**#1. Load the data file using pandas.**

In [**1**]: import pandas as pd

In [**2**]: import numpy as np

In [**3**]: import seaborn as sns

In [**4**]: import matplotlib.pyplot as plt

In [**5**]: data = pd.read\_csv("F:/Courses/Simpli Learn/Data Analytics/3 Data Analytics with Python/Live Classes/Project/googleplaystore/googleplaystore.csv")

In [**6**]: data.head()

Out[**6**]:

App ... Android Ver

0 Photo Editor & Candy Camera & Grid & ScrapBook ... 4.0.3 and up

1 Coloring book moana ... 4.0.3 and up

2 U Launcher Lite – FREE Live Cool Themes, Hide ... ... 4.0.3 and up

3 Sketch - Draw & Paint ... 4.2 and up

4 Pixel Draw - Number Art Coloring Book ... 4.4 and up

[5 rows x 13 columns]

In [**7**]: data.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 10841 entries, 0 to 10840

Data columns (total 13 columns):

App 10841 non-null object

Category 10841 non-null object

Rating 9367 non-null float64

Reviews 10841 non-null object

Size 10841 non-null object

Installs 10841 non-null object

Type 10840 non-null object

Price 10841 non-null object

Content Rating 10840 non-null object

Genres 10841 non-null object

Last Updated 10841 non-null object

Current Ver 10833 non-null object

Android Ver 10838 non-null object

dtypes: float64(1), object(12)

memory usage: 1.1+ MB

**#2. Check for null values in the data. Get the number of null values for each column**.

In [**8**]: data.isna().sum(axis=0)

Out[**8**]:

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. Drop records with nulls in any of the columns.**

In [**9**]: data1 = data.dropna()

In [**10**]: data1.isna().sum(axis=0)

Out[**10**]:

App 0

Category 0

Rating 0

Reviews 0

Size 0

Installs 0

Type 0

Price 0

Content Rating 0

Genres 0

Last Updated 0

Current Ver 0

Android Ver 0

dtype: int64

In [**11**]: data1.info()

<class 'pandas.core.frame.DataFrame'>

Int64Index: 9360 entries, 0 to 10840

Data columns (total 13 columns):

App 9360 non-null object

Category 9360 non-null object

Rating 9360 non-null float64

Reviews 9360 non-null object

Size 9360 non-null object

Installs 9360 non-null object

Type 9360 non-null object

Price 9360 non-null object

Content Rating 9360 non-null object

Genres 9360 non-null object

Last Updated 9360 non-null object

Current Ver 9360 non-null object

Android Ver 9360 non-null object

dtypes: float64(1), object(12)

memory usage: 1023.8+ KB

**#4. Variables seem to have incorrect type and inconsistent formatting. You need to fix them:**

**#4.1. Size column has sizes in Kb as well as Mb. To analyze, you’ll need to convert these to numeric.**

**#4.1.2. Multiply the value by 1,000, if size is mentioned in Mb**

In [**12**]: data1['Size'] = data1['Size'].str.replace('M', '000', regex = True)

\_\_main\_\_:1: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.

Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy

In [**13**]: data1['Size'] = data1['Size'].str.replace('.', '', regex = True)

\_\_main\_\_:1: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.

Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy

In [**14**]: data1['Size'] = data1['Size'].str.replace('k', '', regex = True)

\_\_main\_\_:1: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.

Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy

In [**15**]: data1['Size']

Out[**15**]:

0 19000

1 14000

2 87000

3 25000

4 28000

...

10834 26000

10836 53000

10837 36000

10839 Varies with device

10840 19000

Name: Size, Length: 9360, dtype: object

In [**16**]: data1.dtypes

Out[**16**]:

App object

Category object

Rating float64

Reviews object

Size object

Installs object

Type object

Price object

Content Rating object

Genres object

Last Updated object

Current Ver object

Android Ver object

dtype: object

**#4.1.1. Extract the numeric value from the column**

In [**17**]: data1.isna().sum(axis=0)

Out[**17**]:

App 0

Category 0

Rating 0

Reviews 0

Size 0

Installs 0

Type 0

Price 0

Content Rating 0

Genres 0

Last Updated 0

Current Ver 0

Android Ver 0

dtype: int64

In [**18**]: data1['Size'] = data1['Size'][data1['Size'].str.isdigit() == True].copy()

\_\_main\_\_:1: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.

Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy

In [**19**]: data2 = data1.dropna()

**#4.2. Reviews is a numeric field that is loaded as a string field. Convert it to numeric (int/float).**

In [**20**]: data2.info()

<class 'pandas.core.frame.DataFrame'>

Int64Index: 7723 entries, 0 to 10840

Data columns (total 13 columns):

App 7723 non-null object

Category 7723 non-null object

Rating 7723 non-null float64

Reviews 7723 non-null object

Size 7723 non-null object

Installs 7723 non-null object

Type 7723 non-null object

Price 7723 non-null object

Content Rating 7723 non-null object

Genres 7723 non-null object

Last Updated 7723 non-null object

Current Ver 7723 non-null object

Android Ver 7723 non-null object

dtypes: float64(1), object(12)

memory usage: 844.7+ KB

In [**21**]: data2.info()

<class 'pandas.core.frame.DataFrame'>

Int64Index: 7723 entries, 0 to 10840

Data columns (total 13 columns):

App 7723 non-null object

Category 7723 non-null object

Rating 7723 non-null float64

Reviews 7723 non-null object

Size 7723 non-null object

Installs 7723 non-null object

Type 7723 non-null object

Price 7723 non-null object

Content Rating 7723 non-null object

Genres 7723 non-null object

Last Updated 7723 non-null object

Current Ver 7723 non-null object

Android Ver 7723 non-null object

dtypes: float64(1), object(12)

memory usage: 844.7+ KB

In [**22**]: data2['Reviews'] = data2['Reviews'].astype(int)

\_\_main\_\_:1: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.

Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy

In [**23**]: data2.info()

<class 'pandas.core.frame.DataFrame'>

Int64Index: 7723 entries, 0 to 10840

Data columns (total 13 columns):

App 7723 non-null object

Category 7723 non-null object

Rating 7723 non-null float64

Reviews 7723 non-null int32

Size 7723 non-null object

Installs 7723 non-null object

Type 7723 non-null object

Price 7723 non-null object

Content Rating 7723 non-null object

Genres 7723 non-null object

Last Updated 7723 non-null object

Current Ver 7723 non-null object

Android Ver 7723 non-null object

dtypes: float64(1), int32(1), object(11)

memory usage: 814.5+ KB

**#4.3. Installs field is currently stored as string and has values like 1,000,000+.**

**#4.3.1. Treat 1,000,000+ as 1,000,000**

**#4.3.2. remove ‘+’, ‘,’ from the field, convert it to integer**

In [**24**]: data2['Installs'] = data2['Installs'].str.replace('+', '', regex = True)

\_\_main\_\_:1: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.

Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy

In [**25**]: data2['Installs'] = data2['Installs'].str.replace(',', '', regex = True)

\_\_main\_\_:1: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.

Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy

In [**26**]: data2['Installs'] = data2['Installs'].astype(int)

\_\_main\_\_:1: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.

Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy

In [**27**]: data2.info()

<class 'pandas.core.frame.DataFrame'>

Int64Index: 7723 entries, 0 to 10840

Data columns (total 13 columns):

App 7723 non-null object

Category 7723 non-null object

Rating 7723 non-null float64

Reviews 7723 non-null int32

Size 7723 non-null object

Installs 7723 non-null int32

Type 7723 non-null object

Price 7723 non-null object

Content Rating 7723 non-null object

Genres 7723 non-null object

Last Updated 7723 non-null object

Current Ver 7723 non-null object

Android Ver 7723 non-null object

dtypes: float64(1), int32(2), object(10)

memory usage: 784.4+ KB

**#4.4. Price field is a string and has $ symbol. Remove ‘$’ sign, and convert it to numeric.**

In [**28**]: data2['Price'] = data2['Price'].str.replace('$', '', regex = True)

\_\_main\_\_:1: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.

Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy

In [**29**]: data2['Price'] = data2['Price'].astype(float)

\_\_main\_\_:1: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.

Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy

In [**30**]: data2.info()

<class 'pandas.core.frame.DataFrame'>

Int64Index: 7723 entries, 0 to 10840

Data columns (total 13 columns):

App 7723 non-null object

Category 7723 non-null object

Rating 7723 non-null float64

Reviews 7723 non-null int32

Size 7723 non-null object

Installs 7723 non-null int32

Type 7723 non-null object

Price 7723 non-null float64

Content Rating 7723 non-null object

Genres 7723 non-null object

Last Updated 7723 non-null object

Current Ver 7723 non-null object

Android Ver 7723 non-null object

dtypes: float64(2), int32(2), object(9)

memory usage: 784.4+ KB

**#5. Sanity checks:**

**#5.1. Average rating should be between 1 and 5 as only these values are allowed on the play store.**

**# Drop the rows that have a value outside this range.**

In [**31**]: data2 = data2.loc[data2.Rating.between(1.0 , 5.0)]

In [**32**]: data2

Out[**32**]:

App ... Android Ver

0 Photo Editor & Candy Camera & Grid & ScrapBook ... 4.0.3 and up

1 Coloring book moana ... 4.0.3 and up

2 U Launcher Lite – FREE Live Cool Themes, Hide ... ... 4.0.3 and up

3 Sketch - Draw & Paint ... 4.2 and up

4 Pixel Draw - Number Art Coloring Book ... 4.4 and up

... ... ... ...

10833 Chemin (fr) ... 2.2 and up

10834 FR Calculator ... 4.1 and up

10836 Sya9a Maroc - FR ... 4.1 and up

10837 Fr. Mike Schmitz Audio Teachings ... 4.1 and up

10840 iHoroscope - 2018 Daily Horoscope & Astrology ... Varies with device

[7723 rows x 13 columns]

**#5.2. Reviews should not be more than installs as only those who installed can review the app.**

**#If there are any such records, drop them.**

In [**33**]: data2['R<=I'] = (data2['Reviews'] <= data2['Installs'])

In [**34**]: data3 = data2.loc[data2['R<=I']]

In [**35**]: data3 = data3.drop(columns = 'R<=I')

In [**36**]: data3.info()

<class 'pandas.core.frame.DataFrame'>

Int64Index: 7717 entries, 0 to 10840

Data columns (total 13 columns):

App 7717 non-null object

Category 7717 non-null object

Rating 7717 non-null float64

Reviews 7717 non-null int32

Size 7717 non-null object

Installs 7717 non-null int32

Type 7717 non-null object

Price 7717 non-null float64

Content Rating 7717 non-null object

Genres 7717 non-null object

Last Updated 7717 non-null object

Current Ver 7717 non-null object

Android Ver 7717 non-null object

dtypes: float64(2), int32(2), object(9)

memory usage: 783.8+ KB

**#5.3. For free apps (type = “Free”), the price should not be >0. Drop any such rows.**

In [**37**]: data4 = data3.loc[(data3['Price'] <= 0) & data3['Type'].isin(['Free']) | (data3['Price'] >= 0) & data3['Type'].isin(['Paid'])]

In [**38**]: data4['Rating'] = data4['Rating'].astype(float)

In [**39**]: data4['Size'] = data4['Size'].astype(int)

In [**40**]: data4.info()

<class 'pandas.core.frame.DataFrame'>

Int64Index: 7717 entries, 0 to 10840

Data columns (total 13 columns):

App 7717 non-null object

Category 7717 non-null object

Rating 7717 non-null float64

Reviews 7717 non-null int32

Size 7717 non-null int32

Installs 7717 non-null int32

Type 7717 non-null object

Price 7717 non-null float64

Content Rating 7717 non-null object

Genres 7717 non-null object

Last Updated 7717 non-null object

Current Ver 7717 non-null object

Android Ver 7717 non-null object

dtypes: float64(2), int32(3), object(8)

memory usage: 753.6+ KB

In [**41**]: data4.describe()

Out[**41**]:

