Predict Student’s Performance from Game Play

By/ Ahmed Rashed Salem GitHub: [ahmedsalem88](https://github.com/ahmedsalem88/student-performance-kaggle) Kaggle: [ahmedrashedsalem](https://www.kaggle.com/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***

# Introduction

Game-based learning is an effective way to make learning more enjoyable and interactive. It involves the use of games to teach educational content, which engages students and makes the learning experience dynamic. However, despite its growing popularity in educational settings, there is a shortage of open datasets that can be used to apply data science and learning analytics principles to enhance game-based learning. Moreover, most game-based learning platforms do not fully utilize knowledge-tracing methods to support individual students. Although knowledge tracing has been studied in the context of online learning environments and intelligent tutoring systems, there has been little emphasis on its application in educational games. To address these issues, Field Day Lab, a publicly-funded research lab at the Wisconsin Center for Educational Research, designs games for various subjects and age groups, using game data to understand how people learn. The lab's commitment to accessibility ensures that all of its games are available to everyone for free. Additionally, the lab collaborates with nonprofits like The Learning Agency Lab to develop the science of learning-based tools and programs for the social good. With this support, game developers can improve educational games and provide educators with dashboards and analytic tools to enhance their use of game-based learning platforms, leading to broader support for this educational approach.

In this Dataset, 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](https://en.wikipedia.org/wiki/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.

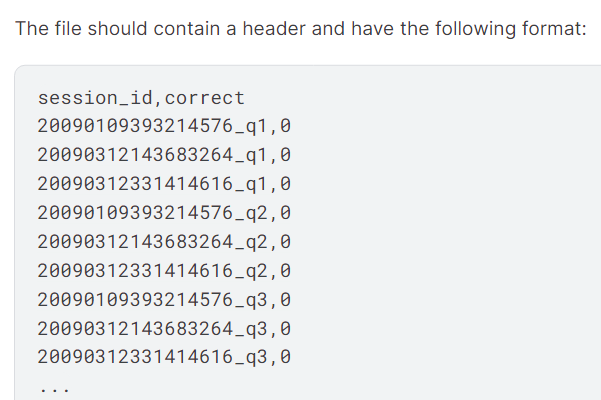


Figure 1.1 – sample of submission

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](https://www.kaggle.com/competitions/predict-student-performance-from-game-play/overview/efficiency-prize-evaluation)

After finishing my work, I upload my code to [GitHub](https://github.com/ahmedsalem88/student-performance-kaggle). GitHub is a web-based platform used for version control and collaborative software development. It provides a centralized location where developers can store and manage their code, as well as collaborate with other developers in real time. GitHub's user-friendly interface and powerful features have made it one of the most popular platforms for hosting and sharing open-source software projects. It also allows developers to track changes to their code, coordinate project updates, and identify and fix bugs quickly. In addition, GitHub offers a wide range of integrations with other tools, such as issue tracking, continuous integration and deployment, and code review, which help streamline the development process. Overall, GitHub has revolutionized the way software development is done, enabling developers to work more efficiently and collaboratively than ever before.

# 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.

# 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.

## 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.

Figure 2.A.1 – reading the dataset

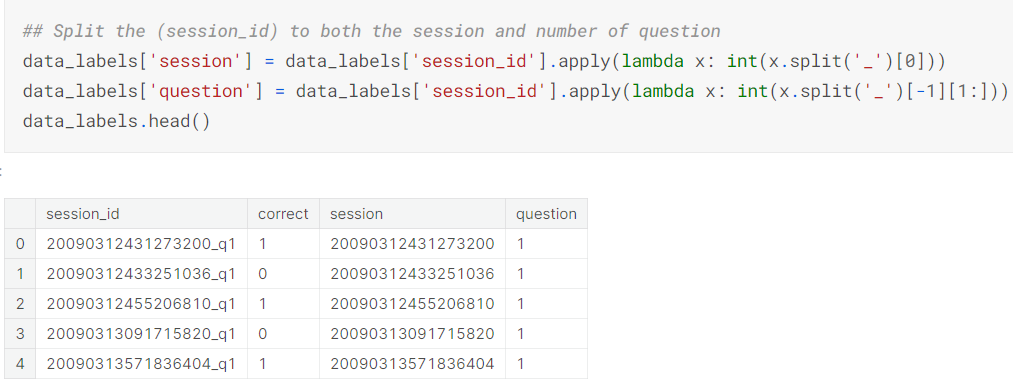
The labels are stored in a separated file, It contains the session\_id and correct columns. I will split the session\_id column to (session and question number)

