COGS 109 Final Project

By (Group 28): Bao Nguyen, Rem Talde, Shadman Noor, Yubi Quinzon

BACKGROUND ¶

Introduction

With the pandemic still in full effect, many activities have been limited to the home. During this time of social distancing, many have taken to different hobbies, such as building different Lego sets. And with the constant stream of Lego sets, the concern for getting a quality set is high. Our group wanted to perform a linear regression to determine which factors, if any, could predict the review rating of a Lego set.

Cleaning our dataset

The dataset we are using was found on kaggle.com containing initially 14 columns(which represent the different dimensionalities or features) and 12262 rows (represent the values of each sampled lego set). To clean the data, we removed rows that had any missing values and columns that were not included in our analysis.

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import statistics as stats
from sklearn.model_selection import train_test_split
plt.rcParams['figure.figsize'] = (10,10)
%matplotlib inline
```

In [2]:
```
## Data preprocessing
df = pd.read_csv("lego_sets.csv") #Creating a Pandas Dataframe

# Dropping all unnecessary columns
df = df.drop(["ages","prod_desc","prod_id","prod_long_desc",
              "review_difficulty", "set_name", "theme_name",
              "country"], axis = 1)

# Dropping all nan values
df = df.dropna(axis = 0, how ='any')

# Resetting index
df = df.reset_index().drop(['index'], axis=1)

df
```

Out[2]:

	list_price	num_reviews	piece_count	play_star_rating	star_rating	val_star_rating
0	29.9900	2.0	277.0	4.0	4.5	4.0
1	19.9900	2.0	168.0	4.0	5.0	4.0
2	12.9900	11.0	74.0	4.3	4.3	4.1
3	99.9900	23.0	1032.0	3.6	4.6	4.3
4	79.9900	14.0	744.0	3.2	4.6	4.1
...	...	...	...	...	...	...
10461	36.5878	6.0	341.0	4.4	4.3	4.2
10462	24.3878	8.0	217.0	4.1	3.6	4.1
10463	24.3878	18.0	233.0	4.6	4.6	4.5
10464	12.1878	1.0	48.0	5.0	5.0	5.0
10465	12.1878	11.0	109.0	4.5	4.7	4.8

10466 rows × 6 columns

In [3]:
```
# Validating all nan were dropped
df["val_star_rating"].unique()
```

Out[3]:
```
array([4. , 4.1, 4.3, 4.4, 4.5, 3.6, 4.2, 3.5, 4.8, 5. , 4.7, 3. , 4.
6,
       4.9, 3.8, 3.3, 2.5, 2.7, 2.2, 2.3, 2.8, 2. , 2.9, 3.2, 3.9, 3.
4,
       2.6, 3.7, 1. , 1.8, 2.4, 1.9, 3.1])
```

METHODS

1. Visualizing our data

The next step preparing our dataset for analysis was to visualize it and choose the appropriate analysis technique. We visualized the dataset with the help of matplotlib and decided that linear regression would be a good fit for the Lego dataset. Linear Regression is a technique that is used to find the best model that predicts continuous outputs using one or multiple input variables from the dataset.

2. Cross Validation

We then used cross validation to split the data in a 80:20 ratio using train_test_split from the sklearn library. After that we created matrix forms of the features and the labels and created the y = Aw matrix form. The best parameter for each of our models, w, was computed from the y = Aw equation by solving for w, using the training data only. After we computed the best parameter, w, we used our models on the test set to make predictions on the dataset. For error measurement we used mean squared error, also known as MSE, to compare our different models and to see which model performs the best on the Lego dataset. With the help of matplotlib, we were able to visualize the results and draw conclusions on our linear regression analysis.

In [4]:
```
# Extracting each column into an array
price, n_reviews, n_pieces, p_rating, rating, v_rating = df.T.values
```

Price vs Rating

In [5]:
```
# Split 80% into a training set, 20% into a test set
price_train, price_test, rating_train, rating_test = train_test_split(
```

In [6]:
```
## Creating the augmented matrix (A1)
# Creating column of ones
ones = np.ones((len(rating_train),1))

# Combining columns to make A1
A1 = np.hstack((ones,price_train.reshape(-1,1)))
```

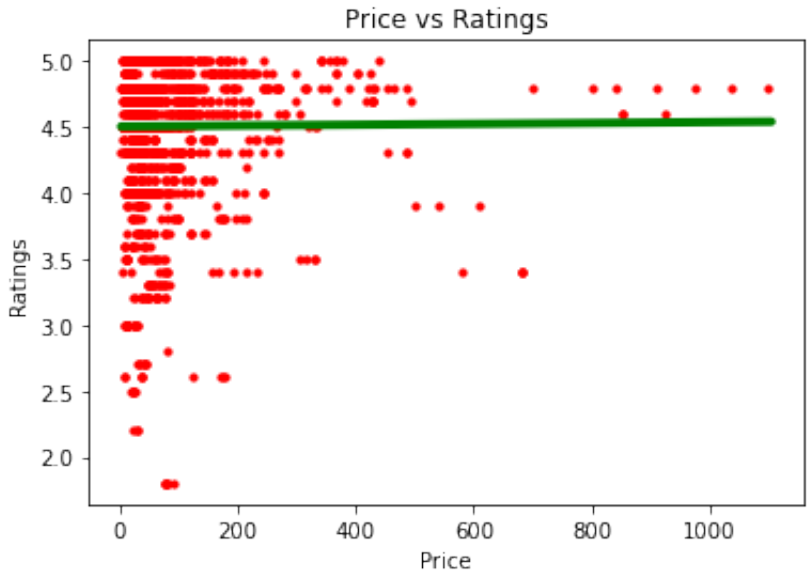
In [7]:
```
# Solving for the weight vector
w1 = np.linalg.lstsq(A1,rating_train,rcond=None)[0]
```

In [8]:
```
# Price vs Rating model:
print("Rating =", w1[0], "+", w1[1],"* price")
```

```
Rating = 4.503169925177405 + 3.2985250544001436e-05 * price
```

In [9]:
```
## Predict ratings according the model
# Creating a smooth range
X = np.linspace(min(price_test), max(price), len(price_test))

# Predicting
rating_pred1 = w1[0] + X*w1[1]
```

In [10]:
```

## Visualize the model
plt.scatter(price_test, rating_test, color='r', label='Test', s=10) #

# Plot the model
plt.title("Price vs Ratings")
plt.ylabel("Ratings")
plt.xlabel("Price")
plt.plot(X, rating_pred1, color='g', label='Price Model', linewidth=4,

plt.show()
```



Number of Reviews vs Rating

In [11]:
```
# Split 80% into a training set, 20% into a test set
reviews_train, reviews_test = train_test_split(n_reviews, train_size=0
```

In [12]:
```
# Combining columns to make A2
A2 = np.hstack((ones,reviews_train.reshape(-1,1)))
```

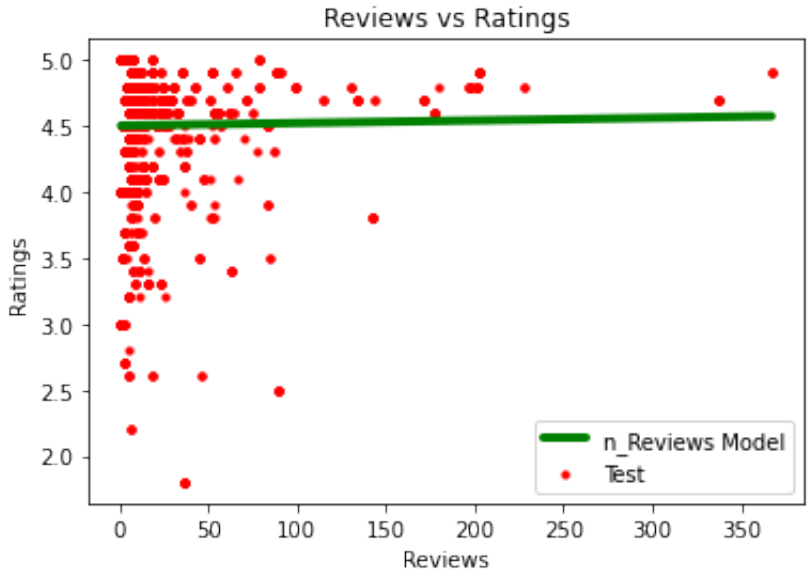
In [13]:
```
# Solving for the weight vector
w2 = np.linalg.lstsq(A2,rating_train,rcond=None)[0]
```

In [14]:
```
# N_reviews vs Rating model:
print("Rating =", w2[0], "+", w2[1],"* n_reviews")
```

```
Rating = 4.502194718317436 + 0.00019477216892437661 * n_reviews
```

In [15]:
```
## Predict ratings according the model
X = np.linspace(min(reviews_test), max(reviews_test), len(reviews_test

# Predicting
rating_pred2 = w2[0] + X*w2[1]
```

In [16]:
```
## Visualize the model
plt.scatter(reviews_test, rating_test, color='r', label='Test', s=10)

## Plot the model
# plt.margins(0.016) # Zooming in

# Plotting
plt.title("Reviews vs Ratings")
plt.ylabel("Ratings")
plt.xlabel("Reviews")
plt.plot(X, rating_pred2, color='g', label='n_Reviews Model', linewidt
plt.legend()
plt.show()
```



Number of Pieces vs Rating

In [17]:
```
# Split 80% into a training set, 20% into a test set
pieces_train, pieces_test = train_test_split(n_pieces, train_size=0.8,
```

In [18]:
```
# Combining columns to make A4
A3 = np.hstack((ones,pieces_train.reshape(-1,1)))
```

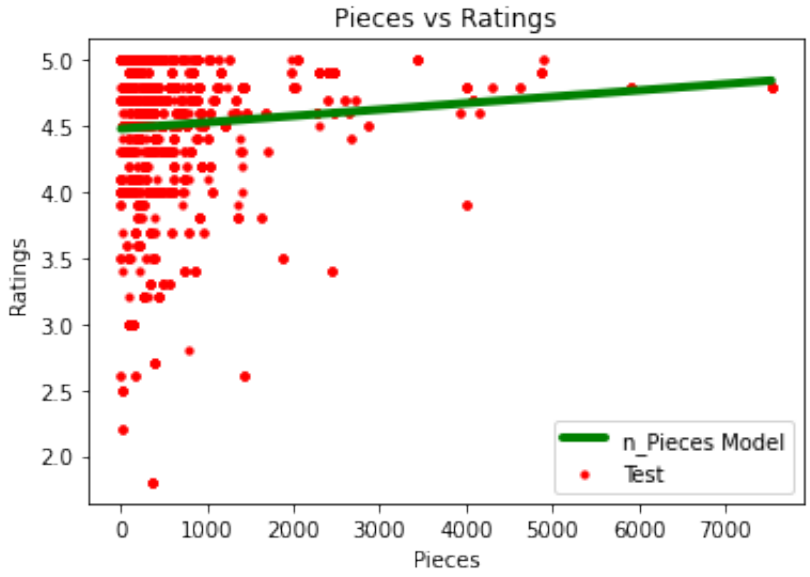
In [19]:
```
# Solving for the weight vector
w3 = np.linalg.lstsq(A3,rating_train,rcond=None)[0]
```

In [20]:
```
# N_reviews vs Rating model:
print("Rating =", w3[0], "+", w3[1],"* n_pieces")
```

```
Rating = 4.478583337287481 + 4.842768165333498e-05 * n_pieces
```

In [21]:
```
## Predict ratings according the model
X = np.linspace(min(pieces_test), max(pieces_test), len(pieces_test))

# Predicting
rating_pred3 = w3[0] + X*w3[1]
```

In [22]:
```
## Visualize the model
plt.scatter(pieces_test, rating_test, color='r', label='Test', s=10) #

## Plot the model
plt.title("Pieces vs Ratings")
plt.ylabel("Ratings")
plt.xlabel("Pieces")
plt.plot(X, rating_pred3, color='g', label='n_Pieces Model', linewidth
plt.legend()
plt.show()
```



Play Rating vs Rating

In [23]:
```
# Split 80% into a training set, 20% into a test set
playrating_train, playrating_test = train_test_split(p_rating, train_s
```

In [24]:
```
# Combining columns to make A4
A4 = np.hstack((ones,playrating_train.reshape(-1,1)))
```

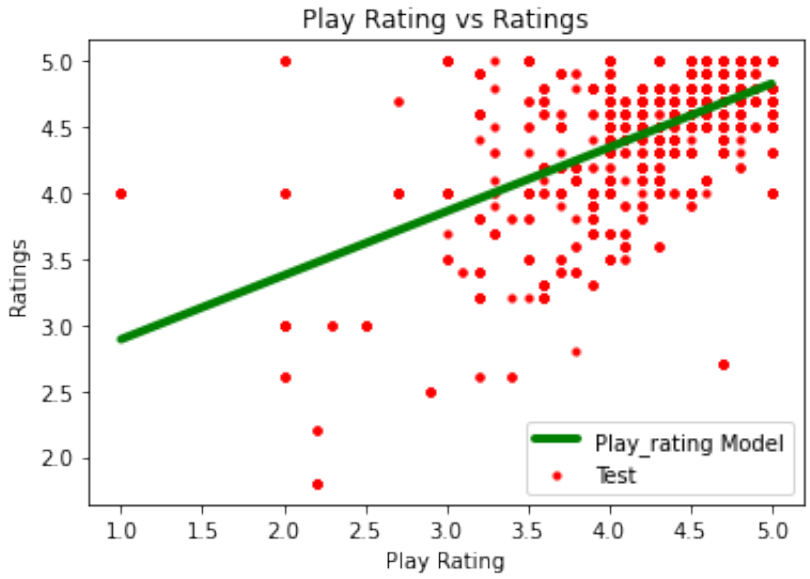
In [25]:
```
# Solving for the weight vector
w4 = np.linalg.lstsq(A4,rating_train,rcond=None)[0]
```

In [26]:
```
# Play Rating vs Rating model:
print("Rating =", w4[0], "+", w4[1],"* play_rating")
```

```
Rating = 2.4062602492476746 + 0.48429280936512337 * play_rating
```

In [27]:
```
## Predict ratings according the model
X = np.linspace(min(playrating_test), max(playrating_test), len(playra

# Predicting
rating_pred4 = w4[0] + X*w4[1]
```

In [28]:
```
## Visualize the model
plt.scatter(playrating_test, rating_test, color='r', label='Test', s=1

## Plot the model
plt.title("Play Rating vs Ratings")
plt.ylabel("Ratings")
plt.xlabel("Play Rating")
plt.plot(X, rating_pred4, color='g', label='Play_rating Model', linewi
plt.legend()
plt.show()
```



Value Rating vs Rating

In [29]:
```
# Split 80% into a training set, 20% into a test set
valrating_train, valrating_test = train_test_split(v_rating, train_siz
```

In [30]:
```
# Combining columns to make A5
A5 = np.hstack((ones,valrating_train.reshape(-1,1)))
```

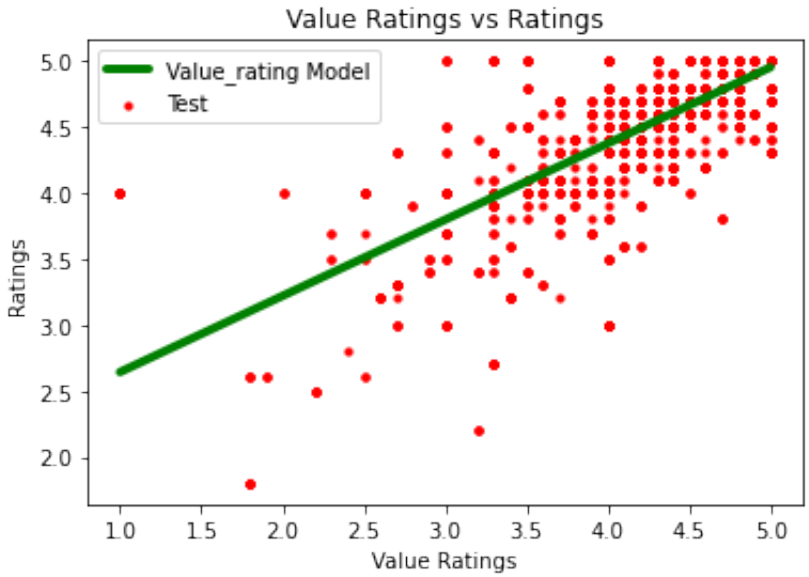
In [31]:
```
# Solving for the weight vector
w5 = np.linalg.lstsq(A5,rating_train,rcond=None)[0]
```

In [32]:
```
# Value Rating vs Rating model:
print("Rating =", w5[0], "+", w5[1],"* value_rating")
```

```
Rating = 2.0648407368524744 + 0.577468286723199 * value_rating
```

In [33]:
```
## Predict ratings according the model
X = np.linspace(min(valrating_test), max(valrating_test), len(valratin

# Predicting
rating_pred5 = w5[0] + X*w5[1]
```

In [34]:
```
## Visualize the model
plt.scatter(valrating_test, rating_test, color='r', label='Test', s=10

## Plot the model
plt.title("Value Ratings vs Ratings")
plt.ylabel("Ratings")
plt.xlabel("Value Ratings")
plt.plot(X, rating_pred5, color='g', label='Value_rating Model', linew
plt.legend()
plt.show()
```



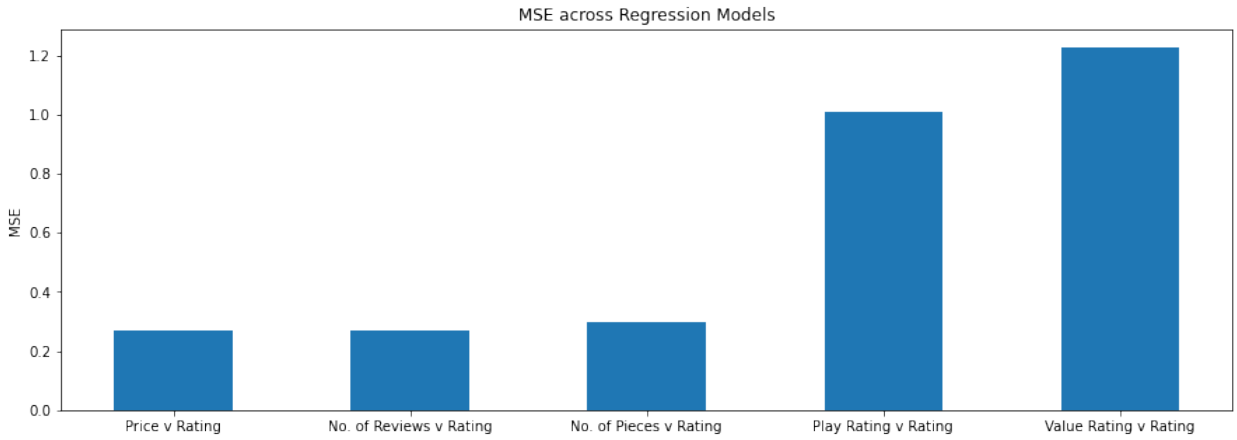
RESULTS

In [35]:
```
# MSE for the different predictions
from sklearn.metrics import mean_squared_error
mse1 = mean_squared_error(rating_test,rating_pred1)
mse2 = mean_squared_error(rating_test,rating_pred2)
mse3 = mean_squared_error(rating_test,rating_pred3)
mse4 = mean_squared_error(rating_test,rating_pred4)
mse5 = mean_squared_error(rating_test,rating_pred5)
```

In [36]:
```
print("MSE for Price vs Rating:", mse1)
print("MSE for Number of Reviews vs Rating:", mse2)
print("MSE for Number of Pieces vs Rating:", mse3)
print("MSE for Play Rating vs Rating:", mse4)
print("MSE for Value Rating vs Rating:", mse5)
```

```
MSE for Price vs Rating: 0.2687761322273113
MSE for Number of Reviews vs Rating: 0.26937877543208266
MSE for Number of Pieces vs Rating: 0.29933878038548184
MSE for Play Rating vs Rating: 1.0104633997284733
MSE for Value Rating vs Rating: 1.2268373081932378
```

In [37]:
```
#Plot to show the different MSE values
MSE_dict = {'Price v Rating':mse1, 'No. of Reviews v Rating':mse2, 'No
        'Play Rating v Rating':mse4, 'Value Rating v Rating':mse5}
dict_keys = list(MSE_dict.keys())
dict_values = list(MSE_dict.values())

fig = plt.figure(figsize = (15, 5))
plt.bar(dict_keys, dict_values, width = 0.5)
plt.title('MSE across Regression Models')
plt.ylabel('MSE')
plt.show()
```



After applying regression analysis, rounding and calculating the mean squared error (MSE) for every predictor, we found their MSEs to be 0.269 (for the price model), 0.269 (number of reviews model), 0.299 (number of ratings model), 1.010 (play ratings model) and 1.227 (value ratings model). As evidenced by graphs of the univariate models, price and number of reviews have identical correlation to ratings and their line graphs are less steep compared to the other three predictors. Price and number of reviews models have the same and lowest error. These models are comparable to the model using number of ratings as a predictor, differing in MSE only by 0.03. Models for the last two predictors, play and value ratings, are influenced by many outliers that scatter all over their graphs and introduce more variance, leading to greater MSEs than the first three. Their slopes are also distinctly steeper than the first three. In general, these univariate models proved to perform better than the multivariate model as the latter has a much higher MSE of 1.760. This is understandable because it combines predictors having greater amounts of variance and will therefore generate a larger MSE.

DISCUSSION

Discussing our results

Based off of the individual MSE values of each feature, we see that Price and the Number of reviews would provide the best prediction for a lego set's rating. Another interesting result is that our multivariate predictor (Where we take account of all the different features as a predictor for rating) to have the highest MSE. This may be a result of giving all of the different features the same weight - as variables with LOW impact but HIGH MSE introduce relatively more error to the model. However, based of our results, we do believe "Price vs Rating" or "Number of reviews vs rating" models to be sufficient in predicting a lego set's rating. However, with a few changes, we believe that the multivariate model could provide better results predicting the ranking of a lego set

How to improve

To improve the accuracy of our multivariate model, we could weigh the features proportionally to their impact. This would lessen the error introduced from features with high MSE but low impact. Another way to improve our model is to reduce the dimensionality of our dataset using PCA. This way we can work with only 2 or 3 principal components of our dataset and be more efficient in generating regression models.