Milestone 1 Report

- Dataset Analysis:

- Rows count = 4801 entries
- Columns count = 16 columns

```
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   id                4798 non-null   float64
 1   track_name        4787 non-null   object
 2   size_bytes        4782 non-null   float64
 3   currency          4616 non-null   object
 4   price             4784 non-null   float64
 5   rating_count_tot  4784 non-null   float64
 6   rating_count_ver  4784 non-null   float64
 7   vpp_lic           4773 non-null   float64
 8   user_rating_ver   4784 non-null   float64
 9   ver               4783 non-null   object
 10  cont_rating       4782 non-null   object
 11  prime_genre       4778 non-null   object
 12  sup_devices.num   4782 non-null   float64
 13  ipadSc_urls.num   4783 non-null   float64
 14  lang.num          4783 non-null   float64
 15  user_rating       4798 non-null   float64
dtypes: float64(11), object(5)
```

- Necessary categorical features:

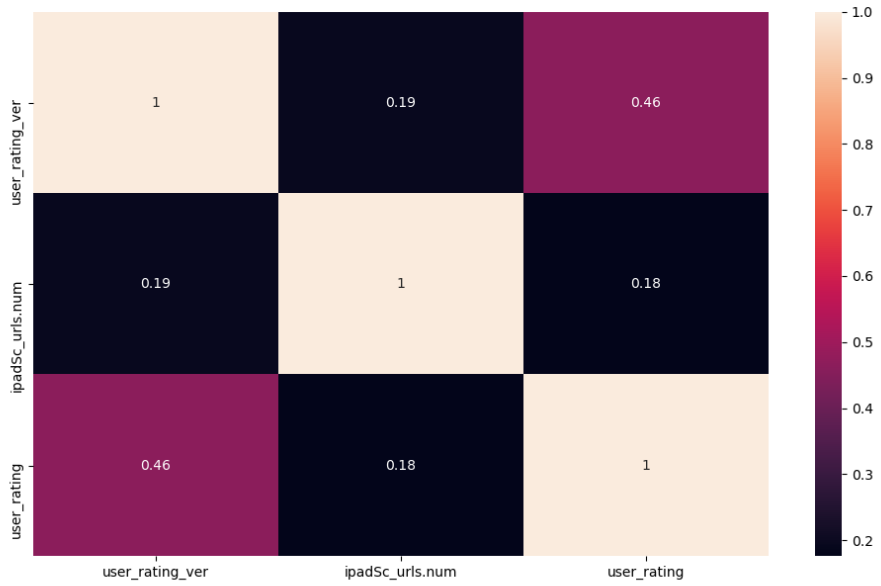
1. ‘prime_genre’ with unique categories:

```
['Games' 'Productivity' 'Weather' 'Shopping' 'Reference' 'Finance' 'Music'
 'Utilities' 'Travel' 'Social Networking' 'Sports' 'Business'
 'Health & Fitness' 'Entertainment' 'Photo & Video' 'Navigation'
 'Education' 'Lifestyle' 'Food & Drink' 'News' 'Book' '0' 'Medical'
 'Catalogs']
```

2. ‘cont_rating’ with unique categories:

```
['4+' '12+' '17+' '9+']
```

- Data correlation:



As shown in the correlation heatmap, the effect of each feature on the others is somehow poor, but the most effective features on **'user_rating'** is:

1. **'user_rating_ver'**
2. **'ipadSc_urls.num'**

So, some features will not be needed to be used in the model like:

1. Id
2. Track_name
3. Currency (It is all the same value)
4. Version

As they do not strongly affect the **'user_rating'.**

- ## Pre-processing techniques:

1. Discarding unnecessary features:

```
data = data.drop(['id', 'track_name', 'currency', 'ver'], axis=1)  # discard unnecessary features
```

Usually there are some data that are not useful to the machine learning model. It does not have an effect on the desired prediction, so it must be dropped in order to have an efficient model.

2. Checking out the missing values:

```
id                   3
track_name          14
size_bytes          19
currency           185
price               17
rating_count_tot    17
rating_count_ver    17
vpp_lic             28
user_rating_ver     17
ver                 18
cont_rating         19
prime_genre         23
sup_devices.num     19
ipadSc_urls.num     18
lang.num            18
user_rating          3
```

The concept of missing values is important to understand in order to successfully manage data. If the missing values are not handled properly, then the model may end up drawing an inaccurate inference about the data. Due to improper handling, the result obtained by the model will differ from ones where the missing values are present. It can be handled by a lot of ways. Here, we will delete a particular row if it has a null value for a particular feature.

And this way is suitable as the dataset has a large number of samples so it will not be highly affected.

```
data.dropna(how='any', inplace=True)  # dropping rows with null values
```

And for the feature ‘user_rating_ver’ as it has a good effect on the ‘user_rating’, its null values will be replaced with the median of the column, they will not be dropped from the data.

```
data['user_rating_ver'] = data['user_rating_ver'].fillna(data['user_rating_ver'].median())
```

3. **Cleaning data with unreasonable values:**

   In the categorical feature **‘prime_genre’**, a category with value **‘0’** was found and it does not have a meaning among the rest of the categories. So, the records with this value should be dropped.

```
i = data[data['prime_genre'] == '0'].index  # get the index of the noisy row which has prime_genre = 0
print(data.loc[i])
data = data.drop(i)  # drop this row
```

4. **Process the categorical data:**

   Since machine learning models are based on mathematical equations and you can intuitively understand that it would cause some problem if we can keep the categorical data in the equations because we want only want numbers in the equations.

   So, we need to encode the categorical variables with numeric values:

   - **cont_rating:** will be encoded using label encoder. Label encoder is an object which is I use to help us in transferring Categorical data into

numerical data. Next, I fitted this object to the column **'cont_rating'** of our matrix X and all this return it encoded. It encodes target labels with values between **0** and **n-1** classes.

```
le = LabelEncoder()
X['cont_rating'] = le.fit_transform(X['cont_rating'])
```

- **prime_genre**: will be encoded using One-Hot Encoder. It's one that takes the value 0 or 1 to indicate the absence or presence of some categorical effect that may be expected to shift the outcome. Instead of having one column, we are going to have n columns (n = #classes).

```
col_trans = make_column_transformer((OneHotEncoder(), ['prime_genre']), remainder='passthrough')
X = col_trans.fit_transform(X)
```

make_column_transformer is a function in **sklearn.compose** that's used to perform some operation on a specific columns in the given matrix. It takes the operation object and the column name as parameters. **(remainder = 'passthrough')** means that you only edit the given column and keep the rest as they are. Then the returned object will be activated through fit_transform that takes the data matrix and returns it after performing the operation.

5. **Feature Scaling:**

It is a method to limit the range of variables so that they can be compared on common grounds. As we see in the dataset below, there are features like **'size_bytes'** and '**rating_count_tot'** that have large different scaled ranges. If feature scaling is not done, then a machine learning algorithm tends to weigh

greater values, higher and consider smaller values as the lower values, regardless of the unit of the values.

B	C	D	E	F	G	H	I	J	K	L
id	track_name	size_bytes	currency	price	rating_count_tot	rating_count_ver	user_rating	ver	cont_rating	prime_genre
281656475	PAC-MAN Premium	100788224	USD	3.99	21292	26	4	6.3.5	4+	Games
281796108	Evernote - stay organiz	158578688	USD	0	161065	26	4	8.2.2	4+	Productivity
281940292	WeatherBug - Local We	100524032	USD	0	188583	2822	3.5	5.0.0	4+	Weather
282614216	eBay: Best App to Buy,	128512000	USD	0	262241	649	4	5.10.0	12+	Shopping
282935706	Bible	92774400	USD	0	985920	5320	4.5	7.5.1	4+	Reference
283619399	Shanghai Mahjong	10485713	USD	0.99	8253	5516	4	1.8	4+	Games
283646709	PayPal - Send and requ	227795968	USD	0	119487	879	4	6.12.0	4+	Finance
284035177	Pandora - Music & Radi	130242560	USD	0	1126879	3594	4	8.4.1	12+	Music
284666222	PCalc - The Best Calcula	49250304	USD	9.99	1117	4	4.5	3.6.6	4+	Utilities
284736660	Ms. PAC-MAN	70023168	USD	3.99	7885	40	4	4.0.4	4+	Games
284791396	Solitaire by MobilityWa	49618944	USD	4.99	76720	4017	4.5	4.10.1	4+	Games
284815117	SCRABBLE Premium	227547136	USD	7.99	105776	166	3.5	5.19.0	4+	Games

The used technique for features scaling in our model is **Standardization**. It is a very effective technique which re-scales a feature value so that it has distribution with 0 mean value and variance equals to 1. By importing **StandardScaler** from **sklearn.preprocessing**:

```
standard = StandardScaler()
X = standard.fit_transform(X)
```

6. **Splitting data into training and testing sets:**

Generally, we split the dataset into 70:30 ratio. It means that 70 percent data take in train and 30 percent data take in test. However, this Splitting can vary according to the dataset shape and size.

```
# Splitting Data
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.30, shuffle=True)
```

x_train: is the training part of the matrix of features.

x_test: is the test part of the matrix of features.

y_train: is the training part of the label values.

y_test: is the test part of the label values.

- ## Regression techniques:

1. **Multiple Linear Regression:**

   Multiple linear regression looks at the relationships within a bunch of information. Instead of just looking at how one thing relates to another thing (simple linear regression).

2. **Polynomial Regression:**

   A regression analysis in which the relationship between the independent variable x and the dependent variable y is modelled as an nth degree polynomial in x. This is still considered to be linear model as the coefficients/weights associated with the features are still linear. x^2 is only a feature. However, the curve that we are fitting is quadratic in nature.

3. **Support Vector Regression (SVR):**

   SVR gives us the flexibility to define how much error is acceptable in our model and will find an appropriate line (or hyperplane in higher dimensions) to fit the data which produces significant accuracy with less computation power.

4. **Ridge Regression:**

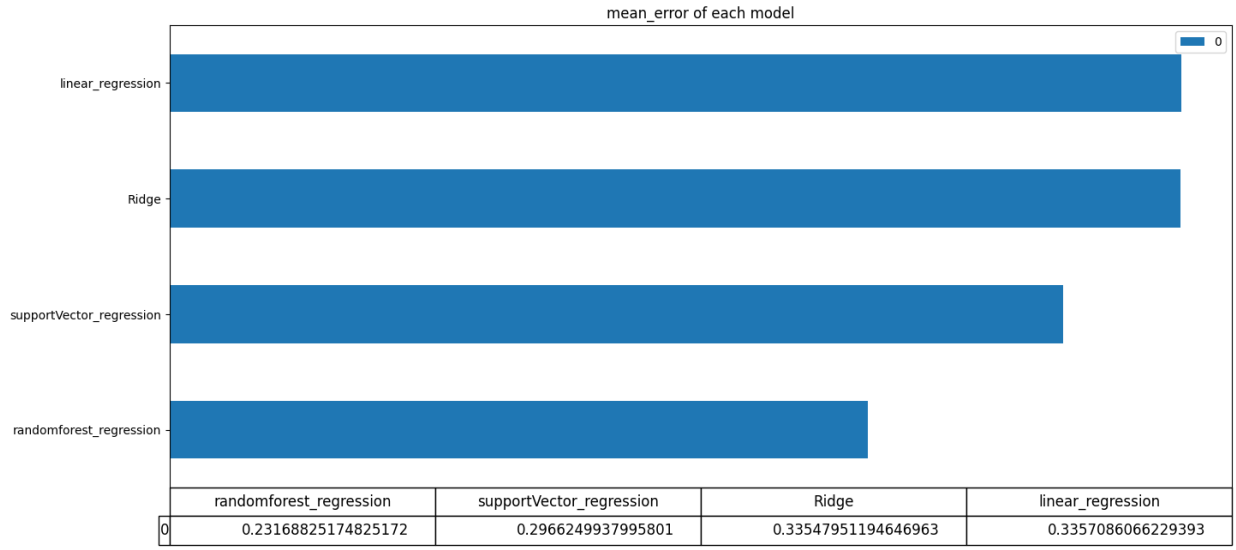
   Ridge regression is an extension of linear regression that adds a regularization penalty to the loss function during training. This penalty has the effect of shrinking the coefficients for those input variables that do not contribute much to the prediction task. The default value is 1.0 or a full penalty.

5. **Random Forest Regression:**

   Random forest Regression is a collection, or ensemble, of several decision trees. Decision trees work by splitting the data into two or more homogeneous sets based on the most significant splitter among the independent variables. The best differentiator is the one that minimizes the cost metric. In a random forest, instead of trying splits on all the features, a sample of features is selected for each split, thereby reducing the variance of the model.

- **Difference between mean square error and training time for each model:**

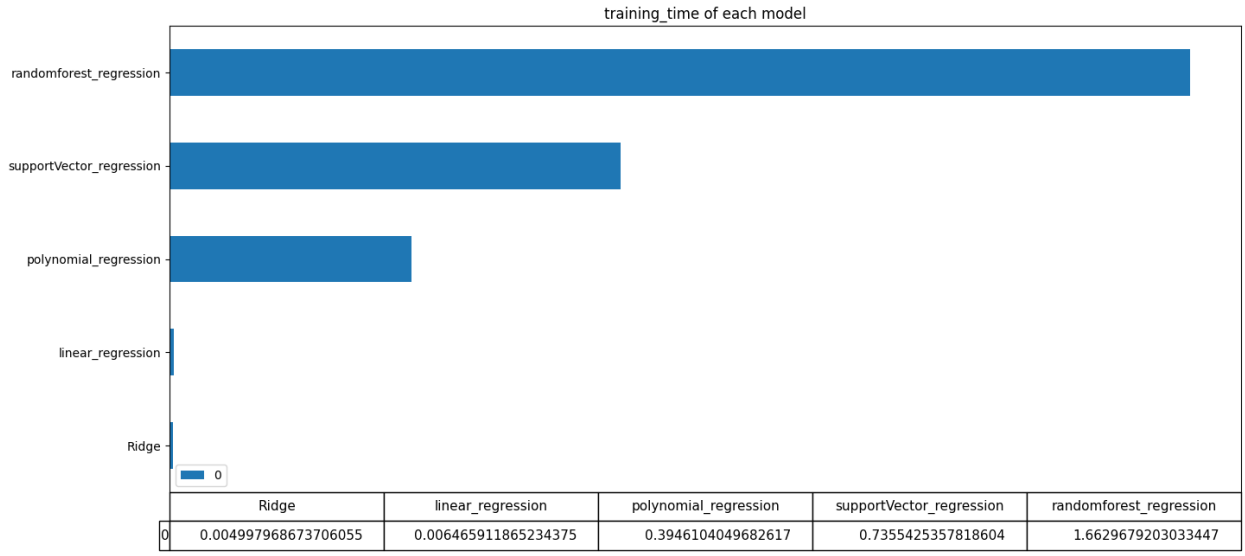
  - **Mean square error:**



	randomforest_regression	supportVector_regression	Ridge	linear_regression
0	0.23168825174825172	0.2966249937995801	0.33547951194646963	0.3357086066229393

	randomforest_regression	supportVector_regression	Ridge	linear_regression	polynomial_regression
0	0.2343245629370629	0.32219514432440455	0.36657009702923565	0.3666150065781729	5.512871570569071e+21

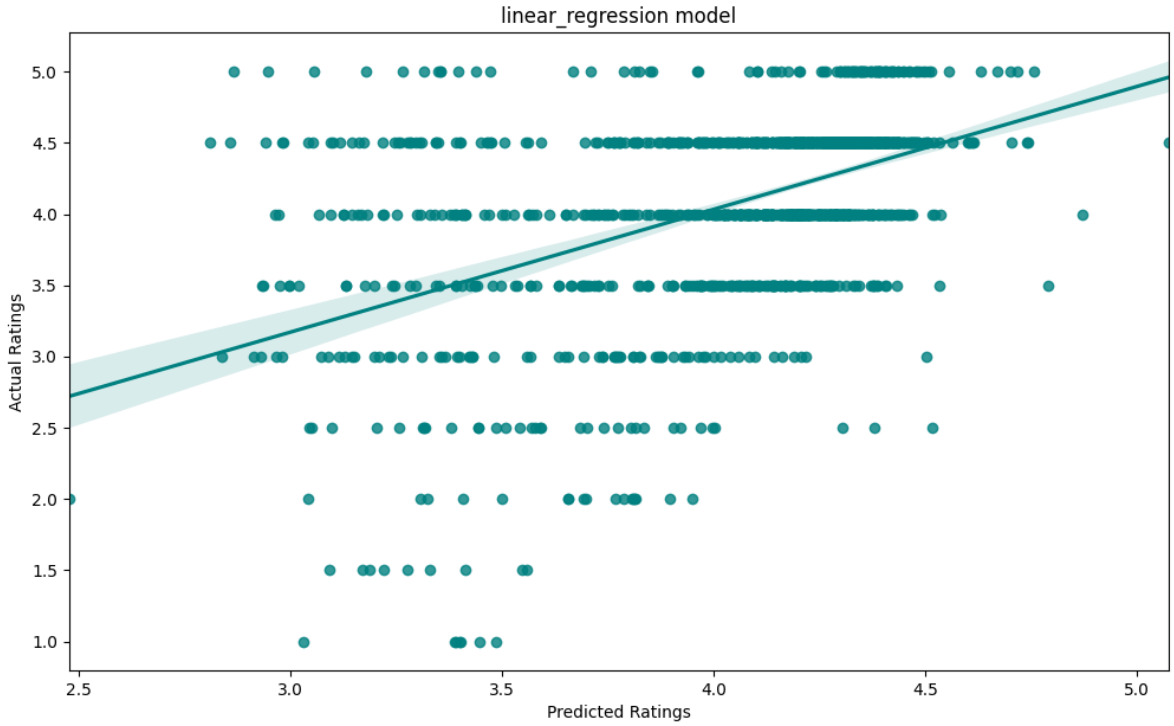
  - **Training time:**



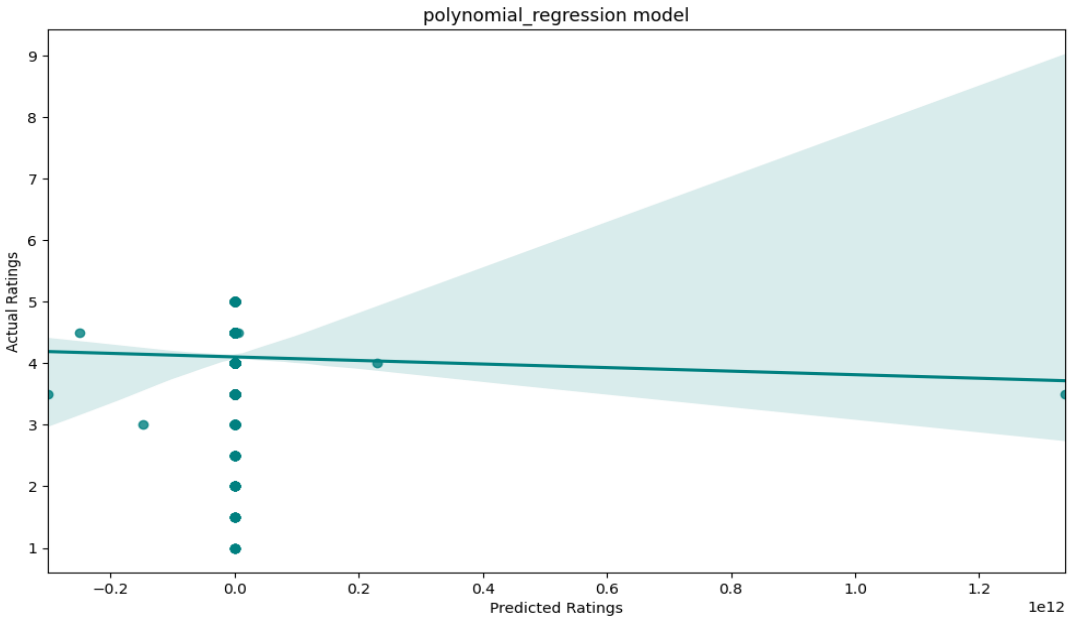
	Ridge	linear_regression	polynomial_regression	supportVector_regression	randomforest_regression
0	0.004997968673706055	0.006465911865234375	0.3946104049682617	0.7355425357818604	1.6629679203033447

- **Screenshots of the resultant regression line plot of each model:**

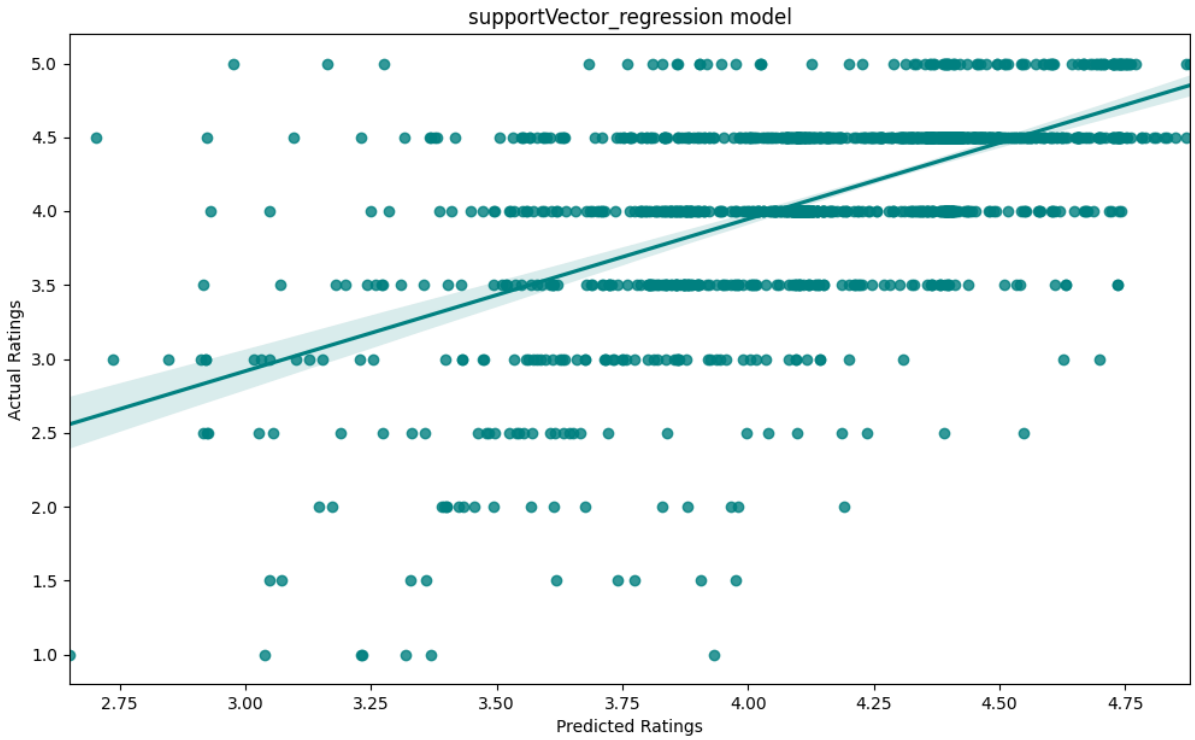
  - **Multiple Linear Regression:**



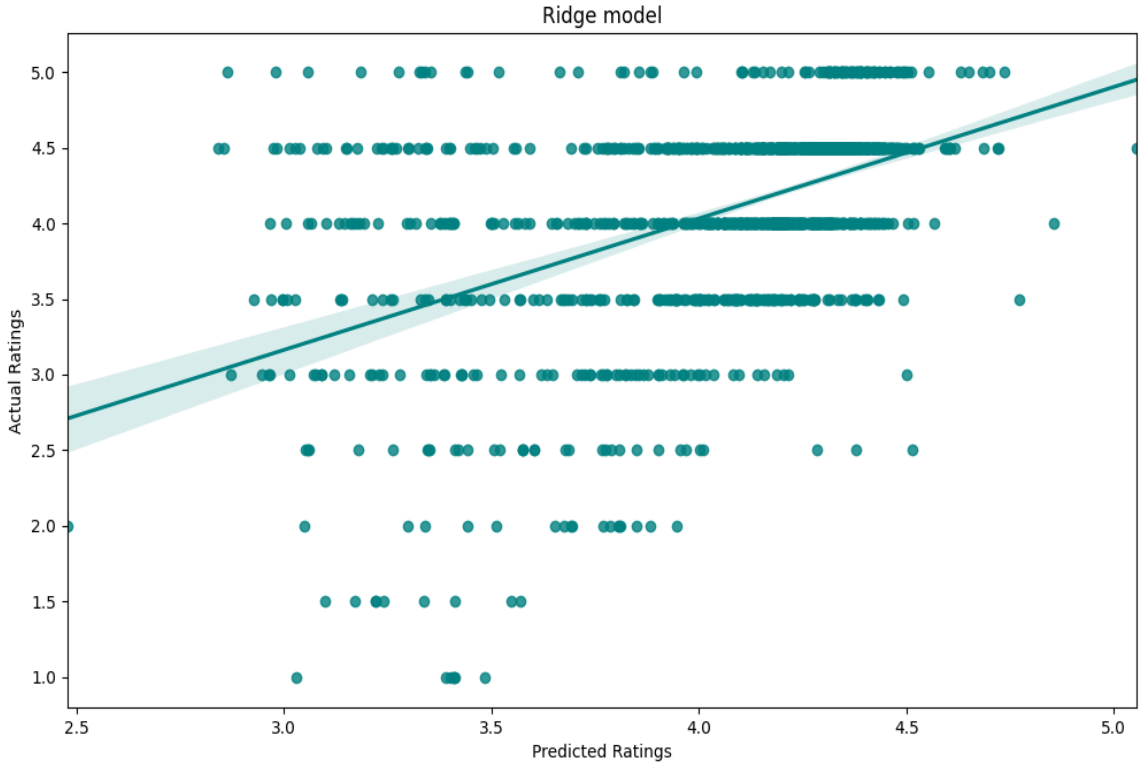
  - **Polynomial Regression:**



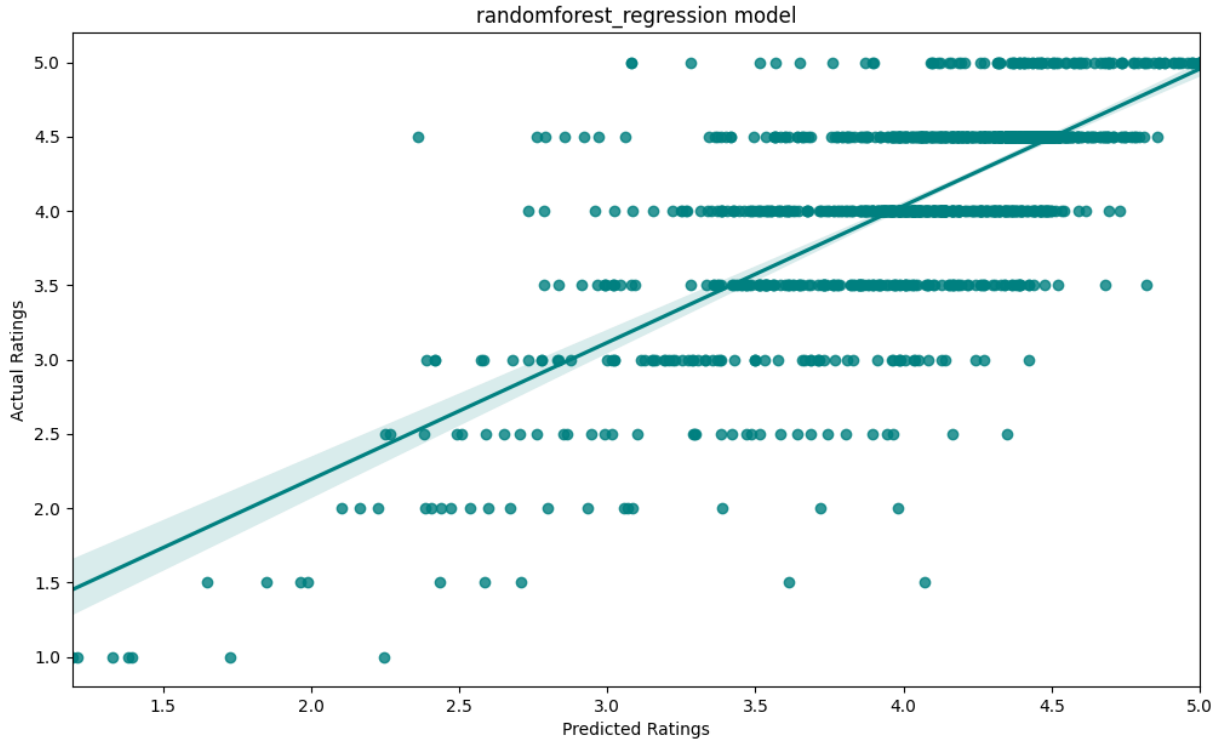
- **Support vector Regression (SVR):**



- **Ridge Regression:**



- **Random forest Regression:**



Conclusion:

In a nutshell, this phase of the project was very useful for us to understand how our mindset should move towards solving these types of machine learning problems. The feature engineering steps are very important because they are the base of solving the problem as well as being a main reason for selecting the appropriate machine learning model that will get the best desired result by using the processed dataset. Also trying several regression techniques helped us to recognize the differences between them and when we should select one of them to solve a specific problem.