## LEGO Database Study Report

Xingyu Lu & Ruijuan Niu

### Introduction

LEGO is one of the most popular toys in the world. The company has launched over 15000 sets since 1932. The accumulated data allows for the construction of a huge dataset for many explorations. This project aims to explore the LEGO dataset compiled by Rebrickable, examining the connections between different components and study how the trends in the sets sizes, themes, etc. have changed over time. By processing and analyzing the data using R, we hope to answer the following questions:

1. What are the most common or rare themes of all time?
2. How have the sizes of sets changed over time?
3. Can we predict which theme a set is from by the bricks it contains?

The structure of this report is as follows: Chapter II states the background on LEGO datasets and briefly introduces the components contained in each .csv files. Chapter III is the implementation and results of the solutions we proposed for each question. Chapter IV gives the conclusion of this project and further exploration we could do on the dataset.

### 

### Background

A LEGO database on Kaggle contains LEGO parts/sets/colors and inventories of every official LEGO set as of July 2017. The database contains 7 .csv files and a schema image which shows the relationship between different .csv files.

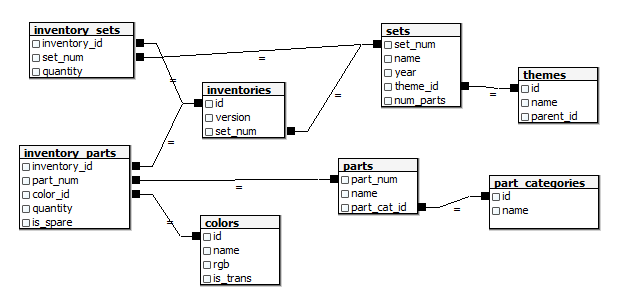


Fig 1. Schema of LEGO database

Figure 1 gives a visual representation of the relationship between the .csv files contained in the dataset package. Each file contains the information as Table 1:

Table 1. Data files information

| **File Name** | Description |
| --- | --- |
| **colors.csv** | This file contains the information about the colors of each piece of a LEGO brick, including a unique color id, a color name, and a RGB value. |
| **inventories.csv** | An inventory represents the small bag of parts that come with LEGO sets. It might be used in multiple sets, and a set could have multiple inventories. The file contains the unique ID, version number, and the related set number of each inventory. |
| **Inventory\_parts.csv**  &  **inventory\_sets.csv** | These two files contain detailed information on the inventories. Compared to the “inventories” file, the “inventory\_sets” gives the quantity of each inventory that is included in the set. Each LEGO set could be separated into several parts. And those parts could be appearing in different inventories. The “inventory\_parts” contains information about the relationship between parts and inventories. |
| **parts.csv**  &  **parts\_categories.csv** | These datasets include information on LEGO parts and defines the parts into 57 different categories by giving each category a unique ID. |
| **sets.csv** | The file contains the information on LEGO sets including the name, year released, theme, and how many parts each set includes. |
| **themes.csv** | The file includes a unique theme ID, a theme name, and a parent ID of each theme. The parent ID is defined as a bigger category which a theme might be a part of. |

### Implementation and Results

#### Problem 1: What are the most common or rare themes of all time?

LEGO has released 614 themes since 1950, we are curious about what are the most common and popular themes, and what are the rarest themes of all time. In this section, we mainly focused on sets.csv and themes.csv.

We found that there are 0’s in column “num\_parts”, which is not logically reasonable because a company would never sell a product with 0 content in it. The set names of these rows are duplicates of other rows, therefore we pre-treated the dataset and omitted the rows with num\_parts = 0. By merging sets and themes datasets, we were able to connect each theme ID with theme name. The parent ID is a larger category which contains similar themes, but since we only knew the ID number not the parent category name, it doesn’t make much sense exploring the parent ID for this problem.

We used the “groupby” command in R and found the top 10 most-sets LEGO themes, shown in Figure 2. The top 10 most-sets themes include Technic, Supplemental, Duplo, City, Ninjago, Friends, Service Packs, Police and Creator. Basic and multi-use themes like Technic, Supplemental and Service Packs are reasonable to appear in the figure. It is interesting to find out that the police theme is in the top 10. Meanwhile the rarest include over 90 themes with only one set released. We noticed that rarest themes also include Supplemental and Friends. The issue is although the theme ID is unique for each theme, the name may have duplicates. If we want to know what exact themes are most common or rare, we may need to check the corresponding theme names in the raw data from LEGO.

