightful [discussion](#) and [notebook](#) by Chris Deotte, [this notebook](#) also has an interactive view on clicks: it allows you to select a session and watch the session click events at each stage of the game.

There are 2 text variables:

- This is the word cloud of text messages:



2.9. *page*

Page Number	Frequency (Thousands of events)
5.0	101.2
1.0	99.9
6.0	90.7
4.0	88.2
0.0	73.5
3.0	62.6
2.0	48.5

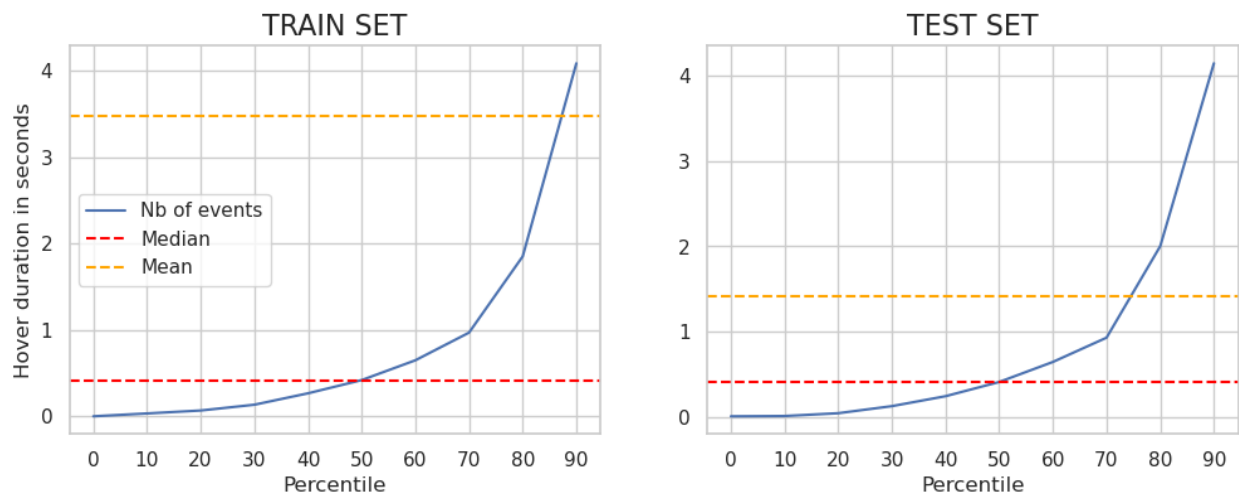


2.10. *hover*

`hover_duration`: how long (in milliseconds) the hover happened for (only for hover events). There are two types of hover events:

- *Object_Hover* - In each chapter of the game, some tasks have to be performed by the student, like clicking the slip on the t-shirt in the 1st Chapter of the game. This slip is an object. If the student takes the mouse pointer above this object, an `object_hover` is recorded. The duration for which the pointer stays above this object, is recorded in the `hover_duration`. If he/she clicks on the object, it's an `object_click`
- *Map_Hover* - In the map, there are many places a student can click to go there. When he/she takes the pointer above any place in the map, a `map_hover` event is recorded. Just like in `object_hover`, `hover_duration` is the duration for which the pointer stays above that place. And when the student clicks on any place, it's a `map_click`

This is the hover distribution chart:



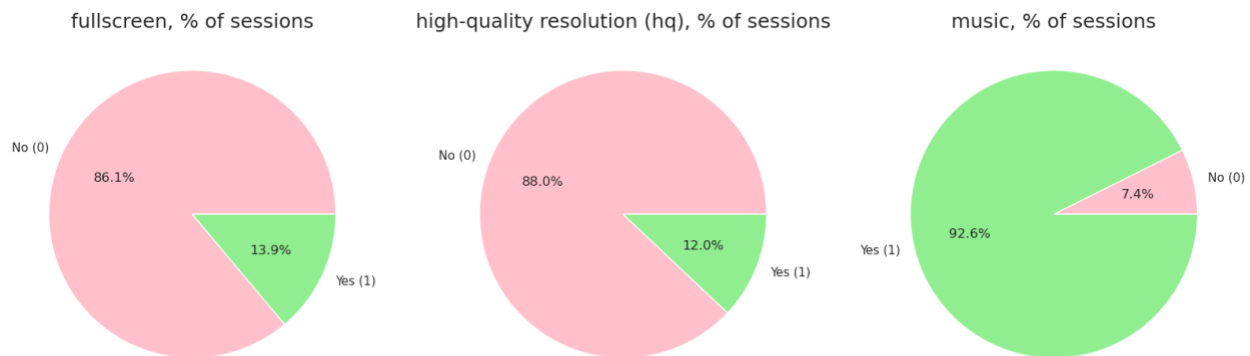
Similar to the page feature, the hover duration is recorded only for a few rows during a hover event. Typically, the duration ranges from a few milliseconds to a few seconds, with rare instances lasting more than 4 seconds. In the training set, the mean hover duration is significantly higher than the median, likely due to outliers where the user hovered for an extended period. These types of examples should be excluded.

2.11. *game properties: fullscreen, hq, music*

Finally, we also have game properties:

- fullscreen - whether the player is in fullscreen mode
- hq - whether the game is in high-quality
- music - whether the game music is on or off

We can assume that 0 in each of these columns corresponds to No, and 1 to Yes. The chart below show % of session with these properties:



III - DATA PREPROCESSING

1. Train and labels data

In data preprocessing, first, I simply load the data with defined scheme as mentioned in Data Discovery Process to handel memory issue. Then with 3 level groups, and every questions of each group will happen after a level checkpoint (4, 12, 22). The main idea is splitting data needed for training into 3 groups for each checkpoint:

- level_group == '0-4' \Rightarrow question 1->3
- level_group == '5-12' \Rightarrow question 4->13
- level_group == '13-22' \Rightarrow question 14->18

And because we're going to use all previous data in a session to predict the correction, therefore, I split the train data to 3 dataframe:

- dataset_df_1 within level 0->4
- dataset_df_2 within level 0->12
- dataset_df_3 within level 0->22

```
dataset_df_1: (3981005, 20)
dataset_df_2: (12825243, 20)
dataset_df_3: (26296946, 20)
```

With labels data, I simply split session_id into session and q features.

	session_id	correct	session	q
0	20090312431273200_q1	1	20090312431273200	1
1	20090312433251036_q1	0	20090312433251036	1
2	20090312455206810_q1	1	20090312455206810	1
3	20090313091715820_q1	0	20090313091715820	1
4	20090313571836404_q1	1	20090313571836404	1

2. Feature selection data

For feature engineering in the next part, I load an additional dataset feature_sort.csv.

	nabor	tip	quest	kol_col	col1	val1	col2	val2	col3	val3	...	kach4	kach5	ka
0	5		18	1	text_fqid	tunic.flaghouse.entry.flag_girl.symbol_recap	0	0	0	0	...	0.551005	0.548130	0.5
1	1		18	1	name	undefined	0	0	0	0	...	0.546731	0.550173	0.5
2	4		18	1	room_fqid	tunic.drycleaner.frontdesk	0	0	0	0	...	0.546318	0.548125	0.5
3	1		18	1	name	close	0	0	0	0	...	0.543156	0.547942	0.5
4	8		18	1	text	Make sure to get some old photos for the exhib...	0	0	0	0	...	0.551282	0.547190	0.5

This dataset, sourced from [Vadim Kamaev's notebook](#), is generated to support analysis and model training by capturing detailed relationships between features for this particular competition. For a deeper understanding of how this dataset was constructed, refer to [this link](#).

Here are some *key features* used in this model building:

- quest: questions (from 1 to 18).
- (col1, val1), (col2, val2), (col3, val3): pairs of column names and their corresponding unique values (col1 and col2 are categorical columns; col3 is a numerical column).
- kol_col: the number of columns involved in generating the feature (1 or 2).
- kach: the quality score for each feature, likely based on F1-score, calculated during training or validation.

With this additional dataset, I chose features with high kach score to generate some new features for each of 3 groups of dataset above.

For example, the below features will be used to training for dataset_df_1

	kol_col	col1	val1	col2	val2
0	1	name	basic	0	0
1	1	room_fqid	tunic.capitol_0.hall	0	0
2	1	room_fqid	tunic.historicalsociety.collection	0	0
3	1	room_fqid	tunic.historicalsociety.stacks	0	0
4	2	room_fqid	tunic.historicalsociety.entry	fqid	tostacks

IV - FEATURE ENGINEERING PROCESS

For feature engineering, I create new features to prepare for model building. The main idea is to generate a new dataframe where each row represents a session. For each session, I aggregate features that summarize the player's activities in the levels before answering the questions.

1. The *elapsed_time* is one of most impact factors so many features are created from it. First, due to the "reverse" phenomenon mentioned in the discovery section, I sort the data in each session based on *elapsed_time* from the raw dataset. Next, I create features related to the elapsed time between two consecutive events, called *d_time*. To manage outliers, I limit their range and store the results in a new feature called *delt_time* and its statistic features, including mean, std, min, max, 0.3, 0.5, 0.65, and 0.8 quantiles. The *delt_time* for subsequent events is also considered valuable, so I create a feature called *delt_time_next*. Finally, I also count the number of indices in each session.
2. For each categorical feature in the raw dataset, I count the number of unique values in each session. For numerical columns, I compute the average and the sum of the values. With *hover_duration*, I compute its mean and std.
3. Date and time components are also extracted from the *session_id* to create temporal features. The function extracts the year, month, day, and session time (hour and minute) from the *session_id* and stores them as separate features (*year*, *month*, *day*, and *sess_time*). This temporal information can help capture seasonality and trends related to user activity.
4. Finally, I generate conditional features based on the configuration provided in *feats_sel*. For single-condition features (where *kol_col* equals 1), I create a mask based on specific conditions and generate features such as the sum and mean of *delt_time_next* for sessions that match the condition. Similarly, for two-condition features (where *kol_col* equals 2), I create a combined mask based on two conditions and generate additional features based on the matching sessions. These conditional features allow for a more granular understanding of user behavior under specific conditions.

Then, I apply this function to 3 above datasets and drop those columns having high missing ratios or only having 1 value. After all these steps, I have the dataset already prepared for training:

```
100%|██████████| 128/128 [00:00<00:00, 1405.80it/s]
100%|██████████| 374/374 [00:00<00:00, 1657.57it/s]
100%|██████████| 551/551 [00:00<00:00, 1718.45it/s]
```

```
We'll train 'dataset_df_1' with 128 features
We'll train 'dataset_df_2' with 353 features
We'll train 'dataset_df_3' with 523 features
```

V - MODEL BUILDING

1. Main idea

The main idea is that I'll train GroupKFold models with XGBoost baseline for each question in the game. The models will use all previously data to predict the correctness of the session for the current question, including events that occurred in the corresponding levels.

This model is copied and modified from [Aravind Pillai's Kaggle notebook](#).

2. About XGBoost

2.1. XGBoost Introduction

XGBoost is a highly efficient and scalable machine learning algorithm based on gradient boosting. It was developed by Tianqi Chen and has become one of the most popular algorithms for structured/tabular data. XGBoost is known for its performance and speed, making it a go-to tool for data science competitions (like Kaggle).

2.2. How it work

XGBoost works by building an ensemble of decision trees sequentially, where each tree corrects the errors of the previous one. It uses gradient boosting, optimizing the model by minimizing a loss function through gradient descent. During training, XGBoost adjusts the weights of misclassified samples and adds regularization to avoid overfitting. The trees are built iteratively, with each new tree focusing on the residuals (errors) of the previous trees, ultimately improving the model's predictions.

2.3. Pros

- **High Performance:** XGBoost is known for its speed and accuracy, often winning Kaggle competitions.
- **Handles Missing Data:** It can handle missing values in the dataset naturally.
- **Parallelization:** Supports parallel and distributed computing, making it faster than other gradient boosting algorithms.

- **Regularization:** Built-in regularization (L1 and L2) helps prevent overfitting.
- **Flexibility:** Works for both regression and classification problems.
- **Scalability:** Can handle large datasets efficiently due to optimizations.

2.4. Cons

- **Complexity:** Tuning the model can be complex with many hyperparameters.
- **Interpretability:** Like most ensemble methods, XGBoost models are less interpretable compared to simpler models (e.g., linear regression).
- **Memory Consumption:** For very large datasets, it can consume a significant amount of memory.
- **Overfitting Risk:** While regularization helps, overfitting can still occur if hyperparameters are not properly tuned.

VI - EXPERIMENT

1. Training

First, I create the XGBoost model with params as below:

```
xgb_params = {
    'objective': 'binary:logistic', # Specifies binary classification (logistic regression)
    'eval_metric': 'logloss',      # Evaluation metric used during training (log loss for binary classification)
    'learning_rate': 0.05,         # Step size during training (smaller values require more boosting rounds)
    'max_depth': 4,                # Maximum depth of trees (helps control model complexity)
    'n_estimators': 6000,          # Number of boosting rounds (trees) to build
    'early_stopping_rounds': 50,   # Stop training early if validation score doesn't improve for 50 rounds
    'tree_method': 'hist',         # Tree-building method (histogram-based method for faster training)
    'subsample': 0.8,              # Fraction of samples to use for each tree (helps prevent overfitting)
    'colsample_bytree': 0.4,        # Fraction of features to use for each tree (helps with regularization)
    'use_label_encoder': False     # Disables label encoding for the target variable (to avoid warnings)
}
```

Then I train models for each of the 18 questions by categorizing them into 3 level groups (0-4, 5-12, 13-22). For each group, I select the corresponding dataset and features then perform 5-fold cross-validation using GroupKFold (for none-overlapping groups), where the data is split into training and validation sets based on *session_id*. For each fold, extract the labels (correct answers) for the relevant question and session and train an XGBoost classifier on the training set and validate the model using the validation set.

An image about the training progress:

```
### q_no 17 grp 13-22 feats : 523
Done for 13-22 17 0
Done for 13-22 17 1
Done for 13-22 17 2
Done for 13-22 17 3
Done for 13-22 17 4
### q_no 18 grp 13-22 feats : 523
Done for 13-22 18 0
Done for 13-22 18 1
Done for 13-22 18 2
Done for 13-22 18 3
Done for 13-22 18 4
```

After having the models, I calculate out-of-fold (OOF) probability predictions for all sessions and questions. This is the result:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
20090312431273200	0.881475	0.992872	0.967973	0.891234	0.667184	0.898649	0.895482	0.680301	0.851501	0.631158	0.779715	0.912305	0.321548	0.828352	0.6
20090312433251036	0.815894	0.989989	0.957041	0.487681	0.168589	0.415500	0.487956	0.543411	0.444420	0.228523	0.481028	0.762087	0.120662	0.270547	0.0
20090312455206810	0.804497	0.981032	0.969205	0.558306	0.709395	0.794033	0.755876	0.696865	0.792604	0.658639	0.706066	0.890314	0.460072	0.519252	0.3
20090313091715820	0.536271	0.971299	0.926895	0.827294	0.533711	0.731084	0.791153	0.555691	0.696316	0.492798	0.675896	0.893706	0.129585	0.731431	0.4
20090313571836404	0.942762	0.998094	0.990665	0.979001	0.852007	0.952956	0.937087	0.842838	0.927818	0.777648	0.799010	0.942125	0.509552	0.882008	0.7
...
22100215342220508	0.826889	0.996679	0.959871	0.968835	0.725543	0.853032	0.825838	0.643127	0.823795	0.526427	0.575858	0.922747	0.202798	0.791189	0.6
22100215460321130	0.462524	0.976916	0.853742	0.885819	0.588612	0.816780	0.723744	0.594588	0.820762	0.542596	0.711282	0.905633	0.171986	0.770272	0.5
22100217104993650	0.754615	0.985878	0.974652	0.872002	0.693690	0.890691	0.879477	0.668096	0.846325	0.604576	0.707722	0.911637	0.218484	0.911347	0.7
22100219442786200	0.650519	0.988792	0.932136	0.816956	0.477832	0.835379	0.715597	0.648719	0.770222	0.579957	0.685784	0.904950	0.321814	0.789533	0.5
22100221145014656	0.464392	0.956740	0.911421	0.750425	0.394571	0.664749	0.644435	0.477266	0.658065	0.310452	0.523531	0.840168	0.101881	0.567071	0.2

23562 rows x 18 columns

One more step before the final, we need to find the threshold because the results above are still probabilities. By iterating through potential thresholds (0.4 to 0.8 in steps of 0.01), I calculate the F1 score for each threshold and track the best score and its corresponding threshold. This process identifies the threshold that maximizes the F1 score, balancing precision and recall for the model's predictions.

This is the result:

Best threshold 0.6200000000000002 F1 score 0.696613666809234

2. Testing


Before testing on test data, I once evaluate the model using the best threshold and calculate the f1-score of each model and the overall score:

```
When using optimal threshold...
Q0: F1 = 0.6678643298891116
Q1: F1 = 0.5261196890476701
Q2: F1 = 0.5079421982478146
Q3: F1 = 0.6773461025874492
Q4: F1 = 0.6369255385736874
Q5: F1 = 0.6432572373431327
Q6: F1 = 0.6300663952001252
Q7: F1 = 0.5714563228701321
Q8: F1 = 0.6283704217674289
Q9: F1 = 0.5856005244390425
Q10: F1 = 0.6102647885269563
Q11: F1 = 0.5186331466589013
Q12: F1 = 0.45966236142531197
Q13: F1 = 0.6354450526291678
Q14: F1 = 0.6053883336992303
Q15: F1 = 0.49842194969282805
Q16: F1 = 0.5541053108013009
Q17: F1 = 0.4956494787254747
==> Overall F1 = 0.696613666809234
```

Finally, apply the model to predict on competition's test data.

```
session_id,correct
20090109393214576_q1,1
20090109393214576_q2,1
20090109393214576_q3,1
20090109393214576_q4,1
20090109393214576_q5,0
20090109393214576_q6,1
20090109393214576_q7,1
20090109393214576_q8,1
20090109393214576_q9,1
```

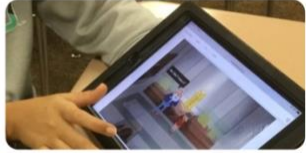
3. Results achieved, comments and analysis

 THE LEARNING AGENCY LAB · FEATURED CODE COMPETITION · A YEAR AGO

Late Submission ...

Predict Student Performance from Game Play

Trace student learning from Jo Wilder online educational game



OverviewDataCodeModelsDiscussionLeaderboardRulesTeamSubmissions


Submissions

You selected 0 of 3 submissions to be evaluated for your final leaderboard score. Since you selected less than 3 submissions, Kaggle auto-selected up to 3 submissions from among your public best-scoring unselected submissions for evaluation. The evaluated submission with the best Private Score is used for your final score.

0/3

Submissions evaluated for final score

AllSuccessfulSelectedErrorsRecent

Submission and Description	Private Score ⓘ	Public Score ⓘ	Selected
 HCMUS-2425-21127038 Model Building - Version 15 Succeeded (after deadline) · 20h ago · Notebook HCMUS-2425-21127038 Model Building Version 15	0.700	0.698	<input type="checkbox"/>

Finally, I submitted to the competition and got the result: **0.700 Private Score** and **0.698 Public Score**. They are higher than the overall F1 on train set, which may infer that the models might are pretty well-regularized and prevent overfitting. I don't have the test data of the competition so further analysis can be perform.

4. Conclusion and development direction

In conclusion, the result is pretty good. In the progresss of discover the data and building the model, I learned many things. For development, I did not try to combine models, there're many Kagglers on the Leaderboard use more than one model for training. There're maybe more features can improve the model too. Additionally, I don't really understand how they build the feature selection dataset but it helps a lot.

5. References

Data Discovery:

<https://www.kaggle.com/code/paulbacher/detailed-eda-student-perf-from-game-play>

<https://www.kaggle.com/code/demche/student-performance-from-game-play-eda>

<https://www.kaggle.com/competitions/predict-student-performance-from-game-play/discussion>

Model Building:

<https://www.kaggle.com/code/aspillai/predict-student-performance-xgb-fold-sel-feats>