BUAN 6341

THE UNIVERSITY OF TEXAS AT DALLAS

APPLIED MACHINE LEARNING

SALES PREDICTION FOR VIDEO GAME COMPANIES

By,

VISHAL KANNA NATARAJAN MANOHAR

VINAY DINESH BHANUSHALI

SRIRAM KOUSHIK MAJETI

INTRODUCTION

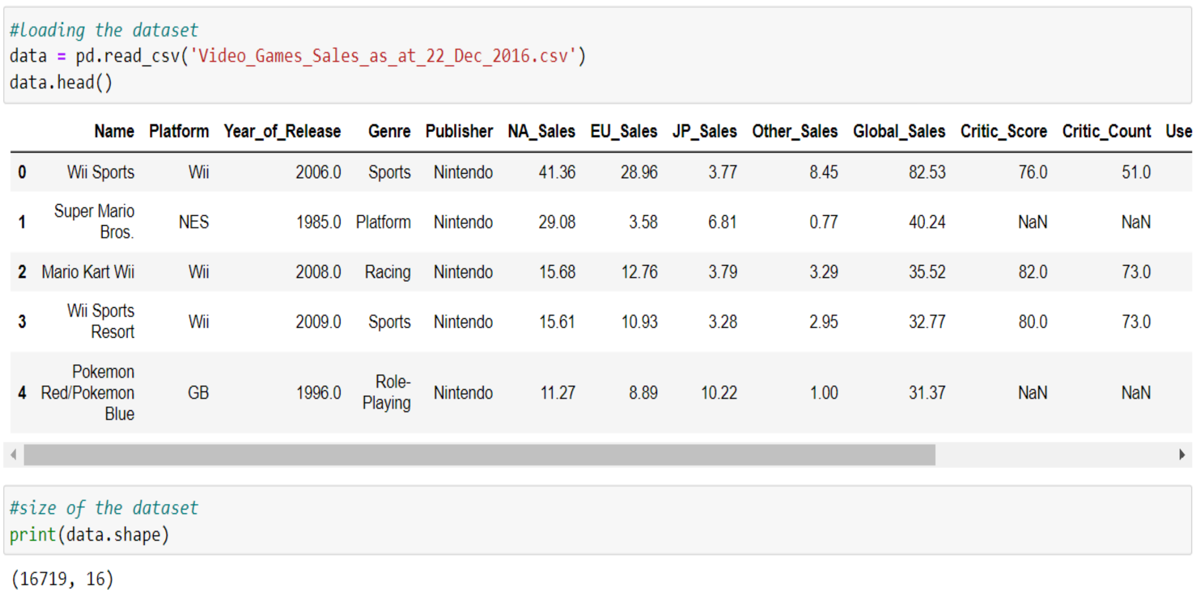
The video game industry is a highly competitive and dynamic market, where new games are released every day, and the demand for the latest and greatest games can change rapidly. Video game companies need to have a good understanding of market trends and consumer preferences to make informed decisions about which games to develop, when to release them, and how to price them. Machine learning can help video game companies to forecast video game sales accurately and make data-driven decisions.

This project aims to build a predictive model to forecast video game sales for video game companies. We will use machine learning techniques to analyse historical data on video game sales, including factors such as genre, platform, release date, and user ratings. The dataset used for this project will be sourced from Kaggle and will include data on video game sales from different regions. The project will involve data pre-processing, exploratory data analysis, feature engineering, model selection, model training, and evaluation.

The final outcome of the project will be a machine learning model that can predict the sales of a new video game based on its features, providing valuable insights to video game companies for better decision-making.

EXPLORATORY DATA ANALYSIS

Data Loading:



```python
#loading the dataset
data = pd.read_csv('Video_Games_Sales_as_at_22_Dec_2016.csv')
data.head()
```

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales	JP_Sales	Other_Sales	Global_Sales	Critic_Score	Critic_Count	Use
0	Wii Sports	Wii	2006.0	Sports	Nintendo	41.36	28.96	3.77	8.45	82.53	76.0	51.0	
1	Super Mario Bros.	NES	1985.0	Platform	Nintendo	29.08	3.58	6.81	0.77	40.24	NaN	NaN	
2	Mario Kart Wii	Wii	2008.0	Racing	Nintendo	15.68	12.76	3.79	3.29	35.52	82.0	73.0	
3	Wii Sports Resort	Wii	2009.0	Sports	Nintendo	15.61	10.93	3.28	2.95	32.77	80.0	73.0	
4	Pokemon Red/Pokemon Blue	GB	1996.0	Role-Playing	Nintendo	11.27	8.89	10.22	1.00	31.37	NaN	NaN	