Rating Reviews Size Installs Price

count 7717.000000 7.717000e+03 7717.000000 7.717000e+03 7717.000000

mean 4.173293 2.951275e+05 39789.025398 8.430620e+06 1.128725

std 0.544362 1.864640e+06 25192.753032 5.017636e+07 17.414784

min 1.000000 1.000000e+00 14.000000 5.000000e+00 0.000000

25% 4.000000 1.090000e+02 20000.000000 1.000000e+04 0.000000

50% 4.300000 2.351000e+03 34000.000000 1.000000e+05 0.000000

75% 4.500000 3.910900e+04 57000.000000 1.000000e+06 0.000000

max 5.000000 4.489389e+07 100000.000000 1.000000e+09 400.000000

**#6. Performing univariate analysis:**

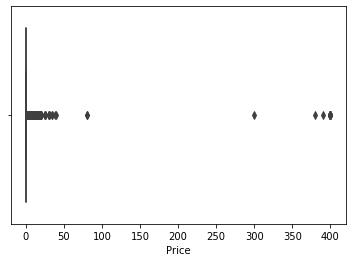
**#• Boxplot for Price**

**#• Are there any outliers? Think about the price of usual apps on Play Store.**

Some apps in the range of 4400 $ are there but maximum number of observation ar near to 0 $.

In [**42**]: sns.boxplot(data4.Price)

Out[**42**]: <matplotlib.axes.\_subplots.AxesSubplot at 0x1f6c1de30c8>



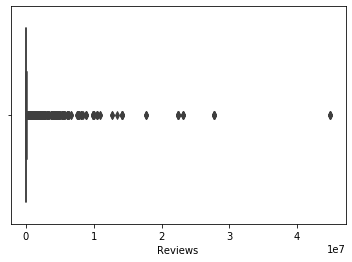
**#• Boxplot for Reviews**

**#• Are there any apps with very high number of reviews? Do the values seem right?**

Many fake reviews are there also so we have analyse further by checking number of installations.

In [**43**]: sns.boxplot(data4.Reviews)

Out[**43**]: <matplotlib.axes.\_subplots.AxesSubplot at 0x1f6c1d0dfc8>



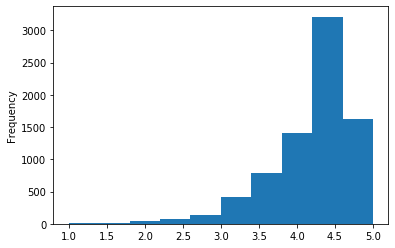
**#• Histogram for Rating**

**#• How are the ratings distributed? Is it more toward higher ratings?**

yes rating are densily populated around 4.5 rating.

In [**44**]: data4.Rating.plot.hist()

Out[**44**]: <matplotlib.axes.\_subplots.AxesSubplot at 0x1f6c1d3df08>



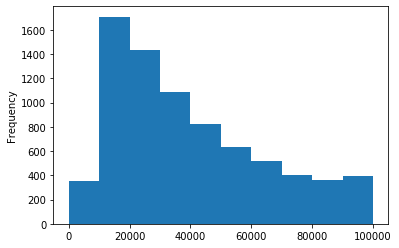
**#• Histogram for Size**

**#Note down your observations for the plots made above. Which of these seem to have outliers?**

Size is populated around 20000 kb. No outliers for size as per the graph.

In [**45**]: data4.Size.plot.hist()

Out[**45**]: <matplotlib.axes.\_subplots.AxesSubplot at 0x1f6c1f5ad88>



**#7. Outlier treatment:**

**#1. Price: From the box plot, it seems like there are some apps with very high price.**

**#A price of $200 for an application on the Play Store is very high and suspicious!**

**#1. Check out the records with very high price**

**#1. Is 200 indeed a high price?**

Check the quantiles to get view about the distribution of prices,200 is very high price for an app and make outliers.

In [**46**]: data4.Price.quantile([0.1, 0.25, 0.5, 0.70, 0.9, 0.95, 0.99])

Out[**46**]:

0.10 0.00

0.25 0.00

0.50 0.00

0.70 0.00

0.90 0.00

0.95 1.99

0.99 9.99

Name: Price, dtype: float64

**#2. Drop these as most seem to be junk apps**

In [**47**]: data5 = data4[data4.Price < 200].copy()

In [**48**]: data5.info()

<class 'pandas.core.frame.DataFrame'>

Int64Index: 7702 entries, 0 to 10840

Data columns (total 13 columns):

App 7702 non-null object

Category 7702 non-null object

Rating 7702 non-null float64

Reviews 7702 non-null int32

Size 7702 non-null int32

Installs 7702 non-null int32

Type 7702 non-null object

Price 7702 non-null float64

Content Rating 7702 non-null object

Genres 7702 non-null object

Last Updated 7702 non-null object

Current Ver 7702 non-null object

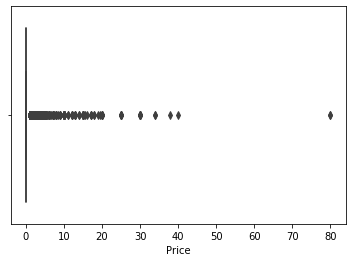
Android Ver 7702 non-null object

dtypes: float64(2), int32(3), object(8)

memory usage: 752.1+ KB

In [**49**]: sns.boxplot(data5.Price)

Out[**49**]: <matplotlib.axes.\_subplots.AxesSubplot at 0x1f6c1ff8b48>



**#2. Reviews: Very few apps have very high number of reviews.**

**#These are all star apps that don’t help with the analysis and, in fact, will skew it.**

**#Drop records having more than 2 million reviews**.

In [**50**]: data6 = data5[data5.Reviews < 2000000].copy()

In [**51**]: data6.info()

<class 'pandas.core.frame.DataFrame'>

Int64Index: 7483 entries, 0 to 10840

Data columns (total 13 columns):

App 7483 non-null object

Category 7483 non-null object

Rating 7483 non-null float64

Reviews 7483 non-null int32

Size 7483 non-null int32

Installs 7483 non-null int32

Type 7483 non-null object

Price 7483 non-null float64

Content Rating 7483 non-null object

Genres 7483 non-null object

Last Updated 7483 non-null object

Current Ver 7483 non-null object

Android Ver 7483 non-null object

dtypes: float64(2), int32(3), object(8)

memory usage: 730.8+ KB

**#3. Installs: There seems to be some outliers in this field too.**

**#Apps having very high number of installs should be dropped from the analysis.**

**#1. Find out the different percentiles – 10, 25, 50, 70, 90, 95, 99**

**#2. Decide a threshold as cutoff for outlier and drop records having values more than that**

Threshold level taken around 95 % and dropped the value above that.

