FOUNDATIONS OF DATA SCIENCE

ASSIGNMENT 1

Mobile App Success and Rating Prediction Based On App Features Using Regression Model

Submitted by: -

Sreya Venkatesh [2017A7PS0800U]

Anushka Patil [2017A7PS0233U]

CONTENTS

1. INTRODUCTION

Mobile application development is a highly innovative software industry that has turned into an extremely profitable business, with revenues only continuing to rise yearly. Due to immense competition from around the world it is necessary for the app developers to predict the success of their app and whether they are proceeding in the right direction. As most of the apps in the play store are free the revenue generated by the subscriptions, in-app purchases, in-app adverts are practically unknown. The success of an app is usually determined by the user ratings and the number of installs rather than the revenue it generates.

Looking into the revenue growth and history of the google play store, between 2016-17 there was a revenue growth of 34.2% and the rise in app downloads was about 16.7%. The play store gets 30% of the earnings while the developer gets 70%. Building an app is not an easy task as there is always a new app coming in every now and then. Study shows that around one-third of the apps have less than 10K downloads in every region and only 15% crossed the 1000K download mark. Hence if the app has a low number of downloads it has a less chance to do better business in the future android market. This is one of the most important problem that is to be addressed in this paper.

The user sentiments are the most important aspect which decides fate of application. If the comments are positive then there can be a scope of enhancement but if comments are negative, then there is trouble with app like it is not working, security or privacy issue.The sentiment detection of texts has been given a lot of importance lately, due to the increased availability of online reviews in digital form and the need to organize and extract meaning out of them.

Due to the expansion and competition in the mobile app industry which threatens every app developer as the value of their app will degrade with the availability of newer ones, we chose to study and analyze a topic relevant to this sector. Through this we hope to know the success rate of the apps and the features that need to be maintained or changed based on the current status of the app. The main motive of this paper is to find something meaningful through the analysis which might be useful to the developers or the end users.

In this paper the app features of the apps available google play store are used which are made available on Kaggle. Kaggle provides a rich source of information regarding the apps which includes many attributes such as name, category, review, rating, price etc. Based on the features offered and user reviews of the apps the rating shall be predicted with the help of Python libraries and packages like sklearn, statsmodel, seaborn, matplotlib etc.

2. LITERATURE SURVEY

Over the years, many researchers have worked towards publishing articles on the sentiment analysis of web scrapped opinions and rating predictions.

Sentiment detection has various applications in various fields such as review classification, synonyms and antonyms differentiation, search engines, summarizing reviews,survey response analysis etc. Mainly four different problems that are

common in this field are addressed in [9], namely, subjectivity classification, word sentiment classification, document sentiment classification and opinion extraction. Subjectivity classification discards irrelevant and misleading text in the reviews. Document sentiment classification and opinion extraction follow word sentiment classification.

[10] presents a Sentiment Analyzer to extract opinions about a relevant subject from online information and text documents. Sentiment Analyzer finds all the references to the subject and extracts the sentiment using Natural Language Processing techniques. It uses two linguistic resources , namely, sentiment lexicon and sentiment pattern database.

[7] focused on opinion summarization which contains three main tasks. Firstly, the features of a product are extracted and opinions associated with the features are identified in each sentence.Next, the opinion orientation is identified and finally, a structured sentence list accordingly with the opinion and feature pairs is produced. This journal uses maximum entropy model to forecast the relationship between opinion and the relevant product feature. A dependency and semantic based approach is used for opinion mining from online reviews.

[8] summarized the work by not just predicting the average rating but also providing a detailed analysis of the reviews. The paper provides a list of attributes that are relevant to the way in which the developer tries to understand the reviews. In addition to that it provides an average rating that represents general sentiment for that attribute.

In [1] more than 5000 apps from Google play store were taken as the database, obtained from Kaggle and about 170000 reviews were scrapped off the collected URLs of those apps.The sentimental mean was calculated for all the reviews using TextBlob, a Python library. [1] used Random Forest Algorithm as the dataset contained both types of features, numerical as well as categorical.[6] is based on android applications too and uses an expert system that consists of various rules for the classification of the calculated sentiment scores. The sentiments of the users are evaluated using python library tools which gives the scores in the form of polarity as well as subjectivity. This expert system integrated in the form of an android app then classifies the score of the user sentiment and displays on the mobile screen.

A machinery was used by [5] to download and extract features about the apps from Google Play Store and the obtained data set was used for the training of three separate models to predict a mobile application's success. A text classifier based on naive bayes for the description of the application, a generalized linear model to categorize the application as successful or not successful and to predict the rating of the application, linear regression was used.

[2] focused on predicting star ratings by studying around 1,760 annotated reviews in German provided by Sanger. Two different groups of experiments were conducted based on the reviews and they all relied on polarity of the subjective phrases. The first group dealt with understanding the significance of the sentiments used in the reviews while the second group focused on other attributes and features in addition to the result of the first group. Reviews dataset was split into training and test set and each experiment along with Monte Carlo cross-validation was done 10,000 times.

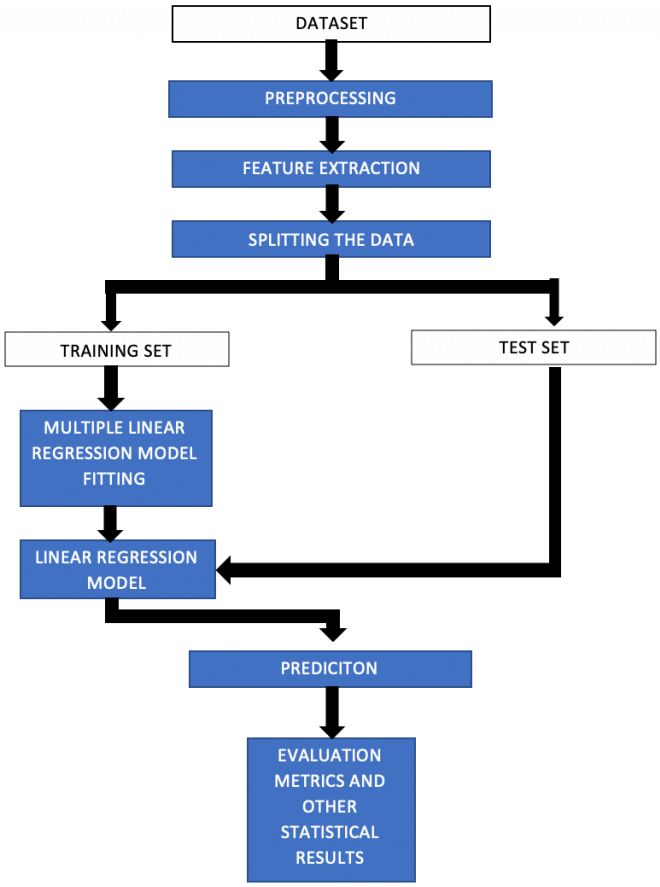
Multi-variate(linear regression based predictors) and Univariate (average based predictors) were studied separately.

[4] built a prediction model using only the claims that developers make about the features. The approach was applied to apps from Blackberry App World and Samsung App Store. Findings suggested the use of app features extraction using Natural Language Processing and Machine learning to predict ratings. The prediction system for rating used a case based reasoning approach. This system requires very little tuning and is easily deployable.

[3] addressed the mismatch between positive sentiments recorded in the reviews and star ratings that suggest otherwise. A survey consisting of data from various Android Apps was conducted to help understand the factors that cause this mismatch. 8600 reviews from 10 of the most popular apps were manually investigated. Machine Learning and Deep learning methods were adopted to automatically detect mismatched ratings with the reviews. Sentiment Score and Readability score were selected as the parameters for Machine Learning. Three models were built, model with handcrafted features, model with word vectors and model using Dependency based Convolutional Neural Network. The later model gave the best results with utmost accuracy.

3. METHOD: MULTIPLE LINEAR REGRESSION

3.1. BLOCK DIAGRAM



3.2. INTRODUCTION TO LINEAR REGRESSION

Multiple linear regression attempts to model the relationship between two or more explanatory(independent) variables and a response (dependent) variable by fitting a linear equation to observed data.

- The dependent features are called the dependent variables, outputs, or responses.
- The independent features are called the independent variables, inputs, or predictors.

It is used to answer whether some phenomeneon is influenced by several other variables. It is also used for forecasting a response.

Two things that we need to consider when we choose the coefficients are

- The independent variable must have a strong correlation with the dependent variable.
- The independent variable should not have a good correlation with any other independent variable.

3.3. BASIC REGRESSION MODEL

Multiple regression estimates the β's in the equation

$$Y=b0 + bx1j + b2xx2j + b3x3j + ... + bpxpj + \varepsilon j$$

where,

- the x's are the independent variables
- Y is the dependent variable
- j represents the observation (row) number
- β's are the unknown regression coefficients ((population parameter)
- εj is the error (residual) of observation j

The j^{th} regression coefficient b represents the expected change in *y* per unit change in j^{th} independent variable X_j . Assuming $E(e) = 0$,

$$b_j = (\partial E(y))/\partial X_j$$

A model is said to be linear when it is linear in parameters. In such a case $(\partial E(y)/\partial X_j$ should not depend on any β's.

3.4. FORMULA

$$\hat{y}_j = b_0 + b_1 x_{1j} + b_2 x_{2j} + ... + b_p x_{pj}$$

where,

- b is an estimate of the unknown regression coefficient β
- ŷ is the estimated value of the dependent variable Y

Although the regression problem may be solved by a number of techniques, the most-used method is least squares. In least squares regression analysis, the b's are selected so as to minimize the sum of the squared residuals. This set of b's is not necessarily the set you want, since they may be distorted by outliers--points that are not representative of the data. Robust regression, an alternative to least squares, seeks to reduce the influence of outliers.

3.5. DATASET DESCRIPTION

The dataset, 'Google Play Store Apps' was obtained from Kaggle and used for this study.

- No of observation (rows): 10840
- Attributes (columns): 13

Independent variables:

i. App: This contains the application name
ii. Category: Category of the app
iii. Reviews: No. of user reviews
iv. Size: Size of the app
v. Installs: Number of user installs
vi. Type: Paid or Free
vii. Price: Price of the app
viii. Content Rating: Age group the app is targeted at - Children / Mature 21+ / Adult
ix. Genres: multiple genres (For eg, a game can belong to Music, Game, Family genres.
x. Last Updated: Date when the app was last updated
xi. Current Ver: Current version of the app available on Play Store
xii. Android Ver: Min required Android version

Dependent variable:
Rating: Overall user rating of the app

3.6. APPROACH USED IN THIS REPORT

In this report we would be solving the problem with two methods using, Scikit -learn and statsmodel.

- We will start by fitting the model using SKLearn. After we fit the model, unlike with statsmodels, SKLearn does not automatically print the concepts or have a method like summary. So we have to print the coefficients,intercepts etc. separately.
- After fitting the model with SKLearn, we fit the model using statsmodels. Unlike SKLearn, statsmodels doesn't automatically fit a constant, so you need

to use the method sm.add_constant(X) in order to add a constant. Adding a constant, while not necessary, makes your line fit much better.

- Coefficients can be obtained pretty easily from SKLearn, so the main benefit of statsmodels is the other statistics it provides.
- One of the assumptions of a simple linear regression model is normality of our data.The statistics in the summary table in statsmodel are testing the normality of our data.
- If the Prob(Omnibus) is very small, and we took this to mean <.05 as this is standard statistical practice, then our data is not normal. This is a more precise way than graphing our data to determine if our data is normal.

Therefore, SKLearn has more useful features, but statsmodels is a good method to analyze your data before

3.7 BUILDING THE MULTIPLE REGRESSION MODEL

3.7.1. IMPORTING LIBRARIES

```
[ ] #Importing libraries

    import pandas as pd
    import numpy as np
    import seaborn as sns

    from sklearn import metrics

    import matplotlib.pyplot as plt
    %matplotlib inline

    import random

    from sklearn.linear_model import LinearRegression
    from sklearn.model_selection import train_test_split
```

3.7.2. IMPORTING THE DATASET FILE BY INTEGRATING KAGGLE IN COLAB

Next few lines are needed to integrate the kaggle dataset (Google Play Store Apps) into google colab

```
!pip install --quiet kaggle
```

```
[ ] from google.colab import files
    files.upload() #upload the json file that contains api key from kaggle account
```

Choose Files | no files selected — Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving kaggle.json to kaggle.json
{'kaggle.json': b'{"username":"anushkapatil","key":"9c2ef954c959036ef0d22a462cf32d3b"}'}
```

```
[ ] !mkdir -p ~/.kaggle
    !cp kaggle.json ~/.kaggle/.

    !chmod 600 ~/.kaggle/kaggle.json #altering permissions
```

```
[ ] !kaggle datasets download -d lava18/google-play-store-apps #this is the api of the dataset obtained from kaggle
```

```
Downloading google-play-store-apps.zip to /content
  0% 0.00/1.94M [00:00<?, ?B/s]
100% 1.94M/1.94M [00:00<00:00, 61.0MB/s]
```

```
[ ] from zipfile import ZipFile
    zip_file= ZipFile('google-play-store-apps.zip')  #this downloaded zip file contains three csv file
    data=pd.read_csv(zip_file.open('googleplaystore.csv'))  #we choose the googleplaystore.csv and load it into a data
```

3.7.3 DATA DESCRIPTION AND CHECKING FOR MISSING VALUES

'non-null' implies there are no missing values. Now, linear regression can be done on numerical or continuous attributes but not on object type attributes such as app, category, reviews, size, installs etc. Therefore, we would be converting these object type variables into int values (0/1) in preprocessing section.

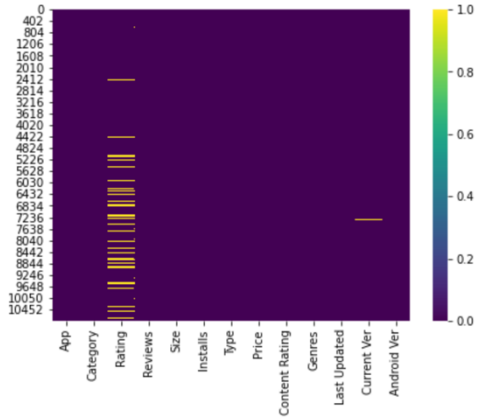
```
[ ] data.info() #result shows there are 10841 entries in the dataframe , it also lists the columns present in the dataset
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10841 entries, 0 to 10840
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   App             10841 non-null  object 
 1   Category        10841 non-null  object 
 2   Rating          9367 non-null   float64
 3   Reviews         10841 non-null  object 
 4   Size            10841 non-null  object 
 5   Installs        10841 non-null  object 
 6   Type            10840 non-null  object 
 7   Price           10841 non-null  object 
 8   Content Rating  10840 non-null  object 
 9   Genres          10841 non-null  object 
 10  Last Updated    10841 non-null  object 
 11  Current Ver     10833 non-null  object 
 12  Android Ver     10838 non-null  object 
dtypes: float64(1), object(12)
memory usage: 1.1+ MB
```

The following screenshot shows that rating, type, price, content rating, current ver and android ver have missing values.

```
plt.figure(figsize=(7, 5))
sns.heatmap(data.isnull(), cmap='viridis')
data.isnull().any() #shows that Rating,Type,Content Rating,Current Ver and Android Ver cloumns have misisng value
```

```
App               False
Category          False
Rating             True
Reviews           False
Size              False
Installs          False
Type               True
Price             False
Content Rating     True
Genres            False
Last Updated      False
Current Ver        True
Android Ver        True
dtype: bool
```



```
data.isnull().sum() # shows the number of missing value in each column respectively
```

```
App                0
Category           0
Rating          1474
Reviews            0
Size               0
Installs           0
Type               1
Price              0
Content Rating     1
Genres             0
Last Updated       0
Current Ver        8
Android Ver        3
dtype: int64
```

3.7.4 DEALING WITH MISSING VALUES

We replace the missing Rating values by the median of all Rating values. Also, we remove the records where Type, Current Ver, Content Rating and Android ver is null.

```
[11] data['Rating'] = data['Rating'].fillna(data['Rating'].median()) #we replace the missing values of column Rating by the median of all Rating values
```

```
[12] #we remove all the entries that have missing column values in Current Ver,Content Rating, Android Ver and Type
     #we remove these entries corresponding to these columns as they have very few missing values
     data = data[pd.notnull(data['Current Ver'])]
     data = data[pd.notnull(data['Content Rating'])]
     data = data[pd.notnull(data['Android Ver'])]
     data = data[pd.notnull(data['Type'])]
```

3.7.5 CLEANING ALL ATTARIBUTES

- **Size**

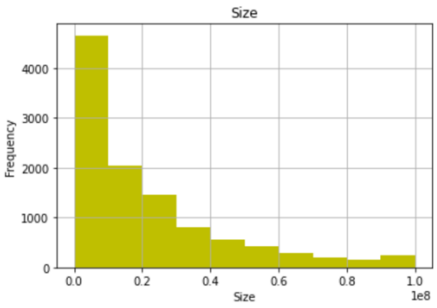
We define a function to convert the size of all the apps in bytes and then fill the missing values.

```
[14] #function to convert MB and KB in bytes
     def change_size(size):
         if 'M' in size:
             x = size[:-1]
             x = float(x)*1000000
             return(x)
         elif 'k' == size[-1:]:
             x = size[:-1]
             x = float(x)*1000
             return(x)
         else:
             return None

     data["Size"] = data["Size"].map(change_size) #update the Size column with these new values
```

```
[15] data.Size.fillna(method = 'ffill', inplace = True) #filling null values
```

```
data.hist(column='Size', color='y')
plt.xlabel('Size')
plt.ylabel('Frequency')
```

```
Text(0, 0.5, 'Frequency')
```



- **Installs**

We remove all the symbols present in the data such as ‘,’ and ‘+’, and then convert it into type float.

```
[17] data.Installs.value_counts()
```

```
1,000,000+        1578
10,000,000+       1252
100,000+          1169
10,000+           1052
1,000+             905
5,000,000+         752
100+               718
500,000+           538
50,000+            478
5,000+             476
100,000,000+       409
10+                385
500+               330
50,000,000+        289
50+                205
5+                  82
500,000,000+        72
1+                  67
1,000,000,000+      58
0+                  14
Name: Installs, dtype: int64
```

```
[18]  #as linear regression deals with float values, we will remove any additional symbols present
      data.Installs=data.Installs.apply(lambda x: x.strip('+'))  #remove symbol '+'
      data.Installs=data.Installs.apply(lambda x: x.replace(',','')) #remove symbol ','
```

```
[19] data['Installs'] = data['Installs'].astype(float)
```

- **Reviews**

We convert object type reviews to type int which stores the number of reviews.

```
[21] data.Reviews.str.isnumeric().sum() #checking if all 10829 Reviews are numeric values
```

```
10829
```

```
[22] data['Reviews'] = data['Reviews'].astype(int) #converting 'object' type Reviews to type 'int'
```

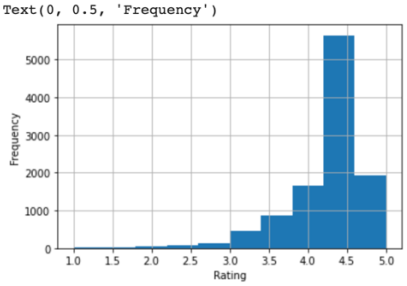
- **Rating**

```
[23] #checking the range of the values of the Rating column
     print("Range: ", data.Rating.min(),"-",data.Rating.max())
```

```
Range:  1.0 - 5.0
```

```
[24] data.Rating.hist();
     plt.xlabel('Rating')
     plt.ylabel('Frequency')
```

```
Text(0, 0.5, 'Frequency')
```



- **Type**

Free apps are denoted as ‘0’ and paid apps are denoted as ‘1’ using the function shown in the screenshot.

```
[25] data.Type.value_counts() #Prints the number of Free and Paid app
```

```
Free    10032
Paid      797
Name: Type, dtype: int64
```

```
[26] #Function that converts the Type value to '0' for free app and '1' for paid app
     def type_cat(types):
         if types == 'Free':
             return 0
         else:
             return 1

     data['Type'] = data['Type'].map(type_cat) #updated type value
```

- **Price**

We remove the symbol ‘$’ and convert it into a float value.

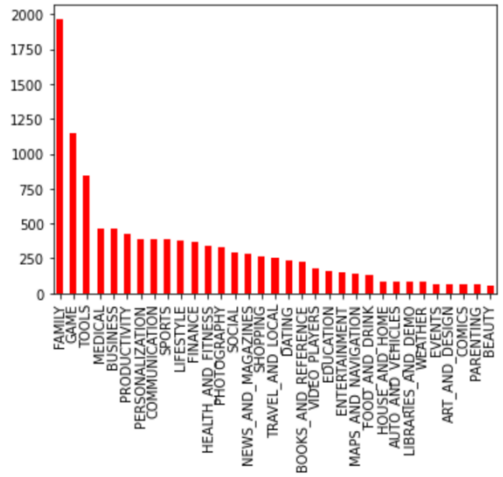
```
[27] data.Price=data.Price.apply(lambda x: x.strip('$'))  #removing the symbol '$'
     data['Price'] = data['Price'].astype(float)
```

- **Category**

Here, we make a new column ‘Category_new’ which stores unique dummy integer values for unique categories.

```
[29] data.Category.value_counts().plot(kind='bar',color='r')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcd287fae10>
```

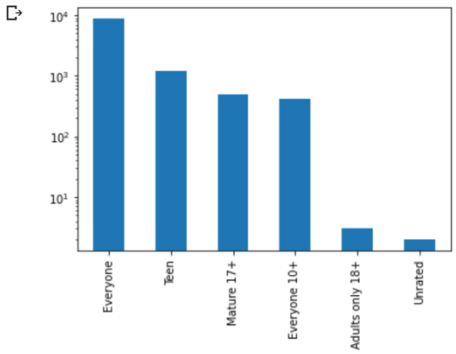



```
[30] #giving discrete dummy values to discrete Categories and adding them in a new Column 'Category_new'
     CategoryL = data.Category.unique()
     CategoryDict = {}
     for i in range(len(CategoryL)):
         CategoryDict[CategoryL[i]] = i
     data['Category_new'] = data['Category'].map(CategoryDict).astype(int)
```

- **Content Rating**

We replace the object type values in the column with dummy integers for different Ratings.

```
[32] data.columns = data.columns.str.replace(' ', '_') #for ex: replacing column name 'Content Rating' with 'Content_Rating'
     data.Content_Rating.value_counts().plot(kind='bar')
     plt.yscale('log')
```




```
[33] #giving discrete dummy values to discrete Content Rating and updating them in the column
     RatingL = data['Content_Rating'].unique()
     RatingDict = {}
     for i in range(len(RatingL)):
         RatingDict[RatingL[i]] = i
     data['Content_Rating'] = data['Content_Rating'].map(RatingDict).astype(int)
```

- **Genres**

Here, we make a new column ‘Genres_new’ which stores unique dummy integer values for unique Genres.

```
[34] #giving discrete dummy values to discrete Genres and adding them in a new Column 'Genre_new'
     GenresL = data.Genres.unique()
     GenresDict = {}
     for i in range(len(GenresL)):
         GenresDict[GenresL[i]] = i
     data['Genres_new'] = data['Genres'].map(GenresDict).astype(int)
```

- **Remaining**

Remaining columns are not relevant for our regression model, and hence are dropped from the data frame.

```
[35] #dropping the columns that are not relevant for our linear regression
     data.drop(labels = ['Last_Updated','Current_Ver','Android_Ver','App'], axis = 1, inplace = True)
```

3.7.6 FINAL DATABASE

```
[36] data.head() #first 5 entries of the updated dataframe
```

	Category	Rating	Reviews	Size	Installs	Type	Price	Content_Rating	Genres	Category_new	Genres_new
0	ART_AND_DESIGN	4.1	159	19000000.0	10000.0	0	0.0	0	Art & Design	0	0
1	ART_AND_DESIGN	3.9	967	14000000.0	500000.0	0	0.0	0	Art & Design;Pretend Play	0	1
2	ART_AND_DESIGN	4.7	87510	8700000.0	5000000.0	0	0.0	0	Art & Design	0	0
3	ART_AND_DESIGN	4.5	215644	25000000.0	50000000.0	0	0.0	1	Art & Design	0	0
4	ART_AND_DESIGN	4.3	967	2800000.0	100000.0	0	0.0	0	Art & Design;Creativity	0	2

3.7.7. MODEL FIT USING SKLEARN

Irrelevant and dependent attributes are dropped from 'X' which stores all the independent variables.

The dataset is split into train and test set. The model is trained using train test and then values are predicted using this model with test set as input.

- **Train set**: 70%
- **Test set**: 30%

Then, we train the model using LinearRegression() function.

```
[85] X = data.drop(labels = ['Category','Rating','Genres'],axis = 1) #We remove the irrelevant columns
     Y = data.Rating #Rating column is to be predicted and is assigned to Y
```

```
[86] X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.30) #split 10829 entries into training sample(70%) amd test sample(30%)
     model = LinearRegression() #model type will be linear regression
```

```
[87] model.fit(X_train,Y_train) #fitting the model with the training set
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

- Intercept and coefficient values

```
print('Intercept: \n', model.intercept_) #value of b0
```

```
Intercept:
 4.188486360330809
```

```
coeff_df = pd.DataFrame(model.coef_, ['Reviews','Size','Installs','Type', 'Price', 'Content_Rating','Category_new','Genres_new'], columns=['Coefficient'])
coeff_df
#these coefficients are the values of b1,b2,b3....b8 respectively and they tell how the nature of dependence of Rating on these column attributes
#if the coefficient is positive/negative then Rating increases/decreases as the value of the attribute increases
```

	Coefficient
Reviews	7.715016e-09
Size	1.294742e-09
Installs	9.842772e-11
Type	1.187820e-01
Price	-7.925088e-04
Content_Rating	-9.658003e-03
Category_new	-4.471535e-04
Genres_new	-2.641828e-04

- Predicted rating values

```
Y_pred = model.predict(X_test) # Ratings are predicted using the regression model and saved in Y_pred
Y_pred
```

```
array([4.15857632, 4.19910952, 4.19990655, ..., 4.28652655, 4.27709548,
       4.25492724])
```

3.7.8. MODEL FITTING USING STATSMODEL, LEAST SQUARE METHOD

To perform this, all values are converted into float64 type first.

```
import statsmodels.api as sm
X_opt = sm.add_constant(X) #constant column is needed in this method for b0 calculation
regressor_OLS = sm.OLS(endog = Y, exog = X_opt).fit() #fitting the model 'regressor_OLS'
regressor_OLS.summary()
```

OLS Regression Results

Dep. Variable:	Rating	**R-squared:**	0.012
Model:	OLS	**Adj. R-squared:**	0.011
Method:	Least Squares	**F-statistic:**	15.77
Date:	Tue, 21 Apr 2020	**Prob (F-statistic):**	2.43e-23
Time:	16:13:11	**Log-Likelihood:**	-7364.8
No. Observations:	10829	**AIC:**	1.475e+04
Df Residuals:	10820	**BIC:**	1.481e+04
Df Model:	8		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	4.2006	0.014	308.972	0.000	4.174	4.227
Reviews	7.958e-09	2.07e-09	3.853	0.000	3.91e-09	1.2e-08
Size	1.349e-09	2.1e-10	6.434	0.000	9.38e-10	1.76e-09
Installs	6.903e-11	7.06e-11	0.978	0.328	-6.93e-11	2.07e-10
Type	0.0944	0.018	5.204	0.000	0.059	0.130
Price	-0.0009	0.000	-3.046	0.002	-0.001	-0.000
Content_Rating	-0.0122	0.006	-1.988	0.047	-0.024	-0.000
Category_new	-0.0012	0.001	-1.048	0.295	-0.003	0.001
Genres_new	-0.0002	0.000	-0.698	0.485	-0.001	0.000

Omnibus:	4855.350	**Durbin-Watson:**	1.804
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	31883.647
Skew:	-2.050	**Prob(JB):**	0.00
Kurtosis:	10.338	**Cond. No.**	3.44e+08

Another model is built, but this time test size is 20% and train set being 80%.

```
X_train2, X_test2, Y_train2, Y_test2 = train_test_split(X_opt, Y, test_size = 0.2, random_state = 0)
model2 = LinearRegression() #building new model
model2.fit(X_train2, Y_train2)
Y_pred2 = model2.predict(X_test2)
Y_pred2
```

```
array([4.20618877, 4.31811391, 4.22513077, ..., 4.28519578, 4.27508703,
       4.20205039])
```

3.7.9 RESIDUAL ANALYSIS OF THE MODEL BUILT USING STATSMODEL PACKAGE

- Calculate residual metrics

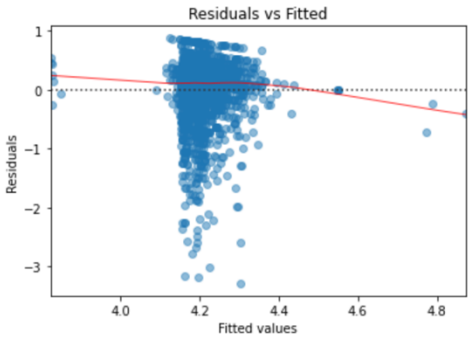
```
residuals= Y_test2-Y_pred2
```

```
# normalized residuals
model_norm_residuals = regressor_OLS.get_influence().resid_studentized_internal
# absolute squared normalized residuals
model_norm_residuals_abs_sqrt = np.sqrt(np.abs(model_norm_residuals))
# absolute residuals
model_abs_resid = np.abs(residuals)
# leverage, from statsmodels internals
model_leverage = regressor_OLS.get_influence().hat_matrix_diag
```

- RESIDUAL Vs. FITTED values graph

When conducting a residual analysis, a "**residuals versus fits plot**" is the most frequently created plot. It is a scatter plot of residuals on the *y* axis and fitted values (estimated responses) on the *x* axis. The plot is used to detect non-linearity, unequal error variances, and outliers.

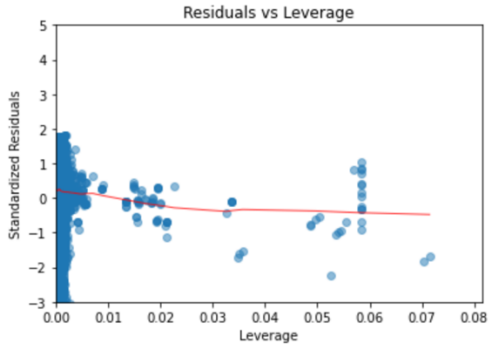
```
plot1 = plt.figure()
plot1.axes[0] = sns.residplot(Y_pred2, residuals,
                          lowess=True,
                          scatter_kws={'alpha': 0.5},
                          line_kws={'color': 'red', 'lw': 1, 'alpha': 0.8})

plot1.axes[0].set_title('Residuals vs Fitted')
plot1.axes[0].set_xlabel('Fitted values')
plot1.axes[0].set_ylabel('Residuals');
```



- RESIDUAL Vs. LEVERAGE GRAPHS

The Residuals vs. Leverage plots helps you identify influential data points on your model. Outliers can be influential, though they don't necessarily have to it and some points within a normal range in your model

```
plot2 = plt.figure();
plt.scatter(model_leverage, model_norm_residuals, alpha=0.5);
sns.regplot(model_leverage, model_norm_residuals,
              scatter=False,
              ci=False,
              lowess=True,
              line_kws={'color': 'red', 'lw': 1, 'alpha': 0.8});
plot2.axes[0].set_xlim(0, max(model_leverage)+0.01)
plot2.axes[0].set_ylim(-3, 5)
plot2.axes[0].set_title('Residuals vs Leverage')
plot2.axes[0].set_xlabel('Leverage')
plot2.axes[0].set_ylabel('Standardized Residuals');
```



4. STATISTICAL RESULTS

4.1. Model MADE USING SKLEARN

Refer to section 3.7.7 to find out how we obtained these values.

Attribute	Var.	value
Intercept	b_0	4.1884
Reviews	b_1	7.715e-09
Size	b_2	1.2947e-09
Installs	b_3	9.842e-11
Type	b_4	1.1878e-01
Price	b_5	-7.925e-04
Content_Rating	b_6	-9.658e-03
Category_new	b_7	-4.47e-04
Genres_new	b_8	-2.64e-04

```
print('Intercept: \n', model.intercept_) #value of b0
```

```
Intercept:
 4.188486360330809
```

```
coeff_df = pd.DataFrame(model.coef_, ['Reviews','Size','Installs'
coeff_df
#these coefficients are the values of b1,b2,b3....b8 respectively
#if the coefficient is positive/negative then Rating increases/de
```

	Coefficient
Reviews	7.715016e-09
Size	1.294742e-09
Installs	9.842772e-11
Type	1.187820e-01
Price	-7.925088e-04
Content_Rating	-9.658003e-03
Category_new	-4.471535e-04
Genres_new	-2.641828e-04

Therefore, the equation is:

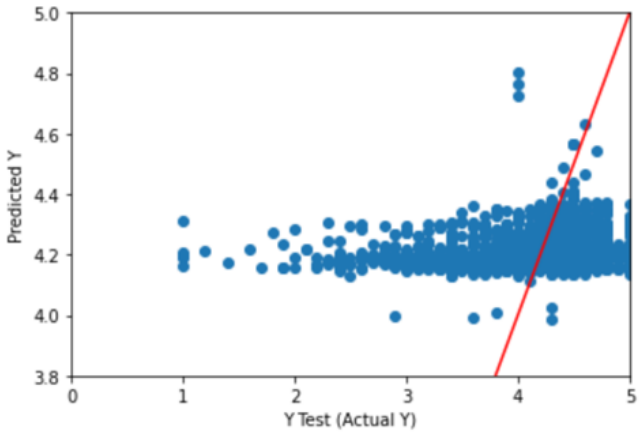
$$\hat{y}_j = 4.1884 + 7.715e\text{-}09x_1 + 1.2947e\text{-}09x_2 + 9.842e\text{-}11\,x_3 + 1.1878e\text{-}01x_4 - 7.925e\text{-}04x_5 - 9.658e\text{-}03x_6 - 4.47e\text{-}04x_7 - 2.64e\text{-}04x_8$$

- We can say that the line intercepts on the positive y axis.
- With one unit increase in reviews, the rating increase by 7.715e-09 units when all other attributes are kept constant and similarly for all others.
- Type attribute is '1' if an app is paid and '0' if it's free. Therefore, for a paid app, rating increases by a factor of 1.1878e-01 units.

- **Comparison b/w actual Rating(Y_test) and predicted Rating(Y_pred)**

```
plt.scatter(Y_test,Y_pred)

plt.ylim(3.8,5)
plt.xlim(0,5)

x = np.linspace(0, 5, 30)
plt.plot(x, x + 0,'-r', linestyle='solid')

plt.xlabel('Y Test (Actual Y)')
plt.ylabel('Predicted Y')
```

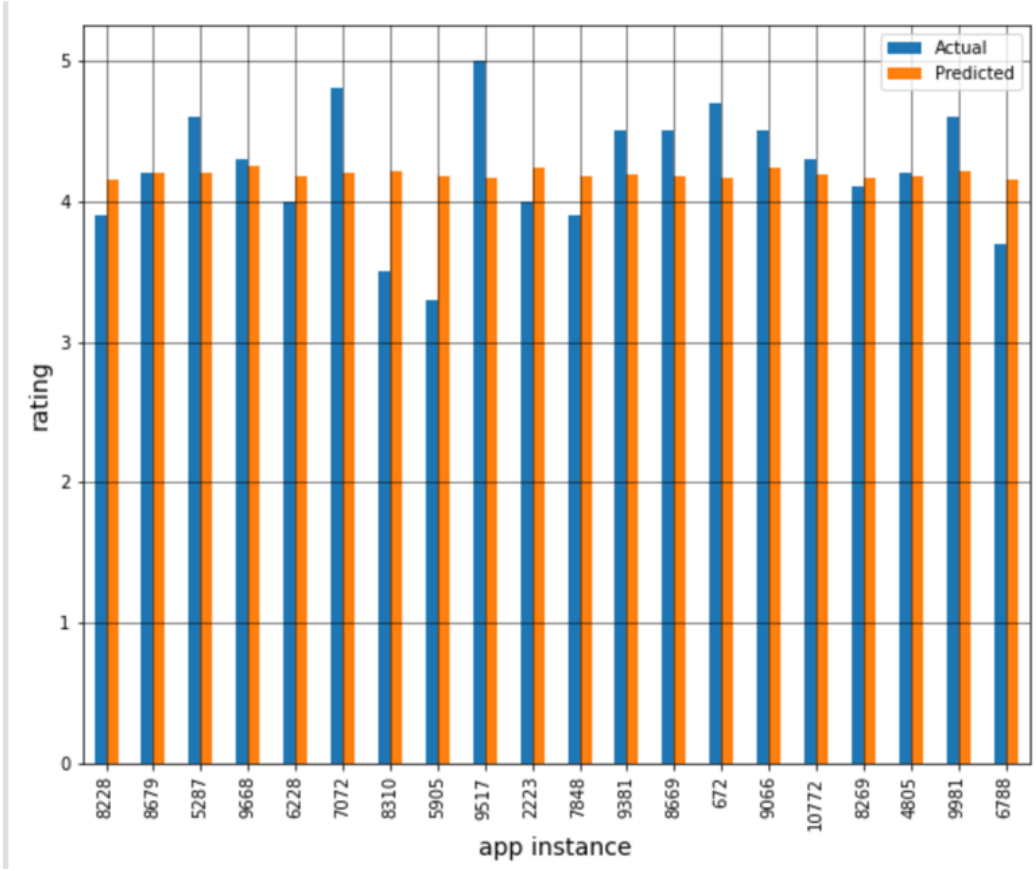
```
Text(0, 0.5, 'Predicted Y')
```



- Following bar graph shows the comparison between actual and predicted rating values for the first 20 app records

```
df1 = pd.DataFrame({'Actual': Y_test, 'Predicted': Y_pred})
df2=df1.head(20)
```

```
df2.plot(kind='bar',figsize=(10,8)) #actual vs predicted Rating values
plt.grid(which='major', linestyle='-', linewidth='0.5', color='black')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='white')
plt.xlabel('app instance',color='black',fontsize=14)
plt.ylabel('rating',color='black',fontsize = 14)

plt.show()
```



- **Evaluation metrics**

```
print ('Mean Squared Error: '+ str(metrics.mean_squared_error(Y_test,Y_pred)))
print ('Mean absolute Error: '+ str(metrics.mean_absolute_error(Y_test,Y_pred)))
print ('Mean squared Log Error: '+ str(metrics.mean_squared_log_error(Y_test,Y_pred)))
```

```
Mean Squared Error: 0.21092912419568813
Mean absolute Error: 0.3154977864080546
Mean squared Log Error: 0.009971592215083131
```

4.2. regressor_OLS MODEL MADE USING STATSMODEL

OLS Regression Results

Dep. Variable:	Rating	**R-squared:**	0.012
Model:	OLS	**Adj. R-squared:**	0.011
Method:	Least Squares	**F-statistic:**	15.77
Date:	Tue, 21 Apr 2020	**Prob (F-statistic):**	2.43e-23
Time:	16:13:11	**Log-Likelihood:**	-7364.8
No. Observations:	10829	**AIC:**	1.475e+04
Df Residuals:	10820	**BIC:**	1.481e+04
Df Model:	8		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	4.2006	0.014	308.972	0.000	4.174	4.227
Reviews	7.958e-09	2.07e-09	3.853	0.000	3.91e-09	1.2e-08
Size	1.349e-09	2.1e-10	6.434	0.000	9.38e-10	1.76e-09
Installs	6.903e-11	7.06e-11	0.978	0.328	-6.93e-11	2.07e-10
Type	0.0944	0.018	5.204	0.000	0.059	0.130
Price	-0.0009	0.000	-3.046	0.002	-0.001	-0.000
Content_Rating	-0.0122	0.006	-1.988	0.047	-0.024	-0.000
Category_new	-0.0012	0.001	-1.048	0.295	-0.003	0.001
Genres_new	-0.0002	0.000	-0.698	0.485	-0.001	0.000

Omnibus:	4855.350	**Durbin-Watson:**	1.804
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	31883.647
Skew:	-2.050	**Prob(JB):**	0.00
Kurtosis:	10.338	**Cond. No.**	3.44e+08

- **R^2 value** of 0.012 is obtained, which means only 1.2% of the variability in the dependent variable (Ratings) can be explained by the independent variables.