Figure 2.A.2 – Labels

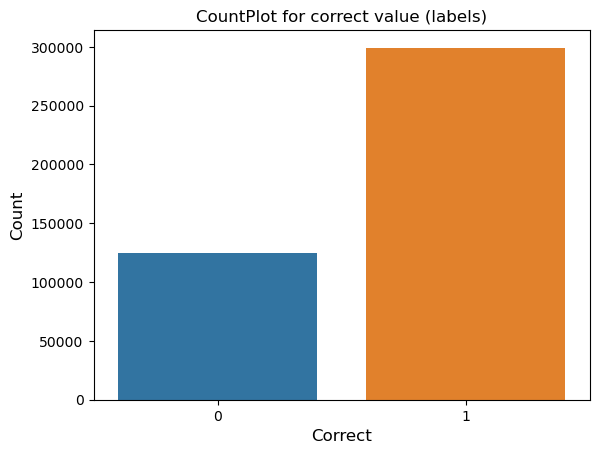
I will make some visualization to check the distribution of the label (correct) column to check the balancing of classes. It shows that the dataset is imbalanced.

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)

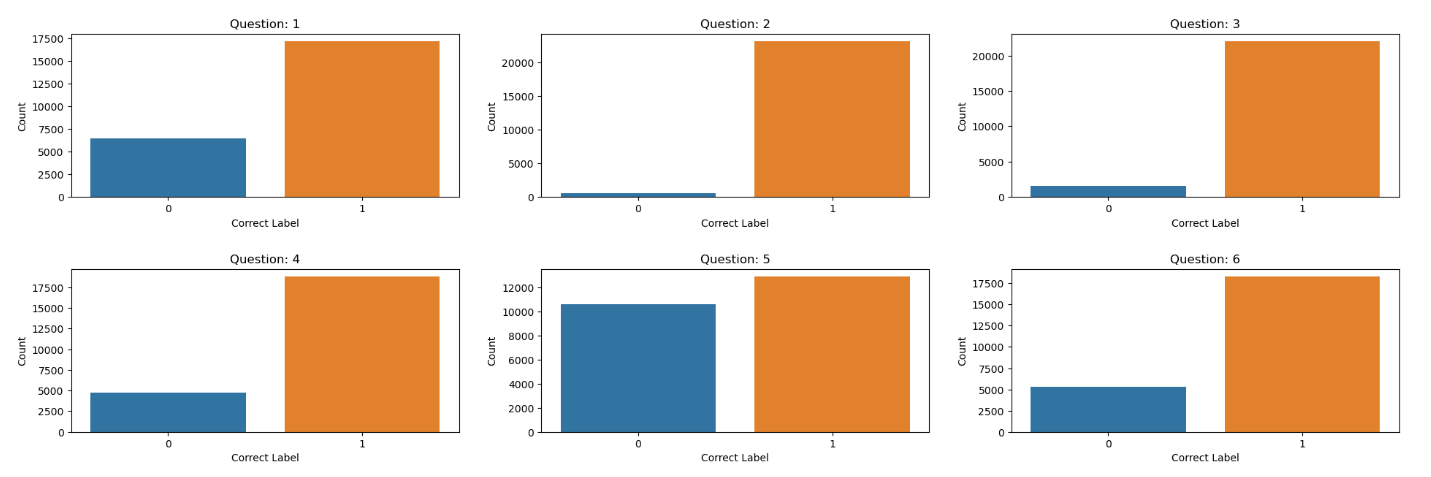
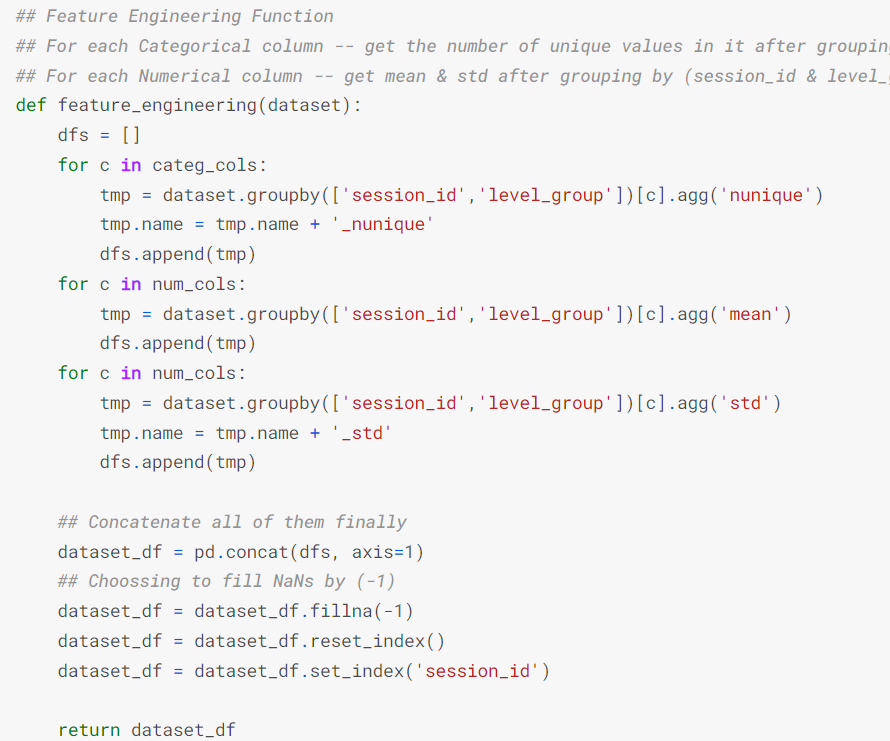