In [**52**]: data6.Installs.quantile([0.1, 0.25, 0.5, 0.70, 0.9, 0.95, 0.99])

Out[**52**]:

0.10 1000.0

0.25 10000.0

0.50 100000.0

0.70 1000000.0

0.90 10000000.0

0.95 10000000.0

0.99 50000000.0

Name: Installs, dtype: float64

In [**53**]: data7 = data6[data6.Installs < 10000000].copy()

In [**54**]: data7.info()

<class 'pandas.core.frame.DataFrame'>

Int64Index: 6489 entries, 0 to 10837

Data columns (total 13 columns):

App 6489 non-null object

Category 6489 non-null object

Rating 6489 non-null float64

Reviews 6489 non-null int32

Size 6489 non-null int32

Installs 6489 non-null int32

Type 6489 non-null object

Price 6489 non-null float64

Content Rating 6489 non-null object

Genres 6489 non-null object

Last Updated 6489 non-null object

Current Ver 6489 non-null object

Android Ver 6489 non-null object

dtypes: float64(2), int32(3), object(8)

memory usage: 633.7+ KB

**#8. Bivariate analysis: Let’s look at how the available predictors relate to the variable of interest,**

**#i.e., our target variable rating. Make scatter plots (for numeric features) and box plots (for character features)**

**#to assess the relations between rating and the other features.**

**#1. Make scatter plot/joinplot for Rating vs. Price**

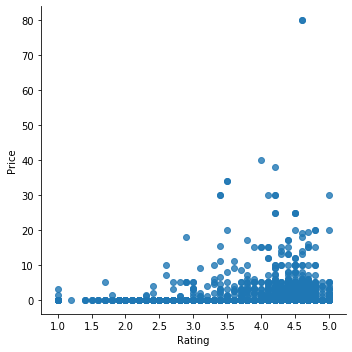
**#1. What pattern do you observe? Does rating increase with price?**

There is very weak positive correlation between price and rating by seeing the graph

In [**55**]: plt.figure(figsize=[20,12])

    ...: sns.lmplot('Rating','Price',data=data7,fit\_reg=False)

Out[**55**]: <seaborn.axisgrid.FacetGrid at 0x1f6c20644c8><Figure size 1440x864 with 0 Axes>



**#2. Make scatter plot/joinplot for Rating vs. Size**

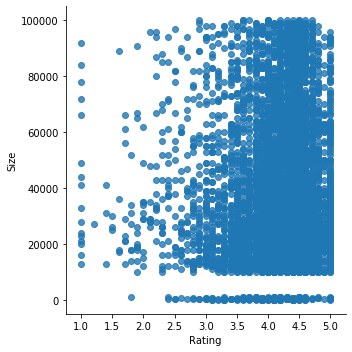
**#1. Are heavier apps rated better?**

We cant really say that by looking over to the graph

In [**56**]: plt.figure(figsize=[20,12])

    ...: sns.lmplot('Rating','Size',data=data7,fit\_reg=False)

Out[**56**]: <seaborn.axisgrid.FacetGrid at 0x1f6c20cf6c8><Figure size 1440x864 with 0 Axes>



**#3. Make scatter plot/joinplot for Rating vs. Reviews**

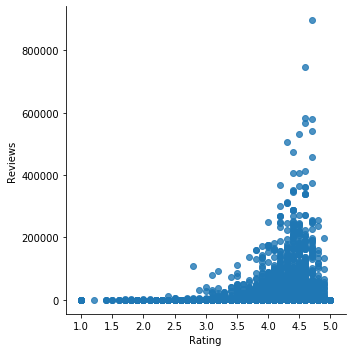
**#1. Does more review mean a better rating always?**

No that’s not always true.

In [**57**]: plt.figure(figsize=[20,12])

    ...: sns.lmplot('Rating','Reviews',data=data7,fit\_reg=False)

Out[**57**]: <seaborn.axisgrid.FacetGrid at 0x1f6c3122a08><Figure size 1440x864 with 0 Axes>



**#4. Make boxplot for Rating vs. Content Rating**

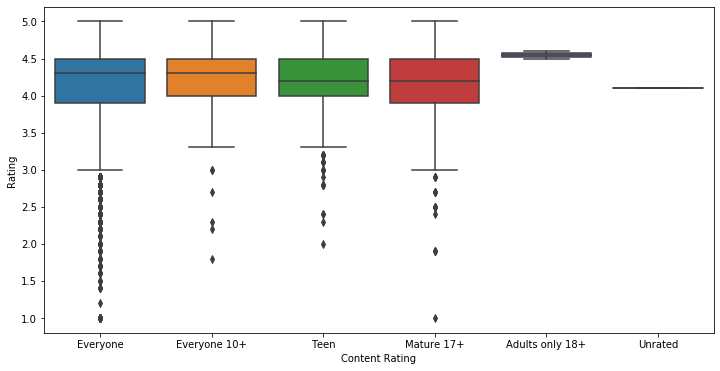
**#1. Is there any difference in the ratings? Are some types liked better?**

Yes it strongly depends on likes.

In [**58**]: plt.figure(figsize=[12,6])

    ...: sns.boxplot("Content Rating", "Rating", data=data7)

Out[**58**]: <matplotlib.axes.\_subplots.AxesSubplot at 0x1f6c2036f88>



**#5. Make boxplot for Ratings vs. Category**

**#1. Which genre has the best ratings?**

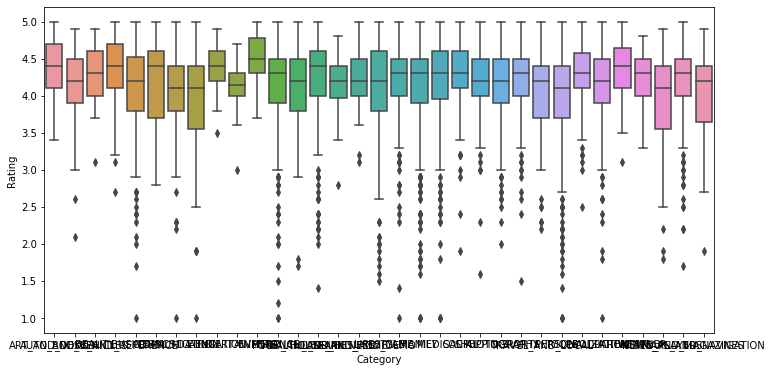
**#For each of the plots above, note down your observation.**

