Forecasting Video Game Sales

Haolan Mai

- **Introduction**
- **Problem statement**
- **Methods**
- **Theory**
- **Dataset description**
- **Results**
- **Conclusion**

Introduction:

Video games have become more and more popular nowadays, not just for teenagers but also for adults. The main reason is because video game it's a temporary escape from the pressures of the real world. No surprise that the tensions of the pandemic brought more people into the gaming fold. A survey by Deloitte, a professional services company, found that 34 percent of those surveyed had tried a new video gaming activity in 2020. Researchers also found that there has been a large growth since the inception in the 1970s. Video industry reached $16.5 billions in U.S. sales in 2015. The PC games market has increased in digital sales and the number of releases after Valve launched its Steam Store. The Steam Store saw a major growth of the video industry, and thanks to the program named Greenlight, which helps developers to release their games on Steam without a publisher. Greenlight has grown more and more larger and increased their number of releases year by year. Because of this, it became increasingly difficult for developers to stand out and even sell enough copies to fund the development of their games. Our goal in our project is to help the developers to make a prediction relating to the global sales of the video games ahead of time so that they can develop a game with higher market competitiveness.

To achieve our goal, we applied the machine learning methods of Neural Network, Linear Regression and Gradient Boosting. With different models , we predicted the global sales for the video games and identified the optimal model with a better performance. We worked on the dataset from Kaggle called "Video Game Sales with Ratings" and did the data cleaning to generate the dataset which is compatible with our model. We would discuss in a further and insightful way with more detailed analysis.

After doing some research, we found some similar projects on kaggle. One of the groups performed seven algorithms and concluded that Gradient Boosting gives the best prediction among others(see reference: https://www.kaggle.com/jruots/forecasting-video-game-sales). Now, we will start our research and see what the result is.

Problem Statement:

If the success of games could be predicted beforehand, it would allow game creators to adjust the development to attract a larger audience or perhaps forsake their attempt to develop a game if it did not lead to a success. Thus, it would be useful to evaluate a concept; not an already finished game. Previous attempts at video games success prediction assumed the game was already released and made predictions based on that knowledge. In addition, they dealt mostly with pre-2010 console games when there was no incentive to study the PC games market. Compared with traditional forecasting, machine learning forecasting allows for more data to be fused into the forecast. We are interested in which machine learning algorithm we would choose to build a decent model for forecasting the global sales of video games. For effective forecasting,

we need to consider the size of training data, the accuracy/interpretability of output, the speed/training time and the number of features, so choosing the right algorithm is very important.

Method:

The methods that we will be using are Neural Network, Random Forest, Linear Regression and Gradient Boosting.

Neural Network(MLP):

Neural Network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. In this sense, neural networks refer to systems of neurons, either organic or artificial in nature.Here, we use MLP (multilayer perceptron) which is a class of feedforward artificial neural network. In MLP, it consists of at least three layers of nodes: an input layer, a hidden layer and an output layer. The input layer receives the input signal to be processed. The required task such as prediction and classification is performed by the output layer. An arbitrary number of hidden layers that are placed in between the input and output layer are the true computational engine of the MLP. Similar to a feed forward network in a MLP the data flows in the forward direction from input to output layer. The neurons in the MLP are trained with the back propagation learning algorithm. MLP is flexible and can be used for both regression and classification problem dealing. MLP are suitable for regression prediction problems where a real-valued quantity is predicted given a set of inputs. Data is often provided in a tabular format, such as the CSV we would use in this project.

For convenience, we would use the sklearn.neural_network.MLPRegressor() which implements a multi-layer perceptron (MLP) that trains using backpropagation with no activation function in the output layer, which can also be seen as using the identity function as activation function.

Random Forest:

Random forest is composed of many decision trees, and there is no correlation between different decision trees.When we perform classification tasks, new input samples are entered, and each decision tree in the forest is judged and classified separately. Each decision tree will get its own classification result. Which of the classification results of the decision tree is classified At most, the random forest will treat this result as the final result.The algorithm would break into 4 steps:

1. A sample with a sample size of N is drawn N times with replacement, one sample is drawn each time, and finally N samples are formed. The selected N samples are used to train a decision tree as the samples at the root node of the decision tree.
2. When each sample has M attributes, when each node of the decision tree needs to be split, randomly select m attributes from these M attributes, and satisfy the condition m < M. Then use a certain strategy (such as information gain) from these m attributes to select 1 attribute as the split attribute of the node.
3. In the process of decision tree formation, each node must be split according to step 2 (it is easy to understand that if the next attribute selected by the node is the attribute that was just used when the parent node was split, then the node has reached the leaf Node, there is no need to continue to split). Until it can no longer be divided. Note that there is no pruning during the entire decision tree formation process.
4. Follow steps 1 to 3 to build a large number of decision trees, which constitutes a random forest.

For convenience, we would use the sklearn.ensemble.RandomForestRegressor().

Linear Regression

In Linear Regression, the algorithm is based on supervised learning and performs a regression task. The Regression models a target prediction value based on independent variables. Our goal is to find out the relationship between variables and forecasting. When training our model, we are given that:
X is our input training data
Y is the labels to the data

When training the model – it fits the best line to predict the value of y for a given value of x. And the model will get the best regression by finding the best θ1 and θ2 value where **θ1:** intercept, **θ2:** coefficient of x

Gradient Boosting

Gradient boosting is a type of machine learning boosting. It mostly relies on the intuition that the best possible next mode when combined with previous models and minimizes the overall prediction error. The main goal of setting up a new model is to minimize the error. And the target outcome depends on each data set and how much changing that case prediction will impact the overall prediction. There's two cases about gradient boosting.

1. If a small change in the prediction for a case causes a large drop in error, then the next target outcome of the case is a high value. Predictions from the new model that are close to its targets will reduce the error.

2. If a small change in the prediction for a case causes no change in error, then the next target outcome of the case is zero. Changing this prediction does not decrease the error.

For convenience, we would use sklearn.ensemble.GradientBoostingRegressor().

Theory:

Linear Regression:

In Linear Regression, we want to obtain a simple function to explain our dependent variable yk in terms of independent variable xk where (x1, y1),...,(xN, yN) is the training dataset.
We consider the following polynomial regression x → yˆ(x;w),

$$\hat{\mathbf{y}}(x;\mathbf{w}) := w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M = \sum_{k=0}^{M} w_k x^M = \boldsymbol{\phi}(x)^T \mathbf{w},$$

where

$$\boldsymbol{\phi}(x) = \begin{bmatrix} 1 & x & \cdots & x^M \end{bmatrix}^T, \qquad \mathbf{w} = \begin{bmatrix} w_0 & w_1 & \cdots & w_M \end{bmatrix}^T \in \mathbb{R}^{M+1}.$$

Our goal is to find the components of w give the coefficients of monomials of x. We want to choose the parameter w so that the corresponding polynomial regression gives a good fit for the training data.

The advantages for using Linear Regression is that it works on any size of the dataset and works very well on non-learner problems. Linear Regression also provides the best approximation of the relationship between dependent and independent variables but there are still some disadvantages and some conditions of using this algorithm. It requires choosing the right polynomial degree for good bias and variance tradeoff and the presence of one or two outliers in the data seriously affect the results of the nonlinear analysis and it is too sensitive to the outliers.

Since linear regression assumes a linear relationship between the input and output variables, it fails to fit complex datasets properly. In most real life scenarios the relationship between the variables of the dataset isn't linear and hence a straight line doesn't fit the data properly.

Random Forest:

There are many advantages for the random forest. It can produce very high-dimensional data without dimensionality reduction or feature selection. It can identify the importance of features and mutual influence between different characteristics. It is not easy to overfit. The training speed is relatively fast, and it is easy to make a parallel method. For unbalanced data sets, it can balance errors.

However, Random Forests have been shown to overfit in some noisy classification or regression problems. Additionally, for data with attributes with different values, attributes with more value divisions will have a greater impact on the random forest, so the attribute weights produced by the random forest on this type of data are unreliable.

Neural Network(MLP)

In regression, the output activation function of MLP is just the identity function.And for regression, MLP uses the Square Error loss function. Starting from initial random weights, multi-layer perceptron (MLP) minimizes the loss function by repeatedly updating these weights. After computing the loss, a backward pass propagates it from the output layer to the previous layers, providing each weight parameter with an update value meant to decrease the loss. MLP is sensitive to feature scaling, so one of the assumptions of MLP is that the data is scaled. Also, Neural networks are good to model with nonlinear data with a large number of inputs, including images. It is reliable in an approach of tasks involving many features. It works by splitting the problem of classification into a layered network of simpler elements. But there are some conditions and disadvantages of using the neural network. Since neural networks are black boxes, which means that we are not able to know whether each independent variable is influencing the dependent variables. Since Neural networks depend a lot on training data. This leads to the problem of overfitting and generalization. The mode relies more on the training data and may be tuned to the data.

Gradient Boosting

Gradient boosting is a naive algorithm that can easily bypass a training data collection.The regulatory methods that panelize different parts of the algorithm will benefit from increasing the algorithm;s efficiency by minimizing over fitness.

In the mathematical understanding, we have a range of attributes x and goals which is nothing but the target we are going to predict. {(xi,Yi)}i=1,...,n, we use to

restore the y = f(x) dependency where xi is the features and Yi is the respective target. We restore dependency by approximating f^(x) and knowing the approximation is stronger by using the L(y,f) loss function . To get the loss, we are going to pass both the f(x) output and the actual target value Y which returns the loss of using this function f(x).

When we discuss the advantages of Gradient Boosting, it can provide us with predictive accuracy that cannot be easily beat. It also has lots of flexibility which can optimize different loss functions and provides several hyperparameter tuning options that make the function fit very flexible. Plus, this algorithm does not require data pre-processing which can deal with numerical and categorical values.

However, gradient boosting could also overemphasize outliers and cause overfitting since it improves to minimize all errors. So we have to use the cross-validation to avoid the issue. Additionally, gradient boosting is computationally expensive since it needs too many trees that could consum dratimally large amount of time and memory.

Data description:

The data we choose is the dataset from kaggle, called Video Game Sales with Ratings. This dataset simply extends the number of variables with another web scrape from Metacritic. This dataset consist 16,719 video games with attributes:

- Name(Name of the game)
- Platform: Console on which the game is running
- Year of Release: Year in which the game was released
- Genre: Genre the game belongs to
- Publisher: Name of the publisher who created the game
- NA Sales: Sales in North America (in millions of units)
- EU Sales: Sales in the European Union (in millions of units)
- JP Sales: Game Sales in Japan (in millions of units)
- Other Sales: Sales in the rest of the world, i.e. Africa, Asia excluding Japan, Australia, Europe excluding the E.U. and South America (in millions of units)
- Global Sales: Total Sales in the world (in millions of units)
- Critic Score: Aggregate score compiled by Metacritic staff
- Critic Count: The number of critics used in coming up with the Critic score
- User Score: Score by Metacritic's subscribers
- User Count: Number of users who gave the User score
- Developer: Party responsible for creating the game
- Rating: The ESRB ratings (E.g. Everyone, Teen, Adults Only, etc.)

Here is the initial dataset:

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales	JP_Sales	Other_Sales	Global_Sales	Critic_Score	Critic_Count	User_Score	User_Count	Developer	Rating
count	16717	16719	16450.000000	16717	16665	16719.000000	16719.000000	16719.000000	16719.000000	16719.000000	8137.000000	8137.000000	10015	7590.000000	10096	9950
unique	11562	31	NaN	12	581	NaN	NaN	NaN	NaN	NaN	NaN	NaN	96	NaN	1696	8
top	Need for Speed: Most Wanted	PS2	NaN	Action	Electronic Arts	NaN	NaN	NaN	NaN	NaN	NaN	NaN	tbd	NaN	Ubisoft	E
freq	12	2161	NaN	3370	1356	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2425	NaN	204	3991
mean	NaN	NaN	2006.487356	NaN	NaN	0.263330	0.145025	0.077602	0.047332	0.533543	68.967679	26.360821	NaN	162.229908	NaN	NaN
std	NaN	NaN	5.878995	NaN	NaN	0.813514	0.503283	0.308818	0.186710	1.547935	13.938165	18.980495	NaN	561.282326	NaN	NaN
min	NaN	NaN	1980.000000	NaN	NaN	0.000000	0.000000	0.000000	0.000000	0.010000	13.000000	3.000000	NaN	4.000000	NaN	NaN
25%	NaN	NaN	2003.000000	NaN	NaN	0.000000	0.000000	0.000000	0.000000	0.060000	60.000000	12.000000	NaN	10.000000	NaN	NaN
50%	NaN	NaN	2007.000000	NaN	NaN	0.080000	0.020000	0.000000	0.010000	0.170000	71.000000	21.000000	NaN	24.000000	NaN	NaN
75%	NaN	NaN	2010.000000	NaN	NaN	0.240000	0.110000	0.040000	0.030000	0.470000	79.000000	36.000000	NaN	81.000000	NaN	NaN
max	NaN	NaN	2020.000000	NaN	NaN	41.360000	28.960000	10.220000	10.570000	82.530000	98.000000	113.000000	NaN	10665.000000	NaN	NaN

There are about 17k games, but some of the data is missing. For instance, only around half of all games have a critic_score. This might be a problem for the prediction model, as critic_score can be one of the major factors determining the Global Sales. User_Score has a non-numeric format. Most of the games have a pretty good score, 7+ (or 70+ for critic score)

We are doing 3 things:

- Dropping games without a year of release or genre
- Creating a new column for age of the game
- Converting User_Score to float and replacing the tbd value in dataset with *NA*
- Renaming columns for ease of use

Here is the overview of the data after cleaning:

	Name	Platform	Year	Genre	Publisher	NA	EU	JP	Other	Global	Critic_Score	Critic_Count	User_Score	User_Count	Developer	Rating	Age
count	16448	16448	16448.000000	16448	16416	16448.000000	16448.000000	16448.000000	16448.000000	16448.00000	7983.000000	7983.000000	7463.000000	7463.000000	9907	9769	16448.000000
unique	11429	31	NaN	12	579	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1680	8	NaN
top	Need for Speed: Most Wanted	PS2	NaN	Action	Electronic Arts	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Ubisoft	E	NaN
freq	12	2127	NaN	3308	1344	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	201	3922	NaN
mean	NaN	NaN	2006.488996	NaN	NaN	0.263965	0.145895	0.078472	0.047583	0.53617	68.994363	26.441313	7.126330	163.015141	NaN	NaN	11.511004
std	NaN	NaN	5.877470	NaN	NaN	0.818286	0.506660	0.311064	0.187984	1.55846	13.920060	19.008136	1.499447	563.863327	NaN	NaN	5.877470
min	NaN	NaN	1980.000000	NaN	NaN	0.000000	0.000000	0.000000	0.000000	0.01000	13.000000	3.000000	0.000000	4.000000	NaN	NaN	-2.000000
25%	NaN	NaN	2003.000000	NaN	NaN	0.000000	0.000000	0.000000	0.000000	0.06000	60.000000	12.000000	6.400000	10.000000	NaN	NaN	8.000000
50%	NaN	NaN	2007.000000	NaN	NaN	0.080000	0.020000	0.000000	0.010000	0.17000	71.000000	22.000000	7.500000	24.000000	NaN	NaN	11.000000
75%	NaN	NaN	2010.000000	NaN	NaN	0.240000	0.110000	0.040000	0.030000	0.47000	79.000000	36.000000	8.200000	81.000000	NaN	NaN	15.000000
max	NaN	NaN	2020.000000	NaN	NaN	41.360000	28.960000	10.220000	10.570000	82.53000	98.000000	113.000000	9.700000	10665.000000	NaN	NaN	38.000000

From, the output above, we can see:

- There are high outliers in sales columns (NA, EU, JP, Other, Global) and User_Count columns.

They might be useful for training as they indicate bestseller games, but for now we're going to remove them.

The below function can be used to remove outliers present in the data set. A data entry is called an outlier if:

- value < Q1 - 3 * IQR
- value > Q3 + 3 * IQR

where,

- Q1 - First Quartile
- Q3 - Third Quartile
- IQR - Interquartile range

There are nearly half of the games which do not have scores. In ideal cases, we would like to drop these columns. But dropping over 8000+ entries is not possible in our case as it will heavily affect the models.

Therefore, we're going to build 2 models: a basic one and an advanced model. In a basic model we will drop games without a score (critic or user) and train it on the remaining data. We will also do minimum feature engineering or feature selection.

After we finish with the basic model, we are going to come back to the full dataset and try to input missing values and create new features.

Result:

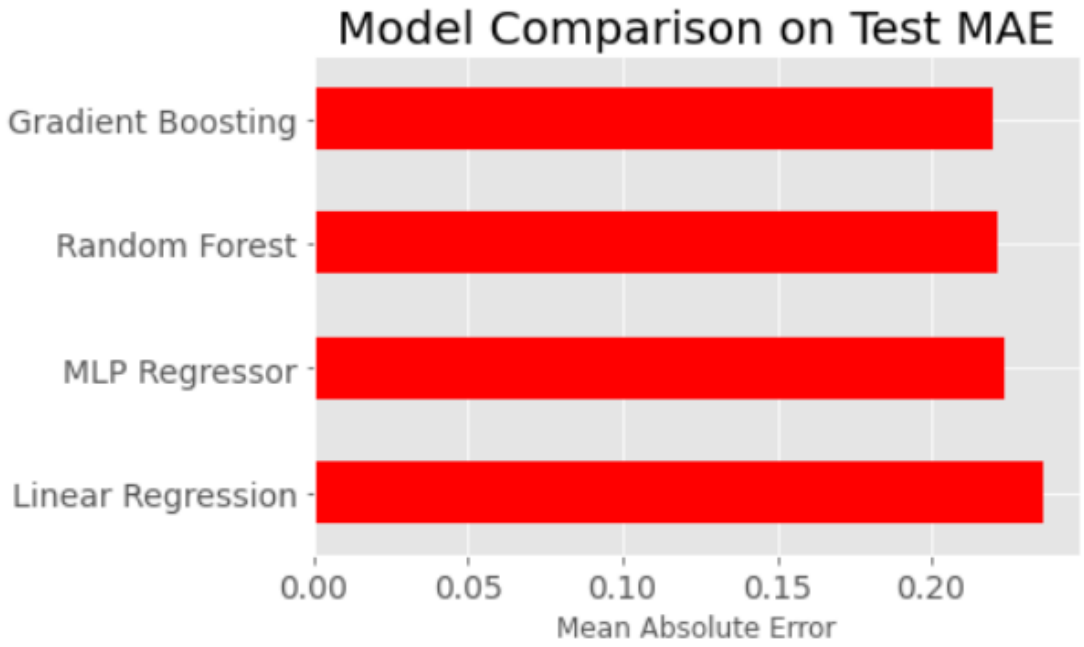
Basic Model:

In our basic model, we are going to drop games that don't have a user score, critic_score or rating. I will also remove outliers in the User_Count column. Only 5.5k games, about 1/3 games in a dataset remaining after doing the above steps.

We now applied one hot encoding to the categorical columns.

```
import category_encoders as ce

# Numeric columns
numeric_subset = scored.select_dtypes("number").drop(columns=["NA", "EU", "JP", "Other", "Year"])

# Categorical column
categorical_subset = scored[["Platform", "Genre", "Rating"]]

# One hot encoding
encoder = ce.one_hot.OneHotEncoder()
categorical_subset = encoder.fit_transform(categorical_subset)

# Column binding to the previos numeric dataset
features = pd.concat([numeric_subset, categorical_subset], axis = 1)

# Find correlations with the score
correlations = features.corr()["Global"].dropna().sort_values()
```

The data was split into training (80%) and testing (20%) sets. For each of the machine learning models mentioned above we calculated the mean absolute error and the respective results are shown below:



From the graph above, we can see that Gradient Boosting is the best model among all the machine learning we've chosen since it has the smallest mean absolute error which is 0.2197. Then, we now fine tune the hyperparameter so that the mean absolute error is as small as possible.

Hyperparameters tuning:

Below are the selected parameters we're using for the tuning.

```
hyperparameter = {"loss": ["ls", "lad", "huber"],
                  "max_depth": [2, 3, 5, 10, 15],
                  "min_samples_leaf": [1, 2, 4, 6, 8],
                  "min_samples_split": [2, 4, 6, 10],
                  "max_features": ["auto", "sqrt", "log2", None]}
```

We're using the randomized search to find the results of the hyperparameter. And then use grid search to find optimal value of the n_estimators parameter which are shown below:

```
GradientBoostingRegressor(loss='huber', max_depth=15, max_features='log2',
                          min_samples_leaf=8, min_samples_split=6,
                          random_state=42)
```

```
GradientBoostingRegressor(loss='huber', max_depth=15, max_features='log2',
                          min_samples_leaf=8, min_samples_split=6,
                          n_estimators=50, random_state=42)
```

Basic Model Result:

```
basic_final_model = grid_search.best_estimator_
basic_final_pred = basic_final_model.predict(X_test)
basic_final_mae = mae(Y_test, basic_final_pred)
print("Final model performance on the test set: MAE = {:.04f}.".format(basic_final_mae))
```

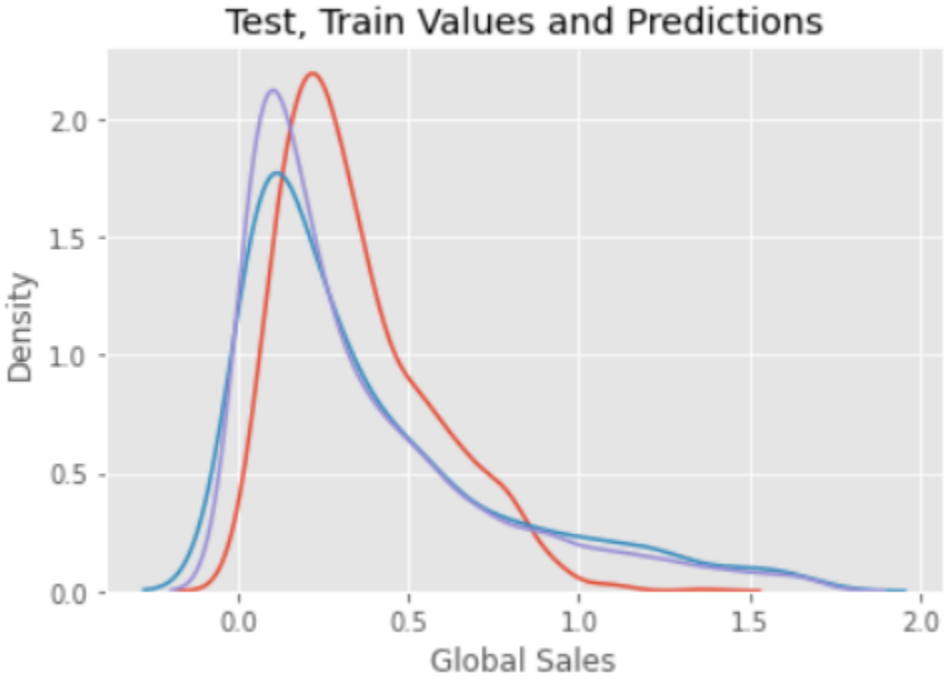
```
Final model performance on the test set: MAE = 0.2086.
```

We could now calculate the mean absolute error by this updated model. The MAE result dropped a little bit, not much(0.2086)! It seems like hyperparameter tuning didn't give a great improvement to the model.

From the graph above, we can know that the model is overfitting. When the test error almost stayed the same as the number of trees growing, the training error kept decreasing which means that the model learned the datasets too well to generalize it. This might be the reason why the MAE of the basic model was not improved a lot.

In the graph below, we made a plotting with comparison on densities of train values, test values and predictions. We can see that the predictions' density is moved a little to the right,

compared to densities of initial values and the tail is also different.



Advanced Model:

Since the improvement of the previous basic model was insignificant, we're going to find a way to overcome the overfitting problem. If we take a closer look at our data, we can find that there are too many different platforms and a lot of them contain very limited games. It is probably a great idea to group platforms in order to reduce the number of features. So, we created a new attribute called 'Group Platform' which groups all the famous platforms together into a single entity. For example, Playstation entity for all PS, PS2, PSP, PS3, etc. Next, we create some new features: weighted score and developer rating. First, we find percent of all games created by each developer, then calculate cumulative percent starting with devs with the least number of games. Finally divide them into 5 groups (20% each). Higher rank means more games developed.

Formula for Weighted score: ((User_Score * 10 * User_Count) + (Critic_Score * Cricti_Count))/ (User_Count + Critic_Count).

Before creating and fitting a model, we consider filling in missing values. We fill scores and counts with zeros since there were no real zero scores or counts in the dataset. So, it will indicate absence of scores.

Here is the overview of the data after modifying:

	Name	Platform	Year	Genre	Publisher	NA	EU	JP	Other	Global	...	Critic_Count	User_Score	User_Count	Developer	Rating	Age	Has_Score	Grouped_Platform	Weighted_Score	Developer_Rank
1058	Cabela's Big Game Hunter 2010	Wii	2009	Sports	Activision Value	1.58	0.00	0.00	0.12	1.69	...	0.0	7.8	4.0	Activision	T	9	False	Nintendo	78.000000	6.0
1059	SOCOM 3: U.S. Navy SEALs	PS2	2005	Shooter	Sony Computer Entertainment	1.22	0.34	0.04	0.10	1.69	...	59.0	8.8	64.0	Zipper Interactive	M	13	True	Playstation	85.121951	4.0
1061	Jampack Winter '99	PS	1999	Misc	Sony Computer Entertainment	0.94	0.64	0.00	0.11	1.69	...	0.0	0.0	0.0	NaN	None	19	False	Playstation	0.000000	0.0
1062	Call of Duty: Black Ops 3	PS3	2015	Shooter	Activision	0.49	0.87	0.07	0.26	1.69	...	0.0	0.0	0.0	NaN	None	3	False	Playstation	0.000000	0.0
1063	WCW vs. nWo: World Tour	N64	1997	Fighting	THQ	1.37	0.28	0.03	0.02	1.69	...	0.0	0.0	0.0	NaN	None	21	False	Other	0.000000	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
16714	Samurai Warriors: Sanada Maru	PS3	2016	Action	Tecmo Koei	0.00	0.00	0.01	0.00	0.01	...	0.0	0.0	0.0	NaN	None	2	False	Playstation	0.000000	0.0
16715	LMA Manager 2007	X360	2006	Sports	Codemasters	0.00	0.01	0.00	0.00	0.01	...	0.0	0.0	0.0	NaN	None	12	False	Xbox	0.000000	0.0
16716	Haitaka no Psychedelica	PSV	2016	Adventure	Idea Factory	0.00	0.00	0.01	0.00	0.01	...	0.0	0.0	0.0	NaN	None	2	False	Portable	0.000000	0.0
16717	Spirits & Spells	GBA	2003	Platform	Wanadoo	0.01	0.00	0.00	0.00	0.01	...	0.0	0.0	0.0	NaN	None	15	False	Portable	0.000000	0.0
16718	Winning Post 8 2016	PSV	2016	Simulation	Tecmo Koei	0.00	0.00	0.01	0.00	0.01	...	0.0	0.0	0.0	NaN	None	2	False	Portable	0.000000	0.0

Now, we will follow the same procedure after Encoding in the basic model and fit various features in the dataset and then finally calculate the updated error value based on the parameters. For the encoding, we will do the same things as we did in the basic model, except for using Ordinal encoding for categorical values instead of OneHot(see below).

```
# Select the numeric columns
numeric_subset = data.select_dtypes("number").drop(columns=["NA", "EU", "JP", "Other", "Year"])

# Select the categorical columns
categorical_subset = data[["Grouped_Platform", "Genre", "Rating"]]

mapping = []
for cat in categorical_subset.columns:
    tmp = scored.groupby(cat).median()["Weighted_Score"]
    mapping.append({"col": cat, "mapping": [x for x in np.argsort(tmp).items()]})

encoder = ce.ordinal.OrdinalEncoder()
categorical_subset = encoder.fit_transform(categorical_subset, mapping=mapping)

# Join the two dataframes using concat. Axis = 1 -> Column bind
features = pd.concat([numeric_subset, categorical_subset], axis = 1)

# Find correlations with the score
correlations = features.corr()["Global"].dropna().sort_values()
```

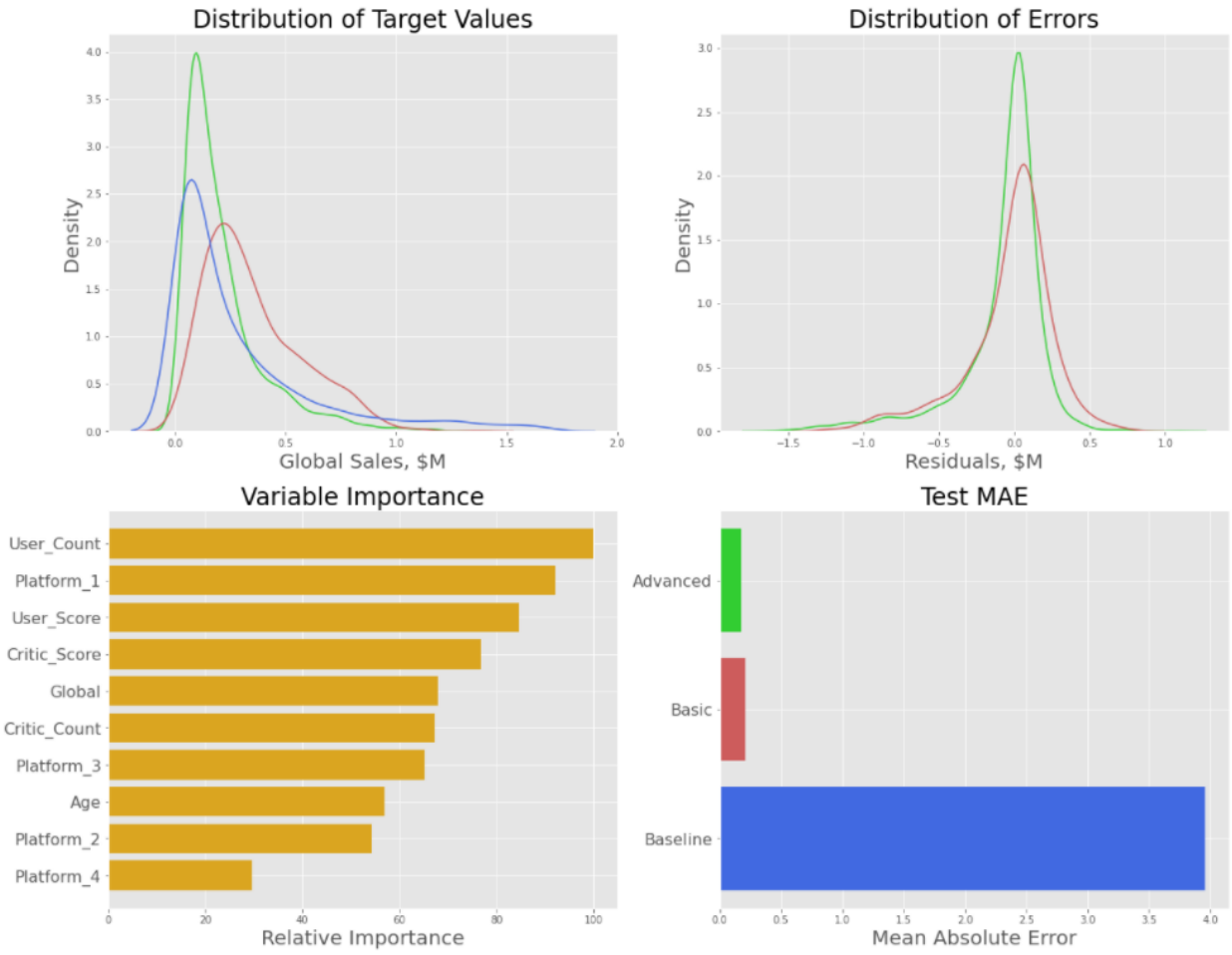
Advanced Model Result:

```
# Getting the final model error
final_model = grid_search.best_estimator_
final_pred = final_model.predict(features_test)
final_mae = mae(target_test, final_pred)
print("Final model performance on the test set: MAE = {:.04f}.".format(final_mae))
```

```
Final model performance on the test set: MAE = 0.1753.
```

From the above result, we know that the mean absolute error of the advanced model is 0.1753 which is lower than the basic model MAE 0.2086. Hence, we can say that the advanced model is a much better and preferable model over the other models.

Summarise:



From the summary above, we're now able to answer the problem we proposed.

The most important features affecting the model are Age, Weighted Score, User Count, Critic Count, User Score and Critic Score.

And, the best machine learning algorithm suitable for data is the Gradient Boosting model which gives the minimum mean absolute error 0.1753.

Conclusion:

After running and predicting all the dataset, we are able to perform the exploratory data analysis and gain valuable inferences. And then we are able to further eoncode and try different machine learning algorithms and find the best prediction to create different models on the dataset. And hence we are able to reduce the error respectively and get the final accuracy of 0.1753. After all these steps, we are able to answer our problems. And the answer is, we will choose gradient descent to predict the global sales of video games since it has the smallest error value.

Also, we learn from this project is, Linear Regression is the most efficient since the time elapsed (in seconds) is 0.0196. The second is gradient boosting, it takes 0.67265 seconds to run. The third is random forest which takes 2.8064 seconds to run and the last is MLP which takes 3.3388. So in conclusion, we learn that Gradient Boosting provides the best result with the least error value and Linear Regression provides the most efficient result.

From the above results, we get that which algorithm is important for video game companies, by knowing which feature fits the best, video companies can get what benefit the most and which type of video game they should put the most effort into.

Even though this project provides some benefit for video game companies, there are still some open questions that should be discussed in the future. Since that Model 1 was overfitting the data, in the future, how do we remove the overfitting condition ? And for Model 2, how do we further reduce the error into smaller numbers based on Gradient Boosting ? How do we select a better training dataset to improve the performance of MLP or other algorithms? What other algorithms give better prediction accuracy? There is lots of work we need to do in the future.

We think that this project is not only interesting but also allows us to learn a lot. On one hand, we strengthened our skills in using python to do data mining and machine learning. On the other hand, we learned how to cooperate with each other and at the end of this quarter, we are so glad that we have known each other and finished a project together.

Reference:

1. Kaggle(https://www.kaggle.com/rush4ratio/video-game-sales-with-ratings)
2. Morris, C. “Why Video Games Are Surging in Popularity. AARP” .2021, April 20
3. Hoare, Jake. “Gradient Boosting Explained - The Coolest Kid on The Machine Learning Block”. Displayr. 2020, December 7.
4. “Report.” *Scribd*, Scribd, www.scribd.com/document/510251646/report.
5. Yiu, Tony. Understanding Random Forest. Medium.2019, August 14.