Figure 2.A.4 – Distribution of correct column for each question

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

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).

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

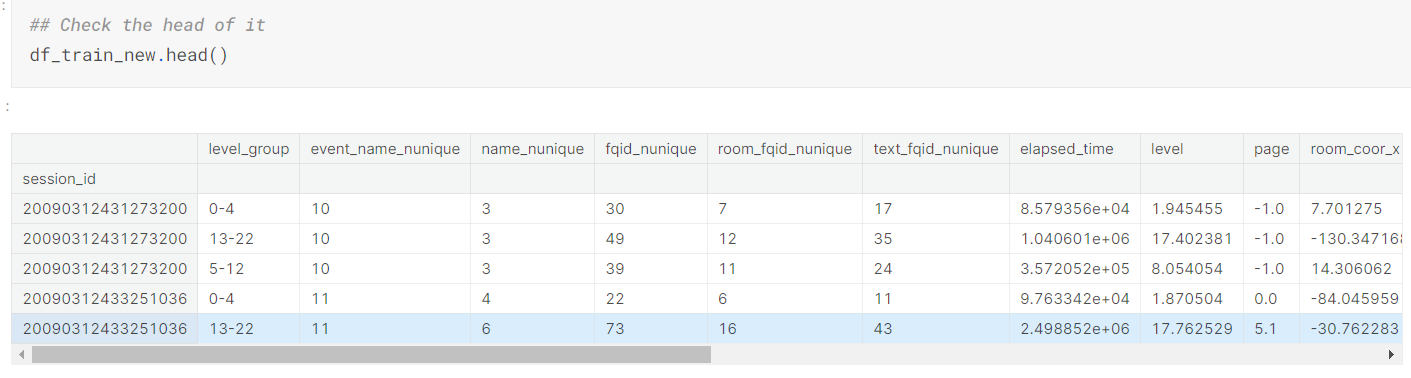


Figure 2.A.7 – Final output of Feature Engineering Function

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

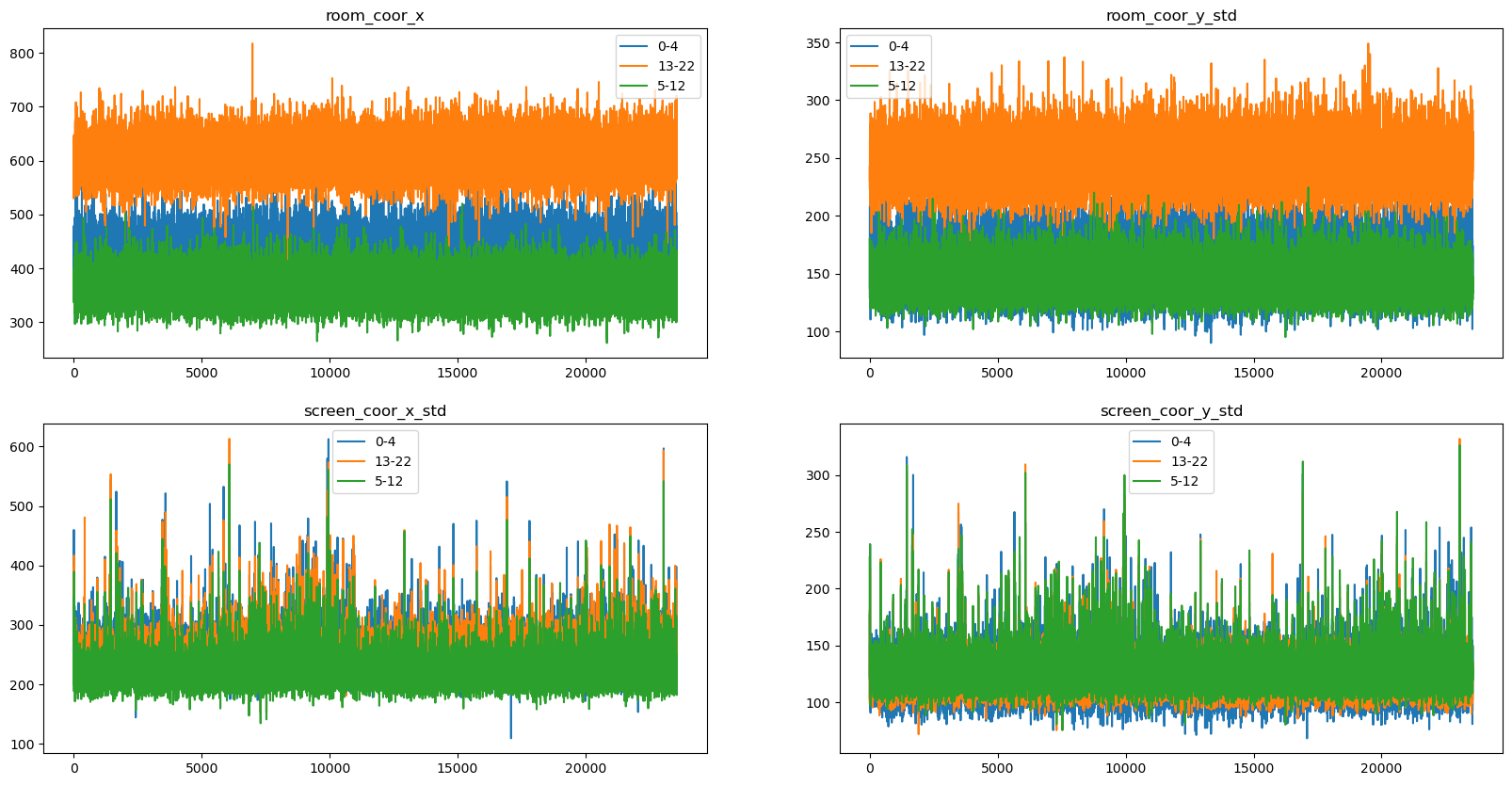


Figure 2.A.8 – Distribution of some Numerical Feature after Processing

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).