Cant coclude from graph

In [**59**]: plt.figure(figsize=[12,6])

    ...: sns.boxplot("Category", "Rating", data=data7)

Out[**59**]: <matplotlib.axes.\_subplots.AxesSubplot at 0x1f6c364eb48>



**#9. Data preprocessing**

**#For the steps below, create a copy of the dataframe to make all the edits. Name it inp1.**

**#1. Reviews and Install have some values that are still relatively very high. Before building a linear regression model**

**#, you need to reduce the skew. Apply log transformation (np.log1p) to Reviews and Installs.**

In [**60**]: inp1 = data7.copy()

In [**61**]: inp1.info()

<class 'pandas.core.frame.DataFrame'>

Int64Index: 6489 entries, 0 to 10837

Data columns (total 13 columns):

App 6489 non-null object

Category 6489 non-null object

Rating 6489 non-null float64

Reviews 6489 non-null int32

Size 6489 non-null int32

Installs 6489 non-null int32

Type 6489 non-null object

Price 6489 non-null float64

Content Rating 6489 non-null object

Genres 6489 non-null object

Last Updated 6489 non-null object

Current Ver 6489 non-null object

Android Ver 6489 non-null object

dtypes: float64(2), int32(3), object(8)

memory usage: 633.7+ KB

In [**62**]: inp1.head()

Out[**62**]:

App ... Android Ver

0 Photo Editor & Candy Camera & Grid & ScrapBook ... 4.0.3 and up

1 Coloring book moana ... 4.0.3 and up

2 U Launcher Lite – FREE Live Cool Themes, Hide ... ... 4.0.3 and up

4 Pixel Draw - Number Art Coloring Book ... 4.4 and up

5 Paper flowers instructions ... 2.3 and up

[5 rows x 13 columns]

In [**62**]:

In [**63**]: inp1['Reviews'] = np.log1p(inp1['Reviews'])

In [**64**]: inp1['Installs'] = np.log1p(inp1['Installs'])

In [**65**]: inp1.head()

Out[**65**]:

App ... Android Ver

0 Photo Editor & Candy Camera & Grid & ScrapBook ... 4.0.3 and up

1 Coloring book moana ... 4.0.3 and up

2 U Launcher Lite – FREE Live Cool Themes, Hide ... ... 4.0.3 and up

4 Pixel Draw - Number Art Coloring Book ... 4.4 and up

5 Paper flowers instructions ... 2.3 and up

[5 rows x 13 columns]

**#2. Drop columns App, Last Updated, Current Ver, and Android Ver. These variables are not useful for our task.**

In [**66**]: cols\_to\_drop = ['App', 'Last Updated', 'Current Ver', 'Android Ver']

In [**67**]: inp1 = inp1.drop(cols\_to\_drop, axis=1).copy()

In [**68**]: inp1.head()

Out[**68**]:

Category Rating ... Content Rating Genres

0 ART\_AND\_DESIGN 4.1 ... Everyone Art & Design

1 ART\_AND\_DESIGN 3.9 ... Everyone Art & Design;Pretend Play

2 ART\_AND\_DESIGN 4.7 ... Everyone Art & Design

4 ART\_AND\_DESIGN 4.3 ... Everyone Art & Design;Creativity

5 ART\_AND\_DESIGN 4.4 ... Everyone Art & Design

[5 rows x 9 columns]

**#3. Get dummy columns for Category, Genres, and Content Rating. This needs to be done as the models do not**

**#understand categorical data, and all data should be numeric. Dummy encoding is one way to convert character fields**

**#to numeric. Name of dataframe should be inp2.**

In [**69**]: inp2 = inp1.copy()

In [**70**]: inp2.head()

Out[**70**]:

Category Rating ... Content Rating Genres

0 ART\_AND\_DESIGN 4.1 ... Everyone Art & Design

1 ART\_AND\_DESIGN 3.9 ... Everyone Art & Design;Pretend Play

2 ART\_AND\_DESIGN 4.7 ... Everyone Art & Design

4 ART\_AND\_DESIGN 4.3 ... Everyone Art & Design;Creativity

5 ART\_AND\_DESIGN 4.4 ... Everyone Art & Design

[5 rows x 9 columns]

In [**71**]: inp2['Type'].value\_counts()

Out[**71**]:

Free 5930

Paid 559

Name: Type, dtype: int64

In [**72**]: inp2['Type'] = inp2['Type'].replace(['Paid', 'Free'], [1, 0])

In [**73**]: inp2.info()

<class 'pandas.core.frame.DataFrame'>

Int64Index: 6489 entries, 0 to 10837

Data columns (total 9 columns):

Category 6489 non-null object

Rating 6489 non-null float64

Reviews 6489 non-null float64

Size 6489 non-null int32

Installs 6489 non-null float64

Type 6489 non-null int64

Price 6489 non-null float64

Content Rating 6489 non-null object

Genres 6489 non-null object

dtypes: float64(4), int32(1), int64(1), object(3)

memory usage: 481.6+ KB

In [**74**]: inp2

Out[**74**]:

Category Rating ... Content Rating Genres

0 ART\_AND\_DESIGN 4.1 ... Everyone Art & Design

1 ART\_AND\_DESIGN 3.9 ... Everyone Art & Design;Pretend Play

2 ART\_AND\_DESIGN 4.7 ... Everyone Art & Design

4 ART\_AND\_DESIGN 4.3 ... Everyone Art & Design;Creativity

5 ART\_AND\_DESIGN 4.4 ... Everyone Art & Design

... ... ... ... ... ...