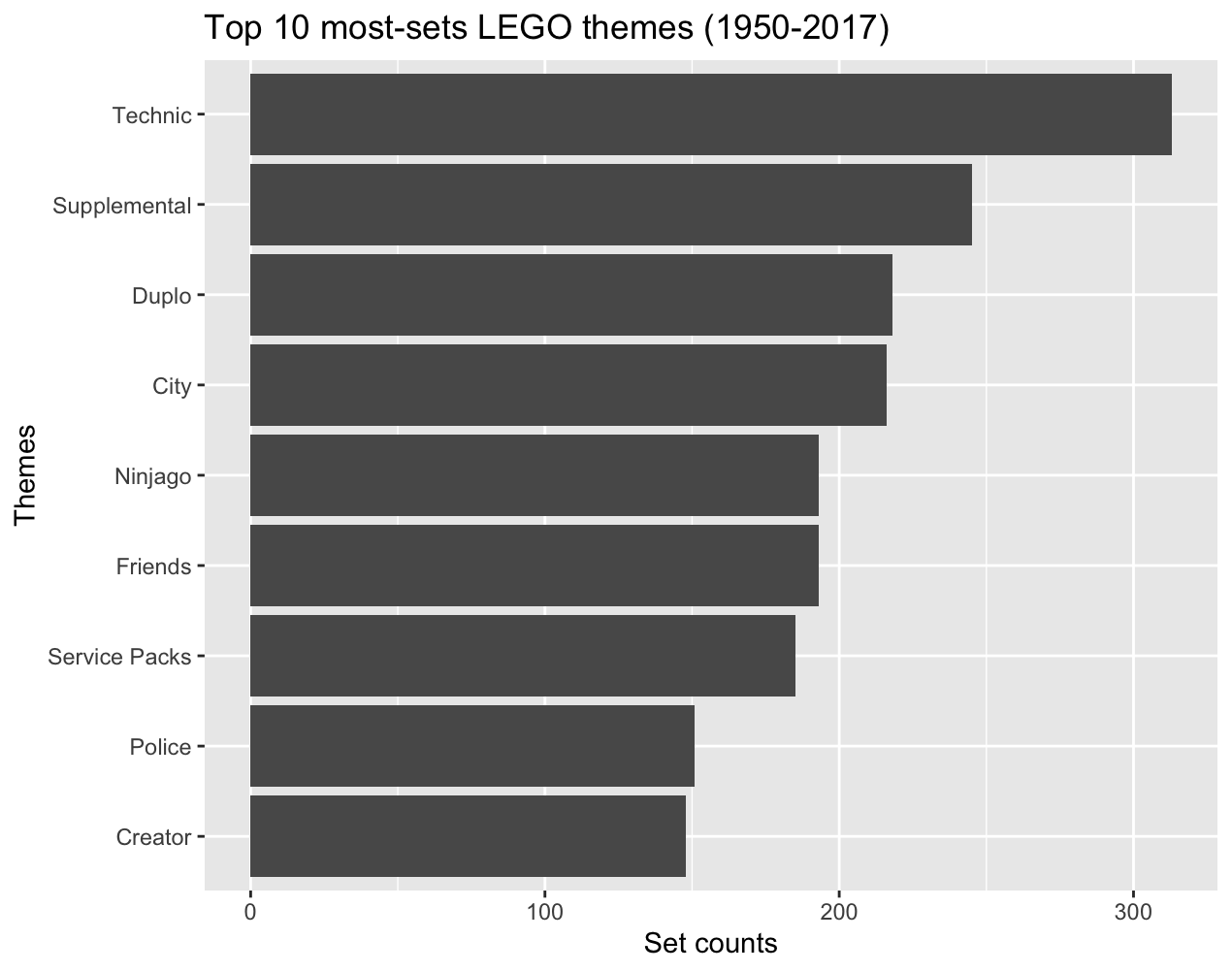


Figure 2. Top 10 most-sets LEGO themes (1950-2017)

When considering the themes with most parts, we compared the average number of parts for all themes and found the top 10 and bottom 10, shown in Figure 3 and 4. We found that the Disney theme is the absolute top 1, which has over 4000 parts on average. Architecture categories, for example Modular Buildings and Town Plan themes also have thousands of parts. As for the least-parts, there are themes like Supplemental and Technic, which makes sense since these may be complement sets and only contain very few parts. The Star Wars theme made it to the bottom 10 for the same reason. We also found the theme IDs are different from the theme IDs in the top 10 most-sets even though the theme names appear to be the same. Again, we need further investigation on the theme ID and name. Overall, the figures present some brief ideas about the LEGO themes of all time.

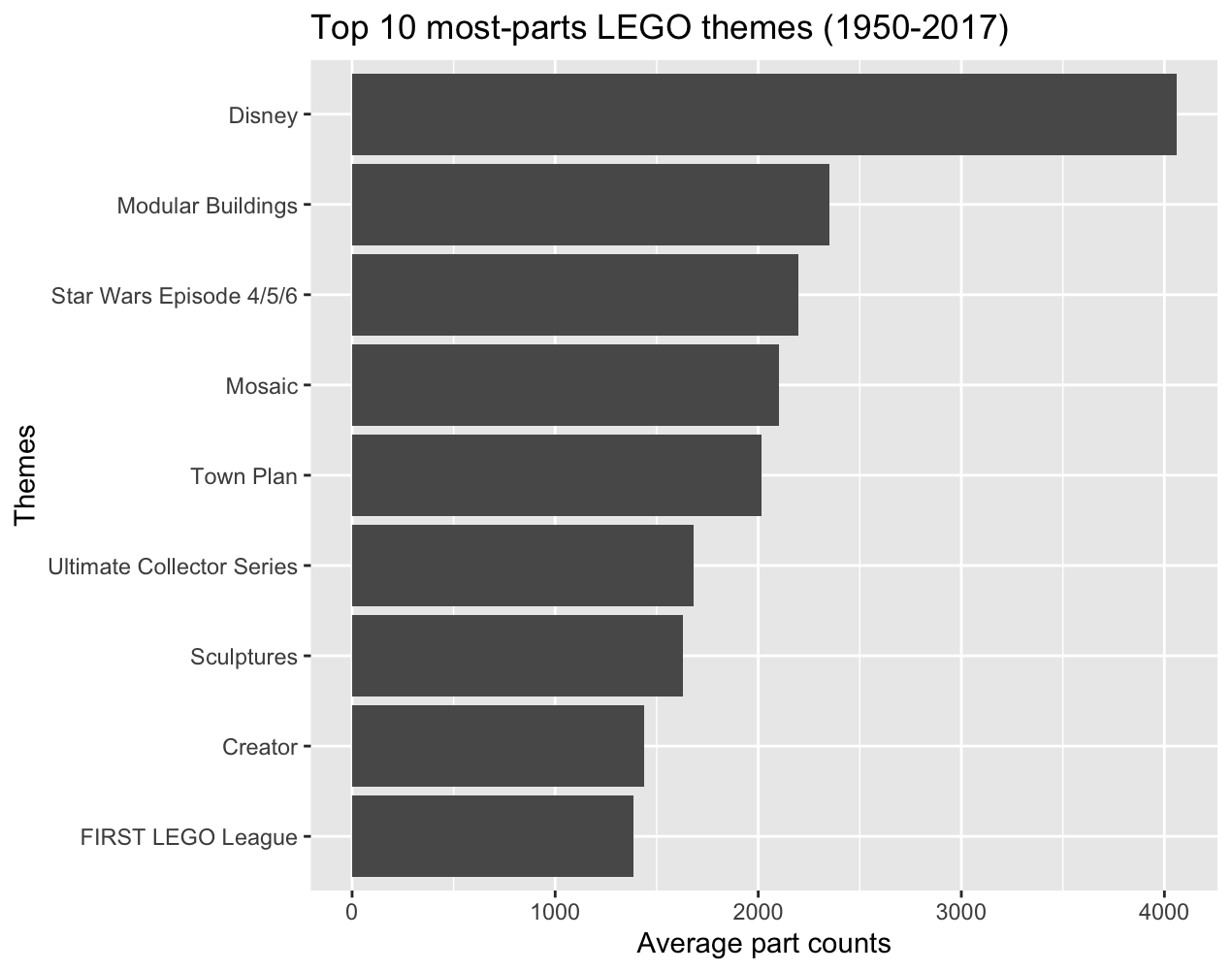


Figure 3. Top 10 most-parts LEGO themes (1950-2017)

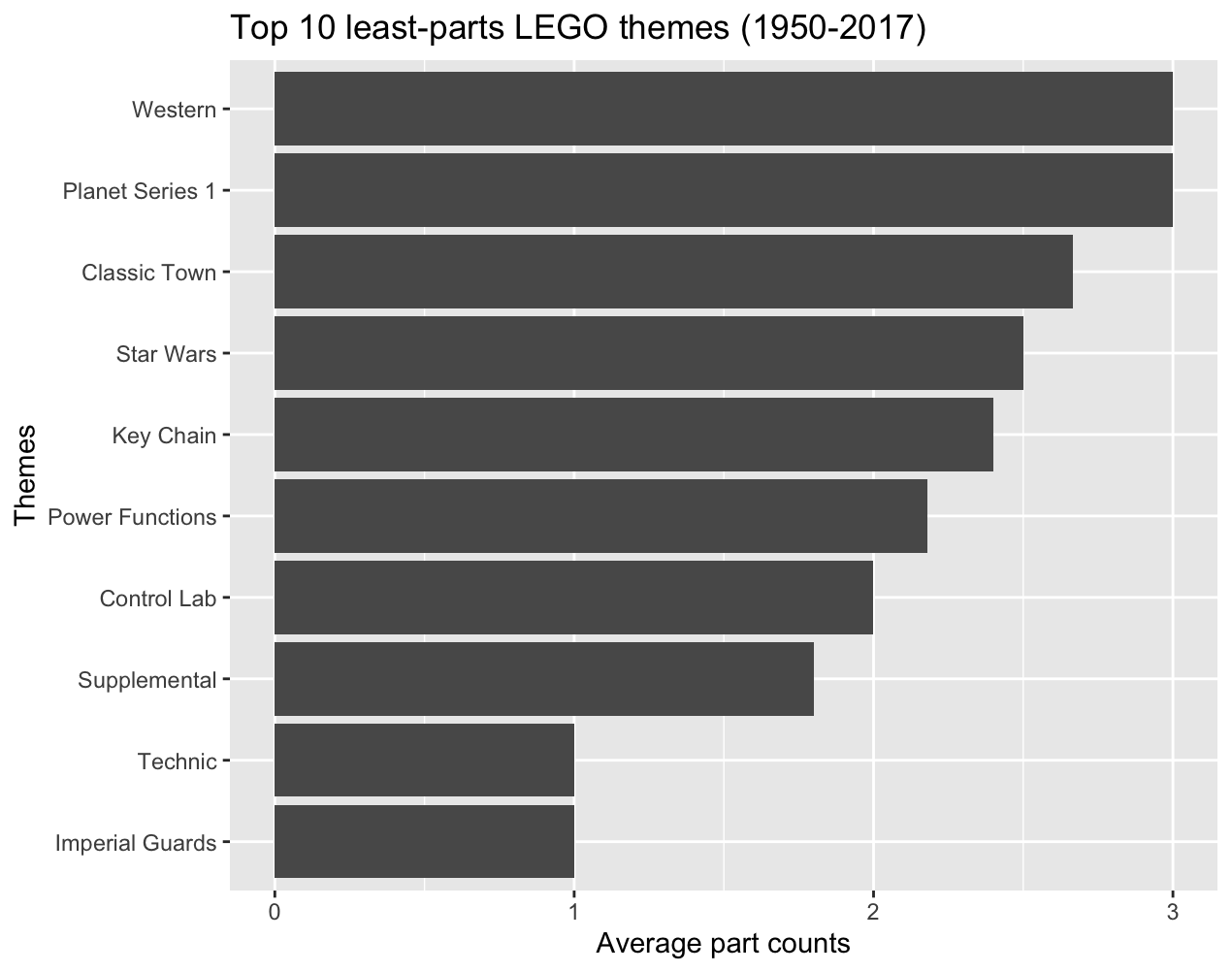


Figure 4. Top 10 least-parts LEGO themes (1950-2017)

#### Problem 2: How have the sizes of sets changed over time?

If we were asked about the set size changing trend without exploring the data, we’d say the size must become larger over time. We aggregated the average set size and the median set size by each year and tried to find the trend.

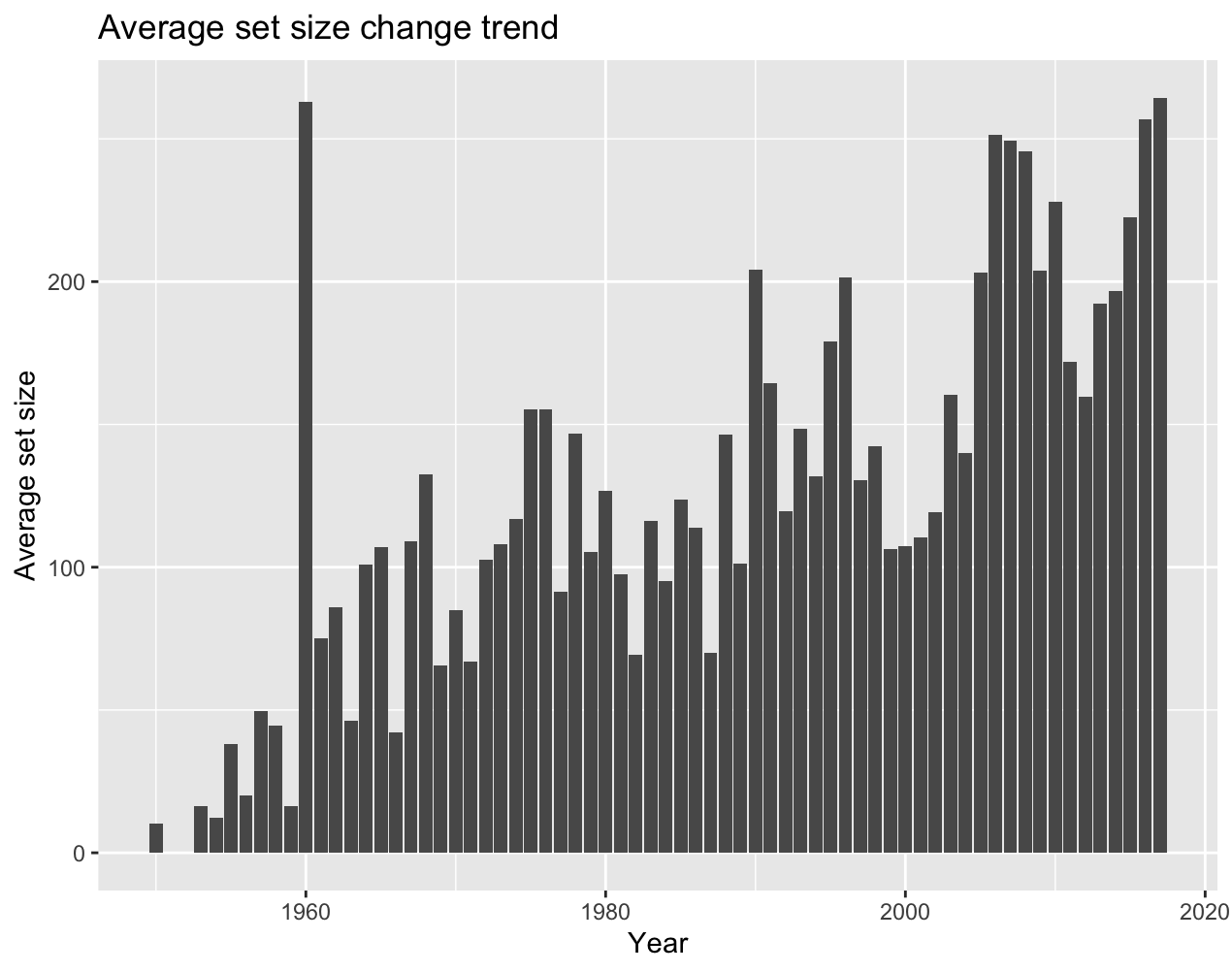


Figure 5. Average set size change trend

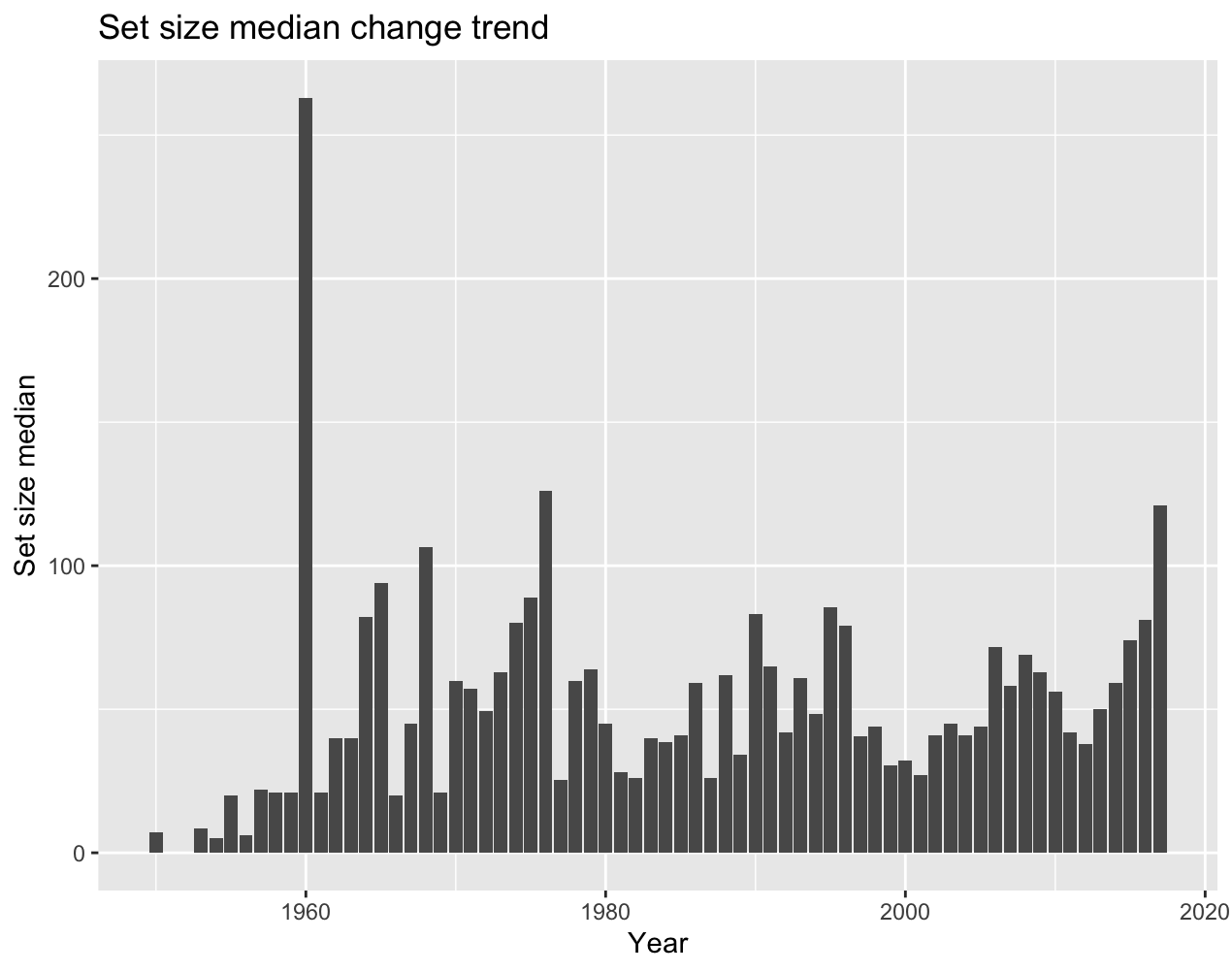


Figure 6. Set size median change trend

From Figure 5, we noticed that the size is becoming larger overall, but in a short period of time, i.e. 5 or 10 years, the average set size may have a W or V shape. The set size median doesn’t show any obvious trend, as Figure 6 showed, except there is one outlier. In 1960, LEGO only released 2 sets, Number Bricks and Kindergarten LEGO Set, with 50 and 476 parts respectively. This is the reason why it has the second largest average set size and the largest set size median from 1950 to 2017.

Next we tried to fit both the kernel and linear regression model on these two datasets. For the kernel regression model, the first step is to find the best bandwidth. We chose to omit the outlier for a more fitted result because the maximum likelihood cross-validation method tends to overweight the outlier. Bandwidths are 5 and 4.78 for the average and median set size respectively. We tried other methods and got even larger bandwidth. It would over-smooth the model and make not much difference compared to the linear model, which is not suitable for these two datasets. If we let the bandwidth be very small, then the model would be less smooth, which would have less bias. And the prediction would be noisier [1]. By using “pcf.kernesti” in package regpro, the kernel regression models were fitted. Figure 7 showed the regression models for average set size, the kernel regression has some curves but overall it is similar to linear regression.

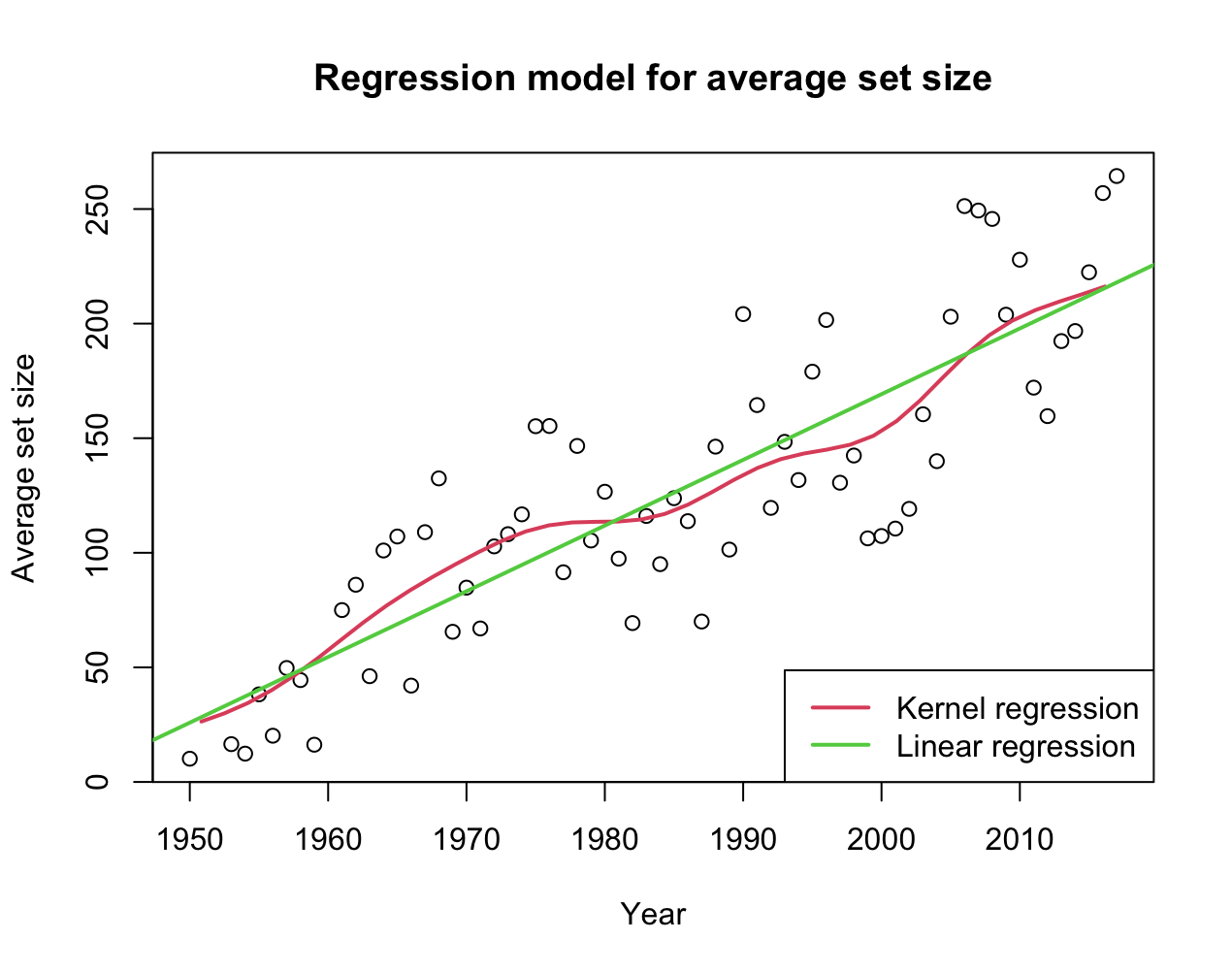


Figure 7. Regression model for average set size

The linear model for median set size is a terrible choice, shown in Figure 8. The kernel regression model fits much better. Since the data points always form a V or W shape, we don’t think those regression models can predict future set sizes. In fact, extrapolation can easily go wrong unless you know the real model of the dataset. We can only use the models to interpolate between the data points [2]. The datasets miss values in 1952. Therefore we were able to “predict” the average and median set size in 1952. The average set size in 1952 is 28.79 and 31.62 for kernel and linear model respectively. The median set size in 1952 is 14.17 and 33.99 for kernel and linear model respectively. Obviously the kernel model is more likely to predict a better value for median set size.

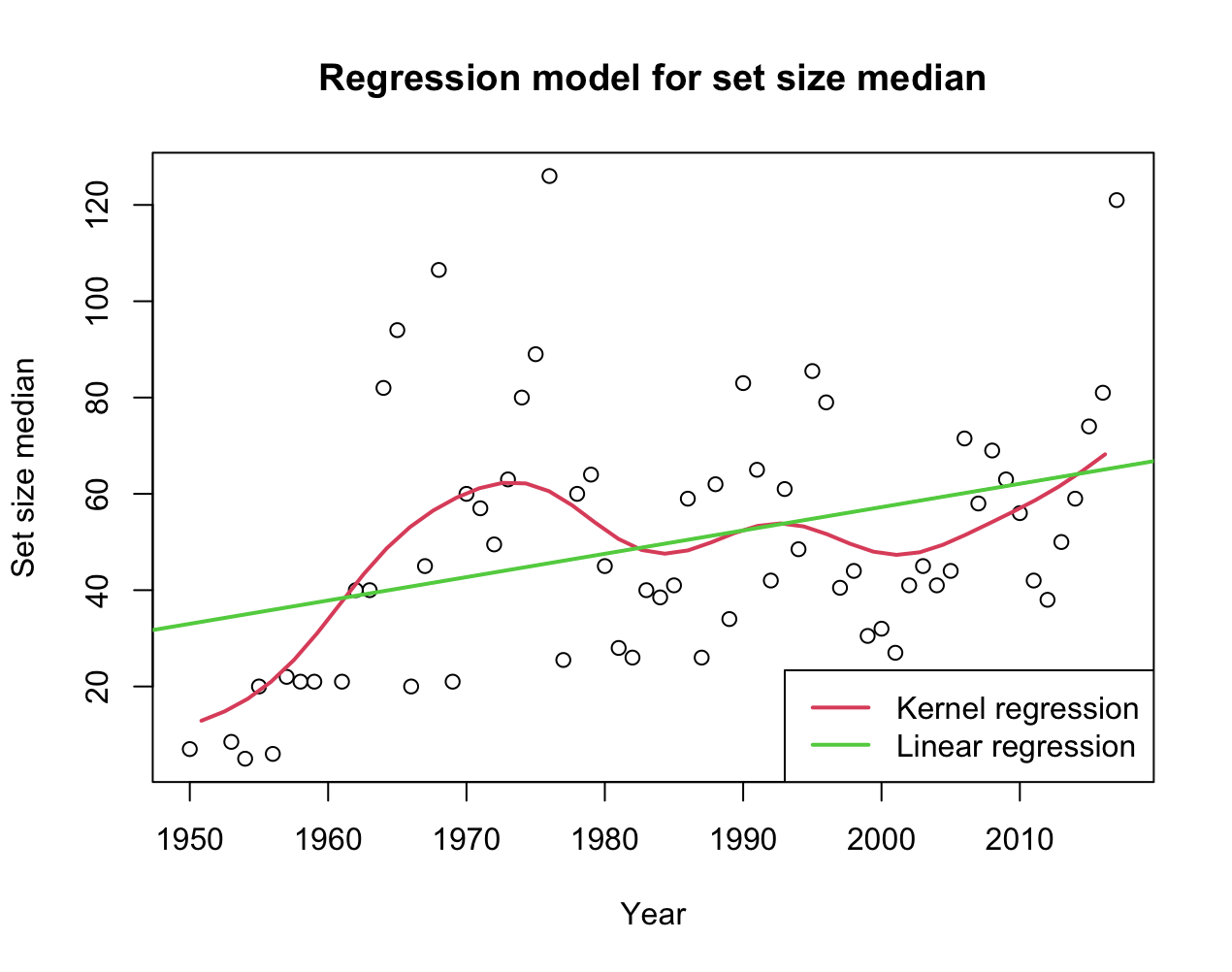


Figure 8. Regression model for set size median

#### Problem 3: Can we predict which theme a set is from by the bricks it contains?

To answer this question, we need to find out a variable that connects the theme of a set and the bricks contained in it. There are more than 400 themes which could be divided into 79 parent IDs in the dataset. We decided to do the classification on the sets. By taking color proportions of the bricks as our predictors and adding a logical variable for parent ID, we want to predict whether the LEGO sets belong to a specific theme category (parent ID X) or not.

The implementation could be divided into following steps:

1. Decide which parent id we would like to predict
2. Pick the most regularly used 10 colors that appear in the sets
3. Give each set a label (‘True’ or ‘False’) to state whether the set belongs to the parent ID X or not. (logical variable)
4. Calculate the proportion of colors in each set. (numeric variables)
5. Split the data into train and test group
6. Use the train data to fit the model
7. Use the fitted model and test data to predict the results
8. Compare the predicted and true outcomes

We got the results as follows:

We took parent ID 217 which is the most common category of themes as our target ID. And picked the most-used 10 colors with RGB code { "05131D", "FFFFFF", "C91A09", "A0A5A9", "F2CD37", "6C6E68", "9BA19D", "0055BF", "237841", "E4CD9E"}. We divided the processed clean data into 10% of test data and 90% of training data. Then we fit a random forest model using ‘caret’ package in R:

| # fit a random forest model (using ranger)  rf\_fit <- train(as.factor(ID1) ~ .,  data = cleaned\_train,  method = "ranger") |
| --- |

And got the fitted model:

| # > rf\_fit  # Random Forest  #  # 6676 samples  # 10 predictor  # 2 classes: 'FALSE', 'TRUE'  #  # No pre-processing  # Resampling: Bootstrapped (25 reps)  # Summary of sample sizes: 6676, 6676, 6676, 6676, 6676, 6676, ...  # Resampling results across tuning parameters:  #  # mtry splitrule Accuracy Kappa  # 2 gini 0.9202881 0.1532812  # 2 extratrees 0.9199904 0.1374270  # 6 gini 0.9201446 0.3557885  # 6 extratrees 0.9218780 0.3340881  # 10 gini 0.9182787 0.3574488  # 10 extratrees 0.9215206 0.3452776  #  # Tuning parameter 'min.node.size' was held constant at a value of 1  # Accuracy was used to select the optimal model using the largest value.  # The final values used for the model were mtry = 6, splitrule = extratrees and min.node.size = 1. |
| --- |

The ‘ranger’ is a method to implement randomForest. The ‘train()’ function re-runs the model over 25 bootstrap samples and across 3 options of the tuning parameter (one of them is ‘mtry’). It represents the number of randomly selected predictors at each cut in the tree.

Then we compare the predicted results with true outcome:

| # predict the outcome on a test set  par\_rf\_pred <- predict(rf\_fit, cleaned\_test)  # compare predicted outcome and true outcome (by hand)  jud =(par\_rf\_pred == 'TRUE')  sum(jud == cleaned\_test$ID1)/length(cleaned\_test$ID1)  #by function:  confusionMatrix(par\_rf\_pred, as.factor(cleaned\_test$ID1)) |
| --- |
| Result:  > sum(jud == cleaned\_test$ID1)/length(cleaned\_test$ID1)  [1] 0.9204852  # Confusion Matrix and Statistics  #  # Reference  # Prediction FALSE TRUE  # FALSE 664 48  # TRUE 11 19  #  # Accuracy : 0.9205  # 95% CI : (0.8986, 0.9389)  # No Information Rate : 0.9097  # P-Value [Acc > NIR] : 0.1686  #  # Kappa : 0.3558  #  # Mcnemar's Test P-Value : 2.775e-06  #  # Sensitivity : 0.9837  # Specificity : 0.2836  # Pos Pred Value : 0.9326  # Neg Pred Value : 0.6333  # Prevalence : 0.9097  # Detection Rate : 0.8949  # Detection Prevalence : 0.9596  # Balanced Accuracy : 0.6336  #  # 'Positive' Class : FALSE |

The accuracy of the model is 0.9205 which means the model made a good prediction on the data. Therefore, with the information of bricks that a set contains, we can predict which theme category(that is, parent ID) does a set belong to.

### Conclusion and Further Works

In conclusion, we solved the problems of finding the most common or rare themes of all time, observing the set size changing trend and predicting the parent themes based on brick colors. Kernel regression model does a great job fitting a dataset without an obvious trend.

We still need to further investigate the raw data and figure out the exact theme name for each theme ID, and the name for parent ID in order to have a better understanding of the LEGO database. Since it is a huge dataset, further exploration could be done in the future.

### Reference

[1] ​​Yee T.W. (2015) VGAMs. In: Vector Generalized Linear and Additive Models. Springer Series in Statistics. Springer, New York, NY. <https://doi.org/10.1007/978-1-4939-2818-7_4>

[2] *How to make predictions with non-parametric regression?* Cross Validated. Retrieved November 22, 2021, from https://stats.stackexchange.com/questions/529677/how-to-make-predictions-with-non-parametric-regression.

[3] *A basic tutorial of caret: the machine learning package in R.* Retrieved November 22, 2021, from

https://www.rebeccabarter.com/blog/2017-11-17-caret\_tutorial/