- The P-value for each term tests the null hypothesis that the coefficient is equal to zero (no effect). A low P-value (< 0.05) indicates that you can reject the null hypothesis. In other words, a predictor that has a low P-value is likely to be a meaningful addition to your model because changes in the predictor's value are related to changes in the response variable.

- From the table, looking at the P-values it shows that attributes such as 'Installs', 'Category_new' and 'Genres_new', may not be meaningful addition to our model.

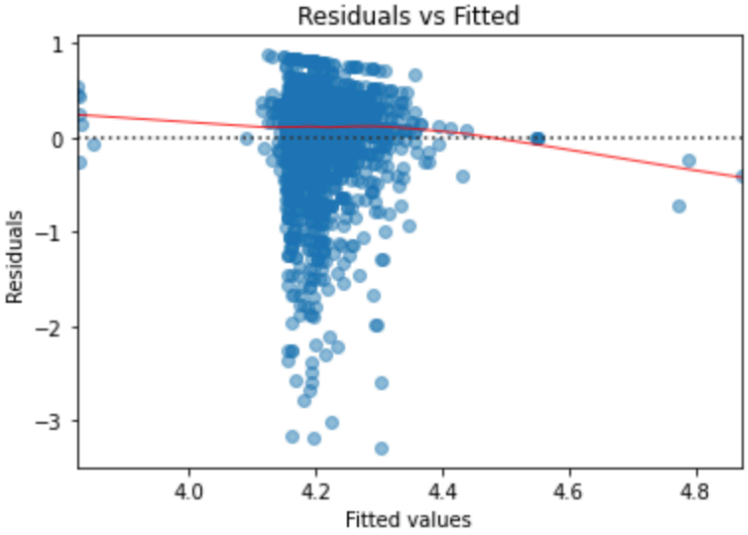
- F-test:
  i) H_0: $\beta_1 = \beta_2 = ... = \beta_p = 0$
  ii) H_1: at least one $\beta_i \neq 0$, i = 1, ..., k
  F-statistic being a huge value, we can reject H_0 that means all beta's have a non-zero value. All the independent variables define variability in the dependent variable.

4.3. RESIDUAL ANALYSIS

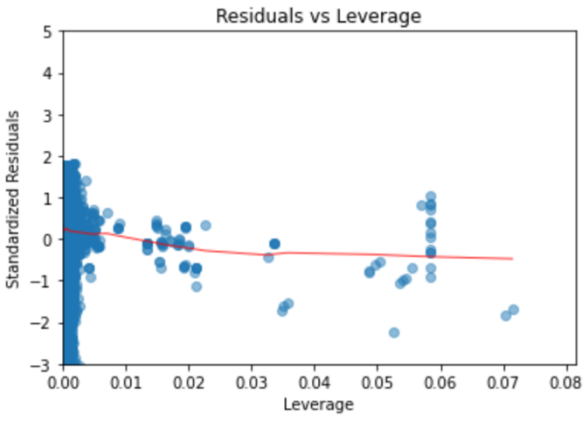
4.3.1. RESIDUAL Vs. FITTED

A good residual vs fitted plot has three characteristics:

- The residuals "bounce randomly" around the 0 line. This suggests that the assumption that the relationship is linear is reasonable. (some of our residual values are far away from the 0 line)
- The residuals values should be equally and randomly spaced around the 0 line.(this is not the ideal case as our fit is not a straight line. This can be so because of the outliers present)
- No one residual "stands out" from the basic random pattern of residuals. This suggests that there are no outliers.(In our case some of the residual values stand out at the extreme bottom. This is so because of inefficient preprocessing)



4.3.2. RESIDUAL Vs. LEVERAGE



- The Residuals vs. Leverage plots helps you identify influential data points on your model. Outliers can be influential, though they don't necessarily have to it and some points within a normal range in your model could be very influential.
- Ideally it should be a horizontal line near the 0-line.
- Our result shows that because of some of the outliers being influential points, our line is slightly curved.

REFERENCES

[1] Abdul Mueez, Khushba Ahmed, Tuba Islam, Waqqas Iqbal, “Exploratory Data Analysis and Success Prediction of Google Play Store Apps”, Department of Computer Science and Engineering, BRAC University ,(2018)

[2] Monett, Dagmar & Stolte, Hermann, “Predicting Star Ratings based on Annotated Reviews of Mobile Apps”, 6th International Workshop on Advances in Semantic Information Retrieval, (2016)

[3] Aralikatte, Sridhara, Gantayat & Mani, “Fault in your stars: an analysis of Android app reviews”, *CoDS-COMAD '18*, (2018)

[4] F. Sarro, M. Harman, Y. Jia & Y. Zhang, “Customer Rating Reactions Can Be Predicted Purely using App Features” ,*IEEE 26th International Requirements Engineering Conference (RE)*, Banff, AB, (2018)

[5] Tuckerman, C.J, “Predicting Mobile Application Success”, cs229 Project, Stanford (2014)

[6] Karan Chimedia, “Sentiment Analysis on User Reviews of Android Apps with Expert System”, Computer Science Department Rochester Institute of Technology, PDF File

[7] Somprasertsri, Gamgarn & Lalitrojwong, Pattarachai, “Mining Feature-Opinion in Online Customer Reviews for Opinion Summarization”, J. UCS. 16. 938-955. 10.3217/jucs-016-06-0938,(2010)

[8] Sangani, Chirag & Sundaram Ananthanarayanan, “Sentiment Analysis of App Store Reviews.” (2013)

[9] Huifeng Tang, Songbo Tan, and Xueqi Cheng, “A survey on sentiment detection of reviews”, Expert Syst. Appl. 36, 7 (2009)

[10] Yi, Jeonghee & Nasukawa, Tetsuya & Bunescu, Razvan & Niblack, Wayne, “ Sentiment Analyzer: Extracting sentiments about a given topic using natural language processing techniques”, IEEE International Conference on Data Mining, ICDM,(2003)