10832 WEATHER 3.8 ... Everyone Weather

10833 BOOKS\_AND\_REFERENCE 4.8 ... Everyone Books & Reference

10834 FAMILY 4.0 ... Everyone Education

10836 FAMILY 4.5 ... Everyone Education

10837 FAMILY 5.0 ... Everyone Education

[6489 rows x 9 columns]

In [**75**]: def preprocess\_features(X):

    ...: ''' Preprocesses the student data and converts non-numeric binary variables into

    ...: binary (0/1) variables. Converts categorical variables into dummy variables. '''

    ...:

    ...: # Initialize new output DataFrame

    ...: output = pd.DataFrame(index = X.index)

    ...:

    ...: # Investigate each feature column for the data

    ...: for col, col\_data in X.iteritems():

    ...:

    ...: # If data type is categorical, convert to dummy variables

    ...: if col\_data.dtype == object:

    ...: # Example: 'school' => 'school\_GP' and 'school\_MS'

    ...: col\_data = pd.get\_dummies(col\_data, prefix = col)

    ...:

    ...: # Collect the revised columns

    ...: output = output.join(col\_data)

    ...:

    ...: return output

In [**76**]: inp2 = preprocess\_features(inp2)

In [**77**]: inp2.info()

<class 'pandas.core.frame.DataFrame'>

Int64Index: 6489 entries, 0 to 10837

Columns: 153 entries, Category\_ART\_AND\_DESIGN to Genres\_Word

dtypes: float64(4), int32(1), int64(1), uint8(147)

memory usage: 1.5 MB

**#10. Train test split and apply 70-30 split. Name the new dataframes df\_train and df\_test.**

In [**78**]: X = inp2.drop(['Rating'], axis=1)

    ...: y = inp2['Rating']

In [**79**]: X

Out[**79**]:

Category\_ART\_AND\_DESIGN ... Genres\_Word

0 1 ... 0

1 1 ... 0

2 1 ... 0

4 1 ... 0

5 1 ... 0

... ... ... ...

10832 0 ... 0

10833 0 ... 0

10834 0 ... 0

10836 0 ... 0

10837 0 ... 0

[6489 rows x 152 columns]

In [**80**]: y

Out[**80**]:

0 4.1

1 3.9

2 4.7

4 4.3

5 4.4

...

10832 3.8

10833 4.8

10834 4.0

10836 4.5

10837 5.0

Name: Rating, Length: 6489, dtype: float64

**#11 Separate the dataframes into X\_train, y\_train, X\_test, and y\_test.**

In [**83**]: from sklearn.model\_selection import train\_test\_split

    ...: X\_train, X\_test, y\_train, y\_test = train\_test\_split (X, y, test\_size = 0.3, random\_state = 42)

In [**84**]: print(X\_train.shape)

    ...: print(X\_test.shape)

    ...: print(y\_train.shape)

    ...: print(y\_test.shape)

(4542, 152)

(1947, 152)

(4542,)

(1947,)

**#12. Model building**

**#• Use linear regression as the technique**

**#• Report the R2 on the train set**

In [**85**]: from sklearn.linear\_model import LinearRegression

    ...: model = LinearRegression(n\_jobs = -1)

    ...: model.fit(X\_train,y\_train)

Out[**85**]: LinearRegression(copy\_X=True, fit\_intercept=True, n\_jobs=-1, normalize=False)

In [**86**]: print(model.intercept\_)

    ...: print(model.coef\_)

4.7666010492718245

[-2.57588805e-01 7.38062569e-02 1.32619748e-01 9.55388581e-02

-4.52423623e-02 -9.48329632e-02 -6.94074580e-02 -1.41784982e-01

5.18419213e-02 -7.21523830e-02 1.41210113e-01 1.35398901e-02

-4.58474472e-02 -1.12811568e-02 2.35069700e-01 -1.68597608e-02

5.52295229e-03 7.49697961e-02 -9.70224173e-03 -8.56465076e-02

6.55541363e-03 -3.91204593e-02 1.04969507e-01 4.93055871e-02

-4.74129578e-02 -5.53662978e-02 1.69250108e-02 1.72771044e-02

4.93707917e-02 -7.75117187e-02 -2.50247438e-02 1.35366573e-02

1.27229384e-02 1.72174260e-01 -6.98655555e-08 -1.55280872e-01

-1.03461010e-01 -5.47812707e-05 2.15541065e-01 -7.00944746e-02

-6.32798345e-02 -3.02962704e-02 -5.18704858e-02 5.30964162e-14

-3.14194541e-01 7.33648281e-02 -3.80948392e-01 1.96445947e-02

2.16493490e-14 -2.55472121e-01 1.05444842e-01 2.07448536e-01

5.54911760e-01 3.40123197e-01 1.79790050e-01 7.38062570e-02

1.32619748e-01 -3.04334313e-01 -9.67732094e-02 1.04968937e-01

6.35939288e-01 9.55388581e-02 -5.10778784e-02 -4.52423623e-02

-4.58504170e-01 -1.93221237e-01 3.55059674e-01 -2.70778660e-01

-2.20050448e-01 -6.24081472e-01 2.97309260e-01 3.96441043e-01

5.01776974e-02 -8.40514196e-02 -9.48329632e-02 -3.06143999e-14

-6.94074580e-02 -1.41784982e-01 9.76809975e-02 4.02888748e-01

-8.14254938e-02 4.91573375e-01 2.57724041e-01 2.65237097e-01

3.17754861e-01 -1.96225416e-01 -1.24344979e-14 2.10316340e-01

7.42665449e-02 1.19879513e-01 7.51867811e-02 -8.95488892e-02

3.60590247e-02 -7.46398666e-02 4.68946908e-01 -2.17187379e-15

-6.19043603e-02 -5.57993727e-01 1.41210113e-01 -4.58474472e-02

-1.12811568e-02 -1.68597608e-02 -4.39888808e-01 1.98231125e-01

5.52295231e-03 7.49697961e-02 -9.70224173e-03 -8.56465076e-02

6.55541363e-03 -4.68694126e-01 3.90912737e-01 1.28239985e-01

-3.91204593e-02 1.47332581e-01 -2.64483231e-01 -1.06828296e-01

3.28948453e-01 4.93055871e-02 -4.74129578e-02 -5.53662978e-02

5.54428352e-02 2.46222804e-01 2.17751620e-01 7.00225973e-02

0.00000000e+00 -3.86877939e-01 1.89288039e-01 0.00000000e+00

-1.67220237e-01 1.96046237e-01 0.00000000e+00 1.69250108e-02

-1.23780410e-01 -1.09506705e-01 -2.83152351e-02 -1.66735924e-01

1.72771044e-02 -1.34273228e-01 -1.43651262e-01 -1.61521972e-01

3.42291911e-01 -1.87155726e-01 0.00000000e+00 -7.75117187e-02

-7.71244763e-02 5.20997325e-02 -5.07566663e-01 -2.22070538e-01

-2.75982631e-01 0.00000000e+00 1.27229384e-02 -3.00503037e-01]