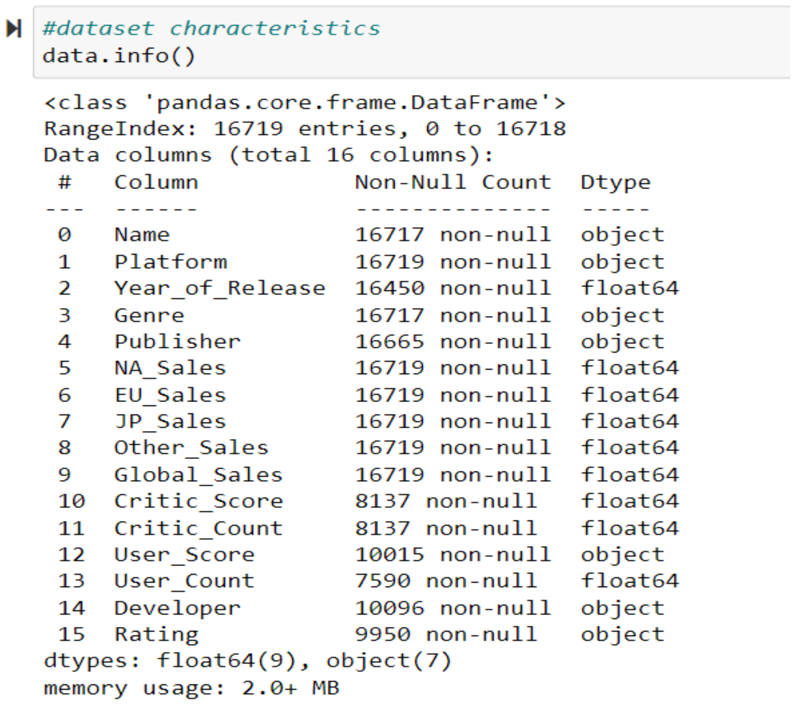
```python
#size of the dataset
print(data.shape)
```

```
(16719, 16)
```

Data was loaded and previewed. In previewing the head of the data, we had 16 columns in total. As the video game industry has grown rapidly, there may be significant time variance if we want to look at all consoles from each generation.

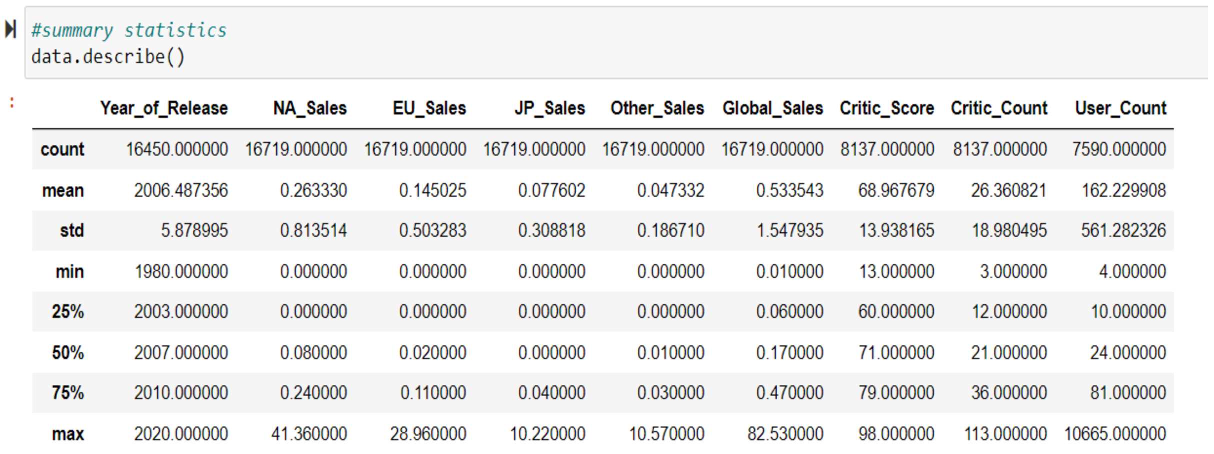
As the video game industry is global, let's have global sales as the dependent variable. This means later on we can drop all other sales features. Before we decide on independent variables, we will perform some data exploration activities.

Dataset Characteristics & Summary Statistics:



```python
#dataset characteristics
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16719 entries, 0 to 16718
Data columns (total 16 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Name             16717 non-null  object 
 1   Platform         16719 non-null  object 
 2   Year_of_Release  16450 non-null  float64
 3   Genre            16717 non-null  object 
 4   Publisher        16665 non-null  object 
 5   NA_Sales         16719 non-null  float64
 6   EU_Sales         16719 non-null  float64
 7   JP_Sales         16719 non-null  float64
 8   Other_Sales      16719 non-null  float64
 9   Global_Sales     16719 non-null  float64
 10  Critic_Score     8137 non-null   float64
 11  Critic_Count     8137 non-null   float64
 12  User_Score       10015 non-null  object 
 13  User_Count       7590 non-null   float64
 14  Developer        10096 non-null  object 
 15  Rating           9950 non-null   object 
dtypes: float64(9), object(7)
memory usage: 2.0+ MB
```

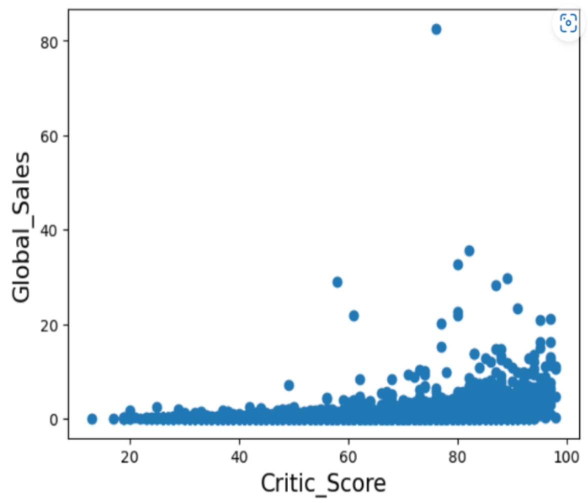


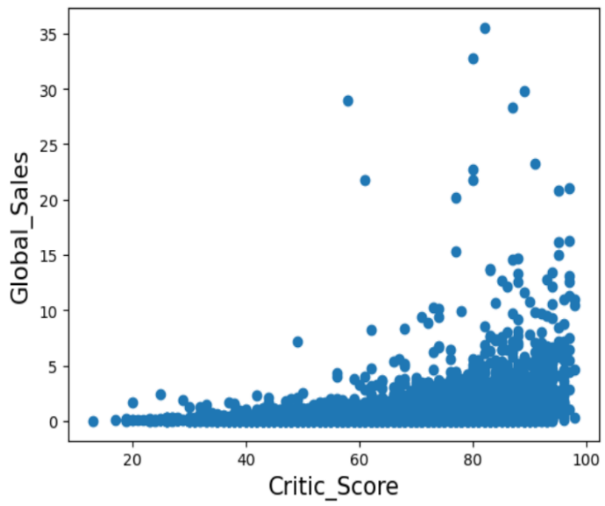
```
#summary statistics
data.describe()
```

	Year_of_Release	NA_Sales	EU_Sales	JP_Sales	Other_Sales	Global_Sales	Critic_Score	Critic_Count	User_Count
count	16450.000000	16719.000000	16719.000000	16719.000000	16719.000000	16719.000000	8137.000000	8137.000000	7590.000000
mean	2006.487356	0.263330	0.145025	0.077602	0.047332	0.533543	68.967679	26.360821	162.229908
std	5.878995	0.813514	0.503283	0.308818	0.186710	1.547935	13.938165	18.980495	561.282326
min	1980.000000	0.000000	0.000000	0.000000	0.000000	0.010000	13.000000	3.000000	4.000000
25%	2003.000000	0.000000	0.000000	0.000000	0.000000	0.060000	60.000000	12.000000	10.000000
50%	2007.000000	0.080000	0.020000	0.000000	0.010000	0.170000	71.000000	21.000000	24.000000
75%	2010.000000	0.240000	0.110000	0.040000	0.030000	0.470000	79.000000	36.000000	81.000000
max	2020.000000	41.360000	28.960000	10.220000	10.570000	82.530000	98.000000	113.000000	10665.000000

Using the above methods, we can see that it will return the complete details about our dataset behaviour.

Feature Engineering:

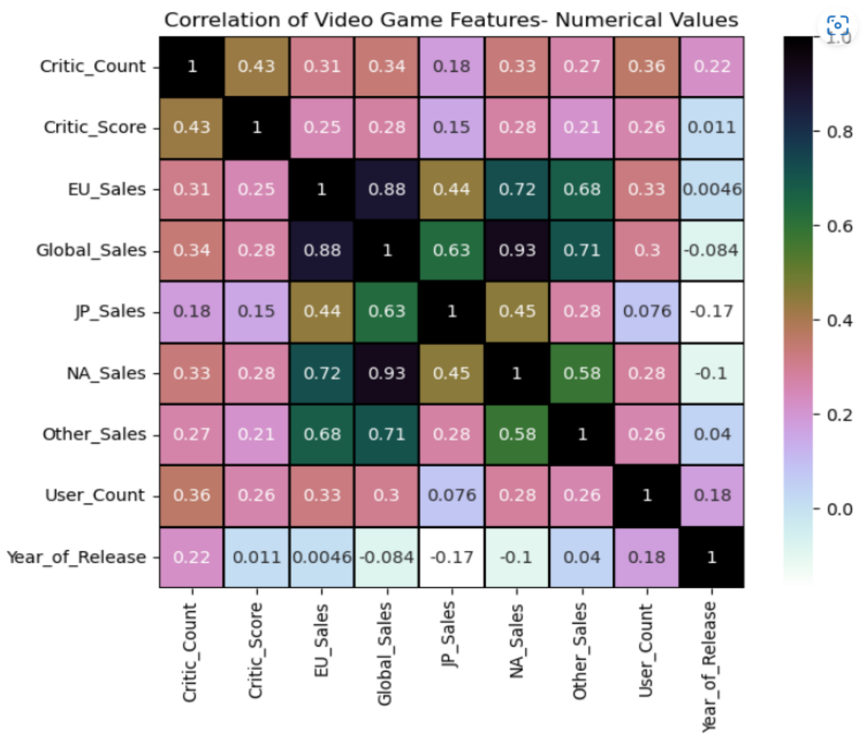




We looked for outliers among the most evident pair of dependent and independent variables. There is one obvious outlier in the sample, so we removed it. This brings us to the topic of "what constitutes an outlier?" When we examine the plot after outlier elimination, it appears that there are still outliers in the data. However, eliminating too many data points that appear to be outliers may constitute data tampering. So, we treated the dataset with the benefit of doubt.

Correlation Check:

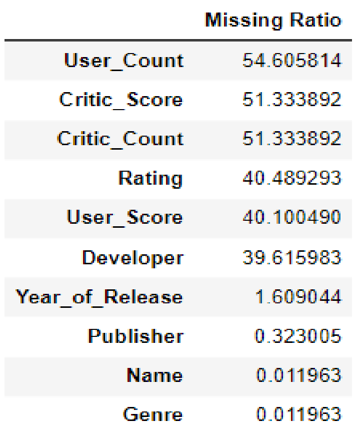
Before we split the data into training and test sets, we looked at some feature correlations to make sure our independent variables don't suffer from autocorrelation, which can be problematic in some linear models like liner regression if we want to look at feature importance through the intercepts.



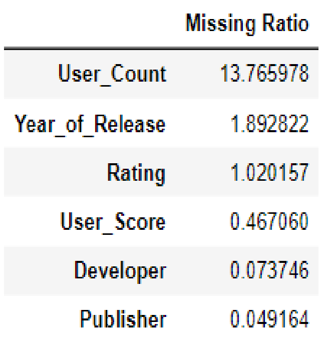
	Critic_Count	Critic_Score	EU_Sales	Global_Sales	JP_Sales	NA_Sales	Other_Sales	User_Count	Year_of_Release
Critic_Count	1	0.43	0.31	0.34	0.18	0.33	0.27	0.36	0.22
Critic_Score	0.43	1	0.25	0.28	0.15	0.28	0.21	0.26	0.011
EU_Sales	0.31	0.25	1	0.88	0.44	0.72	0.68	0.33	0.0046
Global_Sales	0.34	0.28	0.88	1	0.63	0.93	0.71	0.3	-0.084
JP_Sales	0.18	0.15	0.44	0.63	1	0.45	0.28	0.076	-0.17
NA_Sales	0.33	0.28	0.72	0.93	0.45	1	0.58	0.28	-0.1
Other_Sales	0.27	0.21	0.68	0.71	0.28	0.58	1	0.26	0.04
User_Count	0.36	0.26	0.33	0.3	0.076	0.28	0.26	1	0.18
Year_of_Release	0.22	0.011	0.0046	-0.084	-0.17	-0.1	0.04	0.18	1

The prospective independent variables are not overly associated with each other in the correlation heat map in terms of autocorrelation. Only sales figures are highly connected with one another. Higher correlations between sales data may reflect the worldwide character of the video game industry: success on one continent frequently translates to success on another. This increases the likelihood of utilizing only Global_Sales as the dependent variable.

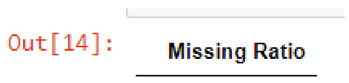
Data Cleaning:



	Missing Ratio
User_Count	54.605814
Critic_Score	51.333892
Critic_Count	51.333892
Rating	40.489293
User_Score	40.100490
Developer	39.615983
Year_of_Release	1.609044
Publisher	0.323005
Name	0.011963
Genre	0.011963



	Missing Ratio
User_Count	13.765978
Year_of_Release	1.892822
Rating	1.020157
User_Score	0.467060
Developer	0.073746
Publisher	0.049164



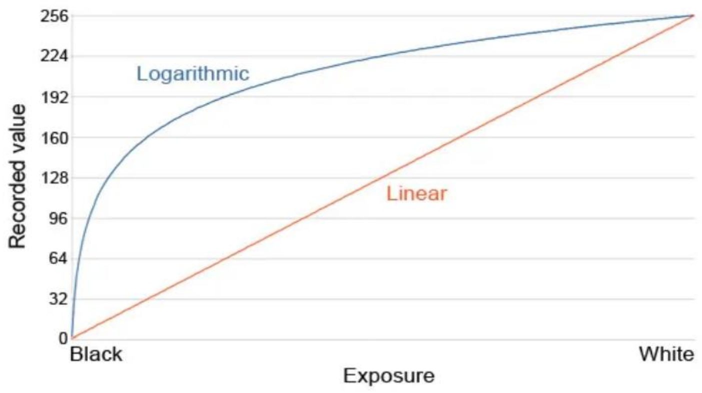
Out[14]:

Missing Ratio

We had a problem here. The most important independent variable, Critic_Score, comprised about 50% of its values as N/A. Similarly, nearly 40% of data points for User_Score, which was most likely one of the most important independent variables, were missing. This missing data ratio was so large that it cannot be filled with median values, per se. We therefore eliminated N/A observations. We've dealt with all the missing values finally.

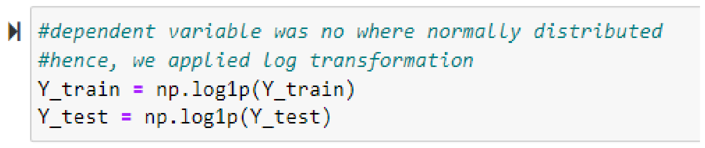
Data Scaling and Transformation:

Global_Sales was not typically distributed, which may have posed a concern for us in the future. Hence, we performed log transformation of the dependent variable in both the training and testing data splits.

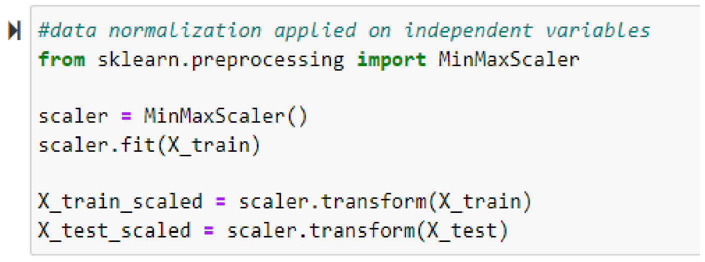


One motivation for log transformation is to standardize the data. When data is skewed, it is difficult to interpret or model because the extreme numbers can dominate the analysis. A log transformation compresses the data and spreads the values more equally across the range, making it easier to analyze or model.

Some models, such as linear regression, assume a linear relationship between the predictor and response variables. If the data contradicts this assumption, like in the case of a curved relationship, a log transformation can sometimes be used to make the relationship linear. The severe case is avoided by compressing the data with a log transformation, which can make the analysis more robust.



```
#dependent variable was no where normally distributed
#hence, we applied log transformation
Y_train = np.log1p(Y_train)
Y_test = np.log1p(Y_test)
```



```
#data normalization applied on independent variables
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
scaler.fit(X_train)

X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

STATISTICAL MODELS FOR REGRESSION

We used seven different Machine Learning algorithms to train the model in hand. The algorithms namely are, 'Linear Regression', 'Lasso Regression', 'Ridge Regression', 'KNN', 'SVM', 'Random Forest', 'Gradient Boosting'.

Performance Metrics

Linear Regression

```
Best parameters: {}
Best cross-validation score: 0.40
Test set score: 0.43
```

Lasso Regression

```
Best parameters: {'alpha': 0.0001, 'max_iter': 1000000}
Best cross-validation score: 0.40
Test set score: 0.43
```

Ridge Regression

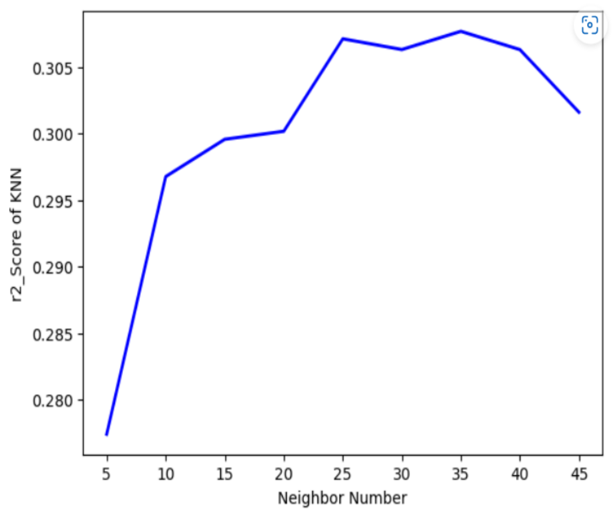
```
Best parameters: {'alpha': 10}
Best cross-validation score: 0.40
Test set score: 0.43
```

K- Nearest Neighbour

```
print(scores_list)
```

```
[0.2774048606835715, 0.29675431773412775, 0.29955567609936273, 0.300158822325738, 0.30710398039898756, 0.30629707209378343, 0.3
0766555184350786, 0.3062906325450926, 0.30158994552871277]
```

```
R2 score: 0.30766555184350786
Mean squared error: 0.13668497373652885
```



From the plot, it can be observed that when the number of neighbours is equal to 35, the r-squared score is optimal. Hence, we will be using the n_neighbors=35 parameter to train the KNN model.

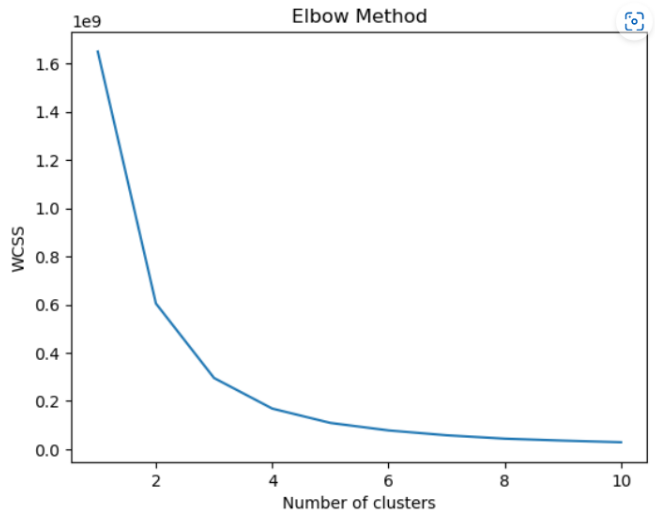
Support Vector Machine

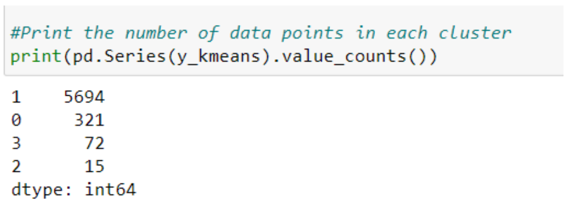
```
Best parameters: {'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}
Best cross-validation score: 0.42
Test set score: 0.42
```

Random Forest Classifier

```
Best parameters: {'max_depth': 9, 'max_features': 12, 'n_estimators': 50}
Best cross-validation score: 0.33
Test set score: 0.35
```

K- Means Clustering





```
#Print the number of data points in each cluster
print(pd.Series(y_kmeans).value_counts())
```

```
1    5694
0     321
3      72
2      15
dtype: int64
```

As a benefit of doubt, we trained this model as an experiment. K-means is a clustering technique that works with continuous numerical data and measures the distance between points using a Euclidean distance metric. However, because categorical data lacks a natural sense of distance, k-means is not directly applicable to categorical data. Hence, we will be discarding this model for further analysis.

Gradient Boosting

```
Best parameters: {'max_depth': 5, 'max_features': 10, 'n_estimators': 200}
Best cross-validation score: 0.31
Test set score: 0.33
```

Note

In GridSearchCV, `scoring='neg_mean_squared_error'` is a parameter that can be used when evaluating the performance of a regression model using cross-validation in Scikit-learn, a popular Python library for machine learning.

The parameter `scoring` specifies the metric to use for evaluating the performance of the model. In this case, `neg_mean_squared_error` is a score function that calculates the negative mean squared error of the predicted values compared to the true values.

The negative sign is used because Scikit-learn expects a score function to maximize, while the mean squared error should be minimized. Therefore, the negative of the mean squared error is used to obtain a score function that can be maximized during the cross-validation process.

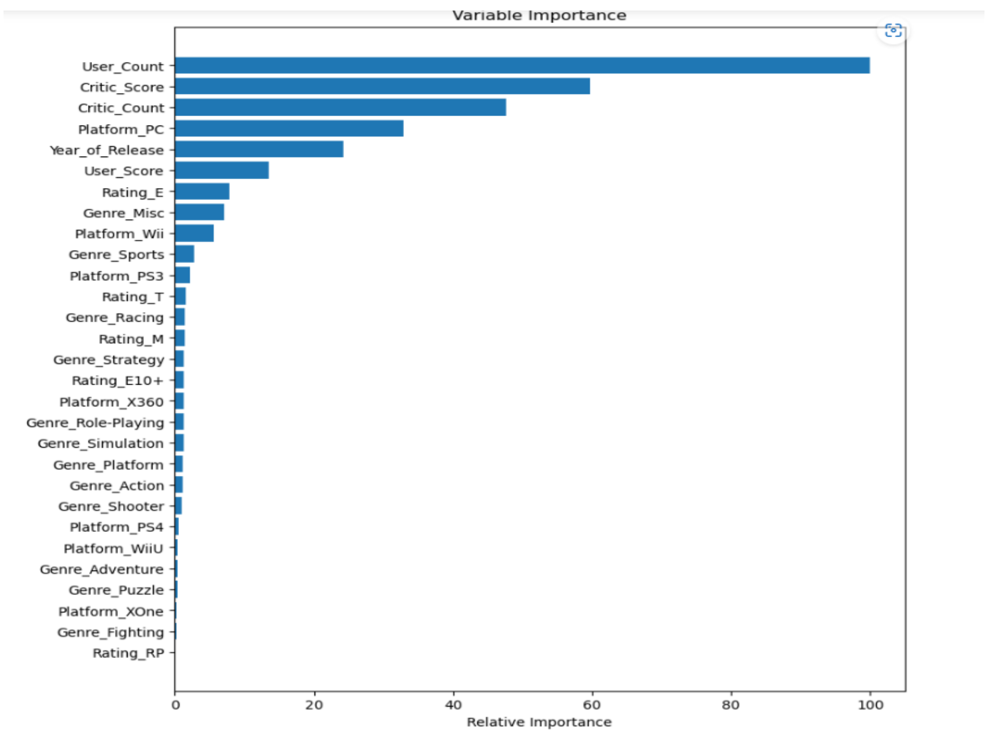
By using this parameter in the cross-validation process, Scikit-learn will calculate the negative mean squared error score for each fold and then return the average score across all the folds. This allows for a quantitative evaluation of the performance of the regression model and can be used to compare the performance of different models or different hyperparameter settings of the same model.

Interpretation

The negative mean squared error is a measure of the quality of a regression model's predictions, and it provides a quantitative measure of how well the model fits the data. The mean squared error (MSE) is a measure of the average squared difference between the predicted values and the true values. A lower MSE indicates a better fit of the model to the data, with smaller differences between the predicted and true values.

By taking the negative of the MSE, the score function returns a value that can be maximized rather than minimized, which is what Scikit-learn expects. Therefore, a higher negative mean squared error score indicates a better performance of the model. Therefore, it is important to compare the negative mean squared error scores of different models or different hyperparameter settings within the context of the specific dataset and problem at hand.

VARIABLE IMPORTANCE



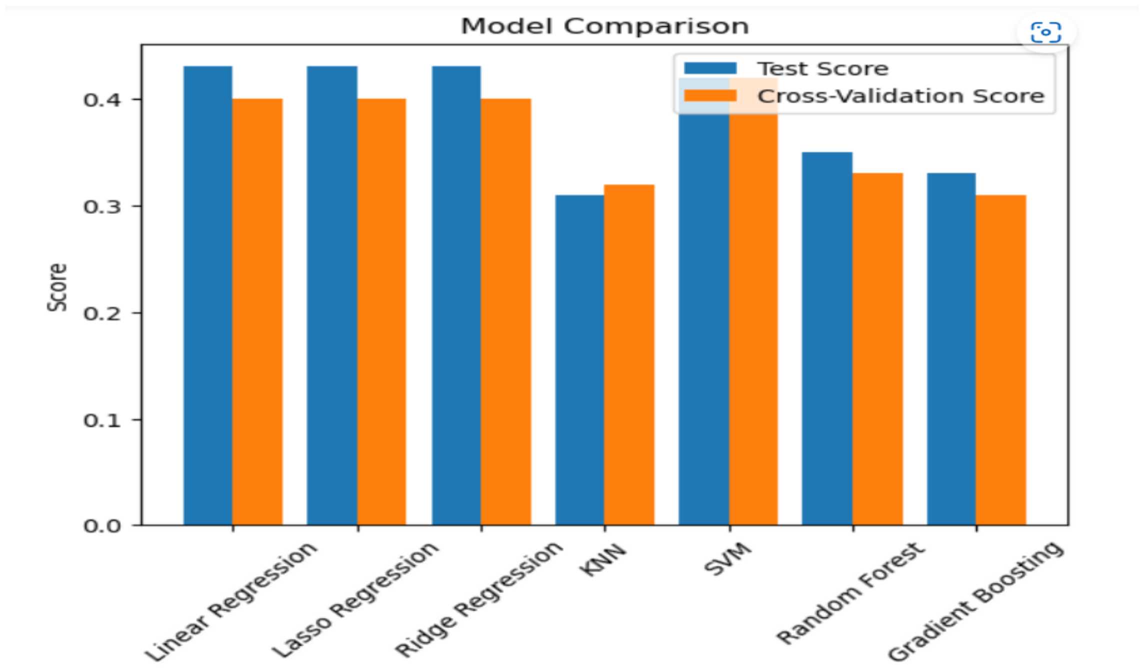
The SVR is the best model out of the seven we trained. Since 'SVR' object has no attribute 'feature_importances_', we considered the second best model- Random Forest for plotting relative variable importance. Moreover, we looked at the model's feature importance as well to see which features are most significant in explaining the outcomes. From the plot, it is evident that User_Count is the most significant variable among the alternatives.

CONCLUSION

Accuracy Comparison Analysis

On a retrospective note, we trained seven different machine learning models on a certain dataset and obtained their test and cross-validation scores. Here are some observations regarding the models and their performance:

The models used were Linear Regression, Lasso Regression, Ridge Regression, KNN, SVM, Random Forest, and Gradient Boosting. Each of these models has its own strengths and weaknesses, and choosing the best one for a given problem requires careful consideration of the data and the task at hand.



The test and cross-validation scores for the three regression models (Linear, Lasso, and Ridge) are all very similar, with a score of 0.43 for the test set and 0.40 for cross-validation. This suggests that these models are not overfitting to the training data, but they may not be capturing all the relevant information in the data either.

The KNN model has a test score of 0.31, which is the lowest among all the models, but its cross-validation score is 0.32, indicating that it is not overfitting to the training data. KNN is known to be sensitive to the choice of hyperparameters, such as the number of neighbors to consider, so tuning these hyperparameters may improve its performance.

The SVM model has a test score of 0.42, which is the second highest among all the models, and its cross-validation score is also high at 0.42. This suggests that SVM is a good choice for this particular problem.

The Random Forest and Gradient Boosting models have test scores of 0.35 and 0.33, respectively, which are both lower than the scores of Linear, Lasso, and Ridge regression models. However, their cross-validation scores are also lower, indicating that they may be overfitting to the training data. Tuning the hyperparameters of these models may help to improve their performance.

In summary, *Support Vector Regressor Model* is the best-performing model among the seven models that we trained, with a test score of 0.42 and a cross-validation score of 0.42. However, it is important to note that this conclusion may change if more information about the data and the task at hand is provided.