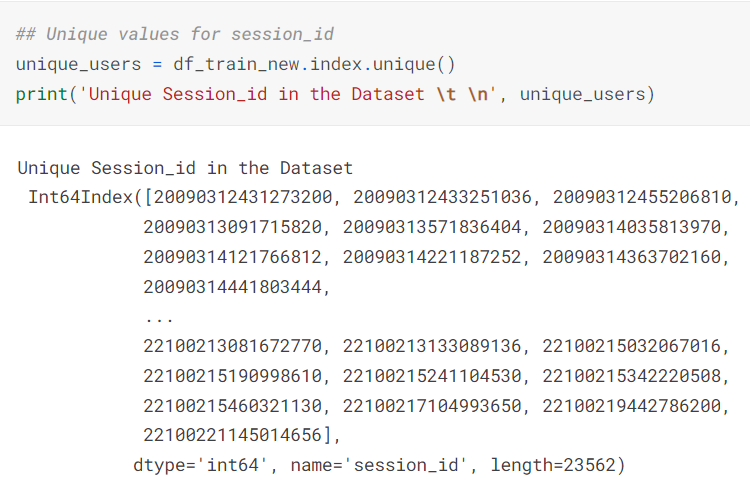


Figure 2.A.9 – Unique Session Ids in dataset

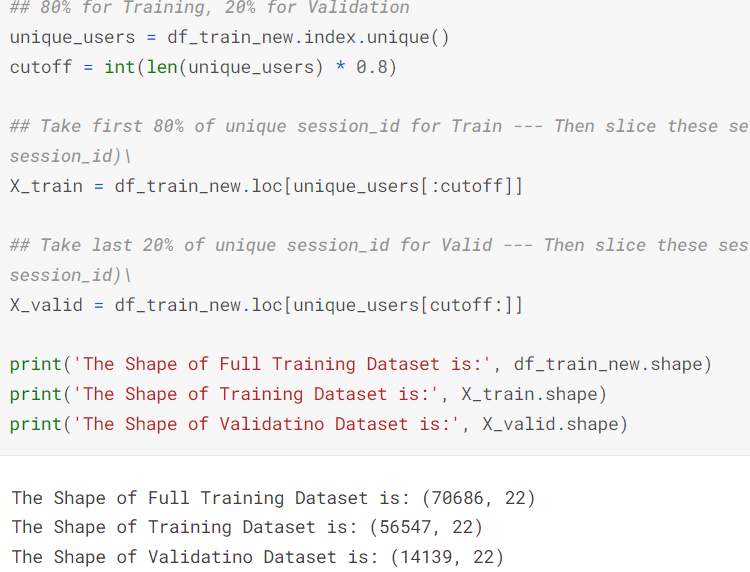
In the above figure, it shows that unique values about 23562 value and the processed dataset contains about 70686 values, of course we will not remove duplicated in session\_id. We will take the first 80% of unique values for the training dataset and the other 20% for validation. I hope this will be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

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

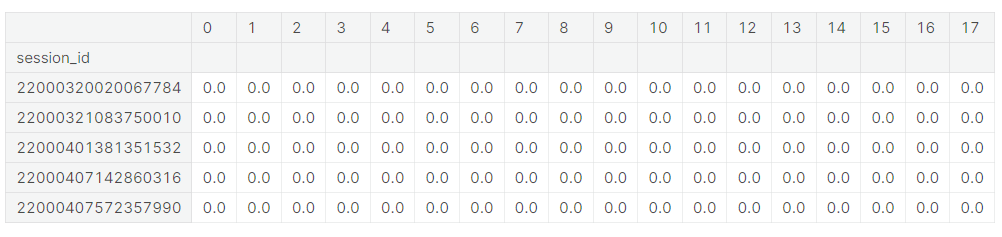
Moving forward to training the models, I will start by GBT (Gradient boosting Trees) but before training, I will prepare a data frame for the prediction of validation dataset indexed by unique values of session\_id in the validation dataset and has 18 columns: one column for each question (we said that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

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

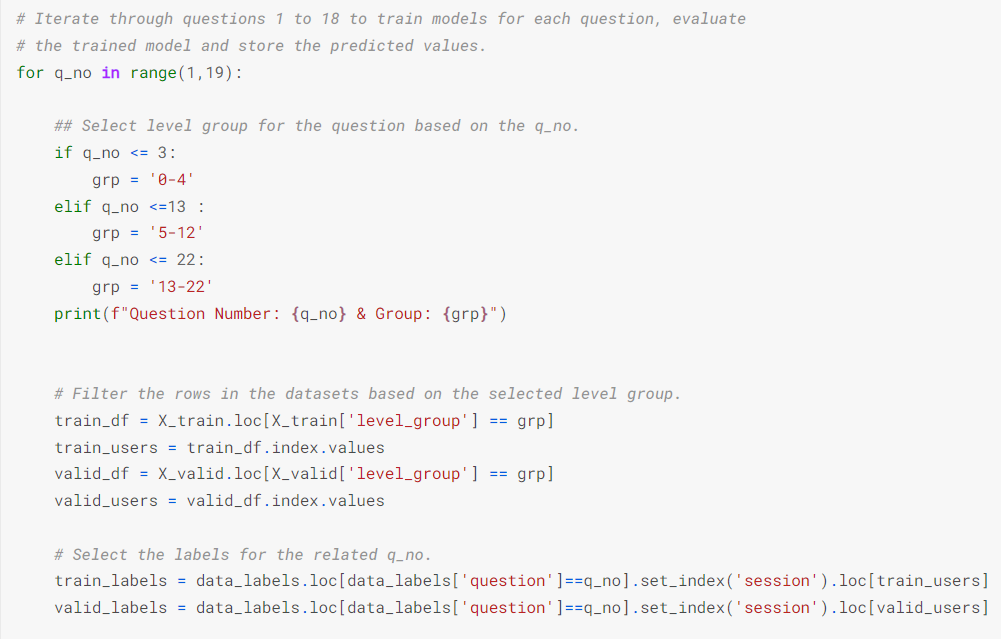
Looping for 18 times, each time for each question, but how to specify the level\_group. That is an important note. Questions 1 to 3 for (level\_group 0-4), Questions 4 to 13 for (level\_group 5-12), and Questions 14 to 18 for (level\_group 13-22). Slicing from the data frame according to the number of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

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

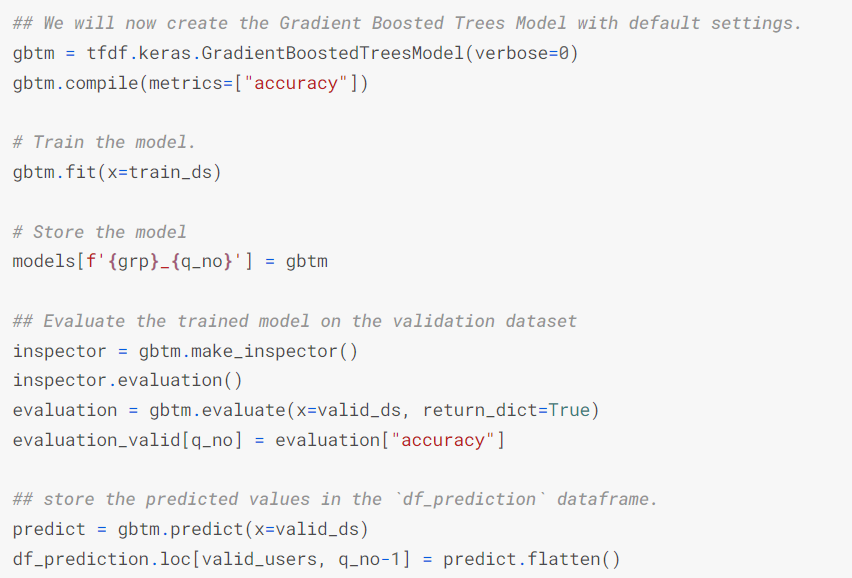
The second part of the loop is training the model, Here I’m training GBT (Gradient Boosting Trees) with number of trees equals 100; each tree is working on a subsample and sub features and finally, we take the majority voting of all trees. It is a type of Ensemble Learning.

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

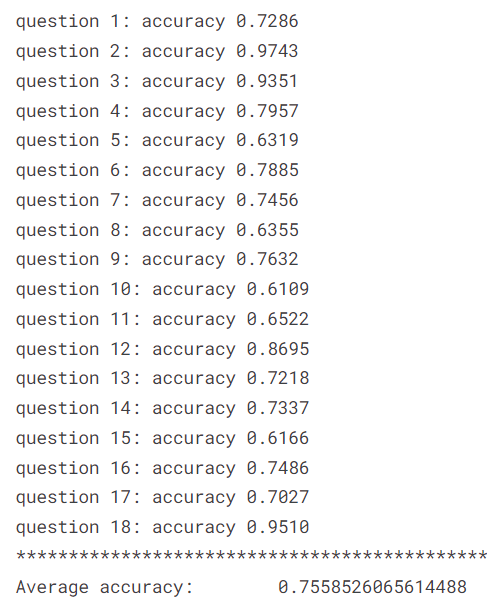
The Average Accuracy of Submission 1 for all Questions is about 0.75.

Figure 2.A.14 – Results of Submission 1

Finally, The submission of the model to Kaggle.



Figure 2.A.15 – Submission to Kaggle

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



## 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.

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

.

## 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.

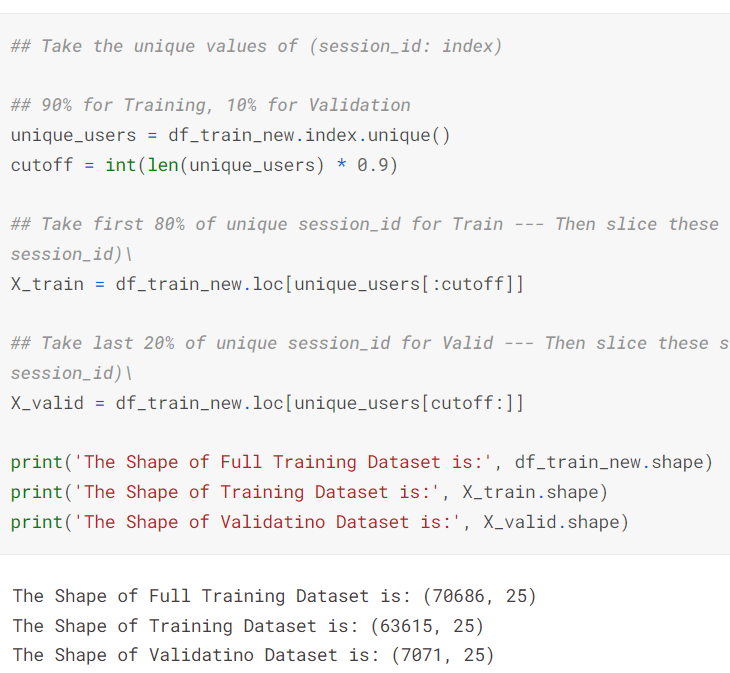


Figure 2.C.1 – Taking 90% for Training, 10% for validation

## Result of Submission 3 – GBT

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

## 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.



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

## Submission 5 – RFM

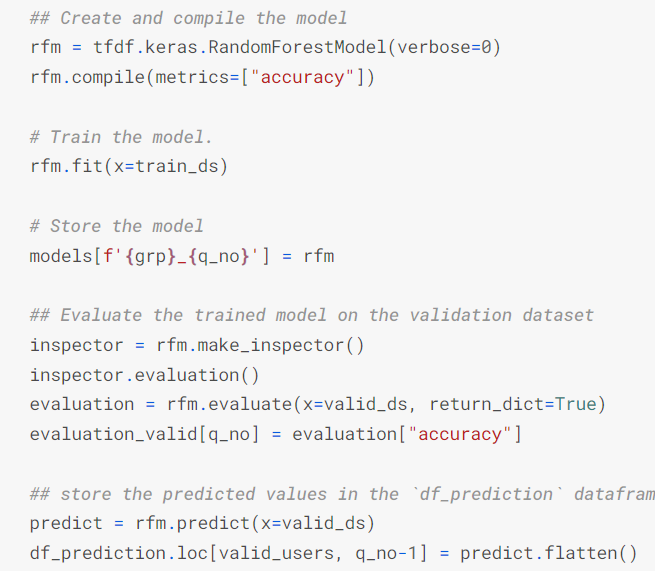
In this Submission, I follow the same steps discussed above about finding the best threshold and using 80% for training, but instead of GBT; I use Random Forest Model (RFM).

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



## Submission 6 – RFM

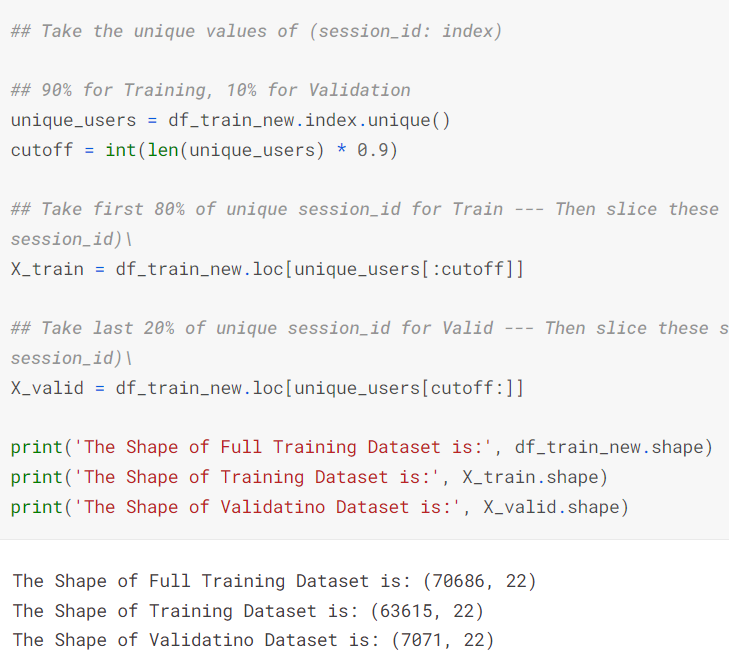
In this Submission, I follow the same steps in Submission 5, but instead, use 90% of the dataset for the training dataset, and 10% of the validation dataset.

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



## Submission 7 – CART

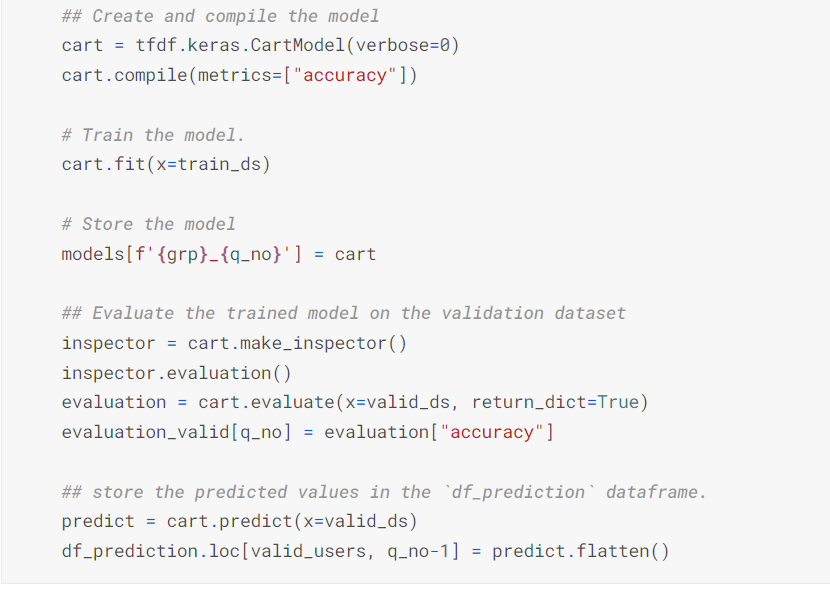
In this Submission, I use CART Model – Classification and Regression Trees – ; I use 80% for the training dataset, and use custom features (take only some features).

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



## Submission 8 – CART

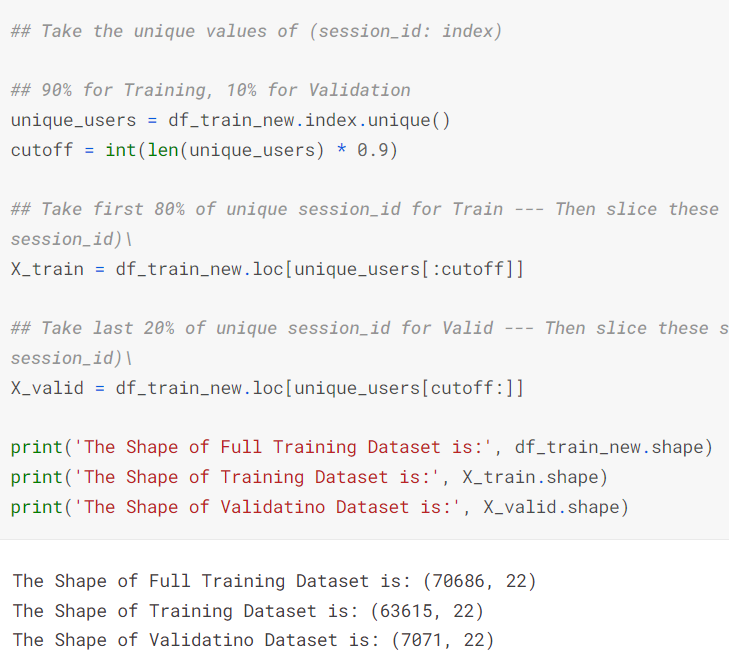
In this Submission, I follow the same steps in Submission 7, but instead taking 90% for the training dataset and 10% for the validation dataset.

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



## 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.

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



## 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

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

# 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.