In [**87**]: categorical\_columns = inp2.columns

In [**88**]: list(zip(categorical\_columns,model.coef\_))

Out[**88**]:

[('Category\_ART\_AND\_DESIGN', -0.2575888054497894),

('Category\_AUTO\_AND\_VEHICLES', 0.07380625692437234),

('Category\_BEAUTY', 0.13261974783660896),

('Category\_BOOKS\_AND\_REFERENCE', 0.09553885809247573),

('Category\_BUSINESS', -0.045242362265390884),

('Category\_COMICS', -0.0948329631832161),

('Category\_COMMUNICATION', -0.06940745799006721),

('Category\_DATING', -0.14178498218830174),

('Category\_EDUCATION', 0.05184192130894752),

('Category\_ENTERTAINMENT', -0.07215238298296517),

('Category\_EVENTS', 0.14121011280559045),

('Category\_FAMILY', 0.013539890126972244),

('Category\_FINANCE', -0.04584744724485665),

('Category\_FOOD\_AND\_DRINK', -0.011281156757240368),

('Category\_GAME', 0.2350696999380064),

('Category\_HEALTH\_AND\_FITNESS', -0.016859760752420763),

('Category\_HOUSE\_AND\_HOME', 0.005522952288982096),

('Category\_LIBRARIES\_AND\_DEMO', 0.07496979611497215),

('Category\_LIFESTYLE', -0.009702241729929362),

('Category\_MAPS\_AND\_NAVIGATION', -0.08564650763119946),

('Category\_MEDICAL', 0.006555413629356875),

('Category\_NEWS\_AND\_MAGAZINES', -0.039120459337725885),

('Category\_PARENTING', 0.1049695069215388),

('Category\_PERSONALIZATION', 0.04930558709821696),

('Category\_PHOTOGRAPHY', -0.047412957828734635),

('Category\_PRODUCTIVITY', -0.055366297845267635),

('Category\_SHOPPING', 0.016925010753818987),

('Category\_SOCIAL', 0.017277104383696562),

('Category\_SPORTS', 0.04937079170023109),

('Category\_TOOLS', -0.07751171872323259),

('Category\_TRAVEL\_AND\_LOCAL', -0.025024743806288368),

('Category\_VIDEO\_PLAYERS', 0.013536657297750079),

('Category\_WEATHER', 0.012722938449914263),

('Rating', 0.17217426007220776),

('Reviews', -6.986555547983109e-08),

('Size', -0.15528087188543596),

('Installs', -0.1034610097814745),

('Type', -5.4781270728151976e-05),

('Price', 0.21554106527449496),

('Content Rating\_Adults only 18+', -0.07009447461166803),

('Content Rating\_Everyone', -0.0632798345079511),

('Content Rating\_Everyone 10+', -0.030296270403343522),

('Content Rating\_Mature 17+', -0.05187048575272522),

('Content Rating\_Teen', 5.309641615269811e-14),

('Content Rating\_Unrated', -0.31419454067604696),

('Genres\_Action', 0.07336482809078976),

('Genres\_Action;Action & Adventure', -0.38094839214534226),

('Genres\_Adventure', 0.01964459470069605),

('Genres\_Adventure;Action & Adventure', 2.1649348980190553e-14),

('Genres\_Adventure;Brain Games', -0.2554721212672611),

('Genres\_Arcade', 0.1054448415612009),

('Genres\_Arcade;Action & Adventure', 0.20744853594439316),

('Genres\_Arcade;Pretend Play', 0.554911760161365),

('Genres\_Art & Design', 0.3401231968038867),

('Genres\_Art & Design;Creativity', 0.17979004953647051),

('Genres\_Art & Design;Pretend Play', 0.07380625698316554),

('Genres\_Auto & Vehicles', 0.13261974783121636),

('Genres\_Beauty', -0.30433431324919474),

('Genres\_Board', -0.09677320938272008),

('Genres\_Board;Action & Adventure', 0.10496893703652571),

('Genres\_Board;Brain Games', 0.6359392879333711),

('Genres\_Board;Pretend Play', 0.09553885808999174),

('Genres\_Books & Reference', -0.05107787842344272),

('Genres\_Books & Reference;Education', -0.04524236226803536),

('Genres\_Business', -0.4585041701001814),

('Genres\_Card', -0.19322123748940748),

('Genres\_Card;Action & Adventure', 0.35505967355845003),

('Genres\_Card;Brain Games', -0.2707786604643782),

('Genres\_Casino', -0.22005044783958078),

('Genres\_Casual', -0.6240814723328835),

('Genres\_Casual;Action & Adventure', 0.2973092600669297),

('Genres\_Casual;Brain Games', 0.39644104328387025),

('Genres\_Casual;Creativity', 0.05017769744536307),

('Genres\_Casual;Education', -0.08405141961578846),

('Genres\_Casual;Pretend Play', -0.0948329631863428),

('Genres\_Comics', -3.061439990403869e-14),

('Genres\_Comics;Creativity', -0.06940745799658157),

('Genres\_Communication', -0.14178498219817107),

('Genres\_Dating', 0.09768099747550596),

('Genres\_Education', 0.4028887476376535),

('Genres\_Education;Action & Adventure', -0.08142549375818464),

('Genres\_Education;Brain Games', 0.49157337542019325),

('Genres\_Education;Creativity', 0.25772404058206216),

('Genres\_Education;Education', 0.2652370972213384),

('Genres\_Education;Music & Video', 0.3177548607620978),

('Genres\_Education;Pretend Play', -0.1962254160362915),

('Genres\_Educational', -1.2434497875801753e-14),

('Genres\_Educational;Action & Adventure', 0.21031634021336226),

('Genres\_Educational;Brain Games', 0.07426654488655475),

('Genres\_Educational;Creativity', 0.11987951273556324),

