Analyzing optimal video game playing conditions

Team GOAT

Keya Shukla (ks6875), Kritik Seth (kls8193), Mary Nikitha Kasipati (mk8463),
Tayyibah Khanam (tk2981), Umair Ayub (ua2057)

1 Introduction

First-person shooter (FPS) games have seen a significant rise in popularity over the past decade. The proliferation of fast internet speeds, online gaming services, growth in the video game industry and technological advancements has made it easier for players to connect and compete with one another in real-time, boosting the popularity of FPS games. The rise of e-sports and competitive gaming has also played a role in the growth of FPS games. Many FPS games, such as Overwatch, Counter-Strike: Global Offensive, and Call of Duty, have professional leagues and tournaments with large prize pools, attracting top players and sponsorships from major brands. The popularity of these e-sport events has helped to draw more attention to FPS games and has contributed to the overall growth of the genre.

The advancement of technology and the increased accessibility of high-quality hardware and software has helped improve the achieved frames per second in different video games over the years. One of the key technological advancements that has contributed to the increase of FPS in games is the increase in processing power. Modern computers and gaming consoles are equipped with powerful CPUs that are capable of handling the complex calculations required for high-quality graphics and realistic physics in FPS games. In addition to processing power, the improvement in graphics card technology has also played a role in the increase of FPS in games. Modern graphics cards are capable of rendering high-quality graphics at high frame rates, which is essential for smooth and responsive gameplay in FPS games. The increase in graphics card performance has enabled developers to create more detailed and realistic environments, as well as more complex character models and special effects, without compromising on the FPS. Combined, the technological advancements in CPU and GPU industry have made it possible for a gamer to play their desired games at higher frame rates than before.

2 Data Description

The original dataset was downloaded from openml.org and has 425,833 rows and 46 columns, sourced from [1]. Each row in this dataset contains the measurement of frames per second achieved for a particular video game on different computers. A computer is characterized by the CPU and GPU, and the dataset contains the technical specifications for each component present in that computer in its factory state. Each game is characterized by name, resolution and the settings the game was running on during the testing. Following are the column names with short descriptions -

1. ID - Unique ID for each computer
2. CPU Name - Name of CPU *(categorical)*
3. CPU Cores - Number of cores in CPU *(continuous)*
4. CPU Threads - Number of threads in CPU *(continuous)*
5. CPU Base Clock - Base clock of CPU *(continuous)*
6. CPU Cache L1 - Type L1 memory built into microprocessor, storing its recently accessed information *(continuous)*
7. CPU Cache L2 - Type L2 memory built into microprocessor, storing its recently accessed information *(continuous)*
8. CPU Cache L3 - Type L3 memory built into microprocessor, storing its recently accessed information *(continuous)*
9. CPU Die Size - Physical dimensions (area) of a bare integrated circuit wafer *(continuous)*
10. CPU Frequency - Number of operations per second
11. CPU Multiplier - Ratio of an internal clock rate to the externally supplied clock in a CPU *(continuous)*
12. CPU Multiplier Unlocked - Denotes if a user is allowed to manually overclock the CPU (1- Unlocked, 0- Locked, *(categorical)*)
13. CPU Process Size - Fabrication process used to embed transistors on silicon wafer of chip *(continuous)*
14. CPU TDP - Maximum heat a CPU can use in watts *(continuous)*
15. CPU Transistors - Number of transistors on silicon wafer of chip *(continuous)*
16. CPU Turbo Clock - Maximum clock of CPU *(continuous)*
17. GPU Name - Name of GPU *(categorical)*
18. GPU Architecture - Design architecture which GPU was made using *(categorical)*

19. GPU Bandwidth - Data transfer speed across GPU and System via Bus *(continuous)*
20. GPU Base Clock - Base clock of CPU *(continuous)*
21. GPU Boost Clock - Maximum clock of GPU *(continuous)*
22. GPU Bus - Memory Interface that connects computing unit to memory unit *(categorical)*
23. GPU Compute Units - Number of processing units in a GPU *(continuous)*
24. GPU Die Size - Physical dimensions (area) of a bare integrated circuit wafer *(continuous)*
25. GPU DirectX - Version of application handling interface for handling multimedia tasks (categorical)
26. GPU Execution Units - Programmable shader units in a GPU *(continuous)*
27. GPU FP32 Performance - "single precision" is a term for a floating point format which occupies 32 bits in computer memory and has a precision between 7 and 8 valid digits *(continuous)*
28. GPU Memory Bus - Memory bandwidth, the rate at which the data can be read or stored into a semiconductor memory by a processor *(continuous)*
29. GPU Memory Size - Type of random access memory, specifically used to store image data for a computer display *(continuous)*
30. GPU Memory Type - Generation of random access memory in GPU *(categorical)*
31. GPU OpenCL - Generation of OpenCL framework that performs computational operations *(categorical)*
32. GPU OpenGL - Generation of OpenGL framework that performs graphical operations *(categorical)*
33. GPU Pixel Rate - Number of pixels graphics processor could write on a video memory (continuous)
34. GPU Process Size - Fabrication process used to embed transistors on silicon wafer of chip (continuous)
35. GPU Number of ROPs - Number of render output units in GPU pipeline *(continuous)*
36. GPU Shader Model - Version of software in GPU that manipulated an image before it is drawn on the screen *(categorical)*
37. GPU Shading Units - Number of small processing units in GPU that are responsible for processing different aspects of image *(continuous)*
38. GPU TMUs - Number of texture mapping units in GPU pipeline *(continuous)*
39. GPU Texture Rate - Pixels rendered per second *(continuous)*
40. GPU Transistors - Number of transistors on silicon wafer of chip *(continuous)*
41. GPU Vulkan - Version of compute API for GPU *(categorical)*
42. Game - Name of game *(categorical)*
43. Game Resolution - Resolution game was played on *(categorical)*
44. Game Settings - Settings that game was played on *(categorical)*
45. Dataset - Dataset where the FPS data was sourced from *(categorical)*
46. FPS - Frames per second achieved *(continuous)*

3 Data Preparation

Figure 1 gives an idea of total missing values in each column before data pre-processing.

3.1 Feature engineering

Assumption: Naming conventions for different CPU Models, Series and Generations follow from what has been used by that company before (example if SKU ID for one of the 2017 Intel CPUs in our data is Intel Core i7-8086K where the Core is Series, the Model is i7, the Generation is 8 and the Type is K, than the SKU ID for next generation of product will be Intel Core i7-9xxxK)

Five new columns were engineered based on domain knowledge:

1. CPU Brand (the company which made the CPU)
2. CPU Mode (the model name in a company's product lineup)
3. CPU Series (class of model in company's product lineup)
4. CPU Generation (indication of year of release for that class of product in a company's product lineup)
5. CPU Type (indication of type of CPU- unlocked / low power / high performance and so on)

3.2 Data Imputation

- Since in our inference we failed to reject our null hypothesis that Moore's Law holds true, we can use Moore's Law to impute the missing values in CPU Transistors column.
- Geometric Sequence was used to impute missing Transistor values for CPUs in our dataset. We first grouped the dataset by CPU Brand, CPU Series, CPU Model, and CPU Generation using mean, and then imputed the missing Transistor values using a sequence that doubled every 2 years. This allowed us to estimate the number of transistors in the missing places based on the trend of increasing transistor count over time.

	Missing Values	% of Total Values
GPU Execution Units	412935	97.0
GPU Compute Units	357107	83.9
CPU Transistors	226039	53.1
CPU Die Size	202534	47.6
GPU Memory Bus	15441	3.6
GPU Bandwidth	15441	3.6
GPU Memory Size	15441	3.6
GPU Memory Type	15441	3.6
GPU Transistors	11539	2.7
GPU Vulkan	11525	2.7
GPU Die Size	11196	2.6
CPU Cache L3	5055	1.2
GPU Open CL	177	0.0
GPU Shading Units	56	0.0
GPU FP32 Performance	56	0.0
GPU Shader Model	5	0.0
CPU Generation	3	0.0

Figure 1: Analysis of missing values

- Linear Regression was used to impute missing Die Size values for CPUs in our dataset. We first grouped the dataset by CPU Brand, CPU Series, CPU Model, and CPU Generation using mean, and then trained linear models on the data of different generations for a particular CPU Brand, Series, and Model. These models were used to impute the missing Die Size values in the dataset.

3.3 Data Cleaning

- Trimmed white spaces in CPU and GPU Names.
- Standardized CPU Naming convention (using regex) in order to enable feature engineering and specification comparison between SKUs of CPU.
- Dropped columns with greater than 50% NA values because any sort of imputation performed on these columns will reduce confidence level in results.
- Identified conflicts in the dataset (same features but different outputs).
- Corrected the conflicts in the dataset by scraping information from the internet and deleting non relevant rows.
- Dropped duplicate rows in the dataset.
- Converted continuous columns to float and categorical columns to object.

4 Exploratory Data Analysis

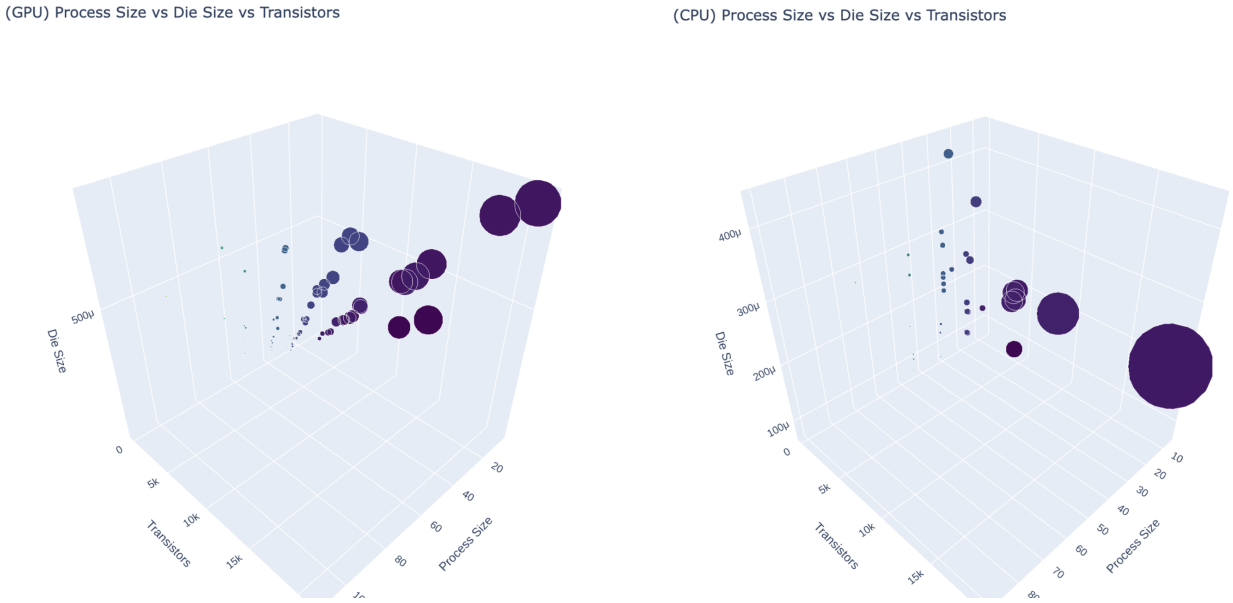
4.1 Die Size vs Process Size vs Transistors

- The process size refers to the width of the transistors on the CPU/GPU chip. Transistors are tiny electronic switches that are used to perform calculations and store data in a computer. The smaller the process size, the more transistors can be placed on the CPU/GPU chip, which can result in improved performance.

- The die size refers to the size of the entire CPU/GPU chip, including the transistors and other components. The larger the die size, the more transistors can be included on the chip, which can also result in improved performance.

- The number of transistors on a CPU/GPU chip can also affect its performance. A CPU/GPU chip with more transistors can perform more calculations simultaneously, which can lead to faster processing speeds. However, adding more transistors also increases the size of the chip and the power it consumes, which can be a trade-off for some applications.

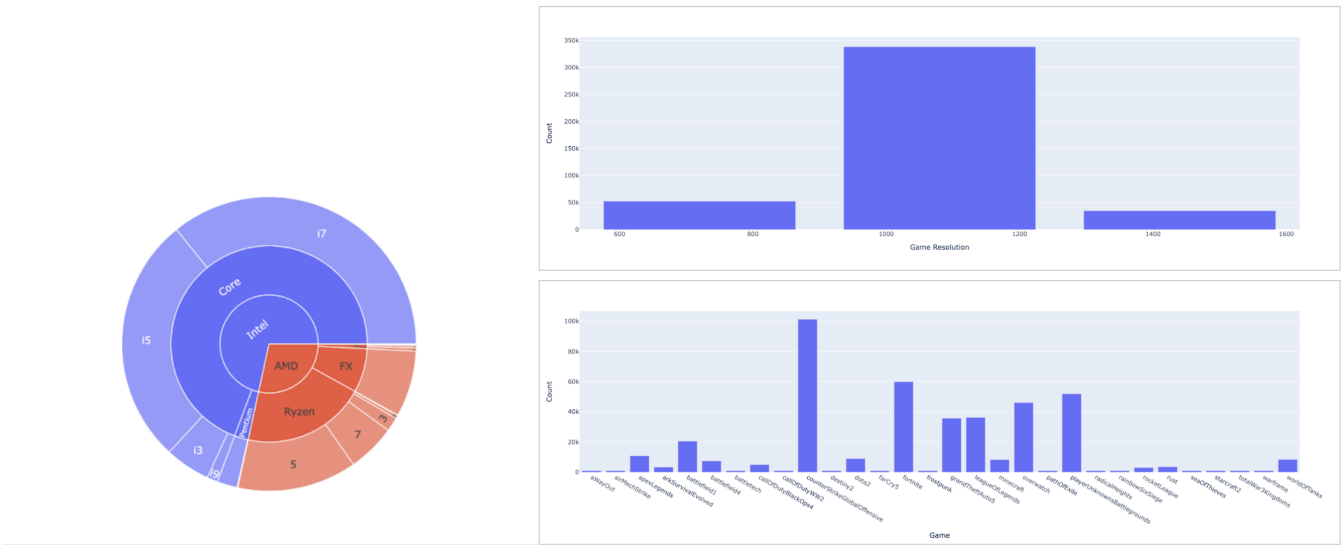
- The process size, die size, and number of transistors on a chip are all factors that can impact the performance of the CPU/GPU. Generally, chips with smaller process sizes, smaller die sizes, and more transistors tend to have better performance, but there are other factors that can also affect a CPU/GPU's performance.
- From figure 2a we observe that when the Die Size in GPU increases, the process size has decreased but the number of transistors on the chip have increased, which is in line with what we are expecting. Size of the circle in this 3D scatter plot is an indication of the number of transistors, larger the circle, more the number of transistors. Color of the circle in this 3D scatter plot is an indication of process size, blue-ish shade indicates larger process size and purple-ish shade indicates smaller process size.



(a) GPU 3D visuals (b) CPU 3D visuals

Figure 2: Process size VS die size VS number of transistors

- From figure 2b we observe that when the Die Size in CPU increases, the process size has generally decreased but the number of transistors on the chip have increased, which is in line with what we are expecting. Size of the circle in this 3D scatter plot is an indication of the number of transistors, larger the circle, more the number of transistors. Color of the circle in this 3D scatter plot is an indication of process size, blue-ish shade indicates larger process size and purple-ish shade indicates smaller process size.



(a) Pie chart with CPU Brands (b) Bar charts of games played and game resolution

Figure 3: Quantifying important data columns

- From figure 3a we can see that there are more Intel CPUs in our database as compared to AMD CPUs. This is a sunburst chart indicating the ratio of CPU Brand, CPU Series and CPU Models in our dataset.
- As we can see in figure 3b most people prefer to play their games at a resolution of around 1000-1200 pixels per inch.
- We also plot correlation for our entire dataset in figure 4.

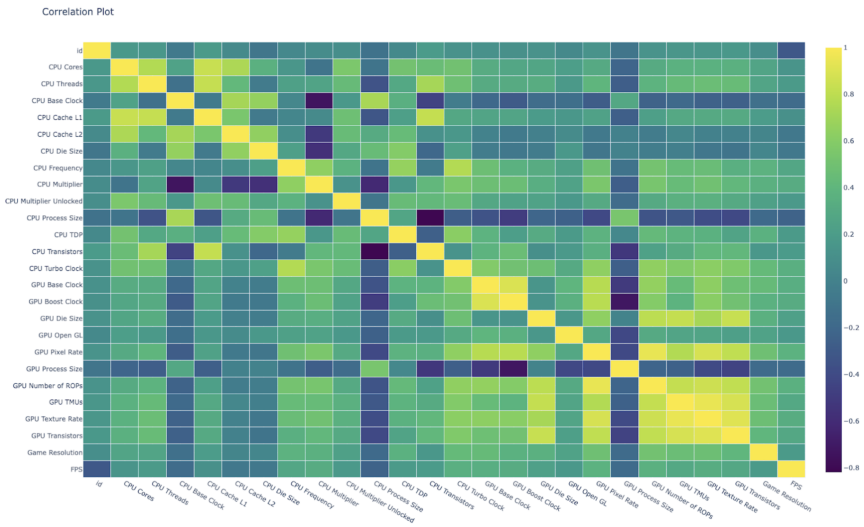


Figure 4: Correlation between feature columns

5 Hypothesis Testing

5.1 Goal

Moore's law is the observation that the number of transistors in a dense Integrated circuit (IC) doubles about every two years [2]. This comes from the prediction that Gordon Moore (former CEO of Intel) made in 1965, where he predicted that with every doubling year, the number of transistor components per integrated circuit would also double. We thus aim to explore whether Moore's law is still valid in today's leading IC technology.

Moore's Law Statement: The number of transistors on a microchip doubles every two years.

Null Hypothesis: Moore's law is valid.

Alternate Hypothesis: Moore's law is not valid.

5.2 Approach

In order to check the validity of Moore's law, we created two synthetic columns depicting the number of transistors after two years; one containing the actual number of transistors and another containing the expected number of transistors in accordance with Moore's law. Finally using an inference test, we compare the two synthetic columns and compute the similarity between them.

5.3 Analysis

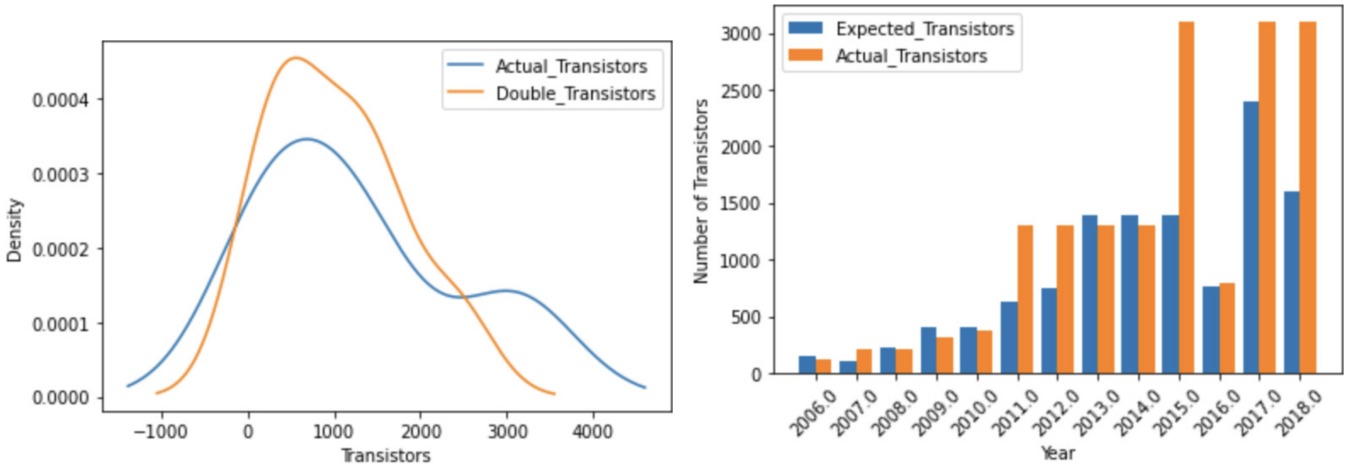
- Corresponding to each unique CPU instance in our video games dataset, we scraped CPU release year data for all unique CPU instances in our dataset from different websites.
- Picked relevant columns for inference such as CPU Name, CPU Number of Cores, CPU Release Year, and Number of Transistors.
- Due to the difference in the number of cores for each unique CPU instance, we standardized the number of transistors for each unique CPU instance to one core.
- For each year, we took only the CPU instance containing the maximum number of transistors. It only makes sense to consider the maximum value of transistors in an IC chip for a particular year, as it was a result of the best chip technology in that year.

- Created two synthetic columns; Double transistors: Number of transistors if Moore's law is assumed to be followed & Actual transistors: Maximum number of transistors in an IC chip 2 years later (from real data).
- Figure 5 shows the structure of final data frame.

Year	Maximum Number of Transistors	Number of Transistors After 2 Years	Expected Number of Transistors According to Moore's Law
2016	N_{2016}	N_{2018}	$2*N_{2016}$
2017	N_{2017}	N_{2019}	$2*N_{2017}$
2018	N_{2018}	N_{2020}	$2*N_{2018}$
2019	N_{2019}	N_{2021}	$2*N_{2019}$

Figure 5: Example columns in final hypothesis testing data frame

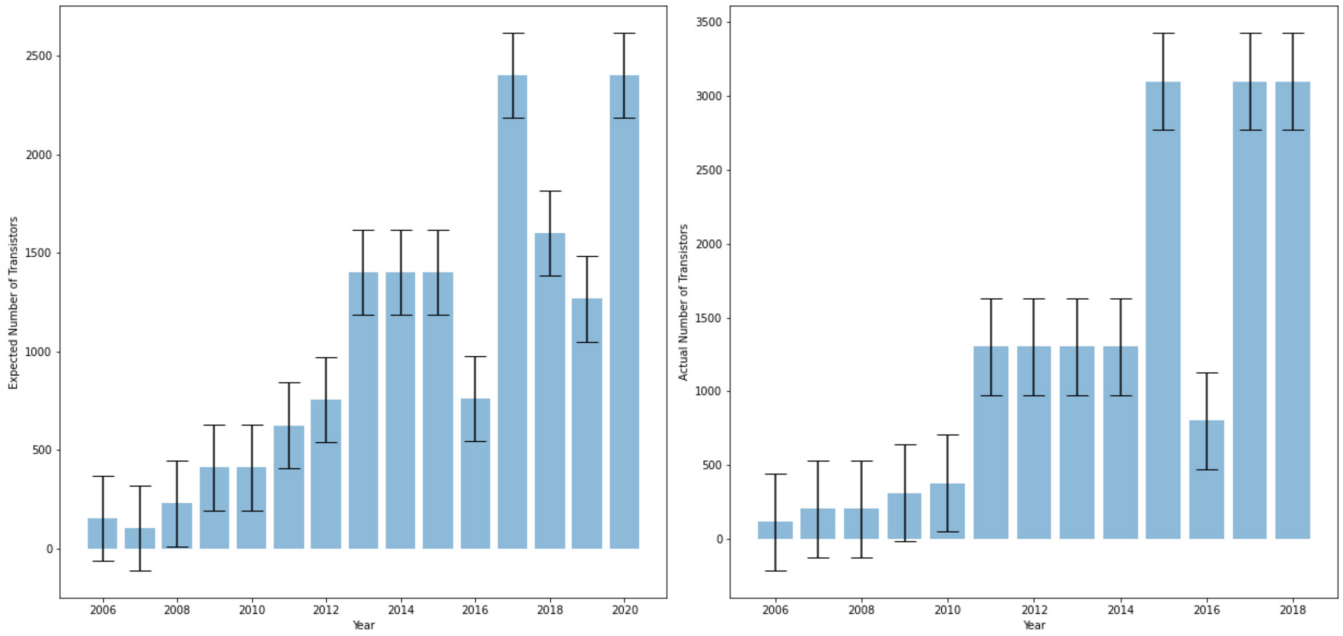
- Data distribution: To visualize the distributions of observations of maximum number of transistors for each year, we use a Kernel Density Estimation (KDE) plot as in Figure 6a. Further, we also plot the bar plots in Figure 6b.
- In order to choose a statistical test, we first ascertained if the data could be reduced to sample means. In our case, we are using the maximum number of transistors for each year and we don't know population parameters, hence we ruled out the option to perform a z-tests and t-tests. Furthermore, t-tests make a strong assumption about the data being normally distributed and given the distributions of our data in figure **??** we can testify against using a parametric t-test. Now, since our data is not categorical, we also eliminated Chi-squared test and instead chose to perform the Kolmogorov–Smirnov test (KS) test as well as the Mann Whitney U (MWU) test to compare the distributions of actual transistors and expected transistors according to Moore's Law. Because we have a small dataset (< 15 data points), a KS test might not be powerful enough.
- Statistical Power: In order to contextualize statistical power, we have calculated the effect size [3] using the Mann-Whitney-U z-value and number of pairs considered in the study. We obtain a medium effect size of 0.3 which is sufficient for our results to be powerful enough for practical application but it could be ameliorated with more data points. Below in figure 7a and 7b, we have also depicted the SEM for the actual and expected number of transistors and plotted the error bars for the same. This gives us an idea of how representative our sample is of the whole population.
- Results: Here we state the p values observed by both, the KS test and Mann Whitney U test for our testing.
  1. Mann–Whitney two-tailed U = 90.0, pvalue = 0.79
  2. Kolmogorov-Smirnov two-tailed D = 0.23, pvalue = 0.89



(a) KDE graphs

(b) Bar charts

Figure 6: Data distributions



(a) Expected number of transistors

(b) Actual number of transistors

Figure 7: SEM plots

Given standard $\alpha = 0.05$, we notice that p values obtained by both inference tests are greater than α and hence we fail to reject our null hypothesis. Thus, our assumption that the null hypothesis is true is accepted.

- **Limitations**: For our inference problem, we have less than 15 instances, and hence the presence of outliers can not be ignored as they have a larger effect on the outcome of our hypotheses test. Secondly, our data is not entirely representative of all the CPUs that have existed and is only a subset of the CPUs present in the original fps video games dataset. Given more data instances and representation of all CPUs, the p values would be much more accurate.

6 Regression

6.1 Goal

Given feature columns including Game Name, Game Settings, CPU type, etc. our aim is to predict the Frames per Second (FPS) obtained using a combination of the CPU, GPU, and game features as the predictors and the FPS achieved as the target.

6.2 Approach

To evaluate best predictors and R^2 scores for this regression, we mainly implement a Shallow Machine Learning model - The Random Forest Regressor as well as a Deep Neural Network. Further, we perform Principal Component Analysis on the dataset, as well as attempt Hyper-parameter tuning.

6.3 Analysis

- Given that our data contains several categorical columns, the best method to treat these categorical columns was by mean encoding. Mean encoding solves the encoding task for each category and also creates a feature that is more representative of the target variable. From a mathematical perspective, mean encoding represents a probability of your target variable, conditional on each value of the feature. In a way, it embodies the target variable in its encoded value. It considerably decreases cardinality and thus becomes a great tool to use in order to reach a better loss with a shorter tree for algorithms such as Decision Tree and Random Forest.
- For the regression problem, we are using a dataset of size 390711 rows and 43 columns imputed by the method described in Section 3.
- We plot feature importance graphs on the original full dataset, as well as feature importance plots of the dataset given different Game Settings. Figure 8 visualizes overall feature importance and figure 9a, 9b, 10a & 10b visualize feature importance specifically at different game settings (as an example).

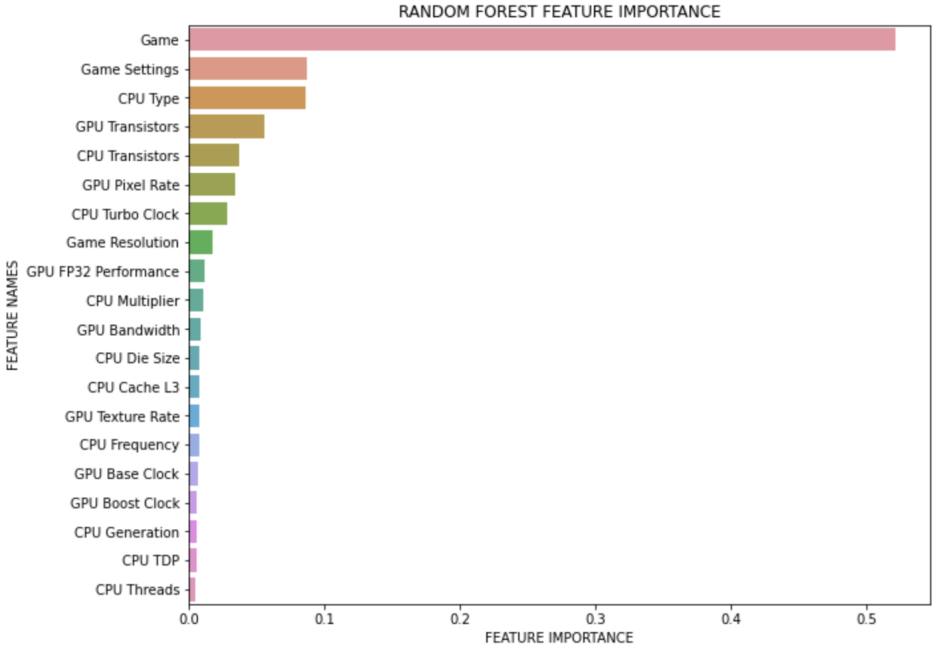
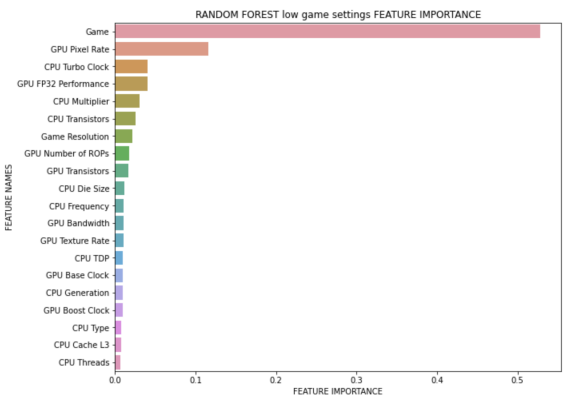
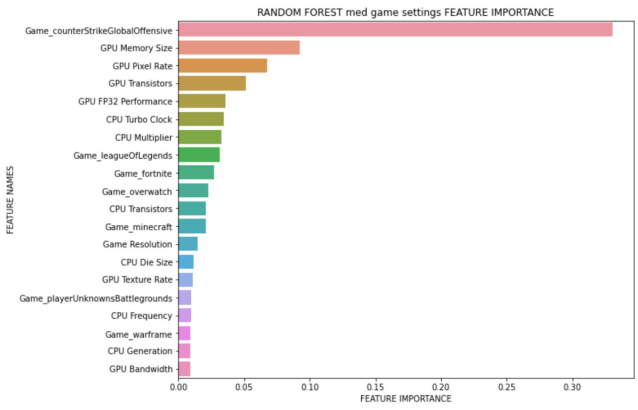


Figure 8: Feature importance for entire dataset

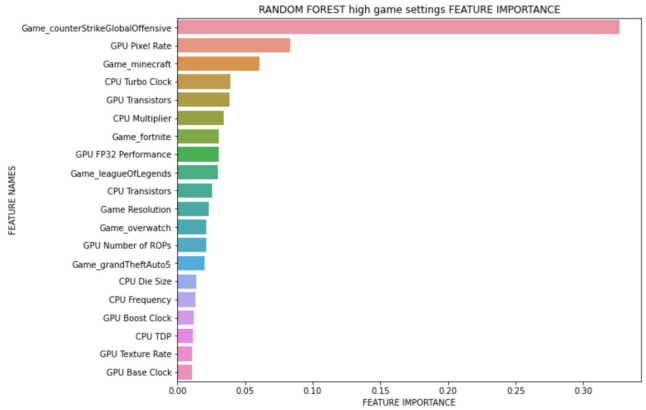


(a) Feature importance for low game settings

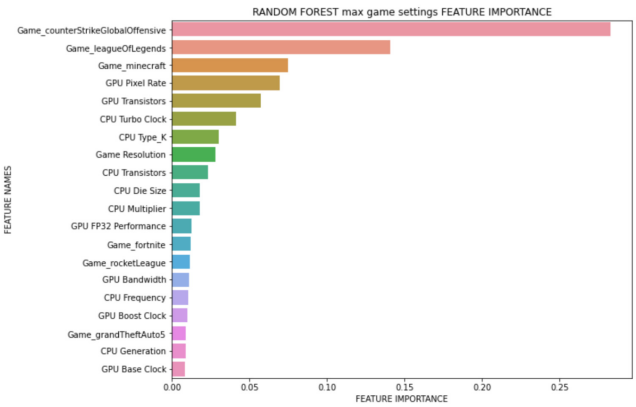


(b) Feature importance for med game settings

Figure 9: Feature importance plots



(a) Feature importance for high game settings

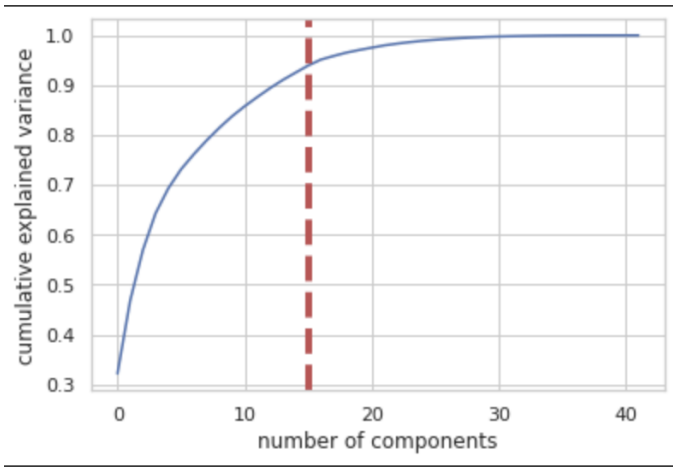


(b) Feature importance for max game settings

Figure 10: Feature importance plots

- At first, we use a Random Forest Regressor, since they are known to be robust as they are an amalgamation of weak decision stumps and incorporate bagging in the algorithm. Bagging includes randomness (essential for large dimensional datasets) and hence, curbs over-fitting.

- Next, we applied Principal Component Analysis on the scaled dataset to approximate the original dataset with fewer variables, while reducing computational power to run our model. Using PCA, we studied the cumulative explained variance ratio of these features to understand which features explain the most variance in the data. As shown in Figure 11a around just 15 components capture about 95% of the variance of the dataset. Figure 11b states the Cumulative Variance Ration and Explained Variance Ratio for these principal components.



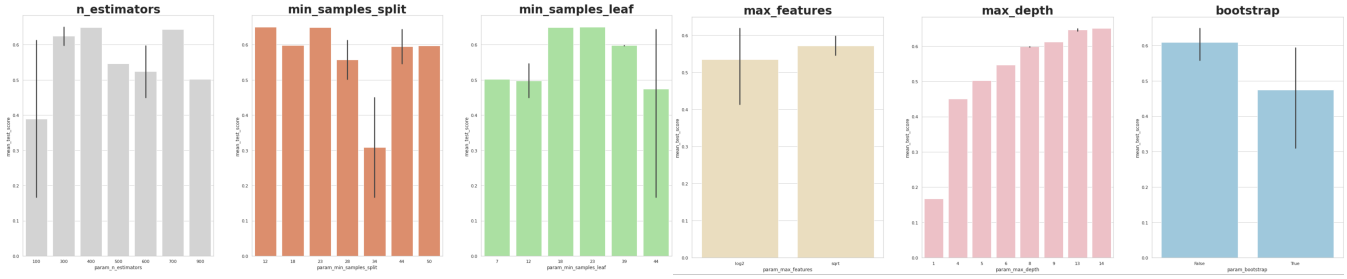
(a) Variance captured graph

S.No.	Cumulative Variance Ratio	Explained Variance Ratio
0	0.321451	0.321451
1	0.468007	0.146556
2	0.569253	0.101246
3	0.643133	0.073880
4	0.693012	0.049879
5	0.731164	0.038152
6	0.761285	0.030121
7	0.788819	0.027534
8	0.814151	0.025332
9	0.837311	0.023160

(b) CVR % EVR

Figure 11: Implementing PCA

- Finally, working with the Random Forest regressor with PCA components, we attempt at tuning hyperparameters such as number of estimators, min leaf samples, max features, max depth, etc using Randomized Search CV. However, since our data is huge, we randomized the search and worked with only 50% of randomly picked data points. Best estimators obtained are $n\_estimators = 300, min\_samples\_split : 12, min\_samples\_leaf = 23, max\_features = log2, max\_depth = 14, bootstrap = False$ which are then utilized for prediction. Figure 12a and 12b plots bar graphs of highest estimator values obtained on mean test scores.



(a) best estimators 1

(b) best estimators part 2

Figure 12: Best estimators obtained by hyperparameter tuning

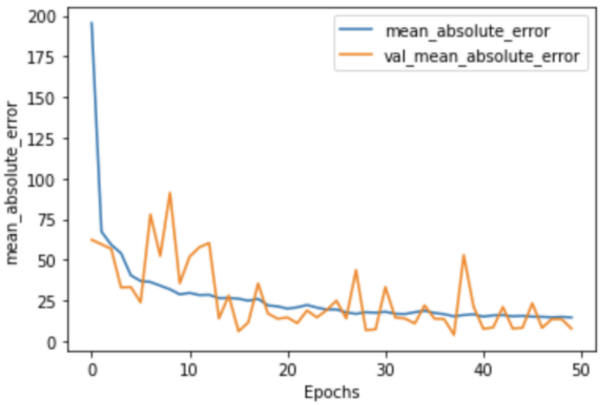
- For further improvements in the results obtained by a Random Forest Regressor, we implemented an Artificial Neural Network having 3 hidden layers and one input and output layer each. With ReLU as the activation function and Adam as an optimizer, we trained the model for 60 epochs and noticed that the Mean Squared Error (MSE) saturated at around 50 epochs. Figure 13b plots the training and validation MSE for 50 epochs.

- With mostly default parameters in the Random Forest Regressor and a test size of 0.3, the model achieves different R^2 scores on full dataset, PCA components after hyperparameter tuning as shown in Figure 13a. It also reports the $RMSE$ and R^2 scores obtained from training of a Neural Network.

- **Limitations**: Due to the large volume of our dataset, we were unable to perform Cross-Validation as it was extremely computationally expensive. As an alternative, we have experimented with different batch sizes and epochs. We were unable to perform hyperparameter tuning for the Artificial Neural Network model as it was not computationally feasible.

	Random Forest Regressor			Artificial Neural Network
Metric	Full Dataset	PCA Components	Tuned Hyper-parameters	
R^2	0.6511	0.6531	0.6589	0.9860
RMSE	0.5110	0.5086	0.5044	0.1025

(a) Result table



(b) MSE during training

Figure 13: Regression outputs

7 Clustering

7.1 Goal

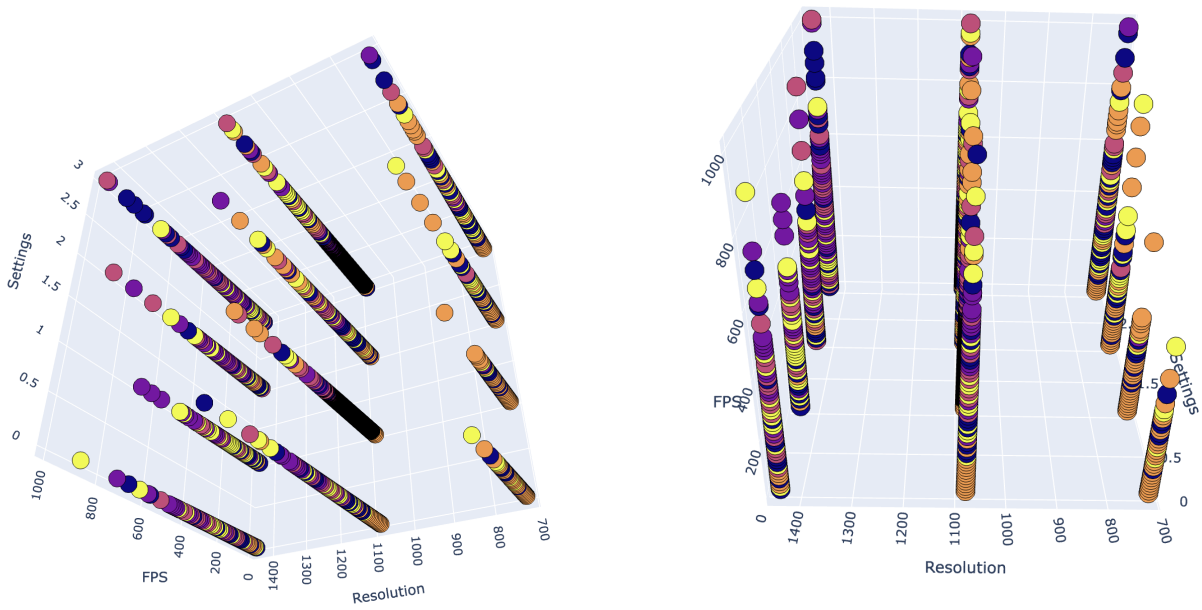
In this section, we aim at dividing the data points into a number of groups such that data points in the same groups are more similar to other data points in the same group than those in other groups. This type of unsupervised machine learning is done to identify distinct groups within a dataset without much concern for a specific outcome.

7.2 Approach

Since our dataset contains both - categorical and continuous feature columns, the biggest challenge was clustering these two different types of feature columns together. Hence, we used a modified version of K means algorithm, the **K-prototype** algorithm and also used the **Gower's distance** approach for the algorithm DBSCAN.

7.3 Analysis

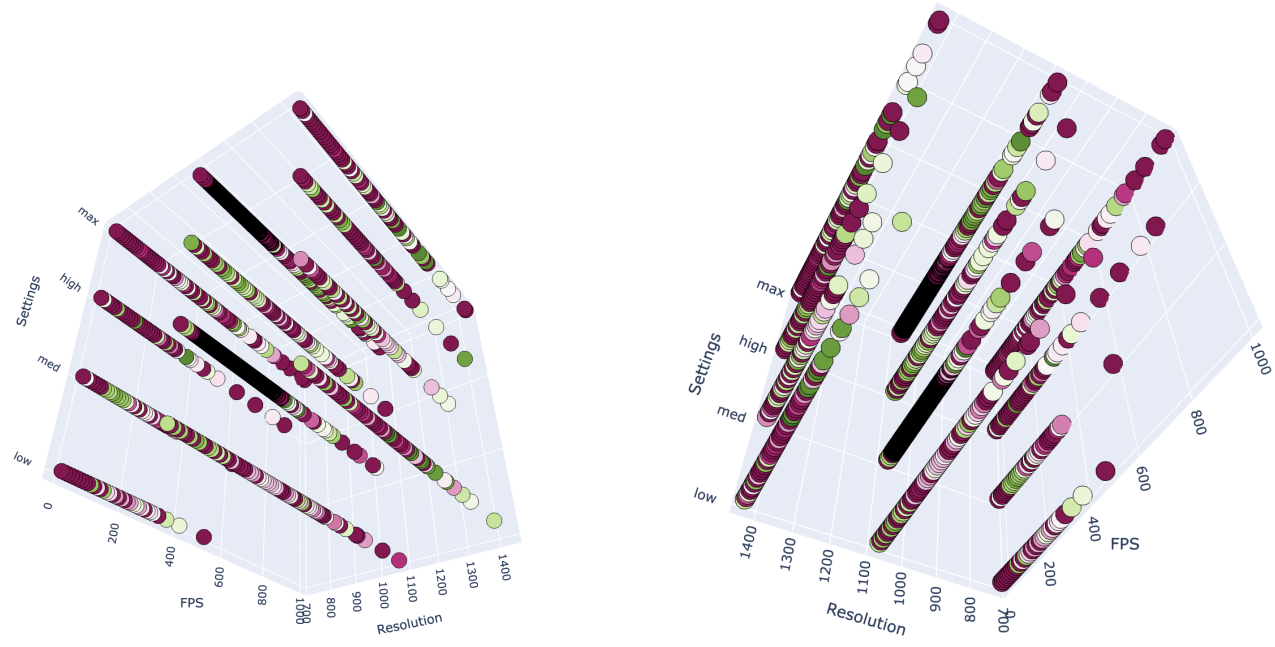
- Here, we work with the original data frame that contains NA values, and we drop the NA values and a few irrelevant columns such as ID, CPU Name, Dataset, etc. Further, we also dropped duplicate rows to ensure uniqueness of each data point. Final dataset is now of the shape: $186120 rows \times 37 columns$.
- Because our data has 37 dimensions and it would be impossible to visualize the clusters, we simply make an assumption here. We assume that the columns characterized by FPS, Game Settings and Display Resolution are dependent on a combination of the rest of the feature columns, and term these as our target columns. While FPS (*continuous*) is already a target column, it would be interesting to notice clusters in a 3D plane with Game Settings (*categorical*) and Display resolutions (*categorical*) as the other two axes.
- First, we implement K-prototype, a partitioning based method developed in order to handle clustering algorithms with the mixed data types (numerical and categorical variables). It has an advantage because it is not too complex and is able to handle large data and is better than hierarchical based algorithms.
    1. Divided the dataset into categorical and continuous column based on a threshold (decided by observing the dataset). Columns that contain less than 16 unique values were treated as categorical.
    2. Used $pd.factorize()$ on categorical columns. This method is useful for obtaining a numeric representation of an array when all that matters is identifying distinct values. factorize is available as both a top-level function [4].
    3. Ran the clustering algorithm on the feature columns (all excluding the three target columns as discussed earlier) and visualized the data points on a 3D space of the target columns.
    4. After trying multiple values for number of clusters, we observed that the data point are best visualized with 5 major clusters. Figure 14a and 14b shows snippets of the 3D plot.
- Next, we implement DBSCAN, a density based clustering algorithm that performs well with data that does not contain normal shaped clusters, and rather contains outliers or specific patterns. It is able to find arbitrary shaped clusters and clusters with noise (i.e. outliers).



(a) View 1 (b) View 2

Figure 14: K-prototypes cluster plots

1. Like K-Means, DBSCAN also does not work with categorical columns and hence we implemented the Gower's distance to measure how different two records are. However, calculating the distance between each record of feature columns with every other record is quite memory intensive and and time taking, hence we used a sliding window method on 18 different parts of the dataset to calculate an $n \times n$ Gower matrix.
2. DBSCAN requires inputs in terms of epsilon value, and the minimum number of samples in a cluster. To find the optimal combination of both, we searched through the space of epsilon in the range of $(0, 1)$ and minimum samples in the range of $(1, 20)$.
3. Finally, a similar 3D plotting procedure was repeated for DBSCAN as of the K prototypes procedure. Figure **??** shows snippets of the 3D plot obtained.



(a) View 1 (b) View 2

Figure 15: DBSCAN cluster plots

- **Limitations**: Since our dataset mostly consisted of categorical features, our clusters were formed in an unconventional manner, which made it cumbersome to analyze.

8 Classification

8.1 Goal

To predict the different settings in which the game is played, namely (low, med, high, or max) using the specifications of the computer and game.

8.2 Approach

Given advantages of Random forest as mentioned in Section 5.3 previously, in this section we train a Random Forest Classifier that used a combination of the CPU, GPU, and game features as the predictors and the Game Settings as the target. We also used LightGBM due to its faster training speed, less memory usage, and its ability to focus on accuracy.

8.3 Analysis

- Initialized the feature columns and target columns after mean encoding (similar to how we did in regression).
- Partitioned our data using a 70-30 train-test-split and trained our models on the training set to make predictions on the test set. Further, similar to the PCA method explained in regression, we again took the principal components, and evaluated results for both classification models on PCA components as well (figure 16). To evaluate the performance of our models, we generated the Classification Report and the ROC-AUC plots as in figure 17a, 17b, 18a, 18b.

Data	Game Resolution	Accuracy	Precision	Recall	F1 Score
Full Dataset	Low	0.66	0.42	0.33	0.37
	Medium		0.61	0.59	0.60
	High		0.76	0.86	0.81
	Max		0.42	0.31	0.36
PCA Components	Low	0.64	0.42	0.32	0.36
	Medium		0.61	0.59	0.60
	High		0.74	0.86	0.79
	Max		0.36	0.24	0.29

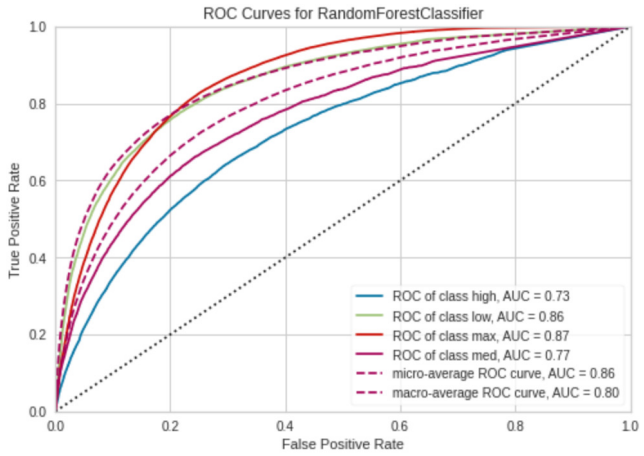
Figure 16: Classification results

- **Limitations**: We were unable to perform hyperparameter tuning as it was computationally expensive given the size of our dataset and the time it takes to run it. It can be seen that F1 scores vary considerably from the confusion matrix and this can cause an imbalance between situations where, for example, low game settings are wrongly classified as high and low game settings are misclassified as max. It would be ideal to further analyze the respective costs and weights to make more accurate conclusions.

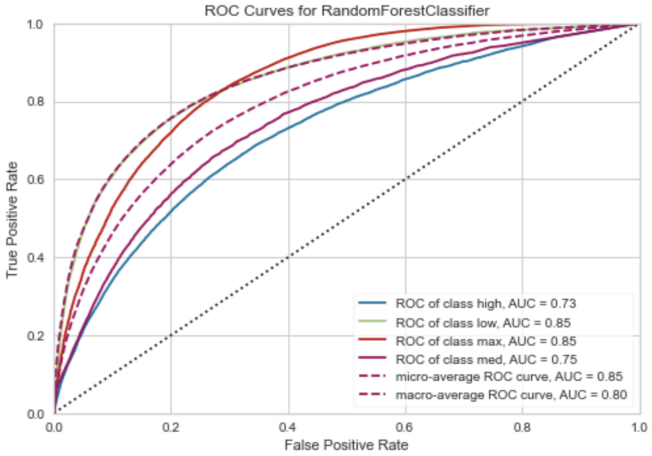
9 Conclusion

The FPS (first-person shooter) gaming industry has experienced significant growth in recent years, with many people turning to gaming as a full-time occupation rather than just a hobby. In this project, we sought to understand the optimal conditions for video game playing by analyzing a dataset from OpenML.

The first step was to clean the dataset, ensuring that it was organized and free of errors. Next, we conducted data exploration to identify trends and patterns in the data. During this process, we discovered a unique relationship between three variables: die size, process size, and transistors. This led us to explore Moore's Law, a theory that

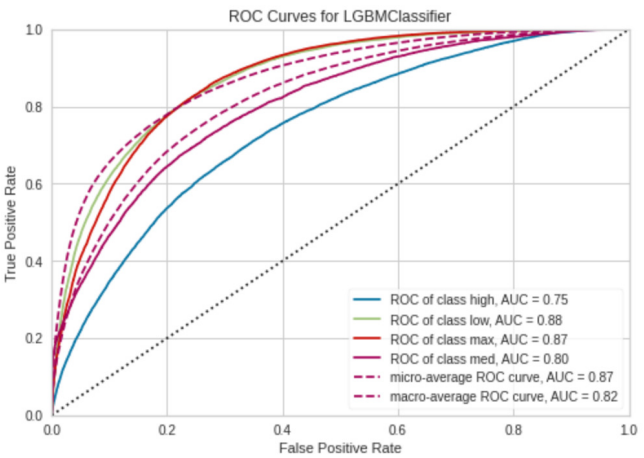


(a) On full dataset

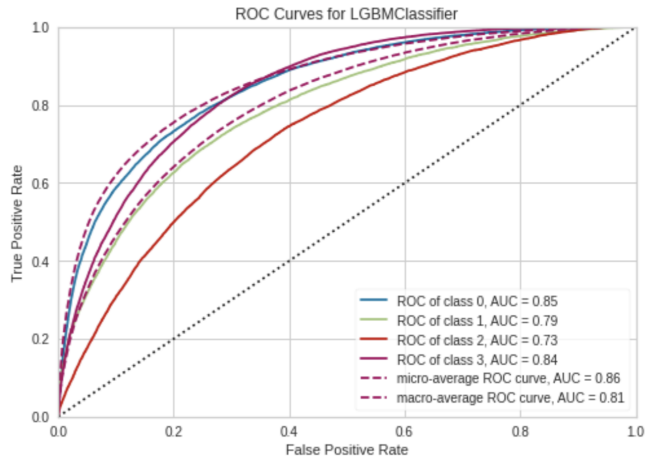


(b) On PCA components

Figure 17: AUC ROC plots of Random Forest Classifier



(a) On full dataset



(b) On PCA components

Figure 18: AUC ROC plots of LightGBM classifier

states that the number of transistors on a microchip will double approximately every two years.

To test the validity of Moore's Law, we performed a hypothesis test, with the null hypothesis being that Moore's Law is still accurate today. After analyzing the data, we were unable to reject the null hypothesis and thus concluded that Moore's Law remains true.

Using this information, we were able to impute missing values for transistor counts using Moore's Law. With the final dataset complete, we then conducted predictive analyses to further understand the optimal conditions for video game playing. We also pursued the prediction of game settings that each configuration could be classified into. Our cleaned dataset also allowed us to cluster our dataset based solely on Game features of Resolution, Settings and FPS. Overall, this project allowed us to gain valuable insights into the FPS gaming industry and the factors that contribute to successful gameplay.

The dataset is a mix of laptop and desktop CPU/GPU. Desktops are much more powerful as compared to laptops as laptops have less number of cores and power consumption. An ideal dataset would have an extra column specifying if it is a laptop or a desktop. If we had the price of each of the components then we could have analyzed the price for each specification of the FPS. To get it we would need to manually scrape a lot of data to get it.

Assumption: Since Moore's Law holds true, either the CPU Die Size is increasing or CPU Process Size is decreasing or both which is causing the number of transistors in a CPU to double every two generations. We observe that Die Size and Process Size are different for different SKUs in a company's product lineup even if all the SKUs were launched in the same year. This may be because any company might want to cut costs in its lower end products and hence may end up not providing the latest in technology. We assume that this is how it is going to continue moving forward.

10 Author Contributions

1. Data cleaning, Pre-processing, Exploratory Data Analysis: Kritik Seth
2. Hypothesis Testing - Tayyibah Khanam, Keya Shukla
3. Clustering - Tayyibah Khanam, Keya Shukla
4. Regression - Umair Ayub, Nikitha Kasipati, Kritik Seth
5. Classification - Umair Ayub, Nikitha Kasipati, Kritik Seth

References

[1] S. Peeters, "fps-in-video-games." [Online]. Available: https://www.openml.org/search?type=data&sort=runs&id=42737&status=active

[2] "Moore's law," Dec 2022. [Online]. Available: https://en.wikipedia.org/wiki/Moore%27s_law

[3] "Test, chi-square, anova, regression, correlation..." [Online]. Available: https://datatab.net/tutorial/mann-whitney-u-test

[4] "Pandas.factorize." [Online]. Available: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.factorize.html