# Predict Student’s Performance from Game Play

By/ Ahmed Rashed Salem GitHub: ahmedsalem88 Kaggle: ahmedrashedsalem

***Abstract* –** Predicting Student’s Performance from Game Play is a Competition on Kaggle, the goal of this competition is to predict student performance during game-based learning in real-time. I developed a model trained on one of the largest open datasets of game logs to do that. The evaluation metric of this competition is the f1_score metric. The prediction is one of zero of column (correct), so it is a binary classification task. Through ten submissions to the competition, I achieved f1_score of about 0.672 in public score using many Machine Learning Models such as Gradient Boosting Trees (GBT), Random Forest Model (RFM), Classification and Regression Trees (CART), and also using feature engineering techniques.

***Keywords – Machine Learning, Feature Engineering, Classification, TensorFlow, TimeSeries***

## 1 Introduction

Learning is meant to be fun, which is where game-based learning comes in. This educational approach allows students to engage with educational content inside a game framework, making it enjoyable and dynamic. Although game-based learning is being used in a growing number of educational settings, there are still a limited number of open datasets available to apply data science and learning analytic principles to improve game-based learning.

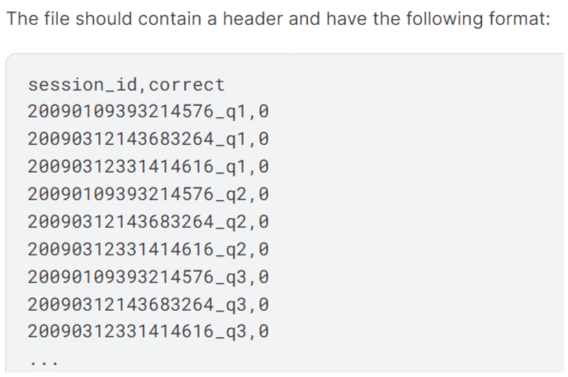
Most game-based learning platforms do not sufficiently make use of knowledge tracing to support individual students. Knowledge tracing methods have been developed and studied in the context of online learning environments and intelligent tutoring systems. But there has been less focus on knowledge tracing in educational games.

Competition host Field Day Lab is a publicly-funded research lab at the Wisconsin Center for Educational Research. They design games for many subjects and age groups that bring contemporary research to the public, making use of the game data to understand how people learn. Field Day Lab's commitment to accessibility ensures all of its games are free and available to anyone. The lab also partners with nonprofits like The Learning Agency Lab, which is focused on developing science of learning-based tools and programs for the social good.

If successful, We 'll enable game developers to improve educational games and further support the educators who use these games with dashboards and analytic tools. In turn, we might see broader support for game-based learning platforms.

For each (session_id / question number) pair in the test set, I must predict a binary label for the `correct` (1 or 0) variable, it is a binary Classification, and submissions will be evaluated based on their F1_score.

Note that the `sample_submission.csv` provided for your usage also includes a grouping variable, `session_level`, that groups the questions by session and level. This is handled automatically by the time-series API, so when making predictions you will not have access to this column. The time-series API presents the questions and data to you in order of levels - level segments 0-4, 5-12, and 13-22 are each provided in sequence, and We will be predicting the correctness of each segment's questions as they are presented below.


The file should contain a header and have the following format:

```
session_id,correct
20090109393214576_q1,0
20090312143683264_q1,0
20090312331414616_q1,0
20090109393214576_q2,0
20090312143683264_q2,0
20090312331414616_q2,0
20090109393214576_q3,0
20090312143683264_q3,0
20090312331414616_q3,0
...
```

Figure 1.1 – sample of submission

The Competition is also hosting a second track that focuses on model efficiency because highly accurate models are often computationally heavy. Such models have a stronger carbon footprint and frequently prove difficult to utilize in real-world educational contexts. You can check more about this track here

## 2 LITERATURE OVERVIEW

ML, due to the concepts it inherits, can be regarded a subfield of Artificial Intelligence (AI). It enables prediction; for this purpose, its basic building blocks are algorithms. ML enables systems to learn on their own rather than being explicitly programmed to do so, resulting in more intelligent behavior. It generates data-driven predictions by developing models that discover patterns in historical data and utilizes those patterns to generate predictions [1] [2]. The general architecture of ML consists of several steps: business understanding (understanding and knowledge of the domain), data acquisition and understanding (gathering and understanding data), modeling (which entails feature engineering, model training, and evaluation), and deployment (deploy the model on the cloud).

Supervised Learning, Unsupervised Learning, and Reinforcement Learning are the three major categories of ML. In Supervised Learning, the model is trained on labeled data and is then used to generate predictions on unlabeled data [3]. In unsupervised learning, a model is trained on unlabeled data and said model automatically learns from that data by extracting features and patterns from it. In Reinforcement Learning, an agent is trained on the environment and this enables said agent to find the optimum solution and accomplish a goal in a complex situation [4].

ML algorithms treat each instance of a dataset as a collection of features. These features may be binary, categorical, or continuous in nature. If the instances are labeled, then this type of learning is termed supervised learning [2]. Supervised Learning involves training the model on labeled data and testing it on unlabeled data. Its fundamental architecture begins with dataset collection; the dataset is then partitioned into testing and training data; and then, the data is preprocessed. Extracted features are fed into an algorithm and the model is then trained to learn the features associated with each label. Finally, the model is supplied with the test data, and said model makes predictions on the test data by providing the expected labels.

Classification and regression are the two broad types of supervised learning. Both are discussed in detail in the following sections.

In classification, a model predicts unknown values (set of Outputs) based on a set of known values (set of inputs) [2]. When the output is in categorical form, the problem is referred to as a classification one [5]. Generally, in classification, a dataset’s instances are categorized according to specified classes [5] [6]. Classification can be applied to both structured and unstructured datasets. Some terms used in Classification are: Classification model, Classification algorithm, and feature. A classification algorithm, alternatively referred to as a classifier, learns from the training dataset and assigns each new data point to a certain class. In comparison, a classification model uses a mapping function, which is concluded by said model from the training dataset, to predict the class label for the test data. Finally, a feature is associated with the dataset, which helps in building a precise predictive model. The classification process is depicted in Figure 5. Data collection and preprocessing are the first steps in building a classification model. Preprocessing is the process of cleansing data by eliminating noise and duplicates. Numerous techniques are used to preprocess data, among which "brute-force" is the simplest and most common one. The data is then split into train and test sets using the cross-validation technique. The next stage is to train the model using class labels; in Python, the sci-kit-learn package has a function called "fit-transform (X, Y)" that maps X (input data) to Y (labels) for the purpose of preparing the classifier. The next step is to forecast the new dataset's class or label. Finally, the classification algorithm is evaluated using the test data. There are two distinct classification types: binary and multi-label [5]. Binary Classification is used when the outcome is binary or has two classes. For instance, in an ambiguity detection process, the model predicts whether or not sentences are ambiguous and as a result, there are only two possible outcomes/classes; this is referred to as binary classification. However, multi-label classification is made up of a distribution of classes. For example, in predicting mental disorders, there are multiple ones such as depression, anxiety, schizophrenia, bipolar disorder, and Post-traumatic stress disorder (PTSD). Thus, the outcome can fall into one of these five categories; this is termed multi-label classification.

Regression is a supervised learning technique that permits the discovery of correlations between variables and the prediction of continuous values based on these variables. When the output is continuous, the problem is referred to as a regression one [5]—for instance, predicting a person's weight, age, or salary, weather forecasting, or housing price forecasting. In regression, X (input variables) is mapped to Y (continuous output). Classification is the process of predicting the discrete labels of the input. Regression, on the other hand, is concerned with the prediction of continuous values. Regression is divided into two main categories: Simple Linear and Multiple. In simple linear regression, a straight line is drawn to define the relationship between two variables

(X and Y). In contrast, Multiple regression encompasses multiple variables and is further divided into linear and non-linear.

## 3 PROPOSED METHODOLOGY

This section describes in detail the methodology used to predict student performance. I tried ten submissions to the competition using different models, different feature engineering and, different splitting of data.

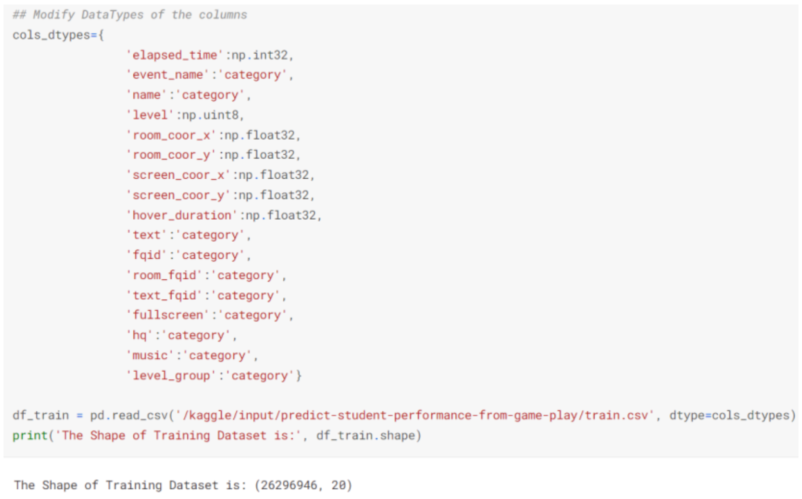
### *Dataset Description*

This competition uses Kaggle's time series API. Test data will be delivered in groupings that do not allow access to future data. The objective of this competition is to use time series data generated by an online educational game to determine whether players will answer questions correctly. There are three question checkpoints (level 4, level 12, and level 22), each with a number of questions.

There are 18 questions for each session – I'm not given the answers but am simply told whether the user for a particular session answered each question correctly. When I'm ready to predict, use the sample notebook to iterate over the test data, which is split as described above and served up as pandas data frames. I made my predictions for each group of questions - at the end of this process, a submission.csv file had been created for me. Simply submit my notebook.

## A Submission 1 – GBT

The Dataset is very big, so it is important to take care of each column data type. Instead of using float 64, we can use float 32. It will not affect too much but it will save more memory. Also, In pandas category is much more efficient than string. After doing that I can read the dataset.

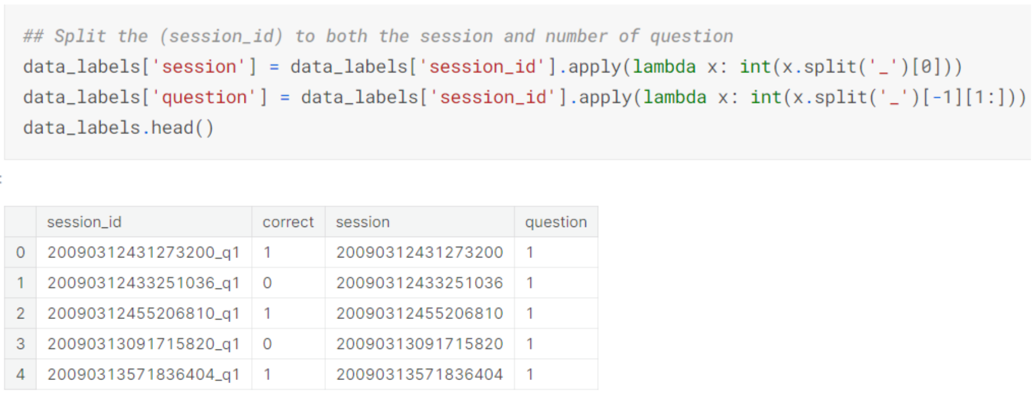


```
## Modify DataTypes of the columns
cols_dtypes={
                'elapsed_time':np.int32,
                'event_name':'category',
                'name':'category',
                'level':np.uint8,
                'room_coor_x':np.float32,
                'room_coor_y':np.float32,
                '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':'category',
                'hq':'category',
                'music':'category',
                'level_group':'category'}

df_train = pd.read_csv('/kaggle/input/predict-student-performance-from-game-play/train.csv', dtype=cols_dtypes)
print('The Shape of Training Dataset is:', df_train.shape)
```

```
The Shape of Training Dataset is: (26296946, 20)
```

Figure 2.A.1 – reading the dataset

The labels are stored in a separated file, It contains the session_id and correct columns. I will split the session_id column to (session and question number)



```
## Split the (session_id) to both the session and number of question
data_labels['session'] = data_labels['session_id'].apply(lambda x: int(x.split('_')[0]))
data_labels['question'] = data_labels['session_id'].apply(lambda x: int(x.split('_')[-1][1:]))
data_labels.head()
```

| | session_id | correct | session | question |
|---|---|---|---|---|
| 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 |

Figure 2.A.2 – Labels

I will make some visualization to check the distribution of the label (correct) column to check the balancing of classes. It shows that the dataset is imbalanced.

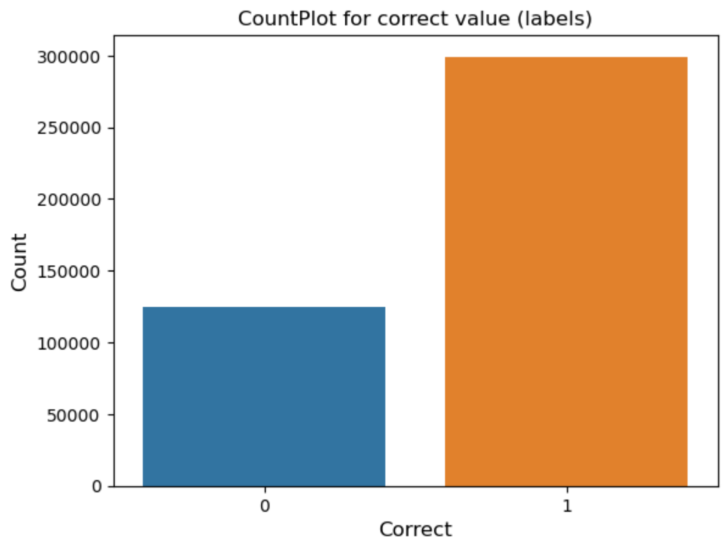


Figure 2.A.3 – Distribution of correct column

Also, I will make a visualization to show the distribution of the label (correct) column but for each question separately. (The below figure shows only for first 6 questions, you can check the notebook for the full code and full output)

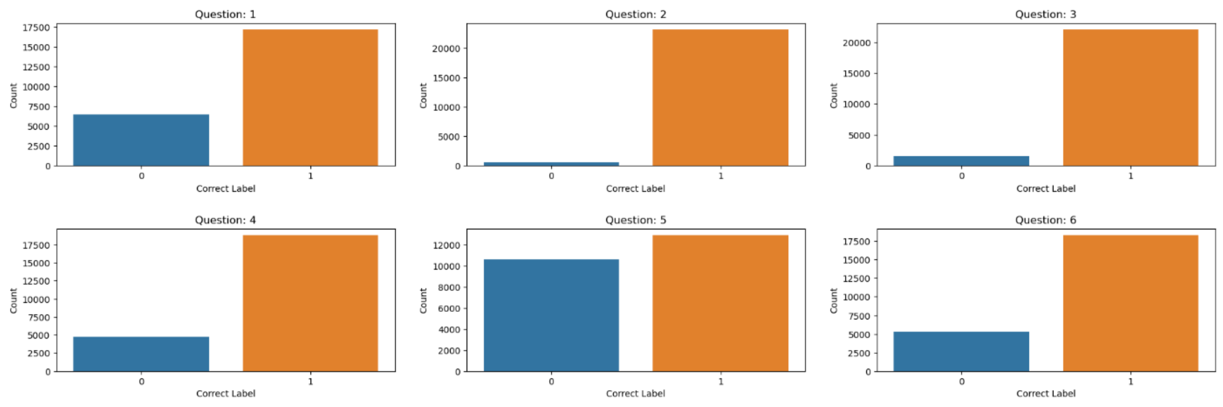
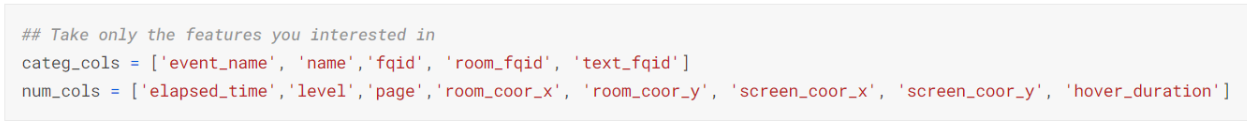


Figure 2.A.4 – Distribution of correct column for each question

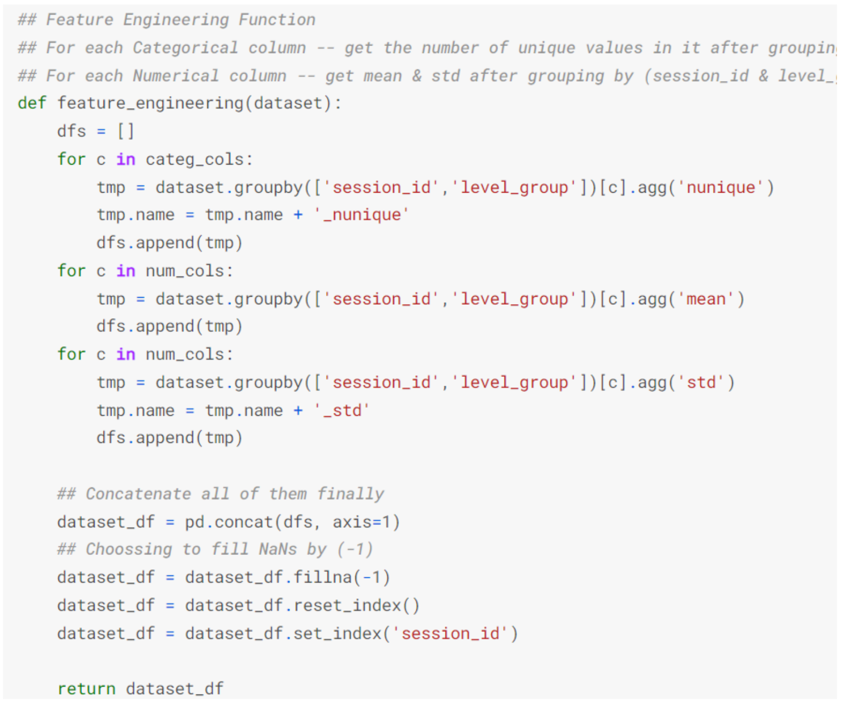
Then Feature Engineering, In this submission I take only some features not all of them.



```
## Take only the features you interested in
categ_cols = ['event_name', 'name','fqid', 'room_fqid', 'text_fqid']
num_cols = ['elapsed_time','level','page','room_coor_x', 'room_coor_y', 'screen_coor_x', 'screen_coor_y', 'hover_duration']
```

Figure 2.A.5 – Relevant Features

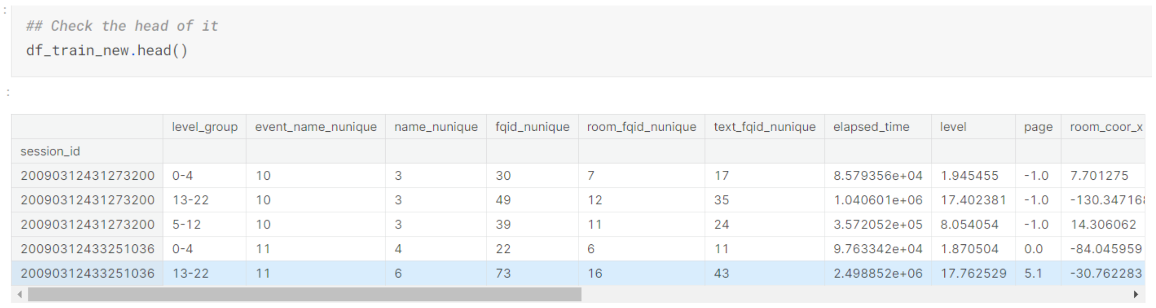
After that, for each categorical column in the list. I will count the number of unique values in that column after grouping by (session_id, level_group) columns, Renaming the output column with its old name plus _nunique. Ex: (event_name_nunique).



```
## Feature Engineering Function
## For each Categorical column -- get the number of unique values in it after groupin
## For each Numerical column -- get mean & std after grouping by (session_id & level_
def feature_engineering(dataset):
    dfs = []
    for c in categ_cols:
        tmp = dataset.groupby(['session_id','level_group'])[c].agg('nunique')
        tmp.name = tmp.name + '_nunique'
        dfs.append(tmp)
    for c in num_cols:
        tmp = dataset.groupby(['session_id','level_group'])[c].agg('mean')
        dfs.append(tmp)
    for c in num_cols:
        tmp = dataset.groupby(['session_id','level_group'])[c].agg('std')
        tmp.name = tmp.name + '_std'
        dfs.append(tmp)

    ## Concatenate all of them finally
    dataset_df = pd.concat(dfs, axis=1)
    ## Choossing to fill NaNs by (-1)
    dataset_df = dataset_df.fillna(-1)
    dataset_df = dataset_df.reset_index()
    dataset_df = dataset_df.set_index('session_id')

    return dataset_df
```

The same thing is done for numerical columns, but instead of taking the number of unique values, I will take the mean and standard deviation for each column

Figure 2.A.6 – Feature Engineering Function



```
## Check the head of it
df_train_new.head()
```

| | level_group | event_name_nunique | name_nunique | fqid_nunique | room_fqid_nunique | text_fqid_nunique | elapsed_time | level | page | room_coor_x |
|---|---|---|---|---|---|---|---|---|---|---|
| session_id | | | | | | | | | | |
| 20090312431273200 | 0-4 | 10 | 3 | 30 | 7 | 17 | 8.579356e+04 | 1.945455 | -1.0 | 7.701275 |
| 20090312431273200 | 13-22 | 10 | 3 | 49 | 12 | 35 | 1.040601e+06 | 17.402381 | -1.0 | -130.34716 |
| 20090312431273200 | 5-12 | 10 | 3 | 39 | 11 | 24 | 3.572052e+05 | 8.054054 | -1.0 | 14.306062 |
| 20090312433251036 | 0-4 | 11 | 4 | 22 | 6 | 11 | 9.763342e+04 | 1.870504 | 0.0 | -84.045959 |
| 20090312433251036 | 13-22 | 11 | 6 | 73 | 16 | 43 | 2.498852e+06 | 17.762529 | 5.1 | -30.762283 |

Figure 2.A.7 – Final output of Feature Engineering Function

I will make visualization for some numerical output, It will show like Time Series Data.

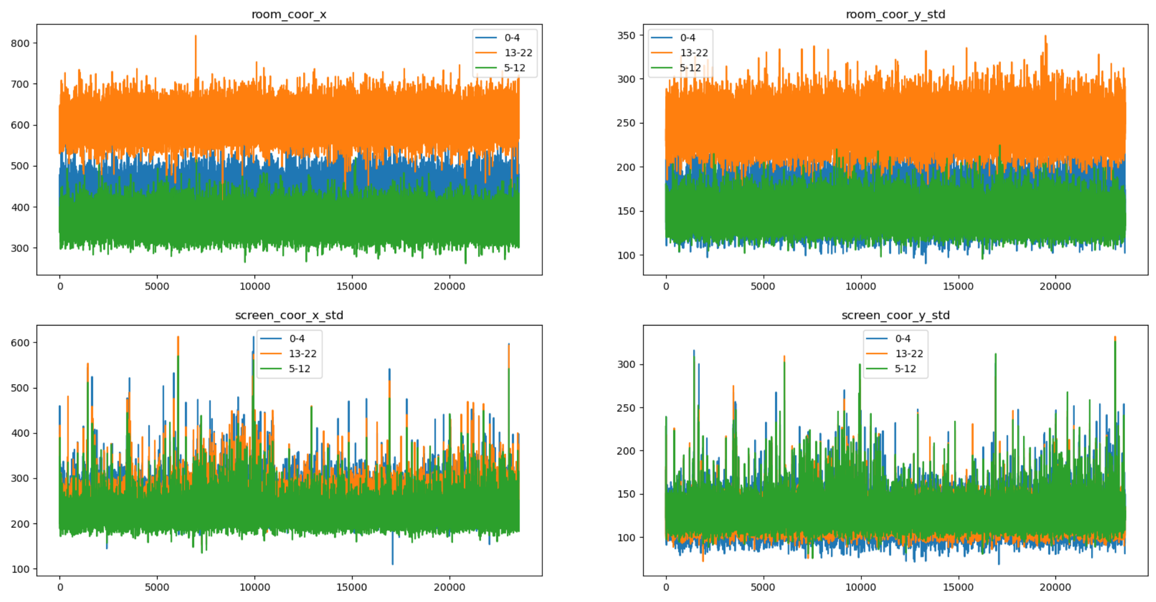
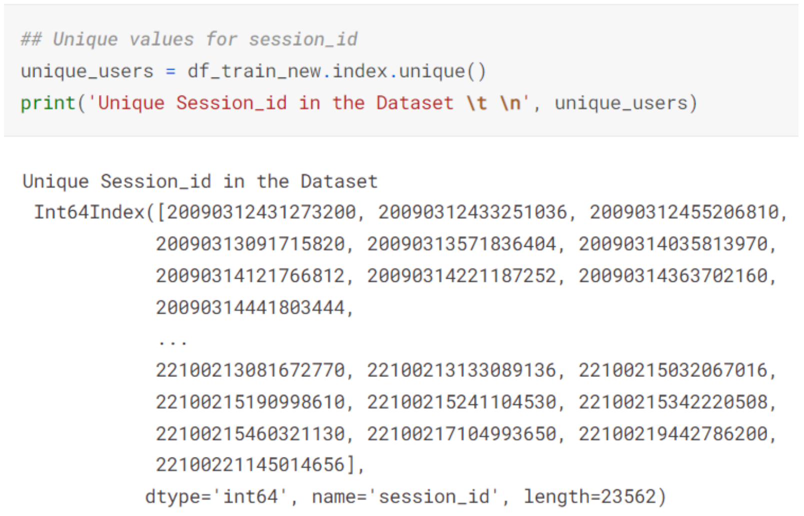


Figure 2.A.8 – Distribution of some Numerical Feature after Processing

Then, I will split the dataset; but take care and don't use the traditional method. The (session_id) column or the index in the processed data frame has duplicates. So we must get unique values and split them from unique values. It doesn't mean that I will return a unique data frame, of course not Let's say We will split 80% for the training dataset and 20% for the validation dataset. Each Training will have duplicated (session_id).

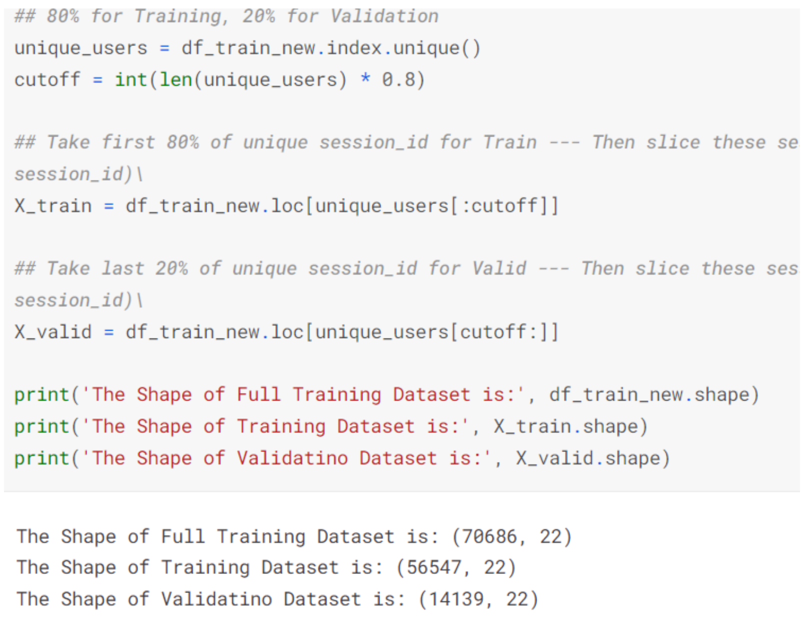


```
## Unique values for session_id
unique_users = df_train_new.index.unique()
print('Unique Session_id in the Dataset \t \n', unique_users)
```

```
Unique Session_id in the Dataset
 Int64Index([20090312431273200, 20090312433251036, 20090312455206810,
            20090313091715820, 20090313571836404, 20090314035813970,
            20090314121766812, 20090314221187252, 20090314363702160,
            20090314441803444,
            ...
            22100213081672770, 22100213133089136, 22100215032067016,
            22100215190998610, 22100215241104530, 22100215342220508,
            22100215460321130, 22100217104993650, 22100219442786200,
            22100221145014656],
           dtype='int64', name='session_id', length=23562)
```

Figure 2.A.9 – Unique Session Ids in dataset

In the above figure, it shows that unique values about 23562 value and the processed dataset contains about 70686 values, of course we will not remove duplicated in session_id. We will take the first 80% of unique values for the training dataset and the other 20% for validation. I hope this will be clear.

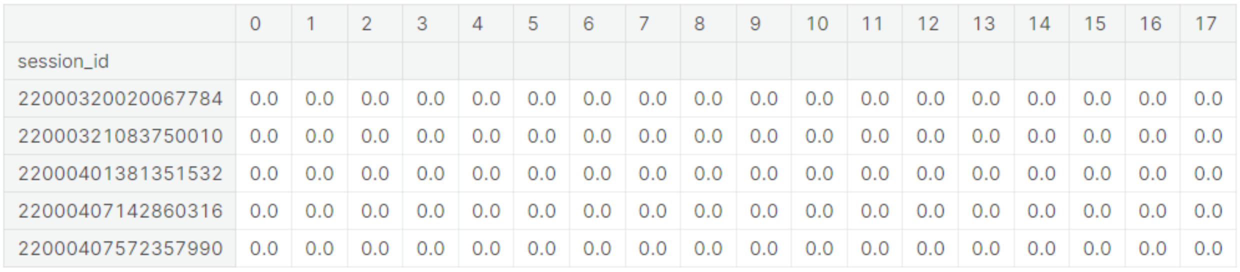


```
## 80% for Training, 20% for Validation
unique_users = df_train_new.index.unique()
cutoff = int(len(unique_users) * 0.8)

## Take first 80% of unique session_id for Train --- Then slice these se
session_id)\
X_train = df_train_new.loc[unique_users[:cutoff]]

## Take last 20% of unique session_id for Valid --- Then slice these ses
session_id)\
X_valid = df_train_new.loc[unique_users[cutoff:]]

print('The Shape of Full Training Dataset is:', df_train_new.shape)
print('The Shape of Training Dataset is:', X_train.shape)
print('The Shape of Validatino Dataset is:', X_valid.shape)
```

```
The Shape of Full Training Dataset is: (70686, 22)
The Shape of Training Dataset is: (56547, 22)
The Shape of Validatino Dataset is: (14139, 22)
```

Figure 2.A.10 – Final Splitting the processed Dataset

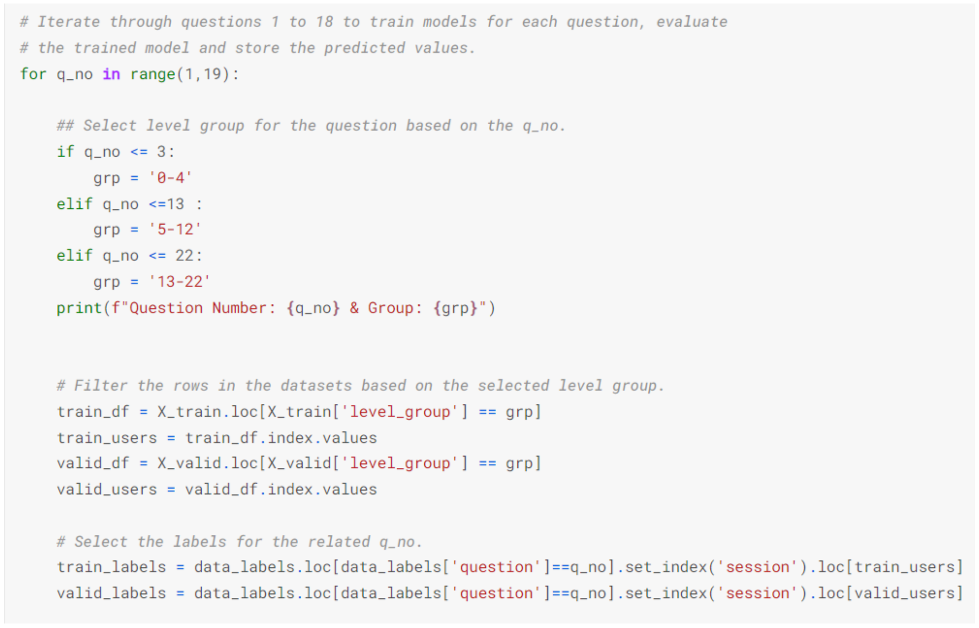
Moving forward to training the models, I will start by GBT (Gradient boosting Trees) but before training, I will prepare a data frame for the prediction of validation dataset indexed by unique values of session_id in the validation dataset and has 18 columns: one column for each question (we said that we have 18 questions).



| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| session_id | | | | | | | | | | | | | | | | | | |
| 22000320020067784 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 22000321083750010 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 22000401381351532 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 22000407142860316 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 22000407572357990 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

Figure 2.A.11 – Empty Data Frame for Prediction of Validation Dataset

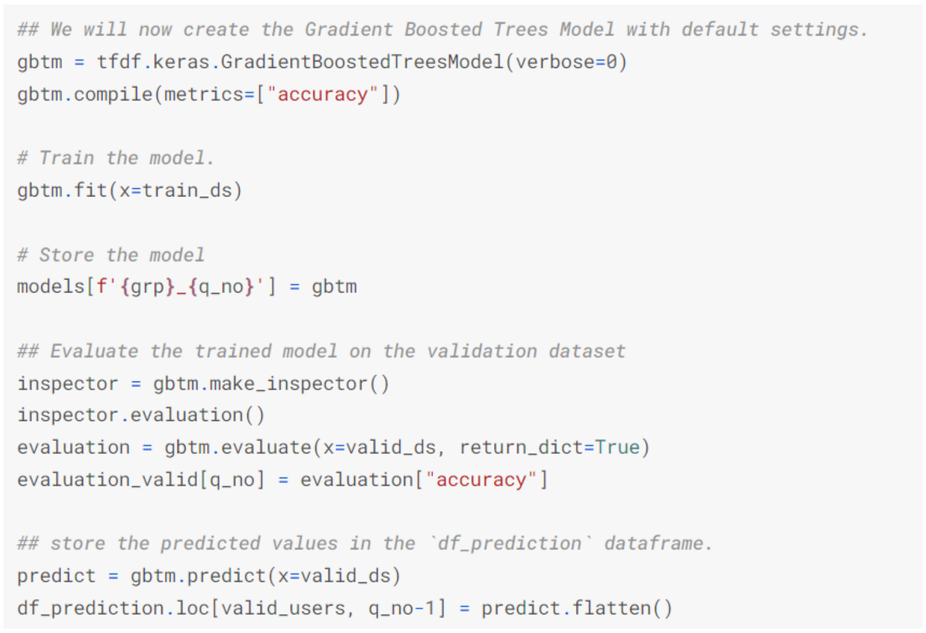
Looping for 18 times, each time for each question, but how to specify the level_group. That is an important note. Questions 1 to 3 for (level_group 0-4), Questions 4 to 13 for (level_group 5-12), and Questions 14 to 18 for (level_group 13-22). Slicing from the data frame according to the number of questions which specify the level_group.



```python
# Iterate through questions 1 to 18 to train models for each question, evaluate
# the trained model and store the predicted values.
for q_no in range(1,19):

    ## Select level group for the question based on the q_no.
    if q_no <= 3:
        grp = '0-4'
    elif q_no <=13 :
        grp = '5-12'
    elif q_no <= 22:
        grp = '13-22'
    print(f"Question Number: {q_no} & Group: {grp}")


    # Filter the rows in the datasets based on the selected level group.
    train_df = X_train.loc[X_train['level_group'] == grp]
    train_users = train_df.index.values
    valid_df = X_valid.loc[X_valid['level_group'] == grp]
    valid_users = valid_df.index.values

    # Select the labels for the related q_no.
    train_labels = data_labels.loc[data_labels['question']==q_no].set_index('session').loc[train_users]
    valid_labels = data_labels.loc[data_labels['question']==q_no].set_index('session').loc[valid_users]
```

Figure 2.A.12 – First part of looping / Getting each question data and label

The second part of the loop is training the model, Here I'm training GBT (Gradient Boosting Trees) with number of trees equals 100; each tree is working on a subsample and sub features and finally, we take the majority voting of all trees. It is a type of Ensemble Learning.



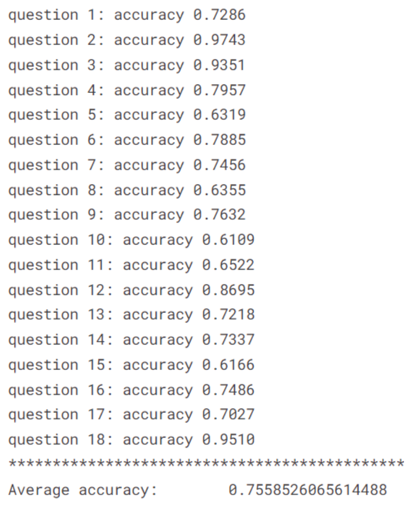
```python
## We will now create the Gradient Boosted Trees Model with default settings.
gbtm = tfdf.keras.GradientBoostedTreesModel(verbose=0)
gbtm.compile(metrics=["accuracy"])

# Train the model.
gbtm.fit(x=train_ds)

# Store the model
models[f'{grp}_{q_no}'] = gbtm

## Evaluate the trained model on the validation dataset
inspector = gbtm.make_inspector()
inspector.evaluation()
evaluation = gbtm.evaluate(x=valid_ds, return_dict=True)
evaluation_valid[q_no] = evaluation["accuracy"]

## store the predicted values in the `df_prediction` dataframe.
predict = gbtm.predict(x=valid_ds)
df_prediction.loc[valid_users, q_no-1] = predict.flatten()
```

Figure 2.A.13 – Second part of looping/ Training & Evaluating for each Question

Because the task is binary classification, we can move the threshold to deal with the imbalanced dataset, as I shown above that the dataset is imbalanced. But Here in the first submission, I use the default threshold equals 0.5.

## ***Results of Submission 1***

The Average Accuracy of Submission 1 for all Questions is about 0.75.



```
question 1: accuracy 0.7286
question 2: accuracy 0.9743
question 3: accuracy 0.9351
question 4: accuracy 0.7957
question 5: accuracy 0.6319
question 6: accuracy 0.7885
question 7: accuracy 0.7456
question 8: accuracy 0.6355
question 9: accuracy 0.7632
question 10: accuracy 0.6109
question 11: accuracy 0.6522
question 12: accuracy 0.8695
question 13: accuracy 0.7218
question 14: accuracy 0.7337
question 15: accuracy 0.6166
question 16: accuracy 0.7486
question 17: accuracy 0.7027
question 18: accuracy 0.9510
**********************************************
Average accuracy:        0.7558526065614488
```

Figure 2.A.14 – Results of Submission 1

Finally, The submission of the model to Kaggle.

```python
## Submission
import jo_wilder
env = jo_wilder.make_env()
iter_test = env.iter_test()

limits = {'0-4':(1, 4), '5-12':(4, 14), '13-22':(14, 19)}

for (test, sample_submission) in iter_test:
    test_df = feature_engineering(test)
    grp = test_df.level_group.values[0]
    a,b = limits[grp]
    for t in range(a,b):
        gbtm = models[f'{grp}_{t}']
        test_ds = tfdf.keras.pd_dataframe_to_tf_dataset(test_df.loc[:, test_df.columns != 'level_group'])
        predictions = gbtm.predict(test_ds)
        mask = sample_submission.session_id.str.contains(f'q{t}')
        n_predictions = (predictions > best_threshold).astype(int)
        sample_submission.loc[mask,'correct'] = n_predictions.flatten()

    env.predict(sample_submission)
```

Figure 2.A.15 – Submission to Kaggle

This Notebook run in about 399.2 seconds, achives a public score 0.639.

| Run | Public Score |
|---|---|
| 399.2s | 0.639 |

## B Submission 2 – GBT

Using the same Submission above, but with some modification. Instead of using custom features: and taking sub-feature, here I will take all features and check the results under the same steps.

```python
## Take only the features you interested in
categ_cols = ['event_name', 'name','fqid', 'room_fqid', 'text_fqid', 'music', 'hq', 'fullscreen']
num_cols = ['elapsed_time','level','page','room_coor_x', 'room_coor_y', 'screen_coor_x', 'screen_coor_y', 'hover_duration']
```

Figure 2.B.1 – Taking more Features in Submission 2

***Result of Submission 2 – GBT***

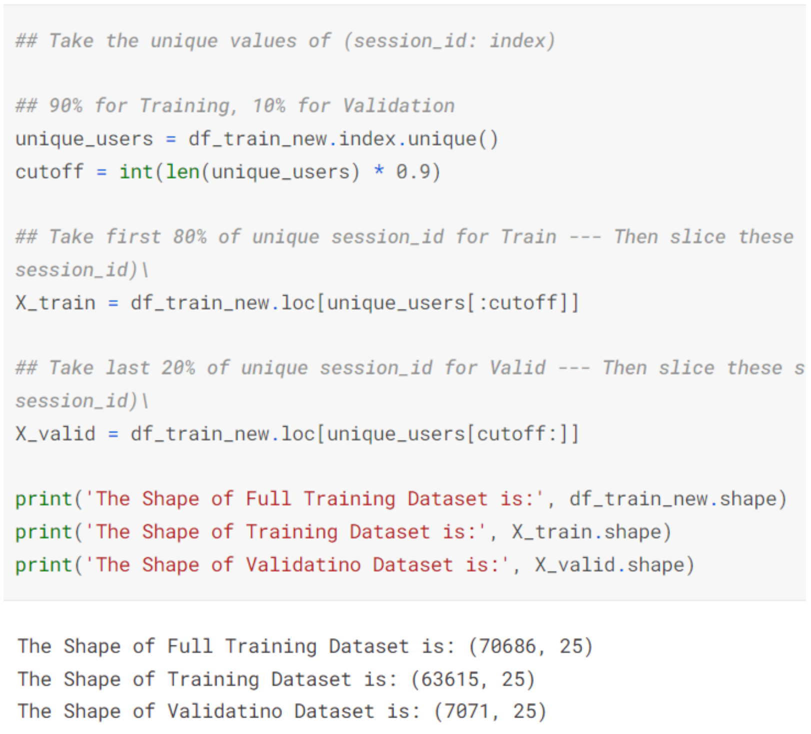
This Submission takes 407.6 seconds for running, and achieves a public score equals 0.639

.

| Run | Public Score |
|---|---|
| 407.6s | 0.639 |

***C Submission 3 – GBT***

Using the same Submission 2, but with some modification. Instead of taking 80% for training dataset, I will try to increase data in training data taking 90%, and 10% for validation.



```
## Take the unique values of (session_id: index)

## 90% for Training, 10% for Validation
unique_users = df_train_new.index.unique()
cutoff = int(len(unique_users) * 0.9)

## Take first 80% of unique session_id for Train --- Then slice these
session_id)\
X_train = df_train_new.loc[unique_users[:cutoff]]

## Take last 20% of unique session_id for Valid --- Then slice these s
session_id)\
X_valid = df_train_new.loc[unique_users[cutoff:]]

print('The Shape of Full Training Dataset is:', df_train_new.shape)
print('The Shape of Training Dataset is:', X_train.shape)
print('The Shape of Validatino Dataset is:', X_valid.shape)
```

```
The Shape of Full Training Dataset is: (70686, 25)
The Shape of Training Dataset is: (63615, 25)
The Shape of Validatino Dataset is: (7071, 25)
```

Figure 2.C.1 – Taking 90% for Training, 10% for validation

***Result of Submission 3 – GBT***

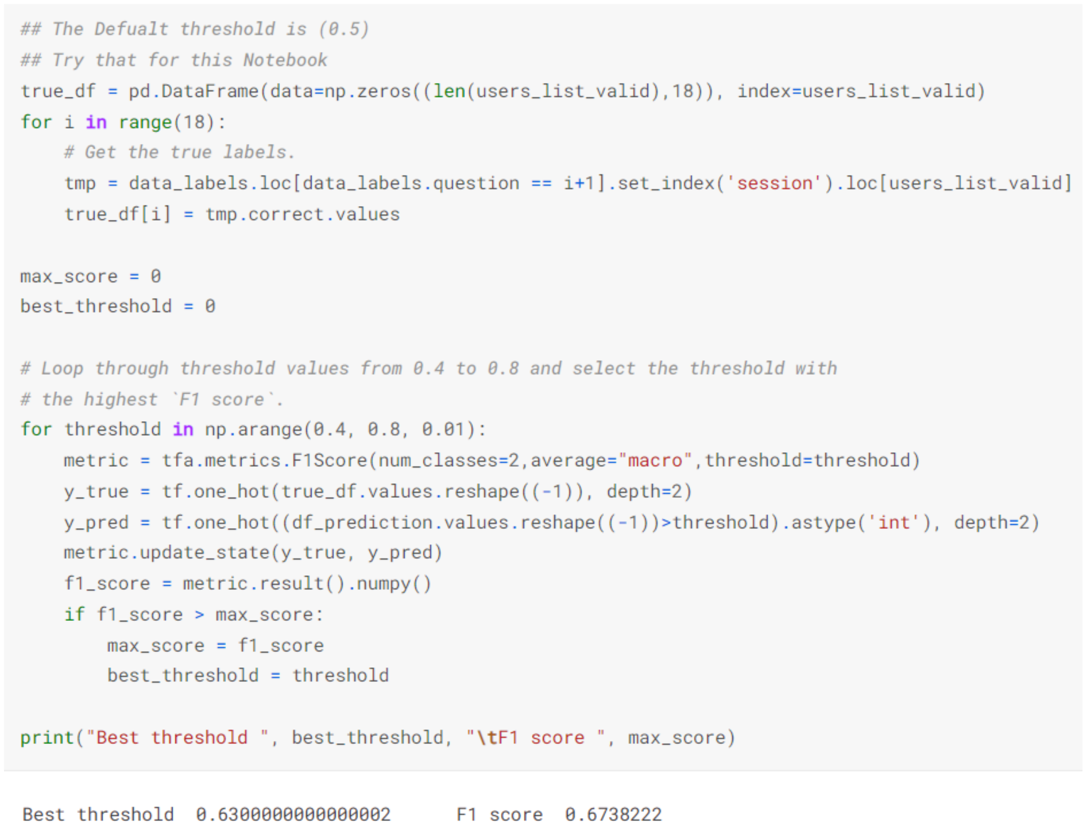
This Submission takes 436.9 seconds for running, and achieves a public score equals 0.639

| Run | Public Score |
|---|---|
| 436.9s | 0.639 |

### *D Submission 4 – GBT*

In this Submission, I use the as the Submission 1, but instead using threshold equals 0.5, I will search in some space to find the best threshold.

In the following figure, I search in space from 0.4 to 0.8 by step 0.01, The best threshold is 0.63. It achieves a score of 0.67 on the validation dataset.



```
## The Defualt threshold is (0.5)
## Try that for this Notebook
true_df = pd.DataFrame(data=np.zeros((len(users_list_valid),18)), index=users_list_valid)
for i in range(18):
    # Get the true labels.
    tmp = data_labels.loc[data_labels.question == i+1].set_index('session').loc[users_list_valid]
    true_df[i] = tmp.correct.values

max_score = 0
best_threshold = 0

# Loop through threshold values from 0.4 to 0.8 and select the threshold with
# the highest `F1 score`.
for threshold in np.arange(0.4, 0.8, 0.01):
    metric = tfa.metrics.F1Score(num_classes=2,average="macro",threshold=threshold)
    y_true = tf.one_hot(true_df.values.reshape((-1)), depth=2)
    y_pred = tf.one_hot((df_prediction.values.reshape((-1))>threshold).astype('int'), depth=2)
    metric.update_state(y_true, y_pred)
    f1_score = metric.result().numpy()
    if f1_score > max_score:
        max_score = f1_score
        best_threshold = threshold

print("Best threshold ", best_threshold, "\tF1 score ", max_score)
```

```
Best threshold  0.6300000000000002      F1 score  0.6738222
```

Figure 2.D.1 – Searching the Best Threshold

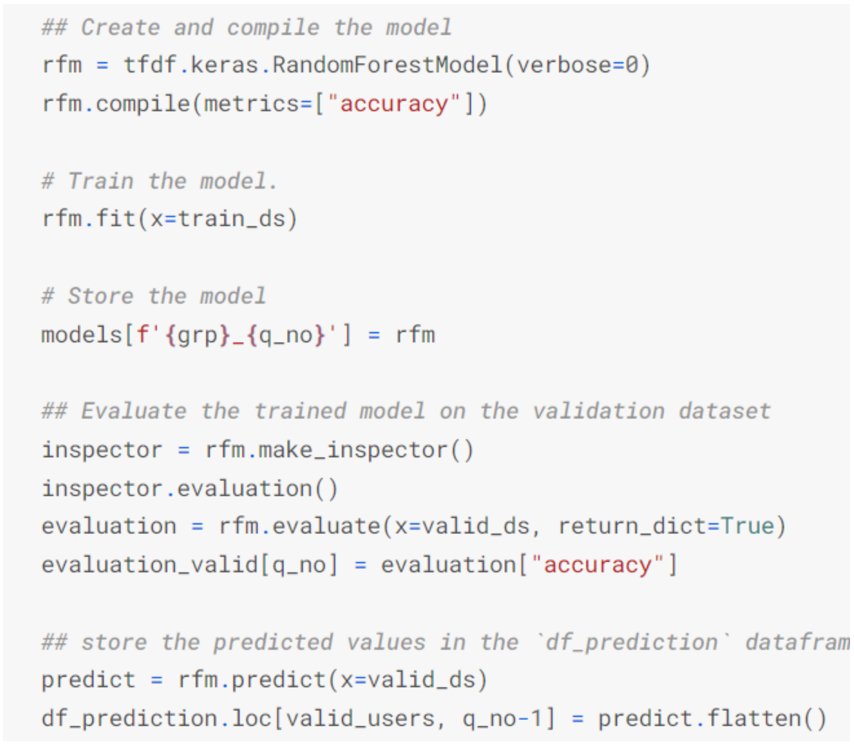
### *Result of Submission 4 – GBT*

This Submission takes 372.6 seconds for running, and achieves a public score equals 0.672

| Run | Public Score |
|---|---|
| 372.6s | 0.672 |

## *E Submission 5 – RFM*

In this Submission, I follow the same steps discussed above about finding the best threshold and using 80% for training, but instead of GBT; I use Random Forest Model (RFM).



```python
## Create and compile the model
rfm = tfdf.keras.RandomForestModel(verbose=0)
rfm.compile(metrics=["accuracy"])

# Train the model.
rfm.fit(x=train_ds)

# Store the model
models[f'{grp}_{q_no}'] = rfm

## Evaluate the trained model on the validation dataset
inspector = rfm.make_inspector()
inspector.evaluation()
evaluation = rfm.evaluate(x=valid_ds, return_dict=True)
evaluation_valid[q_no] = evaluation["accuracy"]

## store the predicted values in the `df_prediction` datafram
predict = rfm.predict(x=valid_ds)
df_prediction.loc[valid_users, q_no-1] = predict.flatten()
```

Figure 2.E.1 – Using Random Forest Model

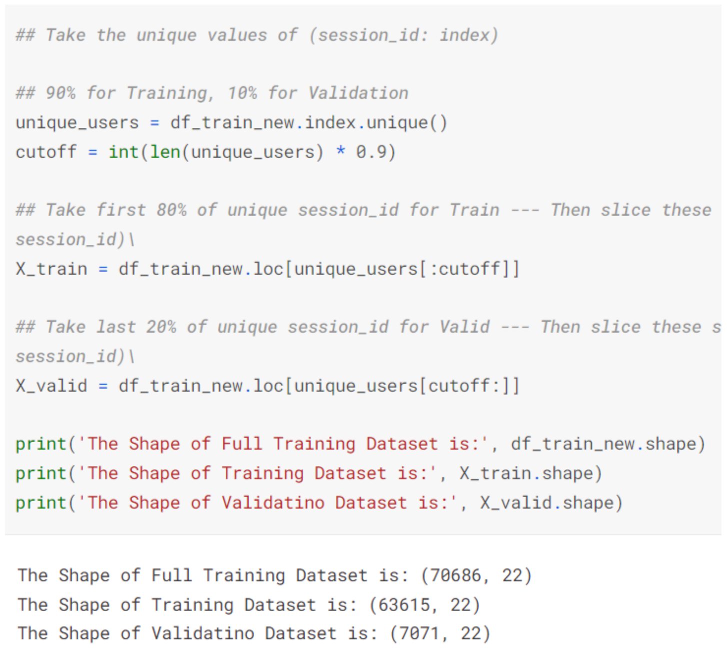
### *Result of Submission 5 – RFM*

This Submission takes 784 seconds for running, and achieves a public score equals 0.669

| Run | Public Score |
|---|---|
| 784.0s | 0.669 |

## *F Submission 6 – RFM*

In this Submission, I follow the same steps in Submission 5, but instead, use 90% of the  dataset for the training dataset, and 10% of the validation dataset.



```
## Take the unique values of (session_id: index)

## 90% for Training, 10% for Validation
unique_users = df_train_new.index.unique()
cutoff = int(len(unique_users) * 0.9)

## Take first 80% of unique session_id for Train --- Then slice these
session_id)\
X_train = df_train_new.loc[unique_users[:cutoff]]

## Take last 20% of unique session_id for Valid --- Then slice these s
session_id)\
X_valid = df_train_new.loc[unique_users[cutoff:]]

print('The Shape of Full Training Dataset is:', df_train_new.shape)
print('The Shape of Training Dataset is:', X_train.shape)
print('The Shape of Validatino Dataset is:', X_valid.shape)
```

```
The Shape of Full Training Dataset is: (70686, 22)
The Shape of Training Dataset is: (63615, 22)
The Shape of Validatino Dataset is: (7071, 22)
```

Figure 2.F.1 – Taking 90% for Training, 10% for Validation

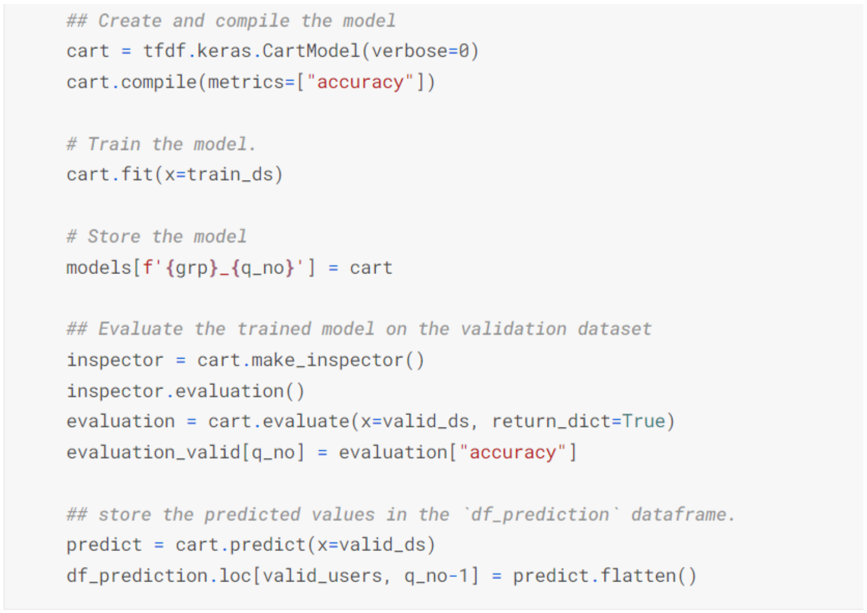
### *Result of Submission 6 – RFM*

This Submission takes 783.5 seconds for running, and achieves a public score equals 0.670

| Run | Public Score |
|---|---|
| 783.5s | 0.670 |

## G Submission 7 – CART

In this Submission, I use CART Model – Classification and Regression Trees – ; I use 80% for the training dataset, and use custom features (take only some features).



```
## Create and compile the model
cart = tfdf.keras.CartModel(verbose=0)
cart.compile(metrics=["accuracy"])

# Train the model.
cart.fit(x=train_ds)

# Store the model
models[f'{grp}_{q_no}'] = cart

## Evaluate the trained model on the validation dataset
inspector = cart.make_inspector()
inspector.evaluation()
evaluation = cart.evaluate(x=valid_ds, return_dict=True)
evaluation_valid[q_no] = evaluation["accuracy"]

## store the predicted values in the `df_prediction` dataframe.
predict = cart.predict(x=valid_ds)
df_prediction.loc[valid_users, q_no-1] = predict.flatten()
```

Figure 2.G.1 – CART Model

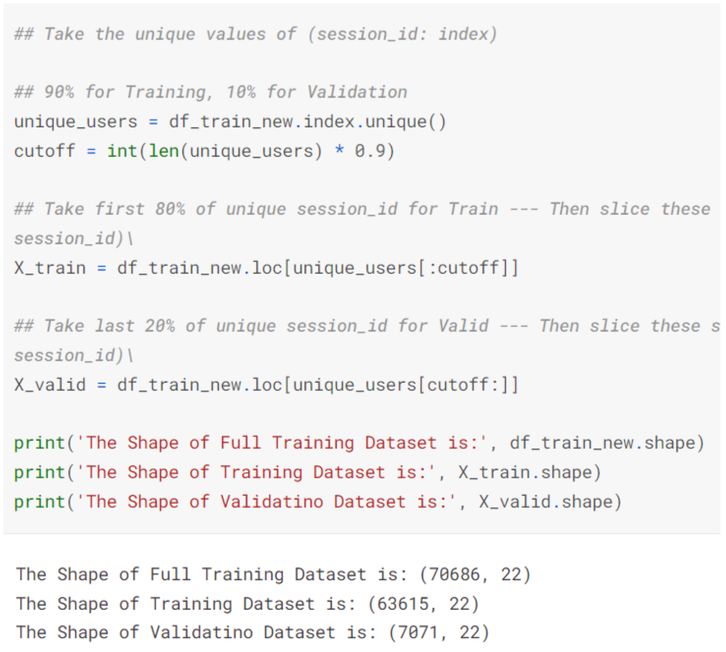
### *Result of Submission 7 – CART*

This Submission takes 269.5 seconds for running and achieves a public score equals 0.612

| Run | Public Score |
|---|---|
| 269.5s | 0.612 |

## *H Submission 8 – CART*

In this Submission, I follow the same steps in Submission 7, but instead taking 90% for the training dataset and 10% for the validation dataset.



```
## Take the unique values of (session_id: index)

## 90% for Training, 10% for Validation
unique_users = df_train_new.index.unique()
cutoff = int(len(unique_users) * 0.9)

## Take first 80% of unique session_id for Train --- Then slice these
session_id)\
X_train = df_train_new.loc[unique_users[:cutoff]]

## Take last 20% of unique session_id for Valid --- Then slice these s
session_id)\
X_valid = df_train_new.loc[unique_users[cutoff:]]

print('The Shape of Full Training Dataset is:', df_train_new.shape)
print('The Shape of Training Dataset is:', X_train.shape)
print('The Shape of Validatino Dataset is:', X_valid.shape)
```

```
The Shape of Full Training Dataset is: (70686, 22)
The Shape of Training Dataset is: (63615, 22)
The Shape of Validatino Dataset is: (7071, 22)
```

Figure 2.H.1 – Taking 90% for Training, 10% for Validation

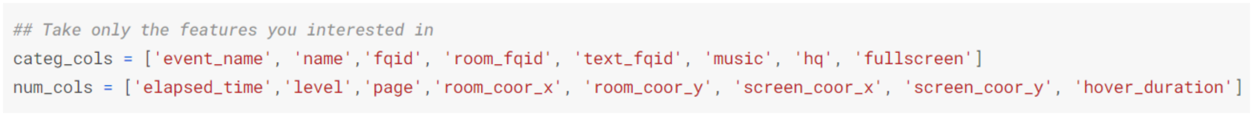
### *Result of Submission 8 – CART*

This Submission takes 269.5 seconds for running and achieves a public score equals 0.612

| Run | Public Score |
|---|---|
| 269.5s | 0.612 |

## *I Submission 9 – CART*

In this Submission, I follow the same steps in Submission 7, but instead some custom features, I will apply on all features.



```
## Take only the features you interested in
categ_cols = ['event_name', 'name','fqid', 'room_fqid', 'text_fqid', 'music', 'hq', 'fullscreen']
num_cols = ['elapsed_time','level','page','room_coor_x', 'room_coor_y', 'screen_coor_x', 'screen_coor_y', 'hover_duration']
```

Figure 2.I.1 – Taking all Features

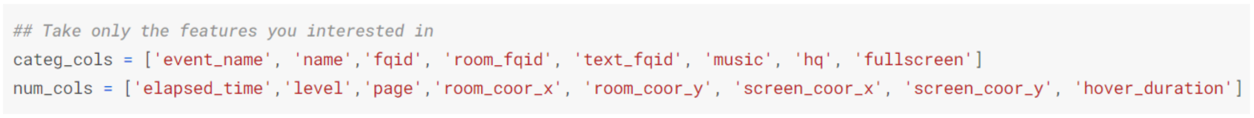
### *Result of Submission 9 – CART*

This Submission takes 280.9 seconds for running and achieves a public score equals 0.613

| Run | Public Score |
|---|---|
| 280.9s | 0.613 |

## *J Submission 10 – RFM*

In this Submission, I use Random Forest Model with searching for the best threshold and training on 80% of the dataset using all features



```
## Take only the features you interested in
categ_cols = ['event_name', 'name','fqid', 'room_fqid', 'text_fqid', 'music', 'hq', 'fullscreen']
num_cols = ['elapsed_time','level','page','room_coor_x', 'room_coor_y', 'screen_coor_x', 'screen_coor_y', 'hover_duration']
```

Figure 2.J.1 – Taking all Features

### *Result of Submission 10 – CART*

This Submission takes 680.7 seconds for running and achieves a public score equals 0.670

| Run | Public Score |
|---|---|
| 680.7s | 0.670 |

## 4 CONCLUSION

In this Competition we have tried to build a machine learning model that is able to predict the student performance of gameplay. The search in space to find the best threshold is very important and it highly affects the results. Adjusting the split of the dataset to 80% or 90% for the training dataset doesn't affect that much. Taking custom features or taking all features doesn't affect that much. Also choosing the model is very important as it differs in both efficiency and accuracy. My outperform model is using Gradient Boosting Trees with a number of trees equal to 100 finding the best threshold via searching and using only custom features.

## REFERENCES

[1] James Cussens, "Machine Learning," IEEE Journal of Computing and Control, Vol.7, No.4, pp.164-168, 1996

[2] Muhammad, I., & Yan, Z., "Supervised Machine Learning Approaches A Survey," ICTACT Journal on Soft Computing, Vol.5, No.3, 2015.

[3] S. B. Kotsiantis, "Supervised Machine Learning: A Review of Classification Techniques," Informatica, Vol.31, No.3, pp.249-268, 2007.

[4] Richard S. Sutton and Andrew G. Barto, "Reinforcement Learning: An Introduction," Cambridge, MA: MIT Press, 1998.

[5] Sen, P. C., Hajra, M., & Ghosh, M., "Supervised classification algorithms in Machine Learning: A survey and review," Emerging technology in modelling and graphics, Springer, pp.99-111, 2020.

[6] Kadhim, A. I.,"Survey on supervised Machine Learning techniques for automatic text classification," AI Review, Vol.52, No.1, pp.273-292, 2019.