('Genres\_Educational;Education', 0.07518678112275708),

('Genres\_Educational;Pretend Play', -0.08954888922461166),

('Genres\_Entertainment', 0.03605902465086425),

('Genres\_Entertainment;Action & Adventure', -0.0746398665815297),

('Genres\_Entertainment;Brain Games', 0.46894690791577887),

('Genres\_Entertainment;Creativity', -2.1718737919229625e-15),

('Genres\_Entertainment;Education', -0.0619043603377474),

('Genres\_Entertainment;Music & Video', -0.5579937267840331),

('Genres\_Entertainment;Pretend Play', 0.14121011280350068),

('Genres\_Events', -0.04584744723690314),

('Genres\_Finance', -0.011281156754842939),

('Genres\_Food & Drink', -0.01685976075058713),

('Genres\_Health & Fitness', -0.43988880784865086),

('Genres\_Health & Fitness;Action & Adventure', 0.19823112482513952),

('Genres\_Health & Fitness;Education', 0.005522952306555612),

('Genres\_House & Home', 0.0749697961135885),

('Genres\_Libraries & Demo', -0.009702241732578117),

('Genres\_Lifestyle', -0.08564650763195811),

('Genres\_Maps & Navigation', 0.00655541362648825),

('Genres\_Medical', -0.4686941256780529),

('Genres\_Music', 0.3909127366774739),

('Genres\_Music & Audio;Music & Video', 0.1282399848538749),

('Genres\_Music;Music & Video', -0.03912045933830973),

('Genres\_News & Magazines', 0.14733258053577877),

('Genres\_Parenting', -0.2644832312334876),

('Genres\_Parenting;Brain Games', -0.10682829559725378),

('Genres\_Parenting;Education', 0.3289484532161463),

('Genres\_Parenting;Music & Video', 0.049305587098223774),

('Genres\_Personalization', -0.04741295783306085),

('Genres\_Photography', -0.05536629784545295),

('Genres\_Productivity', 0.05544283520057118),

('Genres\_Puzzle', 0.2462228035322898),

('Genres\_Puzzle;Action & Adventure', 0.2177516201210315),

('Genres\_Puzzle;Brain Games', 0.07002259733151668),

('Genres\_Puzzle;Creativity', 0.0),

('Genres\_Puzzle;Education', -0.38687793904469114),

('Genres\_Racing', 0.18928803923225893),

('Genres\_Racing;Action & Adventure', 0.0),

('Genres\_Racing;Pretend Play', -0.16722023650345136),

('Genres\_Role Playing', 0.19604623662389226),

('Genres\_Role Playing;Action & Adventure', 0.0),

('Genres\_Role Playing;Pretend Play', 0.01692501075339245),

('Genres\_Shopping', -0.12378041016450436),

('Genres\_Simulation', -0.10950670474904695),

('Genres\_Simulation;Action & Adventure', -0.028315235118776785),

('Genres\_Simulation;Education', -0.16673592383389976),

('Genres\_Simulation;Pretend Play', 0.017277104380957156),

('Genres\_Social', -0.1342732278681365),

('Genres\_Sports', -0.14365126206847267),

('Genres\_Sports;Action & Adventure', -0.16152197237899152),

('Genres\_Strategy', 0.3422919111762496),

('Genres\_Strategy;Action & Adventure', -0.18715572557481902),

('Genres\_Strategy;Creativity', 0.0),

('Genres\_Strategy;Education', -0.07751171872448478),

('Genres\_Tools', -0.07712447629021917),

('Genres\_Travel & Local', 0.05209973248406442),

('Genres\_Travel & Local;Action & Adventure', -0.5075666628620834),

('Genres\_Trivia', -0.22207053783804853),

('Genres\_Video Players & Editors', -0.2759826305083164),

('Genres\_Video Players & Editors;Creativity', 0.0),

('Genres\_Video Players & Editors;Music & Video', 0.01272293844995496),

('Genres\_Weather', -0.30050303662716077)]

**#13. Make predictions on test set and report R2.**

In [**89**]: model.predict(X\_test)

Out[**89**]:

array([4.25098497, 3.83116753, 4.36543238, ..., 4.17686038, 4.47015288,

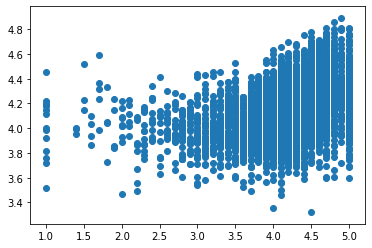
4.05086857])

In [**90**]: print('R2 Value/Coefficient of Determination: {}'.format(model.score(X\_test, y\_test)))

**R2 Value/Coefficient of Determination: 0.1452618347136635**

In [**91**]: plt.scatter(y\_train,model.predict(X\_train))

Out[**91**]: <matplotlib.collections.PathCollection at 0x1f6c33bd648>



In [**92**]: from math import sqrt

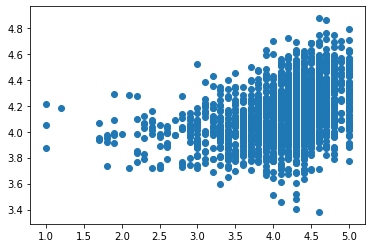
    ...: from sklearn.metrics import mean\_squared\_error

    ...: print(sqrt(mean\_squared\_error(y\_train, model.predict(X\_train))))

0.5375666340157613

In [**93**]: plt.scatter(y\_test,model.predict(X\_test))

Out[**93**]: <matplotlib.collections.PathCollection at 0x1f6c33f1208>



In [**94**]: print(sqrt(mean\_squared\_error(y\_test, model.predict(X\_test))))

0.5322846940094714

In [**95**]: result = pd.DataFrame({'Actual': y\_test, 'Predicted': model.predict(X\_test)})

    ...: print(result)

Actual Predicted

6533 4.3 4.250985

9230 3.4 3.831168

1810 4.5 4.365432

1207 3.8 4.241885

5195 4.0 4.001392

... ... ...

6658 4.3 4.156594

9857 4.2 4.266427

7422 5.0 4.176860

10833 4.8 4.470153

6666 4.5 4.050869

[1947 rows x 2 columns]

In [**96**]: