Life cycle of Machine learning Project

1.Understanding the Problem Statement

2.Data Collection

3.Exploratory data analysis

4.Data Cleaning

5.Data Pre-Processing

6.Model Training

1) Problem statement.

Today, 1.85 million different apps are available for users to download. Android users have even more from which to choose, with 2.56 million available through the Google Play Store. These apps have come to play a huge role in the way we live our lives today. Our Objective is to find the Most Popular Category, find the App with largest number of installs , the App with largest size etc.

2) Data Collection.

The Dataset is collected from https://www.kaggle.com/lava18/google-play-store-apps

Importing the packages

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

Importing the Dataset

```
df = pd.read_csv("google_playstore.csv")
```

Show top 5 records

```
df.head()
```

```
                                                 App        Category
Rating  \
0     Photo Editor & Candy Camera & Grid & ScrapBook  ART_AND_DESIGN
4.1
1                                Coloring book moana  ART_AND_DESIGN
3.9
2  U Launcher Lite – FREE Live Cool Themes, Hide ...  ART_AND_DESIGN
4.7
```

```
3                              Sketch - Draw & Paint  ART_AND_DESIGN
4.5
4              Pixel Draw - Number Art Coloring Book  ART_AND_DESIGN
4.3

  Reviews  Size     Installs  Type Price Content Rating  \
0     159   19M      10,000+  Free     0       Everyone
1     967   14M     500,000+  Free     0       Everyone
2   87510  8.7M   5,000,000+  Free     0       Everyone
3  215644   25M  50,000,000+  Free     0           Teen
4     967  2.8M     100,000+  Free     0       Everyone

                     Genres      Last Updated         Current Ver  \
0              Art & Design   January 7, 2018               1.0.0
1  Art & Design;Pretend Play  January 15, 2018               2.0.0
2              Art & Design    August 1, 2018               1.2.4
3              Art & Design      June 8, 2018  Varies with device
4    Art & Design;Creativity     June 20, 2018                 1.1

    Android Ver
0  4.0.3 and up
1  4.0.3 and up
2  4.0.3 and up
3    4.2 and up
4    4.4 and up
```

Show Last 5 records

```
df.tail()
```

```
                                                 App
Category  \
10836                               Sya9a Maroc - FR
FAMILY
10837               Fr. Mike Schmitz Audio Teachings
FAMILY
10838                         Parkinson Exercices FR
MEDICAL
10839                  The SCP Foundation DB fr nn5n
BOOKS_AND_REFERENCE
10840  iHoroscope - 2018 Daily Horoscope & Astrology
LIFESTYLE

       Rating Reviews                Size     Installs  Type Price  \
10836     4.5      38                 53M       5,000+  Free     0
10837     5.0       4                3.6M         100+  Free     0
10838     NaN       3                9.5M       1,000+  Free     0
10839     4.5     114  Varies with device       1,000+  Free     0
10840     4.5  398307                 19M  10,000,000+  Free     0

      Content Rating            Genres      Last Updated
```

```
Current Ver  \
10836       Everyone          Education     July 25, 2017
1.48
10837       Everyone          Education      July 6, 2018
1.0
10838       Everyone            Medical  January 20, 2017
1.0
10839     Mature 17+  Books & Reference  January 19, 2015  Varies with
device
10840       Everyone          Lifestyle     July 25, 2018  Varies with
device

              Android Ver
10836          4.1 and up
10837          4.1 and up
10838          2.2 and up
10839  Varies with device
10840  Varies with device
```

Shape of the Dataset

```
df.shape
```

```
(10841, 13)
```

The dataset have 10841 rows and 13 columns

Information of the Dataset

```
df.info()
```

```
<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
```

We got some null values in the dataset and also we got except Rating feature every feature is categorical datatype and the Rating feature is float datatype

Checking the null values

```
df.isnull().sum()
```

```
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
```

We got 1474 null values in Rating feature , 1 null value in Type feature and Content Rating feature,

8 null value in Current Ver , 3 null value in Android Ver

Statistical Analysis

```
df.describe(include='all')
```

```
           App Category       Rating Reviews                Size
Installs  \
count    10841    10841  9367.000000   10841               10841
10841
unique    9660       34          NaN    6002                 462
22
top     ROBLOX   FAMILY          NaN       0  Varies with device
1,000,000+
freq         9     1972          NaN     596                1695
1579
mean       NaN      NaN     4.193338     NaN                 NaN
NaN
std        NaN      NaN     0.537431     NaN                 NaN
NaN
min        NaN      NaN     1.000000     NaN                 NaN
NaN
25%        NaN      NaN     4.000000     NaN                 NaN
NaN
50%        NaN      NaN     4.300000     NaN                 NaN
NaN
75%        NaN      NaN     4.500000     NaN                 NaN
```

```
NaN
max         NaN       NaN     19.000000     NaN                 NaN
NaN

         Type  Price Content Rating Genres    Last Updated  \
count   10840  10841          10840  10841           10841
unique      3     93              6    120            1378
top      Free      0       Everyone  Tools  August 3, 2018
freq    10039  10040           8714    842             326
mean      NaN    NaN            NaN    NaN             NaN
std       NaN    NaN            NaN    NaN             NaN
min       NaN    NaN            NaN    NaN             NaN
25%       NaN    NaN            NaN    NaN             NaN
50%       NaN    NaN            NaN    NaN             NaN
75%       NaN    NaN            NaN    NaN             NaN
max       NaN    NaN            NaN    NaN             NaN

               Current Ver Android Ver
count                10833       10838
unique                2832          33
top     Varies with device  4.1 and up
freq                  1459        2451
mean                   NaN         NaN
std                    NaN         NaN
min                    NaN         NaN
25%                    NaN         NaN
50%                    NaN         NaN
75%                    NaN         NaN
max                    NaN         NaN
```

The described method will help to see how data has been spread for numerical values.

We can clearly see the minimum value, mean values, different percentile values, and maximum values.

Handling with Categorical Features

We need to convert (Reviews & Size & Installs & Price) to int

we need to Change Last update into a datetime column

(1)Dealing with Review feature

Check if all values in number of Reviews numeric

```
df.Reviews.str.isnumeric().sum()
```

```
10840
```

Since we have 10841 rows but here we got 10840 numeric data in Review features

It clearly mean that 1 data is not numeric

Checking the non-numeric data

```
df[~df['Reviews'].str.isnumeric()]
```

```
                                           App Category  Rating 
Reviews  \
10472  Life Made WI-Fi Touchscreen Photo Frame      1.9    19.0 
3.0M

        Size Installs Type     Price Content Rating 
Genres  \
10472  1,000+     Free    0  Everyone            NaN  February 11, 
2018

      Last Updated Current Ver Android Ver  
10472       1.0.19  4.0 and up         NaN
```

Here we can see the entries are wrong types because the Price features cannot have 'Eeryone' data value

So, we can drop this for now.

Creating a copy of the dataset for further process so that our original dataset should not lost

```
df_copy = df.copy()

df_copy=df_copy.drop(df_copy.index[10472])
```

Converting the Review feature from Categorical to int datatype

```
df_copy['Reviews']=df_copy['Reviews'].astype(int)
```

(2) Dealing with Size feature

```
df_copy['Size'].unique()
```

```
array(['19M', '14M', '8.7M', '25M', '2.8M', '5.6M', '29M', '33M', 
'3.1M',
       '28M', '12M', '20M', '21M', '37M', '2.7M', '5.5M', '17M', 
'39M',
       '31M', '4.2M', '7.0M', '23M', '6.0M', '6.1M', '4.6M', '9.2M',
       '5.2M', '11M', '24M', 'Varies with device', '9.4M', '15M', 
'10M',
       '1.2M', '26M', '8.0M', '7.9M', '56M', '57M', '35M', '54M', 
'201k',
       '3.6M', '5.7M', '8.6M', '2.4M', '27M', '2.5M', '16M', '3.4M',
       '8.9M', '3.9M', '2.9M', '38M', '32M', '5.4M', '18M', '1.1M',
       '2.2M', '4.5M', '9.8M', '52M', '9.0M', '6.7M', '30M', '2.6M',
       '7.1M', '3.7M', '22M', '7.4M', '6.4M', '3.2M', '8.2M', '9.9M',
       '4.9M', '9.5M', '5.0M', '5.9M', '13M', '73M', '6.8M', '3.5M',
       '4.0M', '2.3M', '7.2M', '2.1M', '42M', '7.3M', '9.1M', '55M',
```

```
       '23k', '6.5M', '1.5M', '7.5M', '51M', '41M', '48M', '8.5M',
'46M',
       '8.3M', '4.3M', '4.7M', '3.3M', '40M', '7.8M', '8.8M', '6.6M',
       '5.1M', '61M', '66M', '79k', '8.4M', '118k', '44M', '695k',
'1.6M',
       '6.2M', '18k', '53M', '1.4M', '3.0M', '5.8M', '3.8M', '9.6M',
       '45M', '63M', '49M', '77M', '4.4M', '4.8M', '70M', '6.9M',
'9.3M',
       '10.0M', '8.1M', '36M', '84M', '97M', '2.0M', '1.9M', '1.8M',
       '5.3M', '47M', '556k', '526k', '76M', '7.6M', '59M', '9.7M',
'78M',
       '72M', '43M', '7.7M', '6.3M', '334k', '34M', '93M', '65M',
'79M',
       '100M', '58M', '50M', '68M', '64M', '67M', '60M', '94M',
'232k',
       '99M', '624k', '95M', '8.5k', '41k', '292k', '11k', '80M',
'1.7M',
       '74M', '62M', '69M', '75M', '98M', '85M', '82M', '96M', '87M',
       '71M', '86M', '91M', '81M', '92M', '83M', '88M', '704k',
'862k',
       '899k', '378k', '266k', '375k', '1.3M', '975k', '980k', '4.1M',
       '89M', '696k', '544k', '525k', '920k', '779k', '853k', '720k',
       '713k', '772k', '318k', '58k', '241k', '196k', '857k', '51k',
       '953k', '865k', '251k', '930k', '540k', '313k', '746k', '203k',
       '26k', '314k', '239k', '371k', '220k', '730k', '756k', '91k',
       '293k', '17k', '74k', '14k', '317k', '78k', '924k', '902k',
'818k',
       '81k', '939k', '169k', '45k', '475k', '965k', '90M', '545k',
'61k',
       '283k', '655k', '714k', '93k', '872k', '121k', '322k', '1.0M',
       '976k', '172k', '238k', '549k', '206k', '954k', '444k', '717k',
       '210k', '609k', '308k', '705k', '306k', '904k', '473k', '175k',
       '350k', '383k', '454k', '421k', '70k', '812k', '442k', '842k',
       '417k', '412k', '459k', '478k', '335k', '782k', '721k', '430k',
       '429k', '192k', '200k', '460k', '728k', '496k', '816k', '414k',
       '506k', '887k', '613k', '243k', '569k', '778k', '683k', '592k',
       '319k', '186k', '840k', '647k', '191k', '373k', '437k', '598k',
       '716k', '585k', '982k', '222k', '219k', '55k', '948k', '323k',
       '691k', '511k', '951k', '963k', '25k', '554k', '351k', '27k',
       '82k', '208k', '913k', '514k', '551k', '29k', '103k', '898k',
       '743k', '116k', '153k', '209k', '353k', '499k', '173k', '597k',
       '809k', '122k', '411k', '400k', '801k', '787k', '237k', '50k',
       '643k', '986k', '97k', '516k', '837k', '780k', '961k', '269k',
       '20k', '498k', '600k', '749k', '642k', '881k', '72k', '656k',
       '601k', '221k', '228k', '108k', '940k', '176k', '33k', '663k',
       '34k', '942k', '259k', '164k', '458k', '245k', '629k', '28k',
       '288k', '775k', '785k', '636k', '916k', '994k', '309k', '485k',
       '914k', '903k', '608k', '500k', '54k', '562k', '847k', '957k',
       '688k', '811k', '270k', '48k', '329k', '523k', '921k', '874k',
       '981k', '784k', '280k', '24k', '518k', '754k', '892k', '154k',
```

```
       '860k', '364k', '387k', '626k', '161k', '879k', '39k', '970k',
       '170k', '141k', '160k', '144k', '143k', '190k', '376k', '193k',
       '246k', '73k', '658k', '992k', '253k', '420k', '404k', '470k',
       '226k', '240k', '89k', '234k', '257k', '861k', '467k', '157k',
       '44k', '676k', '67k', '552k', '885k', '1020k', '582k', '619k'],
      dtype=object)
```

Remove all characters from size and convert it to float¶

```
df_copy['Size']=df_copy['Size'].str.replace('M','000')
df_copy['Size']=df_copy['Size'].str.replace('k','')
df_copy['Size']=df_copy['Size'].replace("Varies with device",np.nan)
df_copy['Size']=df_copy['Size'].astype('float')
```

Convert mega bytes to kilo bytes then convert all to mega bytes. So that all size lies in Sync

```
df_copy['Size'].unique()
```

```
array([1.90e+04, 1.40e+04, 8.70e+00, 2.50e+04, 2.80e+00, 5.60e+00,
       2.90e+04, 3.30e+04, 3.10e+00, 2.80e+04, 1.20e+04, 2.00e+04,
       2.10e+04, 3.70e+04, 2.70e+00, 5.50e+00, 1.70e+04, 3.90e+04,
       3.10e+04, 4.20e+00, 7.00e+00, 2.30e+04, 6.00e+00, 6.10e+00,
       4.60e+00, 9.20e+00, 5.20e+00, 1.10e+04, 2.40e+04,      nan,
       9.40e+00, 1.50e+04, 1.00e+04, 1.20e+00, 2.60e+04, 8.00e+00,
       7.90e+00, 5.60e+04, 5.70e+04, 3.50e+04, 5.40e+04, 2.01e+02,
       3.60e+00, 5.70e+00, 8.60e+00, 2.40e+00, 2.70e+04, 2.50e+00,
       1.60e+04, 3.40e+00, 8.90e+00, 3.90e+00, 2.90e+00, 3.80e+04,
       3.20e+04, 5.40e+00, 1.80e+04, 1.10e+00, 2.20e+00, 4.50e+00,
       9.80e+00, 5.20e+04, 9.00e+00, 6.70e+00, 3.00e+04, 2.60e+00,
       7.10e+00, 3.70e+00, 2.20e+04, 7.40e+00, 6.40e+00, 3.20e+00,
       8.20e+00, 9.90e+00, 4.90e+00, 9.50e+00, 5.00e+00, 5.90e+00,
       1.30e+04, 7.30e+04, 6.80e+00, 3.50e+00, 4.00e+00, 2.30e+00,
       7.20e+00, 2.10e+00, 4.20e+04, 7.30e+00, 9.10e+00, 5.50e+04,
       2.30e+01, 6.50e+00, 1.50e+00, 7.50e+00, 5.10e+04, 4.10e+04,
       4.80e+04, 8.50e+00, 4.60e+04, 8.30e+00, 4.30e+00, 4.70e+00,
       3.30e+00, 4.00e+04, 7.80e+00, 8.80e+00, 6.60e+00, 5.10e+00,
       6.10e+04, 6.60e+04, 7.90e+01, 8.40e+00, 1.18e+02, 4.40e+04,
       6.95e+02, 1.60e+00, 6.20e+00, 1.80e+01, 5.30e+04, 1.40e+00,
       3.00e+00, 5.80e+00, 3.80e+00, 9.60e+00, 4.50e+04, 6.30e+04,
       4.90e+04, 7.70e+04, 4.40e+00, 4.80e+00, 7.00e+04, 6.90e+00,
       9.30e+00, 1.00e+01, 8.10e+00, 3.60e+04, 8.40e+04, 9.70e+04,
       2.00e+00, 1.90e+00, 1.80e+00, 5.30e+00, 4.70e+04, 5.56e+02,
       5.26e+02, 7.60e+04, 7.60e+00, 5.90e+04, 9.70e+00, 7.80e+04,
       7.20e+04, 4.30e+04, 7.70e+00, 6.30e+00, 3.34e+02, 3.40e+04,
       9.30e+04, 6.50e+04, 7.90e+04, 1.00e+05, 5.80e+04, 5.00e+04,
       6.80e+04, 6.40e+04, 6.70e+04, 6.00e+04, 9.40e+04, 2.32e+02,
       9.90e+04, 6.24e+02, 9.50e+04, 4.10e+01, 2.92e+02, 1.10e+01,
       8.00e+04, 1.70e+00, 7.40e+04, 6.20e+04, 6.90e+04, 7.50e+04,
       9.80e+04, 8.50e+04, 8.20e+04, 9.60e+04, 8.70e+04, 7.10e+04,
       8.60e+04, 9.10e+04, 8.10e+04, 9.20e+04, 8.30e+04, 8.80e+04,
       7.04e+02, 8.62e+02, 8.99e+02, 3.78e+02, 2.66e+02, 3.75e+02,
       1.30e+00, 9.75e+02, 9.80e+02, 4.10e+00, 8.90e+04, 6.96e+02,
```

```
       5.44e+02, 5.25e+02, 9.20e+02, 7.79e+02, 8.53e+02, 7.20e+02,
       7.13e+02, 7.72e+02, 3.18e+02, 5.80e+01, 2.41e+02, 1.96e+02,
       8.57e+02, 5.10e+01, 9.53e+02, 8.65e+02, 2.51e+02, 9.30e+02,
       5.40e+02, 3.13e+02, 7.46e+02, 2.03e+02, 2.60e+01, 3.14e+02,
       2.39e+02, 3.71e+02, 2.20e+02, 7.30e+02, 7.56e+02, 9.10e+01,
       2.93e+02, 1.70e+01, 7.40e+01, 1.40e+01, 3.17e+02, 7.80e+01,
       9.24e+02, 9.02e+02, 8.18e+02, 8.10e+01, 9.39e+02, 1.69e+02,
       4.50e+01, 4.75e+02, 9.65e+02, 9.00e+04, 5.45e+02, 6.10e+01,
       2.83e+02, 6.55e+02, 7.14e+02, 9.30e+01, 8.72e+02, 1.21e+02,
       3.22e+02, 1.00e+00, 9.76e+02, 1.72e+02, 2.38e+02, 5.49e+02,
       2.06e+02, 9.54e+02, 4.44e+02, 7.17e+02, 2.10e+02, 6.09e+02,
       3.08e+02, 7.05e+02, 3.06e+02, 9.04e+02, 4.73e+02, 1.75e+02,
       3.50e+02, 3.83e+02, 4.54e+02, 4.21e+02, 7.00e+01, 8.12e+02,
       4.42e+02, 8.42e+02, 4.17e+02, 4.12e+02, 4.59e+02, 4.78e+02,
       3.35e+02, 7.82e+02, 7.21e+02, 4.30e+02, 4.29e+02, 1.92e+02,
       2.00e+02, 4.60e+02, 7.28e+02, 4.96e+02, 8.16e+02, 4.14e+02,
       5.06e+02, 8.87e+02, 6.13e+02, 2.43e+02, 5.69e+02, 7.78e+02,
       6.83e+02, 5.92e+02, 3.19e+02, 1.86e+02, 8.40e+02, 6.47e+02,
       1.91e+02, 3.73e+02, 4.37e+02, 5.98e+02, 7.16e+02, 5.85e+02,
       9.82e+02, 2.22e+02, 2.19e+02, 5.50e+01, 9.48e+02, 3.23e+02,
       6.91e+02, 5.11e+02, 9.51e+02, 9.63e+02, 2.50e+01, 5.54e+02,
       3.51e+02, 2.70e+01, 8.20e+01, 2.08e+02, 9.13e+02, 5.14e+02,
       5.51e+02, 2.90e+01, 1.03e+02, 8.98e+02, 7.43e+02, 1.16e+02,
       1.53e+02, 2.09e+02, 3.53e+02, 4.99e+02, 1.73e+02, 5.97e+02,
       8.09e+02, 1.22e+02, 4.11e+02, 4.00e+02, 8.01e+02, 7.87e+02,
       2.37e+02, 5.00e+01, 6.43e+02, 9.86e+02, 9.70e+01, 5.16e+02,
       8.37e+02, 7.80e+02, 9.61e+02, 2.69e+02, 2.00e+01, 4.98e+02,
       6.00e+02, 7.49e+02, 6.42e+02, 8.81e+02, 7.20e+01, 6.56e+02,
       6.01e+02, 2.21e+02, 2.28e+02, 1.08e+02, 9.40e+02, 1.76e+02,
       3.30e+01, 6.63e+02, 3.40e+01, 9.42e+02, 2.59e+02, 1.64e+02,
       4.58e+02, 2.45e+02, 6.29e+02, 2.80e+01, 2.88e+02, 7.75e+02,
       7.85e+02, 6.36e+02, 9.16e+02, 9.94e+02, 3.09e+02, 4.85e+02,
       9.14e+02, 9.03e+02, 6.08e+02, 5.00e+02, 5.40e+01, 5.62e+02,
       8.47e+02, 9.57e+02, 6.88e+02, 8.11e+02, 2.70e+02, 4.80e+01,
       3.29e+02, 5.23e+02, 9.21e+02, 8.74e+02, 9.81e+02, 7.84e+02,
       2.80e+02, 2.40e+01, 5.18e+02, 7.54e+02, 8.92e+02, 1.54e+02,
       8.60e+02, 3.64e+02, 3.87e+02, 6.26e+02, 1.61e+02, 8.79e+02,
       3.90e+01, 9.70e+02, 1.70e+02, 1.41e+02, 1.60e+02, 1.44e+02,
       1.43e+02, 1.90e+02, 3.76e+02, 1.93e+02, 2.46e+02, 7.30e+01,
       6.58e+02, 9.92e+02, 2.53e+02, 4.20e+02, 4.04e+02, 4.70e+02,
       2.26e+02, 2.40e+02, 8.90e+01, 2.34e+02, 2.57e+02, 8.61e+02,
       4.67e+02, 1.57e+02, 4.40e+01, 6.76e+02, 6.70e+01, 5.52e+02,
       8.85e+02, 1.02e+03, 5.82e+02, 6.19e+02])
```

```python
for i in df_copy['Size']:
    if i<10:
        df_copy['Size']=df_copy['Size'].replace(i,i*1000)
df_copy['Size']=df_copy['Size']/1000
df_copy['Size']
```

```
0        19.0
1        14.0
2         8.7
3        25.0
4         2.8
         ...
10836    53.0
10837     3.6
10838     9.5
10839     NaN
10840    19.0
Name: Size, Length: 10840, dtype: float64
```

Dealing with Install and Price feature

```
df_copy['Installs'].unique()
```

```
array(['10,000+', '500,000+', '5,000,000+', '50,000,000+', '100,000+',
       '50,000+', '1,000,000+', '10,000,000+', '5,000+',
'100,000,000+',
       '1,000,000,000+', '1,000+', '500,000,000+', '50+', '100+',
'500+',
       '10+', '1+', '5+', '0+', '0'], dtype=object)
```

```
df_copy['Price'].unique()
```

```
array(['0', '$4.99', '$3.99', '$6.99', '$1.49', '$2.99', '$7.99',
'$5.99',
       '$3.49', '$1.99', '$9.99', '$7.49', '$0.99', '$9.00', '$5.49',
       '$10.00', '$24.99', '$11.99', '$79.99', '$16.99', '$14.99',
       '$1.00', '$29.99', '$12.99', '$2.49', '$10.99', '$1.50',
'$19.99',
       '$15.99', '$33.99', '$74.99', '$39.99', '$3.95', '$4.49',
'$1.70',
       '$8.99', '$2.00', '$3.88', '$25.99', '$399.99', '$17.99',
       '$400.00', '$3.02', '$1.76', '$4.84', '$4.77', '$1.61',
'$2.50',
       '$1.59', '$6.49', '$1.29', '$5.00', '$13.99', '$299.99',
'$379.99',
       '$37.99', '$18.99', '$389.99', '$19.90', '$8.49', '$1.75',
       '$14.00', '$4.85', '$46.99', '$109.99', '$154.99', '$3.08',
       '$2.59', '$4.80', '$1.96', '$19.40', '$3.90', '$4.59',
'$15.46',
       '$3.04', '$4.29', '$2.60', '$3.28', '$4.60', '$28.99', '$2.95',
       '$2.90', '$1.97', '$200.00', '$89.99', '$2.56', '$30.99',
'$3.61',
       '$394.99', '$1.26', '$1.20', '$1.04'], dtype=object)
```

Removing the unneccessery characters

```
chars_to_remove=['+',',','$']
cols_to_clean=['Installs','Price']
for item in chars_to_remove:
```

```
    for col in cols_to_clean:
        df_copy[col]=df_copy[col].str.replace(item,'')
```

```
df_copy['Installs'].unique()
```

```
array(['10000', '500000', '5000000', '50000000', '100000', '50000',
       '1000000', '10000000', '5000', '100000000', '1000000000',
'1000',
       '500000000', '50', '100', '500', '10', '1', '5', '0'],
dtype=object)
```

```
df_copy['Price'].unique()
```

```
array(['0', '4.99', '3.99', '6.99', '1.49', '2.99', '7.99', '5.99',
       '3.49', '1.99', '9.99', '7.49', '0.99', '9.00', '5.49',
'10.00',
       '24.99', '11.99', '79.99', '16.99', '14.99', '1.00', '29.99',
       '12.99', '2.49', '10.99', '1.50', '19.99', '15.99', '33.99',
       '74.99', '39.99', '3.95', '4.49', '1.70', '8.99', '2.00',
'3.88',
       '25.99', '399.99', '17.99', '400.00', '3.02', '1.76', '4.84',
       '4.77', '1.61', '2.50', '1.59', '6.49', '1.29', '5.00',
'13.99',
       '299.99', '379.99', '37.99', '18.99', '389.99', '19.90',
'8.49',
       '1.75', '14.00', '4.85', '46.99', '109.99', '154.99', '3.08',
       '2.59', '4.80', '1.96', '19.40', '3.90', '4.59', '15.46',
'3.04',
       '4.29', '2.60', '3.28', '4.60', '28.99', '2.95', '2.90',
'1.97',
       '200.00', '89.99', '2.56', '30.99', '3.61', '394.99', '1.26',
       '1.20', '1.04'], dtype=object)
```

Changing the datatype of Installs and Price Feature

```
df_copy['Installs']=df_copy['Installs'].astype('int')
df_copy['Price']=df_copy['Price'].astype('float')
```

Dealing with Last update feature

```
df_copy['Last Updated'].unique()
```

```
array(['January 7, 2018', 'January 15, 2018', 'August 1, 2018', ...,
       'January 20, 2014', 'February 16, 2014', 'March 23, 2014'],
      dtype=object)
```

Deriving Day,Month,Year from Last Updated Feature

```
df_copy['Last Updated'] = pd.to_datetime(df_copy['Last Updated'])
df_copy['Day']=df_copy['Last Updated'].dt.day
df_copy['Month']=df_copy['Last Updated'].dt.month
df_copy['Year']=df_copy['Last Updated'].dt.year
```

Checking the datatypes of the features

```
df_copy.dtypes
```

```
App                          object
Category                     object
Rating                      float64
Reviews                       int32
Size                        float64
Installs                      int32
Type                         object
Price                       float64
Content Rating               object
Genres                       object
Last Updated         datetime64[ns]
Current Ver                  object
Android Ver                  object
Day                           int64
Month                         int64
Year                          int64
dtype: object
```

Creating a new clean dataset

```
df_copy.to_csv('googleplaystore_cleaned.csv', index = False)
```

Handling Null Values

```
df_copy.isnull().sum()
```

```
App                  0
Category             0
Rating            1474
Reviews              0
Size              1695
Installs             0
Type                 1
Price                0
Content Rating       0
Genres               0
Last Updated         0
Current Ver          8
Android Ver          2
Day                  0
Month                0
Year                 0
dtype: int64
```

Since the number of null values are high , so, dropping it would not be a good step.

Imputing Mean/Median in place of Null values

```
df_copy_me_mo = df_copy.copy()
```

```
df_copy_me_mo['mean_Size'] =
df_copy_me_mo['Size'].fillna(df_copy_me_mo['Size'].mean())
df_copy_me_mo['median_Size'] =
df_copy_me_mo['Size'].fillna(df_copy_me_mo['Size'].median())
df_copy_me_mo['mean_Rating'] =
df_copy_me_mo['Rating'].fillna(df_copy_me_mo['Rating'].mean())
df_copy_me_mo['median_Rating'] =
df_copy_me_mo['Rating'].fillna(df_copy_me_mo['Rating'].median())
```

```
print('Original Size Variance', df_copy_me_mo['Size'].var())
print('Size Variance After mean imputation',
df_copy_me_mo['mean_Size'].var())
print('Size Variance After median imputation',
df_copy_me_mo['median_Size'].var())
```

```
Original Size Variance 510.5801557864865
Size Variance After mean imputation 430.7357638630519
Size Variance After median imputation 440.28217654605237
```

```
print('Original Rating Variance', df_copy_me_mo['Rating'].var())
print('Rating Variance After mean imputation',
df_copy_me_mo['mean_Rating'].var())
print('Rating Variance After median imputation',
df_copy_me_mo['median_Rating'].var())
```

```
Original Rating Variance 0.26545047227541496
Rating Variance After mean imputation 0.22935175503821595
Rating Variance After median imputation 0.23072842363353122
```

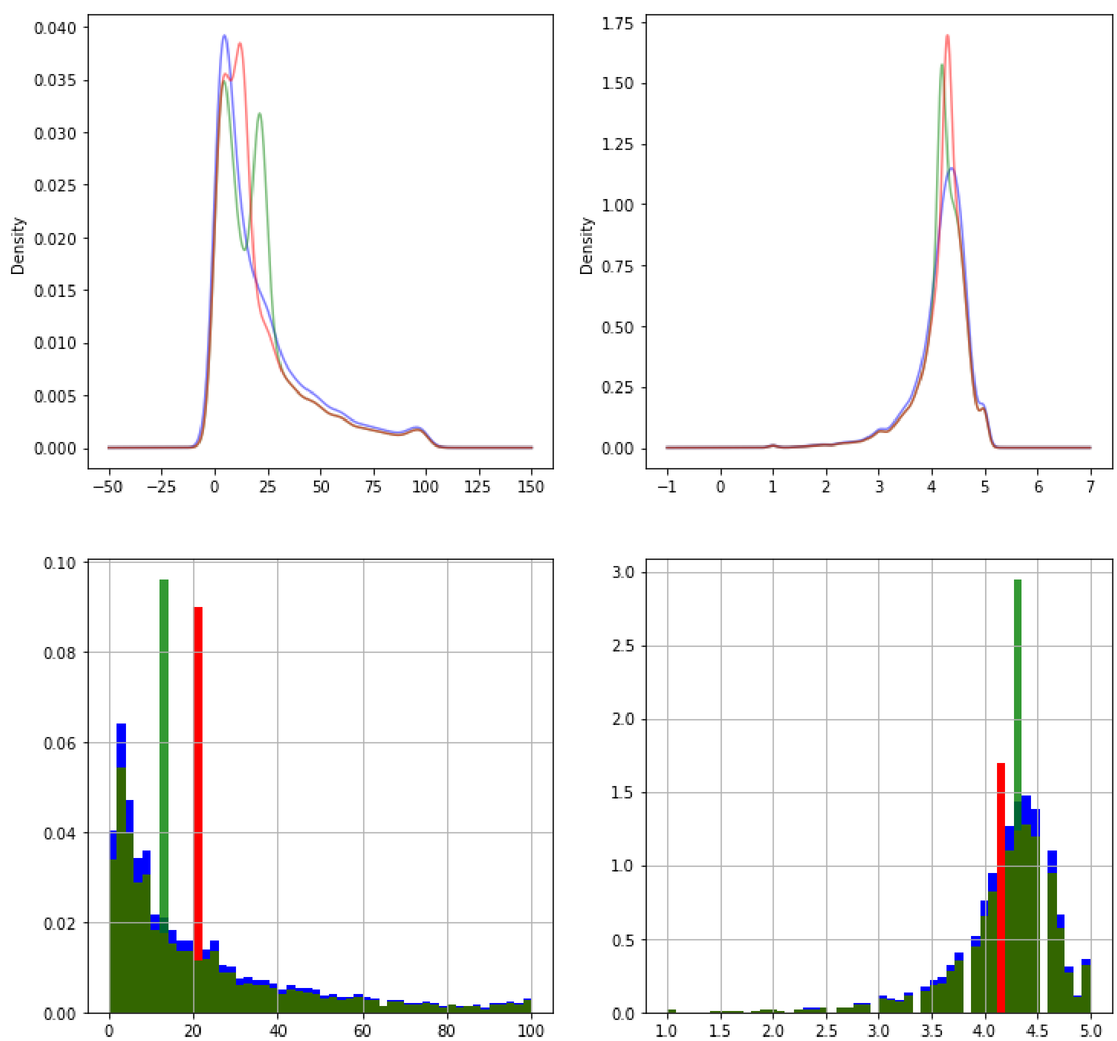
As we can observe Variance is distorted after both mean and median imputation

Graphical Analysis after imputation mean/median

```
fig= plt.figure()

# density plot using seaborn library
fig, axs = plt.subplots(2, 2, figsize=(15, 7))

df_copy_me_mo['Size'].plot.density(color='blue',ax=axs[0,
0],alpha=0.5,label='Size')
df_copy_me_mo['mean_Size'].plot.density(color='green',ax=axs[0,
0],alpha=0.5,label='mean_Size')
df_copy_me_mo['median_Size'].plot.density(color='red',ax=axs[0,
0],alpha=0.5,label='median_Size')

df_copy_me_mo['Rating'].plot.density(color='blue',ax=axs[0,
1],alpha=0.5,label='Rating')
df_copy_me_mo['mean_Rating'].plot.density(color='green',ax=axs[0,
1],alpha=0.5,label='mean_Rating')
df_copy_me_mo['median_Rating'].plot.density(color='red',ax=axs[0,
1],alpha=0.5,label='median_Rating')
```

```
df_copy_me_mo['Size'].hist(bins=50,ax=axs[1,
0],density=True,figsize=(12,12),color='blue')
df_copy_me_mo['mean_Size'].hist(bins=50,ax=axs[1,
0],density=True,figsize=(12,12),color='red')
df_copy_me_mo['median_Size'].hist(bins=50,ax=axs[1,
0],density=True,figsize=(12,12),color='green', alpha=0.8)

df_copy_me_mo['Rating'].hist(bins=50,ax=axs[1,
1],density=True,figsize=(12,12),color='blue')
df_copy_me_mo['mean_Rating'].hist(bins=50,ax=axs[1,
1],density=True,figsize=(12,12),color='red')
df_copy_me_mo['median_Rating'].hist(bins=50,ax=axs[1,
1],density=True,figsize=(12,12),color='green', alpha=0.8)
```

```
<AxesSubplot:>

<Figure size 432x288 with 0 Axes>
```



As we can observe from above plots

Mean and median imputation Technique is changing our distribution pattern. So we will reject mean and median imputation Technique also.

Random Sample Imputation

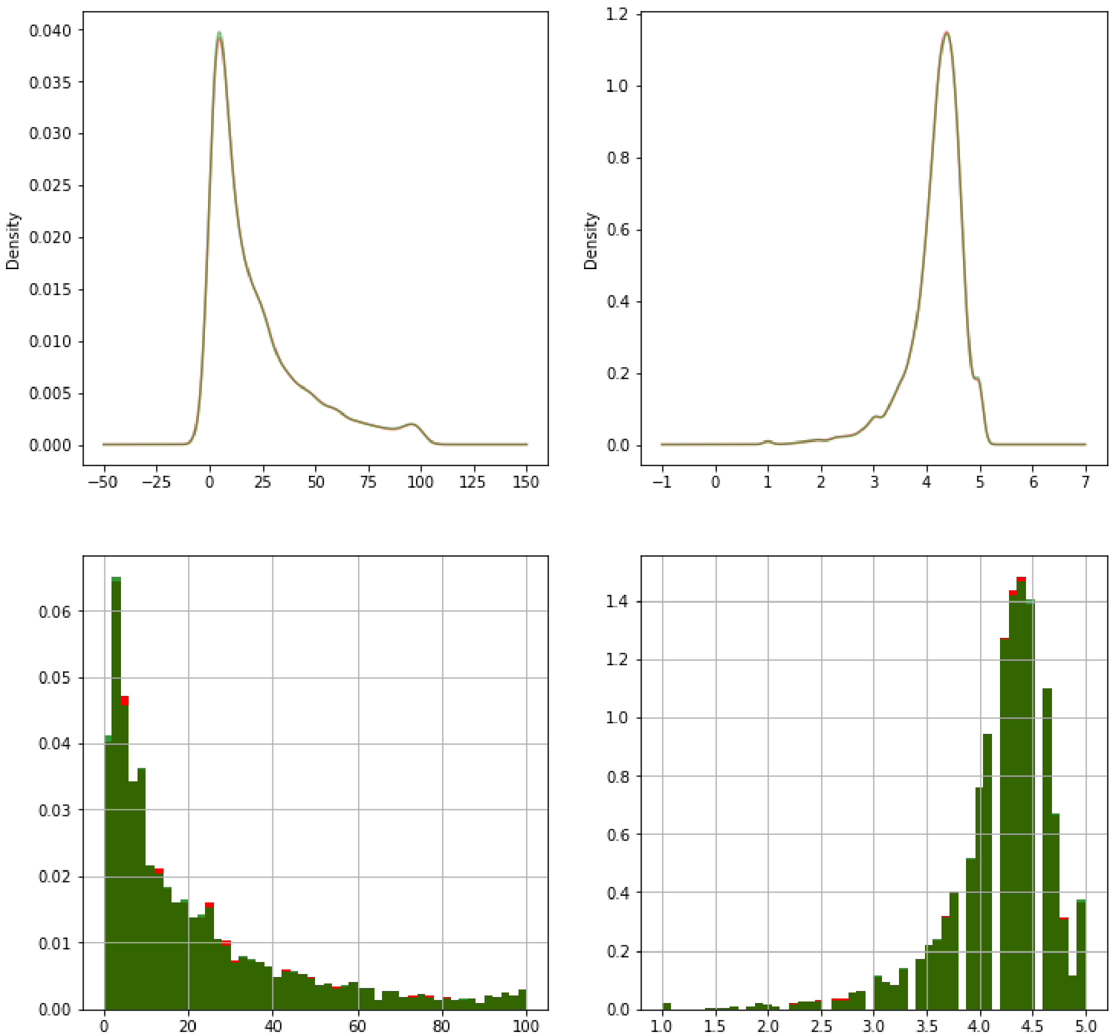
```
df_random = df_copy.copy()

def Random_Sample_imputation(feature):

random_sample=df_random[feature].dropna().sample(df_random[feature].is
null().sum())
    random_sample.index=df_random[df_random[feature].isnull()].index
    df_random.loc[df_random[feature].isnull(),feature]=random_sample

for col in df_random:
    Random_Sample_imputation(col)

print('Original Size Variance', df_copy['Size'].var())
print('Size Variance After Random imputation',
df_random['Size'].var())

Original Size Variance 510.5801557864865
Size Variance After Random imputation 513.6193789230667

print('Original Size Variance', df_copy['Rating'].var())
print('Size Variance After Rating imputation',
df_random['Rating'].var())

Original Size Variance 0.26545047227541496
Size Variance After Rating imputation 0.26446004494157366
```

Statistical Analysis after sample imputation in place of null value

```
fig= plt.figure()

# density plot using seaborn library
fig, axs = plt.subplots(2, 2, figsize=(15, 7))

df_copy['Size'].plot.density(color='red',ax=axs[0,
0],alpha=0.5,label='Size')
df_random['Size'].plot.density(color='green',ax=axs[0,
0],alpha=0.5,label='Size')
df_copy['Rating'].plot.density(color='red',ax=axs[0,
1],alpha=0.5,label='Rating')
df_random['Rating'].plot.density(color='green',ax=axs[0,
1],alpha=0.5,label='Rating')
df_copy['Size'].hist(bins=50,ax=axs[1,
0],density=True,figsize=(12,12),color='red')
df_random['Size'].hist(bins=50,ax=axs[1,
0],density=True,figsize=(12,12),color='green', alpha=0.8)
df_copy['Rating'].hist(bins=50,ax=axs[1,
```

```
1],density=True,figsize=(12,12),color='red')
df_random['Rating'].hist(bins=50,ax=axs[1,
1],density=True,figsize=(12,12),color='green', alpha=0.8)
```

```
<AxesSubplot:>

<Figure size 432x288 with 0 Axes>
```




```
null_df = pd.DataFrame({'Null Values' :
df_random.isna().sum().sort_values(ascending=False), 'Percentage Null
Values' : (df_random.isna().sum().sort_values(ascending=False)) /
(df_random.shape[0]) * (100)})
null_df
```

	Null Values	Percentage Null Values
App	0	0.0
Category	0	0.0
Rating	0	0.0
Reviews	0	0.0
Size	0	0.0

```
Installs                  0                      0.0
Type                      0                      0.0
Price                     0                      0.0
Content Rating            0                      0.0
Genres                    0                      0.0
Last Updated              0                      0.0
Current Ver               0                      0.0
Android Ver               0                      0.0
Day                       0                      0.0
Month                     0                      0.0
Year                      0                      0.0
```

As we can observe from above plots

Random Sample imputation Technique has no impact on distribution pattern. So we will accept Random Sample imputation Technique .

```
df_random.to_csv('googleplaystore_missing_imputed.csv', index = False)
```

Exploratory Data Analysis

```
data = pd.read_csv('googleplaystore_missing_imputed.csv')
data.head()
```

```
                                                 App        Category
Rating  \
0     Photo Editor & Candy Camera & Grid & ScrapBook  ART_AND_DESIGN
4.1
1                                Coloring book moana  ART_AND_DESIGN
3.9
2  U Launcher Lite – FREE Live Cool Themes, Hide ...  ART_AND_DESIGN
4.7
3                              Sketch - Draw & Paint  ART_AND_DESIGN
4.5
4              Pixel Draw - Number Art Coloring Book  ART_AND_DESIGN
4.3

   Reviews  Size  Installs  Type  Price Content Rating  \
0      159  19.0     10000  Free    0.0       Everyone
1      967  14.0    500000  Free    0.0       Everyone
2    87510   8.7   5000000  Free    0.0       Everyone
3   215644  25.0  50000000  Free    0.0           Teen
4      967   2.8    100000  Free    0.0       Everyone

                      Genres Last Updated         Current Ver
Android Ver  \
0               Art & Design   2018-01-07               1.0.0  4.0.3
and up
1  Art & Design;Pretend Play   2018-01-15               2.0.0  4.0.3
and up
```

```
2               Art & Design   2018-08-01               1.2.4  4.0.3
and up
3               Art & Design   2018-06-08  Varies with device    4.2
and up
4    Art & Design;Creativity   2018-06-20                 1.1    4.4
and up

   Day  Month  Year
0    7      1  2018
1   15      1  2018
2    1      8  2018
3    8      6  2018
4   20      6  2018
```

Shape

```
data.shape

(10840, 16)
```

Checking the null values

```
data.isnull().sum()

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
Day               0
Month             0
Year              0
dtype: int64
```

Zero null Value

Checking Datatypes

```
data.dtypes

App                object
Category           object
Rating            float64
Reviews             int64
Size              float64
```

```
Installs              int64
Type                 object
Price               float64
Content Rating       object
Genres               object
Last Updated         object
Current Ver          object
Android Ver          object
Day                   int64
Month                 int64
Year                  int64
dtype: object
```

Separating Numerical and Categorical features

```
numeric_features = [feature for feature in data.columns if
data[feature].dtype !='O']
categoric_features = [feature for feature in data.columns if
data[feature].dtype == 'O']

## Print Columns

print("We have {} numeric features : {}
".format(len(numeric_features),numeric_features))
print("\nWe have {} categorical features : {}
".format(len(categoric_features),categoric_features))
```

```
We have 8 numeric features : ['Rating', 'Reviews', 'Size', 'Installs',
'Price', 'Day', 'Month', 'Year']

We have 8 categorical features : ['App', 'Category', 'Type', 'Content
Rating', 'Genres', 'Last Updated', 'Current Ver', 'Android Ver']
```

Feature Information

1. App :- Name of the App

2. Category :- Category under which the App falls.

3. Rating :- Application's rating on playstore

4. Reviews :- Number of reviews of the App.

5. Size :- Size of the App.

6. Install :- Number of Installs of the App

7. Type :- If the App is free/paid

8. Price :- Price of the app (0 if it is Free)

9. Content Rating :- Appropiate Target Audience of the App.

10. Genres:- Genre under which the App falls.

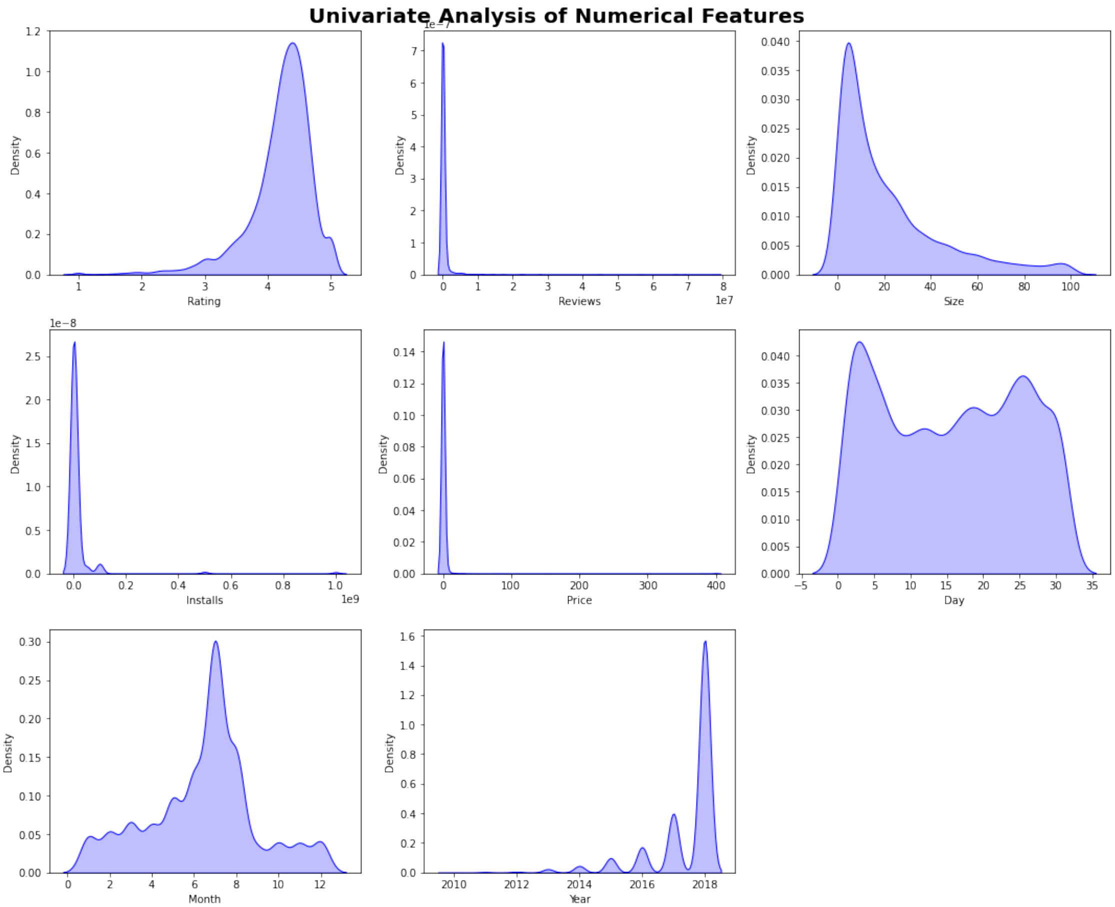
11. Last Updated :- Date when the App was last updated

12. Current Ver :- Current Version of the Application

13. Android Ver :- Minimum Android Version required to run the App

Univariate Analysis

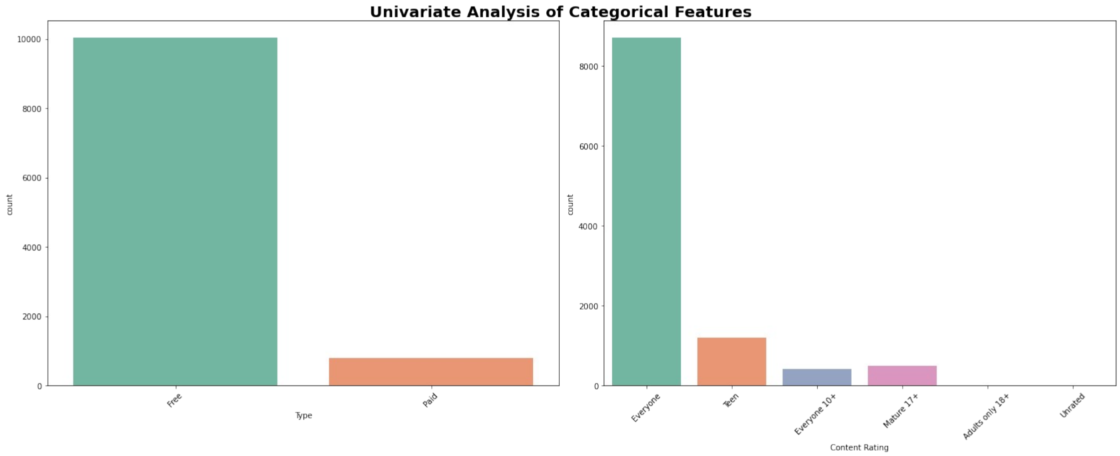
```
plt.figure(figsize=(15,20))
plt.suptitle('Univariate Analysis of Numerical Features', fontsize=20,
fontweight='bold')

for i in range(0, len(numeric_features)):
    plt.subplot(5, 3, i+1)
    sns.kdeplot(x=data[numeric_features[i]],shade=True, color='b')
    plt.xlabel(numeric_features[i])
    plt.tight_layout()
```



Rating and Year is Left skewed while Reviews, Size, Installs and Price are Right Skewed.

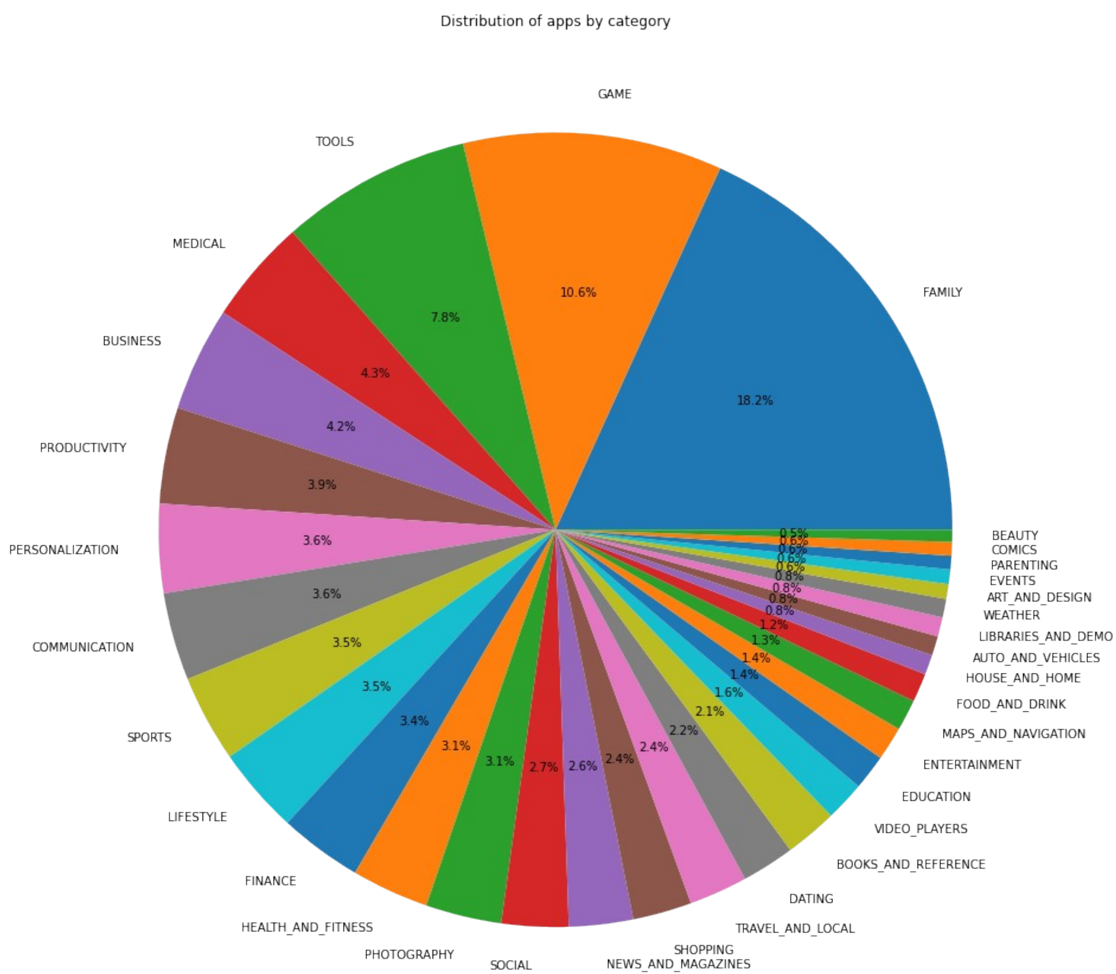
There are outliers in Reviews,Installs, Price and Year

```
# categorical columns
plt.figure(figsize=(20, 15))
plt.suptitle('Univariate Analysis of Categorical Features',
fontsize=20, fontweight='bold')
category = [ 'Type', 'Content Rating']
for i in range(0, len(category)):
    plt.subplot(2, 2, i+1)
    sns.countplot(x=data[category[i]],palette="Set2")
    plt.xlabel(category[i])
    plt.xticks(rotation=45)
    plt.tight_layout()
```



Which is the most popular app category?

```
data['Category'].value_counts().plot.pie(y = data['Category'], figsize = (15, 16), label = '', autopct = '%1.1f%%', title = 'Distribution of apps by category', );# label = '' removes column name
```



Insights

There are more kinds of apps in playstore which are under category of family, games & tools Beatuty,comics,arts and weather kinds of apps are very less in playstore

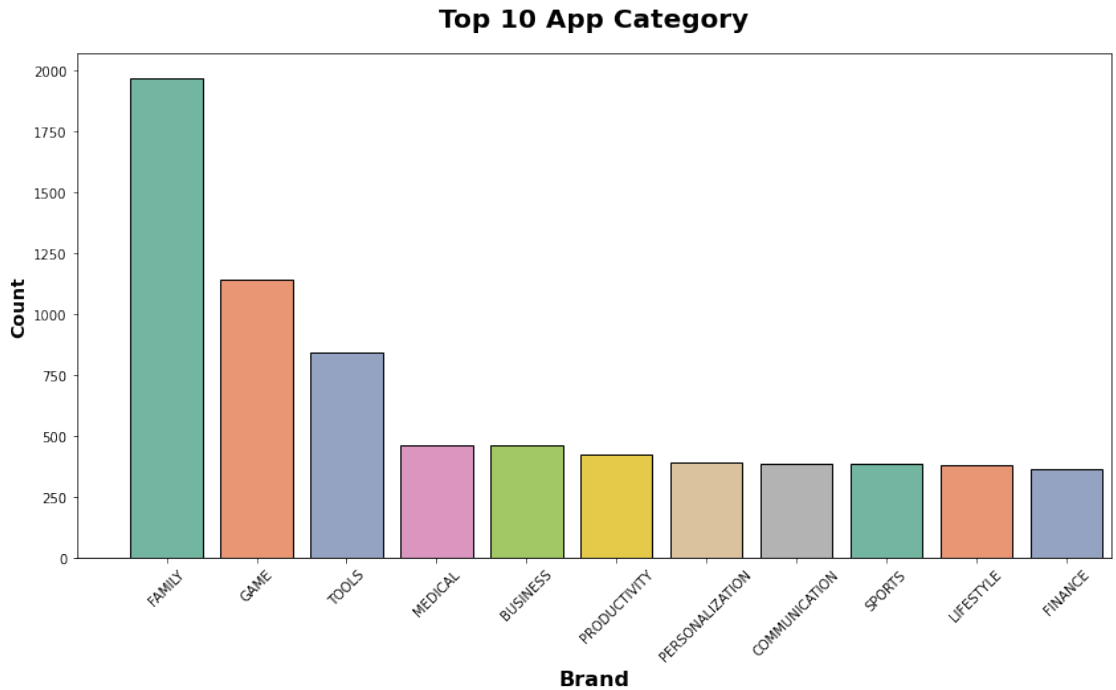
Top 10 App Categories

```
data.Category.value_counts()
```

```
FAMILY                 1972
GAME                   1144
TOOLS                   843
MEDICAL                 463
BUSINESS                460
PRODUCTIVITY            424
PERSONALIZATION         392
COMMUNICATION           387
SPORTS                  384
LIFESTYLE               382
FINANCE                 366
HEALTH_AND_FITNESS      341
PHOTOGRAPHY             335
SOCIAL                  295
NEWS_AND_MAGAZINES      283
SHOPPING                260
TRAVEL_AND_LOCAL        258
DATING                  234
BOOKS_AND_REFERENCE     231
VIDEO_PLAYERS           175
EDUCATION               156
ENTERTAINMENT           149
MAPS_AND_NAVIGATION     137
FOOD_AND_DRINK          127
HOUSE_AND_HOME           88
AUTO_AND_VEHICLES        85
LIBRARIES_AND_DEMO       85
WEATHER                  82
ART_AND_DESIGN           65
EVENTS                   64
PARENTING                60
COMICS                   60
BEAUTY                   53
Name: Category, dtype: int64
```

```
plt.subplots(figsize=(14,7))
sns.countplot(x="Category", data=data,ec =
"black",palette="Set2",order = data['Category'].value_counts().index)
plt.title("Top 10 App Category", weight="bold",fontsize=20, pad=20)
plt.ylabel("Count", weight="bold", fontsize=14)
plt.xlabel("Brand", weight="bold", fontsize=16)
plt.xticks(rotation= 45)
```

```
plt.xlim(-1,10.5)
plt.show()
```



Family category has the most number of apps , followed by Games category apps.

Least number of apps belong to the Beauty category .

Which Category has largest number of installations?

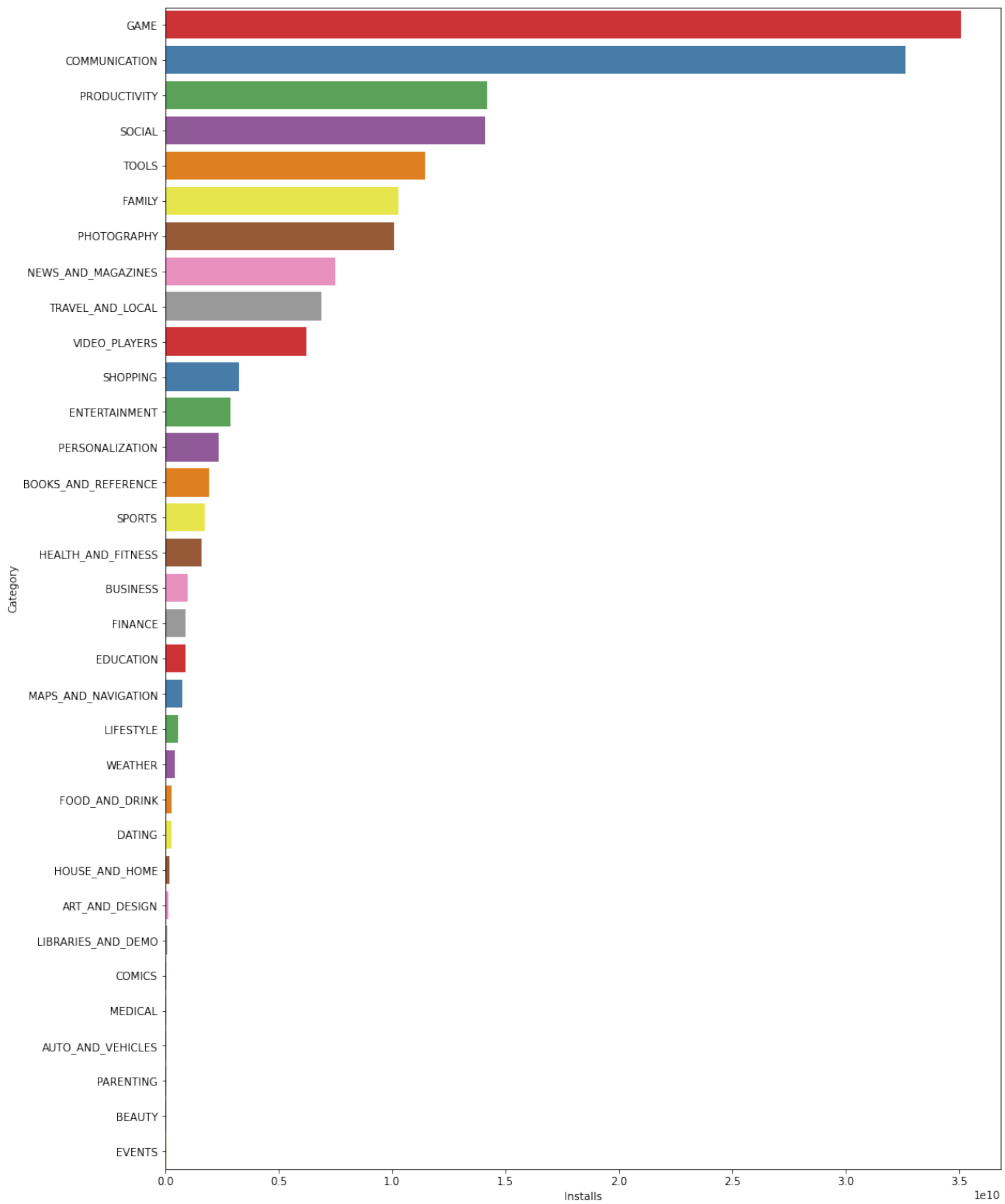
```
data_cat_installs = data.groupby(['Category'])
['Installs'].sum().sort_values(ascending = False).reset_index()

data_cat_installs

               Category     Installs
0                  GAME  35086024415
1         COMMUNICATION  32647276251
2          PRODUCTIVITY  14176091369
3                SOCIAL  14069867902
4                 TOOLS  11452771915
5                FAMILY  10258263505
6           PHOTOGRAPHY  10088247655
7    NEWS_AND_MAGAZINES   7496317760
8       TRAVEL_AND_LOCAL   6868887146
9         VIDEO_PLAYERS   6222002720
10             SHOPPING   3247848785
11        ENTERTAINMENT   2869160000
12      PERSONALIZATION   2325494782
13   BOOKS_AND_REFERENCE   1921469576
14               SPORTS   1751174498
```

```
15   HEALTH_AND_FITNESS   1583072512
16             BUSINESS   1001914865
17              FINANCE    876648734
18            EDUCATION    871452000
19  MAPS_AND_NAVIGATION    724281890
20            LIFESTYLE    537643539
21              WEATHER    426100520
22       FOOD_AND_DRINK    273898751
23               DATING    264310807
24       HOUSE_AND_HOME    168712461
25       ART_AND_DESIGN    124338100
26   LIBRARIES_AND_DEMO     62995910
27               COMICS     56086150
28              MEDICAL     53257437
29    AUTO_AND_VEHICLES     53130211
30            PARENTING     31521110
31               BEAUTY     27197050
32               EVENTS     15973161
```

```
plt.figure(figsize = (14,20))
```

```
sns.barplot(x='Installs', y='Category',
data=data_cat_installs,palette="Set1")
```

```
<AxesSubplot:xlabel='Installs', ylabel='Category'>
```



Out of all the categories "GAME" has the most number of Installations.

With almost 35 Billion Installations GAME is the most popular Category in Google App storens.

How many apps are there on Google Play Store which get 5 ratings??

```
rating = data.groupby(['Category','Installs', 'App'])
['Rating'].sum().sort_values(ascending = False).reset_index()

toprating_apps = rating[rating.Rating == 5.0]
```

```
print("Number of 5 rated apps",toprating_apps.shape[0])
toprating_apps.head(1)
```

```
Number of 5 rated apps 322

    Category  Installs      App  Rating
746   FAMILY       100  CF Life     5.0
```

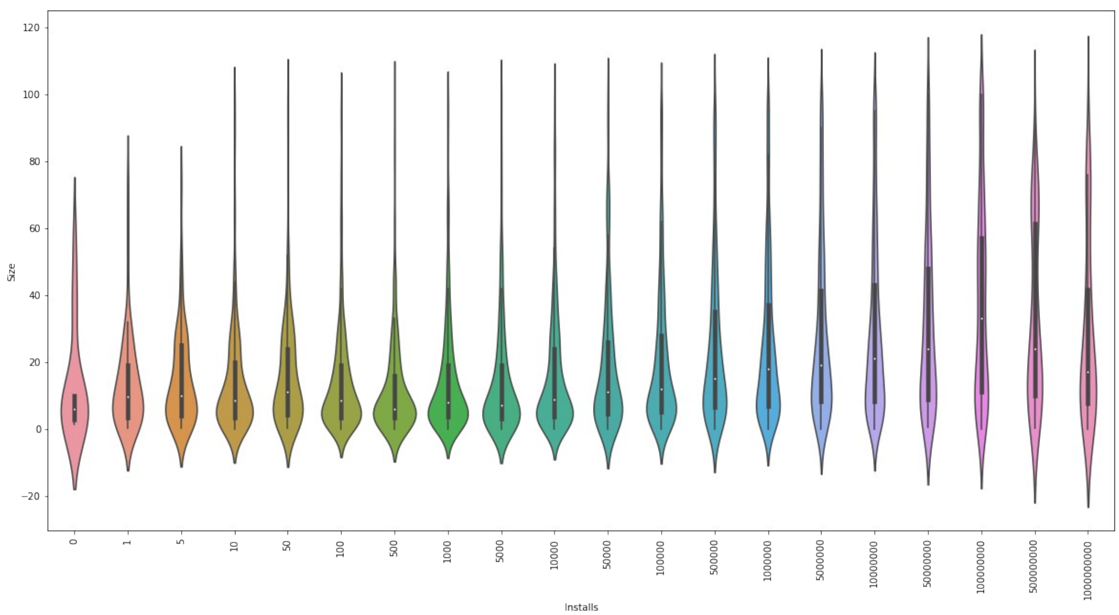
How many apps are there on Google Play Store which get 4.5 ratings??

```
toprating_apps = rating[rating.Rating ==4.5]
print("Number of 4.5 rated apps",toprating_apps.shape[0])
toprating_apps.head(1)
```

```
Number of 4.5 rated apps 924

     Category  Installs                      App  Rating
2642   FAMILY       500  Raytheon F-16 EW 360 VR     4.5
```

Does Size of application has any impact on its popularity ??

```
# Using violin plot to plot the relation
plt.figure(figsize=(20,10))
sns.violinplot(x=data.Installs, y=data.Size)
plt.xticks(rotation = 90)
plt.show()
```



We can observe from the plot that there is a large impact by the size of the app on the number of installations.

The tiny white circle in middle of each plot shows the median value of each value of installations.

Across the plot, we can see the median grows steadliy higher.

Initially on Install axis, there is a higher number of outliers with respect to the size of the apps. As we progress across the Install axis, the number of outliers decreases and the number of installations increases.

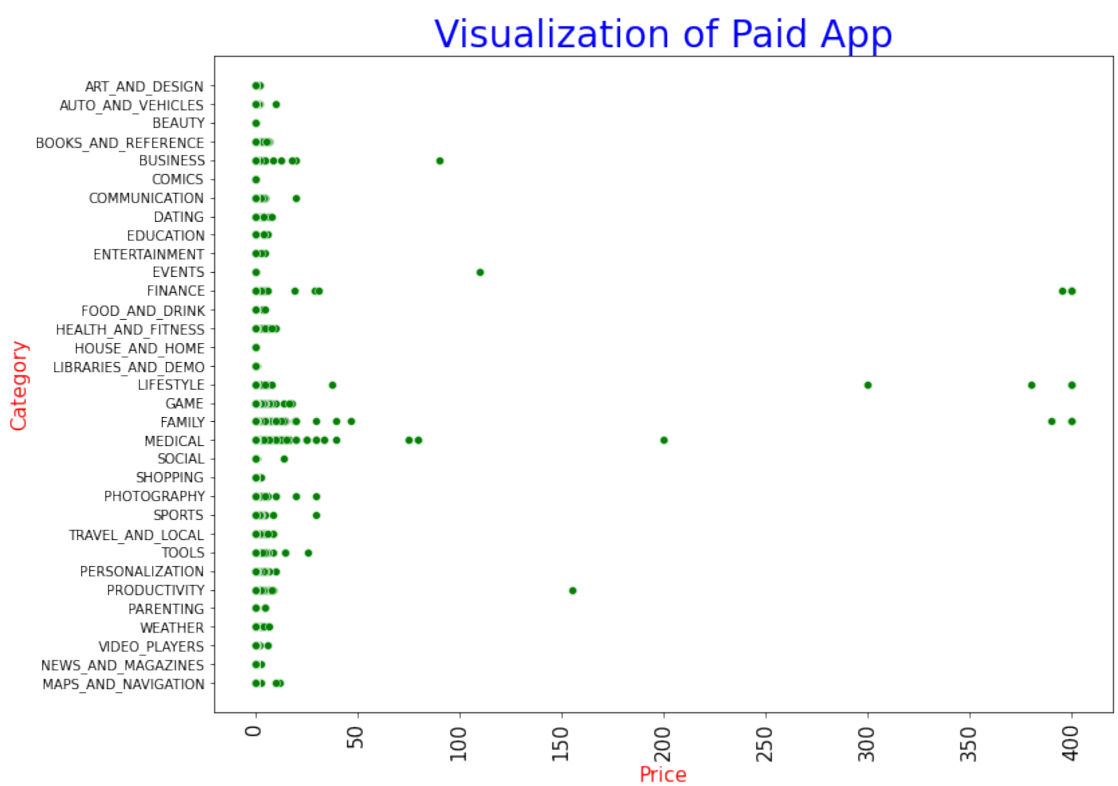
As the number of installations reaches the maximum value, we see that the app size has reached the lowest values, peaking at posiibly, 100 -110 MB.

From this we can conclude that bigger the app, lesser the chance for it to be installed plt.figure(figsize=(12,9))

Category Vs Price

```
plt.figure(figsize = (12,9))

sns.scatterplot(y='Category', x='Price', data=data,color='g')
plt.xticks(rotation='vertical',size=15)
plt.yticks(size=10)
plt.xlabel("Price",size=15,c="r")
plt.ylabel("Category",size=15,c="r")
plt.title("Visualization of Paid App",size=28,c="b")
```

```
Text(0.5, 1.0, 'Visualization of Paid App')
```



There are few Apps related to finance games and lifestyle are costly in paid category apps.

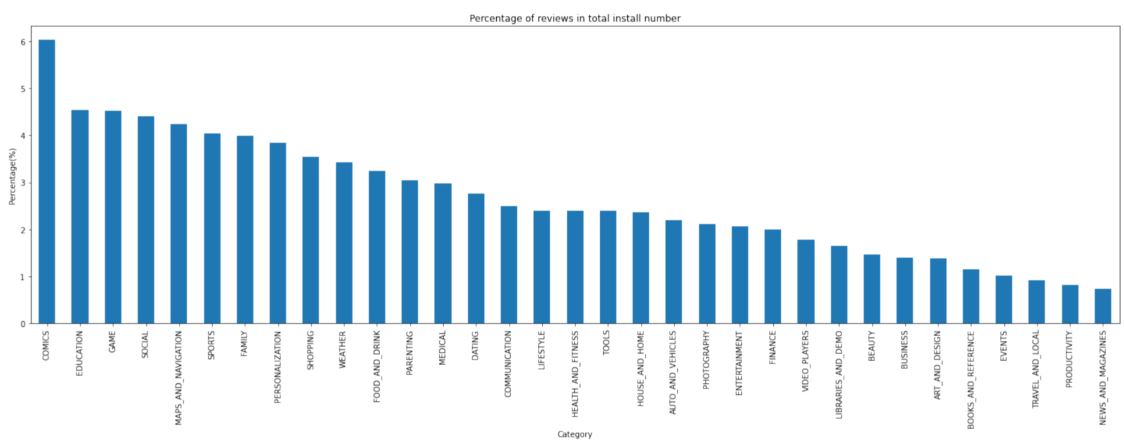
Paid apps are very very less.

0 indicate free apps.

Which Category app users are reviewing the most ??

```
data_installs_reviews =
data.groupby('Category').agg({'Installs':'sum','Reviews':'sum'})
data_installs_reviews['reviews_percent'] =
(data_installs_reviews['Reviews'] / data_installs_reviews['Installs'])
* 100
plt.figure(figsize=(25,7))
reviews_data  = data_installs_reviews.sort_values('reviews_percent',
ascending=False)['reviews_percent'].plot(kind='bar', title='Percentage
of reviews in total install number')
reviews_data.set(ylabel='Percentage(%)')
```

```
[Text(0, 0.5, 'Percentage(%)')]
```



Users downloading "Comics" with higher rate to leave a review but with relative high spread of the rating comparing with others.

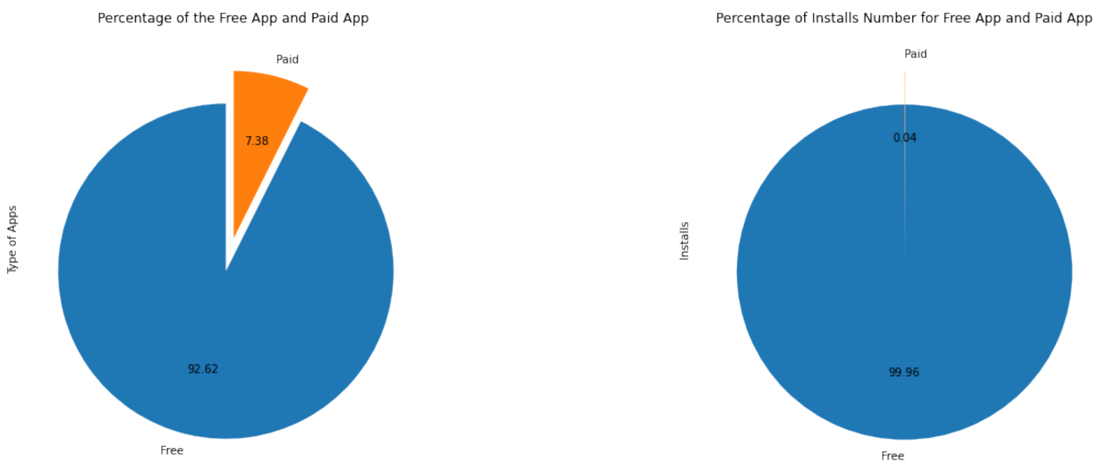
"Comic" ranks 6 in the number of installation.

The two categories with high download rate are having relatively low review rate.

Which kinds of apps users are downloading the most- free/ paid??

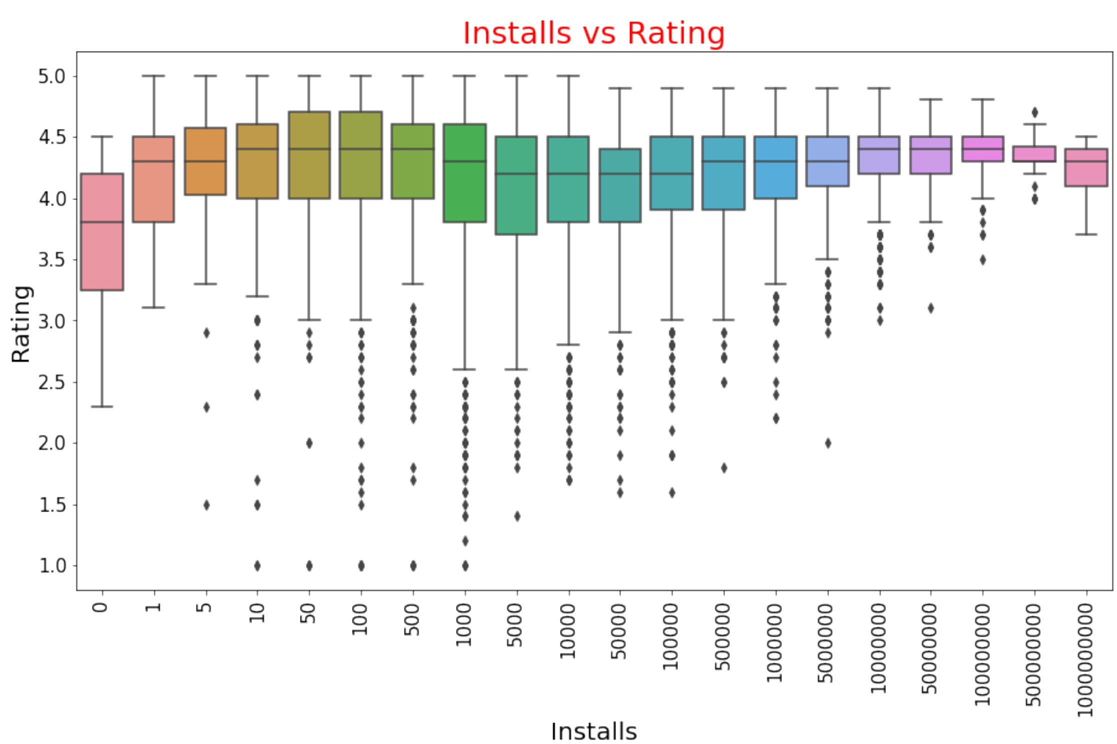
```
fig, ax = plt.subplots(1,2,figsize=(20,7))
data.value_counts('Type').plot.pie(y='Type',startangle=90,
explode=(0.2,0), title='Percentage of the Free App and Paid App',
legend=False, autopct='%.2f', ax=ax[0])
ax[0].set(ylabel='Type of Apps')
data.groupby('Type').agg({'Installs':sum}).plot.pie(y='Installs',
startangle=90, explode=(0.2,0), title='Percentage of Installs Number
for Free App and Paid App', legend=False, autopct='%.2f', ax=ax[1])
```

```
<AxesSubplot:title={'center':'Percentage of Installs Number for Free
App and Paid App'}, ylabel='Installs'>
```



which apps has good ratings on google play store ??

```python
#boxplot plot installs/rates
ax = plt.figure(figsize=(15,8))
sns.boxplot(x="Installs", y="Rating", data=data)
plt.title("Installs vs Rating",size=25,c="r")
plt.xticks(size=15,rotation=90)
plt.yticks(size=15)
plt.xlabel("Installs",size=20)
plt.ylabel("Rating",size=20)
plt.show()
```

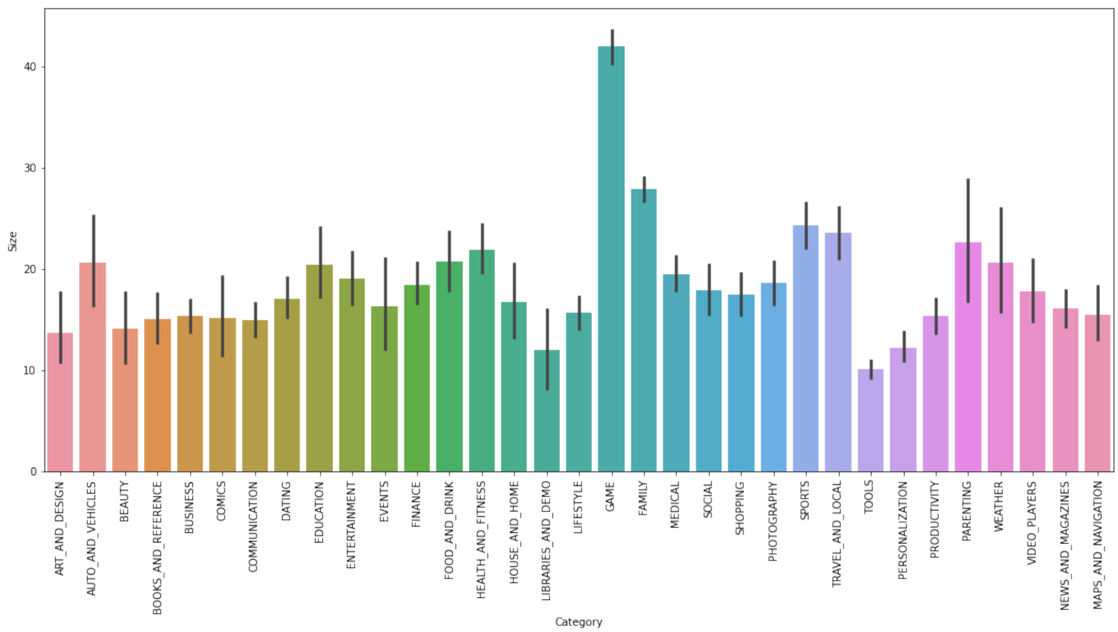


Average rating of all apps is between 3 to 4.5.

Highly installed apps having rating is also good.

Which Category apps are of largest size ?

```
plt.subplots(figsize = (18,8))
plt.xticks(rotation = 90)
sns.barplot('Category','Size', data = data)
plt.show()
```



As is evident that GAME category has the largest size.

We have seen that an average game size exceeds 35MB. Some games are also in the order of GBs.

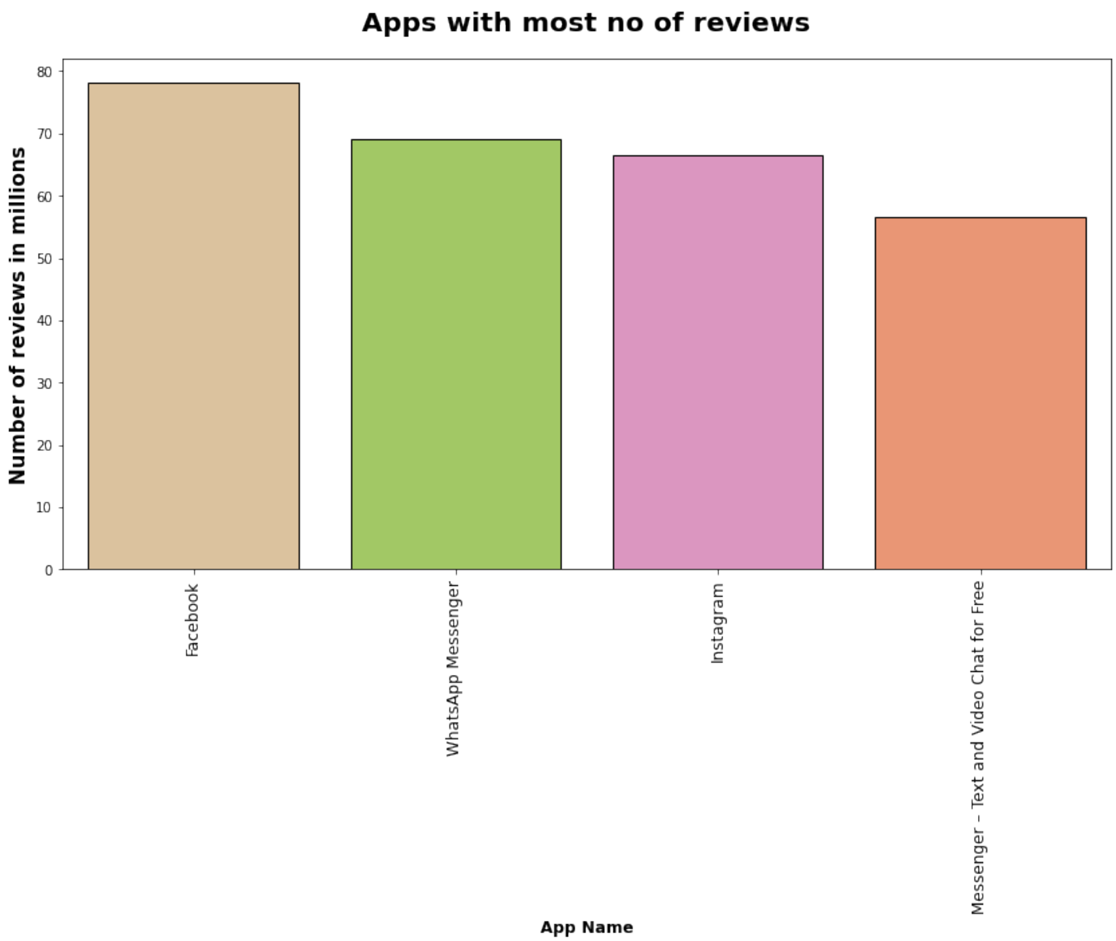
The FAMILY apps have the second largest size.

Which app with most number of reviews and its number of reviews??

```
app_most_reviews = data[data['Reviews'] == data['Reviews'].max()]
[['App', 'Reviews']].reset_index(drop=True)

print('App with most no of reviews is:
{}'.format(app_most_reviews['App'][0]))
print('And it has {} reviews'.format(app_most_reviews['Reviews'][0]))

App with most no of reviews is: Facebook
And it has 78158306 reviews

plt.subplots(figsize=(14,7))
reviews_data = data.sort_values('Reviews', ascending=False)[['App',
'Reviews']][:10]
```

```
reviews_data['Reviews'] = reviews_data['Reviews']/10**6
sns.barplot(x=reviews_data.App, y=reviews_data.Reviews, ec = "black",
palette="Set2_r")
plt.xticks(rotation=90,size=12)
plt.title("Apps with most no of reviews", weight="bold",fontsize=20,
pad=20)
plt.ylabel("Number of reviews in millions", weight="bold",
fontsize=15)
plt.xlabel("App Name", weight="bold", fontsize=12)
```

```
Text(0.5, 0, 'App Name')
```

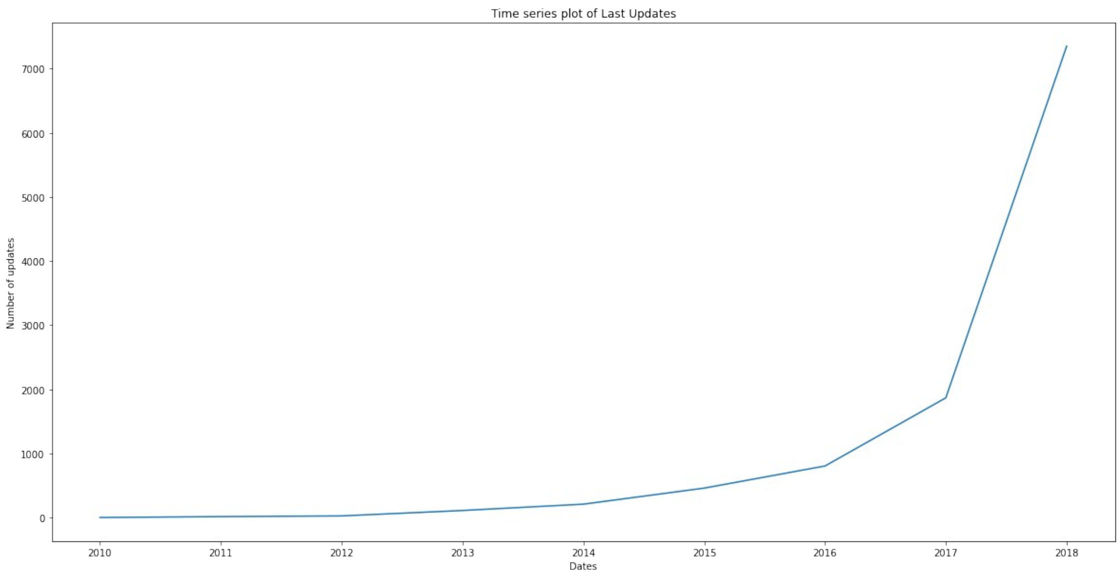


Facebook app has largest number of reviews followed by whatsapp

Time Series

```
plt.subplots(figsize=(20,10))
freq= pd.Series()
freq=data['Year'].value_counts()
freq.plot()
plt.xlabel("Dates")
plt.ylabel("Number of updates")
plt.title("Time series plot of Last Updates")
```

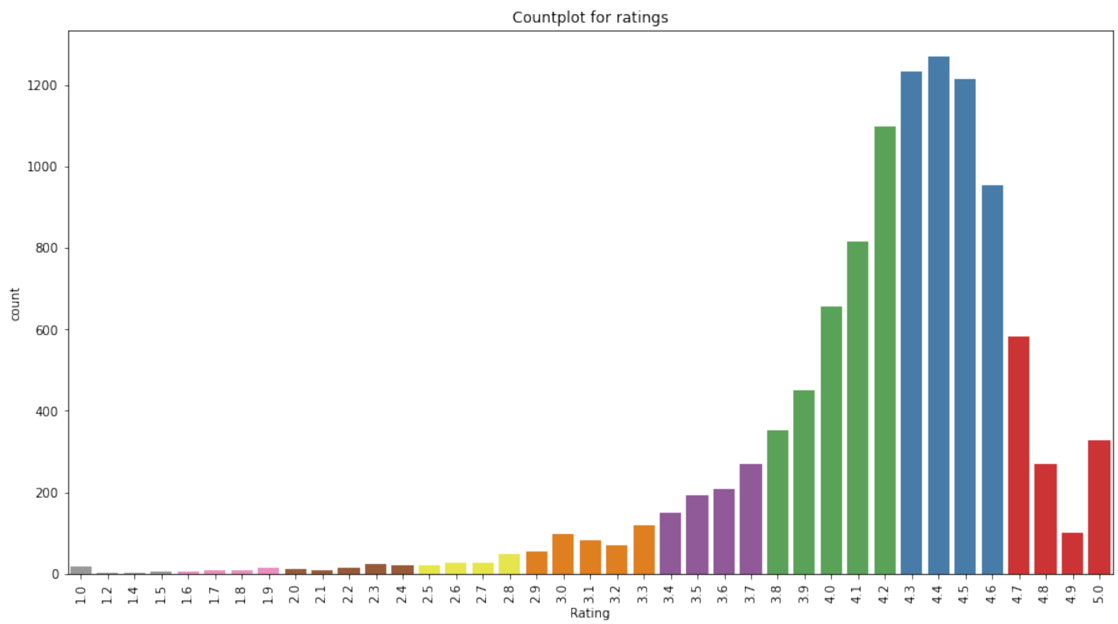
```
Text(0.5, 1.0, 'Time series plot of Last Updates')
```



After 2012 , with year number of updates are also increasing

Countplot for Rating

```
plt.figure(figsize=(15,8))
sns.countplot(x='Rating',data = data,palette="Set1_r")
plt.xticks(rotation =90)
plt.title('Countplot for ratings')
plt.show()
```




```
rating_greater_4 = len(data[data['Rating'] >= 4])/len(data)*100
print('Percentage of Apps having ratings of 4 or greater: {}
%'.format(round(rating_greater_4,2)))
```

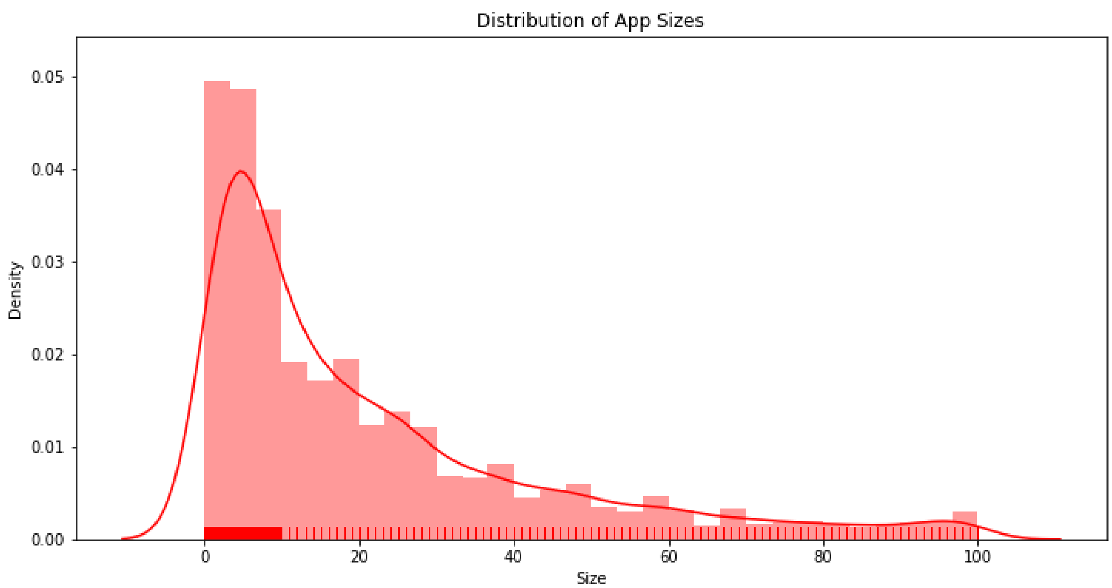
```
Percentage of Apps having ratings of 4 or greater: 78.63%
```

Majority of apps in the playstore have a rating 4 or above

Distribution of App size

```
plt.figure(figsize=(12,6))
plt.title('Distribution of App Sizes')
sns.distplot(data['Size'],bins = 30,rug=True,color="Red")
```

```
<AxesSubplot:title={'center':'Distribution of App Sizes'},
xlabel='Size', ylabel='Density'>
```



Do expensive apps have higher rating?

```
plt.figure(figsize=(12,6))
sns.regplot(x='Price', y='Rating', data=data)
plt.title('Price VS Rating')
```

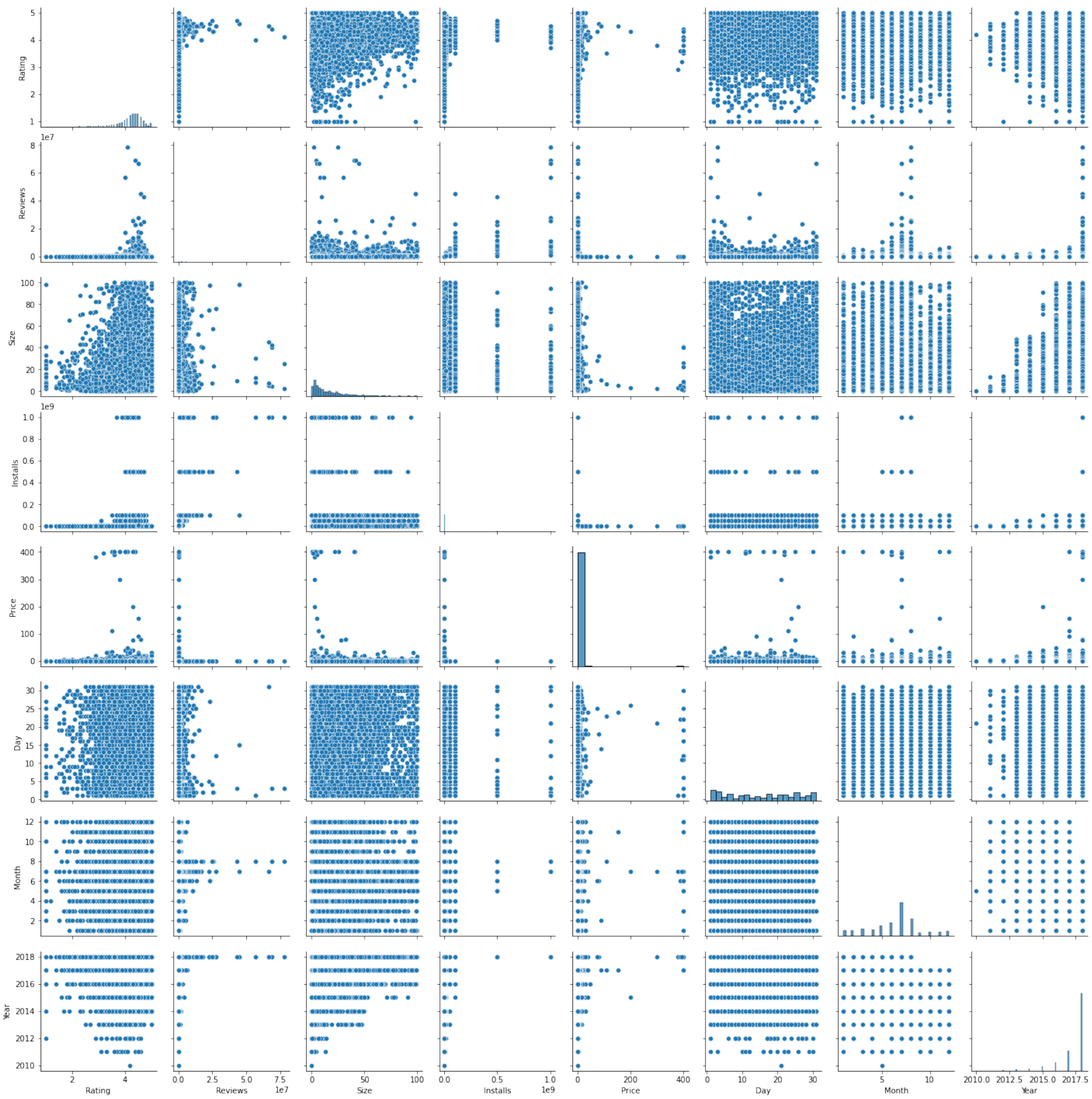
```
Text(0.5, 1.0, 'Price VS Rating')
```

From this plot we can see a slight positive trend between price and rating: apps with higher prices tends to be slightly higher rated.

Pairplot

```
sns.pairplot(data)
```

```
<seaborn.axisgrid.PairGrid at 0x206df10e790>
```



Conclusions:

The higher the rating, more people will be inclined to download the app.

Better the Reviews more are the chances for the app to be downloaded by more people.

People are always inclined to download apps that are free of cost.

Apps that falls under the Content Rating, 'Everyone', 'Teens' and 'Everyone 10+' has the highest chance to be downloaded.

The apps with smaller sizes have more chance to be downloaded.

Subway surfer is the most downloaded app followed by Instagram, Hangouts and Google Drive

92.6% of the apps in the app store are free.

Feature Engineering

1.Exploring Features of the dataset

2.Check Correlation using Heatmap

3.Hypothesis Testing (Check Normal Distribution)

3.1) Shapiro Wick Test

3.2) K^2 Normality Test

4.Checking for Normal Distribution using Transformations

4.1) Log Transformation

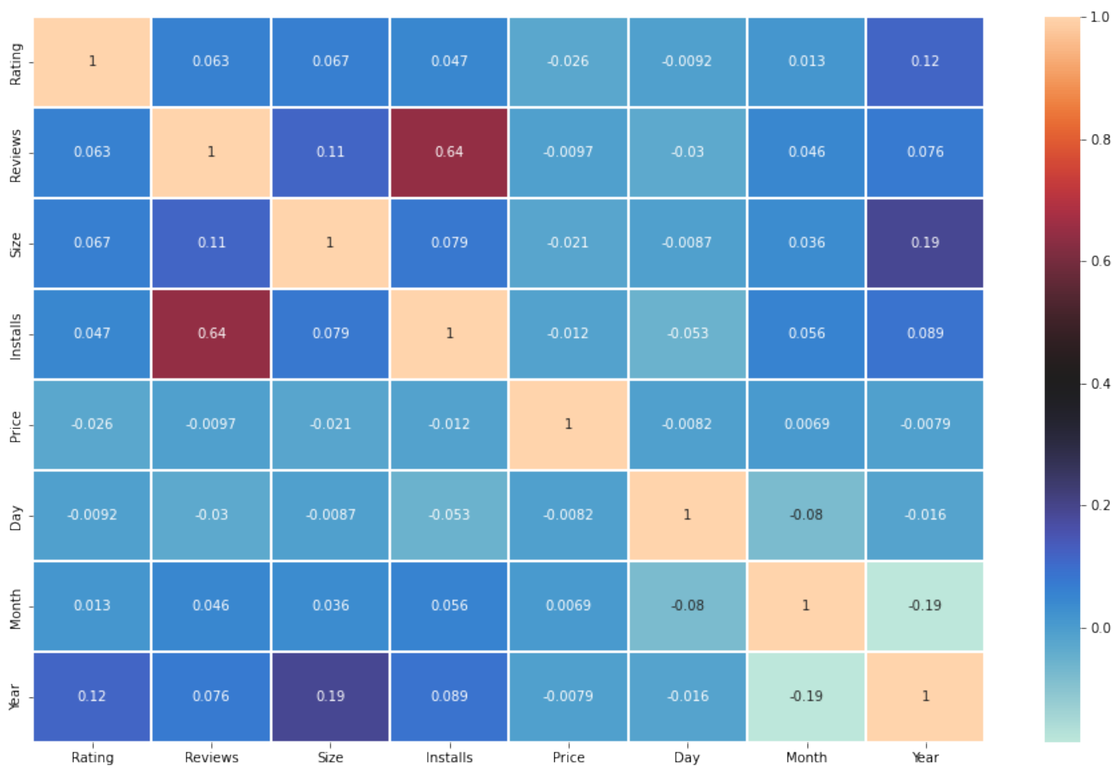
4.2) Square-Root Transformation

4.3) Yeo-Johnson Transformation

2. Check Correlation using Heatmap

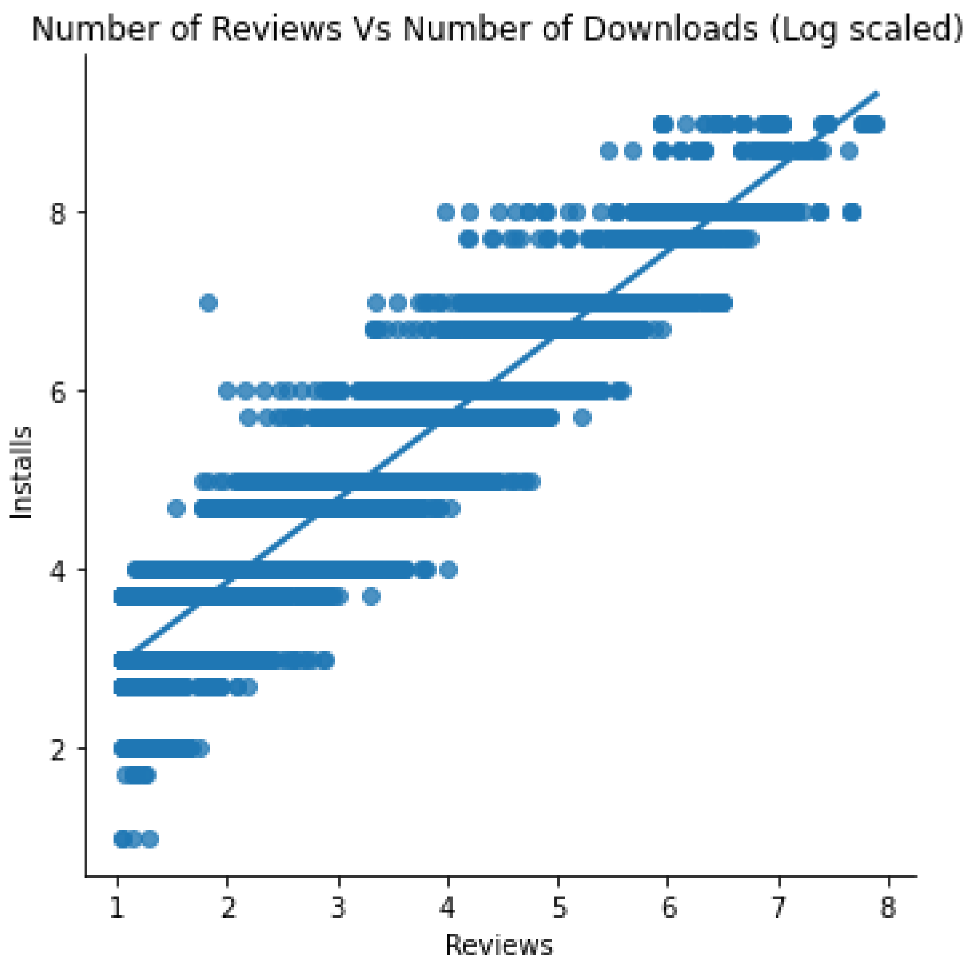
```
plt.figure(figsize=(16,10))

sns.heatmap(data.corr(),annot=True,cmap='icefire',linewidths=0.2)
#data.corr()-->correlation matrix
```

```
<AxesSubplot:>
```



```
plt.figure(figsize=(12,6))
data_copy = data.copy()

data_copy = data_copy[data_copy.Reviews > 10]
data_copy = data_copy[data_copy.Installs > 0]

data_copy['Installs'] = np.log10(data['Installs'])
data_copy['Reviews'] = np.log10(data['Reviews'])

sns.lmplot("Reviews", "Installs", data=data_copy)
ax = plt.gca()
_ = ax.set_title('Number of Reviews Vs Number of Downloads (Log
scaled)')
```

```
<Figure size 864x432 with 0 Axes>
```



Insights

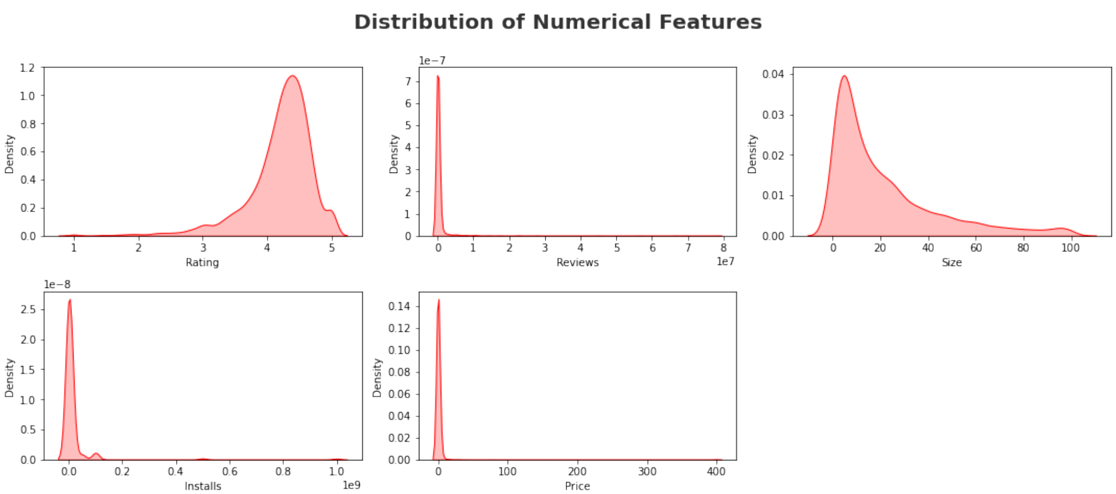
A High positive correlation of 0.9 exists between the number of reviews and number of downloads. This means that customers tend to download a given app more if it has been reviewed by a larger number of people.

This also means that many active users who download an app usually also leave back a review or feedback.

So, getting your app reviewed by more people maybe a good idea to increase your app's capture in the market!

Hypothesis Testing (Check Normal Disrtibution)

```
numeric_features = [feature for feature in data.columns if
data[feature].dtype != 'O']
plt.figure(figsize=(15, 15))
plt.suptitle('Distribution of Numerical Features', fontsize=20,
fontweight='bold', alpha=0.8, y=1.)

res_list = numeric_features[: len(numeric_features) - 3]
num_df = data[res_list]
for i in range(0, len(res_list)):
    plt.subplot(5, 3, i+1)
    sns.kdeplot(x=data[numeric_features[i]],shade=True, color='r')
    plt.xlabel(numeric_features[i])
    plt.tight_layout()
```



Shapiro Wick Test

The Shapiro-Wilk test is a way to tell if a random sample comes from a normal distribution.

Ho : Data is normally distributed

H1 : Data is not normally distributed

```
from sklearn.preprocessing import PowerTransformer
from sklearn.preprocessing import FunctionTransformer

from scipy.stats import shapiro
shapiro_wick_test = []
for column in res_list:
    dataToTest = num_df[column]
```

```
    stat,p = shapiro(dataToTest)
    if p > 0.05:
        shapiro_wick_test.append("Normally Distributed")
    else:
        shapiro_wick_test.append("Not Normally Distributed")
result = pd.DataFrame(data=[res_list, shapiro_wick_test]).T
result.columns = ['Column Name', 'Shapiro Hypothesis Result']
result
```

```
  Column Name Shapiro Hypothesis Result
0      Rating  Not Normally Distributed
1     Reviews  Not Normally Distributed
2        Size  Not Normally Distributed
3    Installs  Not Normally Distributed
4       Price  Not Normally Distributed
```

K^2 Normality Test

Test aims to establish whether or not the given sample comes from a normally distributed population. Test is based on transformations of the sample kurtosis and skewness

Ho : Data is normally distributed

H1 : Data is not normally distributed

```
from scipy.stats import normaltest
normaltest_test = []
for column in res_list:
    dataToTest = num_df[column]
    stat,p = normaltest(dataToTest)
    if p > 0.05:
        normaltest_test.append("Normally Distributed")
    else:
        normaltest_test.append("Not Normally Distributed")
result = pd.DataFrame(data=[res_list, normaltest_test]).T
result.columns = ['Column Name', 'normaltest Hypothesis Result']
result
```

```
  Column Name normaltest Hypothesis Result
0      Rating     Not Normally Distributed
1     Reviews     Not Normally Distributed
2        Size     Not Normally Distributed
3    Installs     Not Normally Distributed
4       Price     Not Normally Distributed
```

Checking for Normal Distribution using Transformations

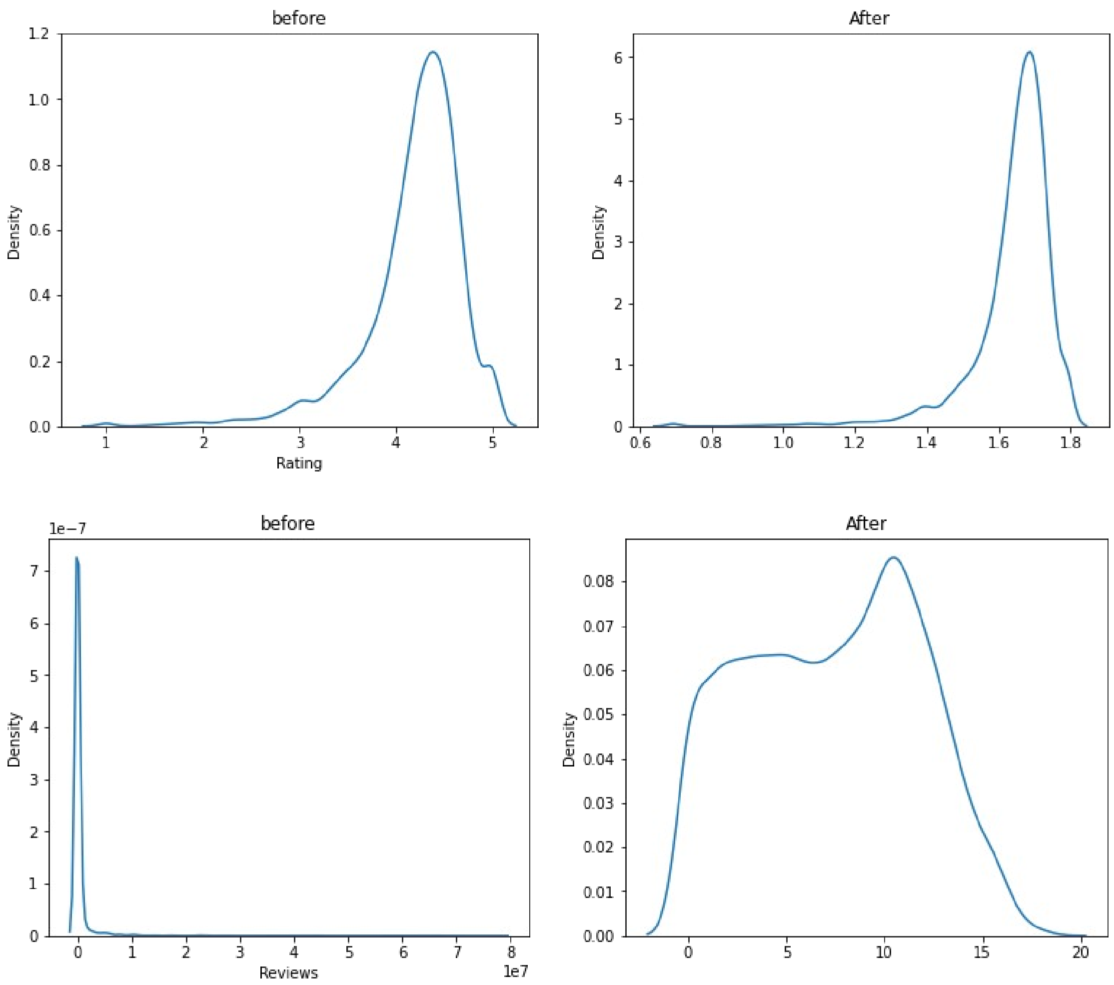
```
def plots(data,var,transformer):
    plt.figure(figsize=(13,5))
    plt.subplot(121)
    sns.kdeplot(data[var])
```

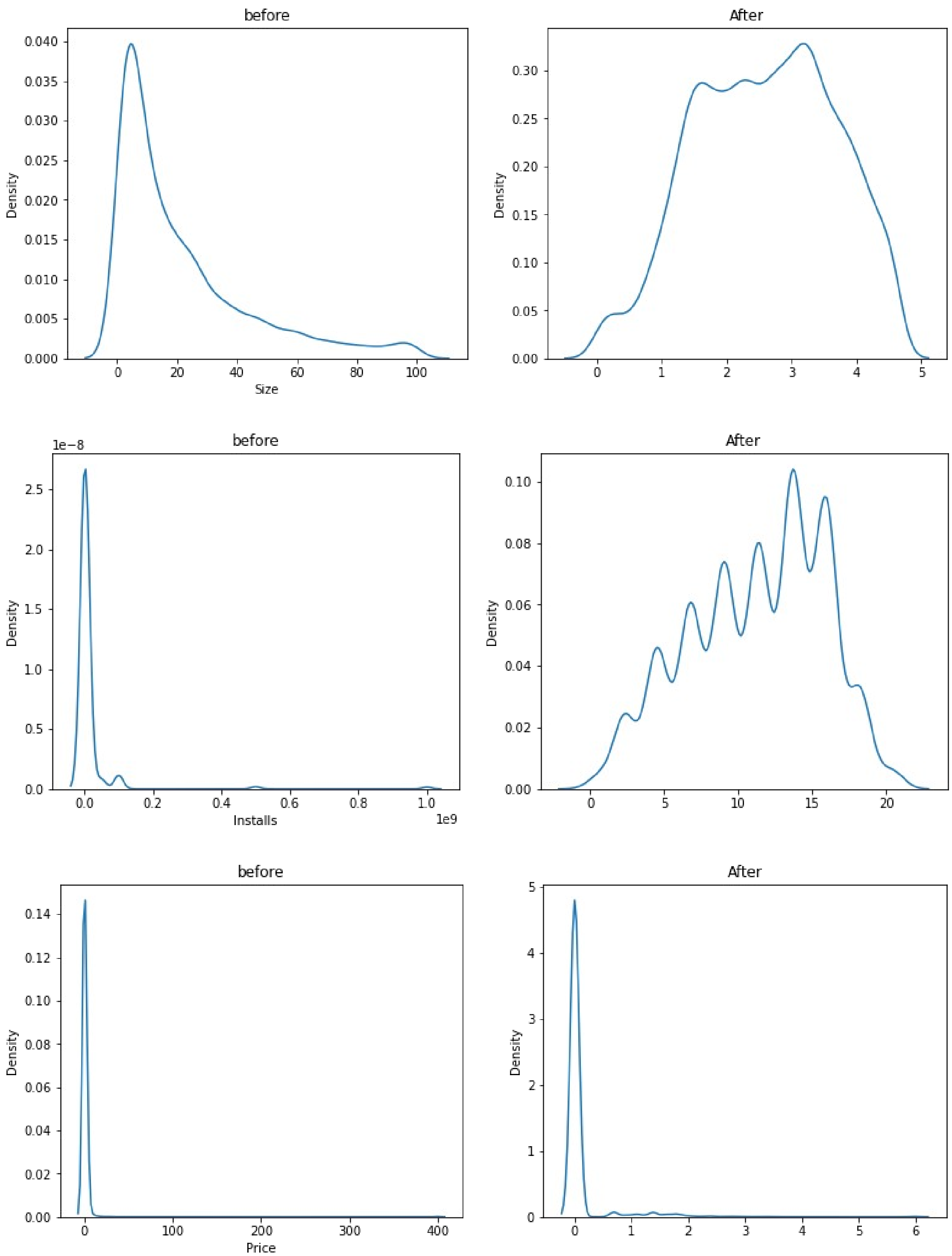
```
    plt.title('before' )
    plt.subplot(122)
    sns.kdeplot(transformer)
    plt.title('After')
```

Log Transformation

In Log transformation each variable of x will be replaced by log(x) with base 10, base 2, or natural log.

```
log_transformer = FunctionTransformer(np.log1p)
for col in res_list:
    X = np.array(data[col])
    Y = log_transformer.transform(X)
    plots(data,col,Y)
```





Insights

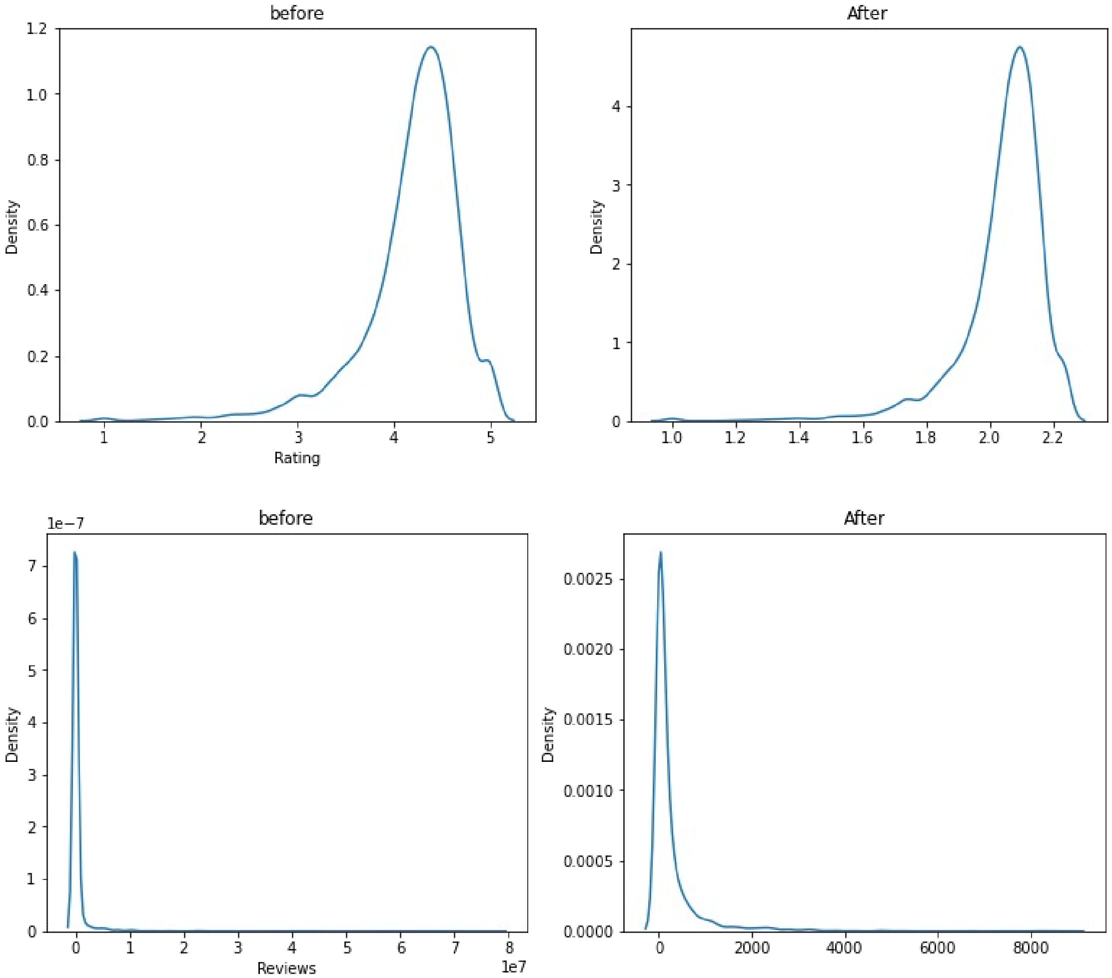
For Reviews , Size and Install features Log tranformation has reduced skewness

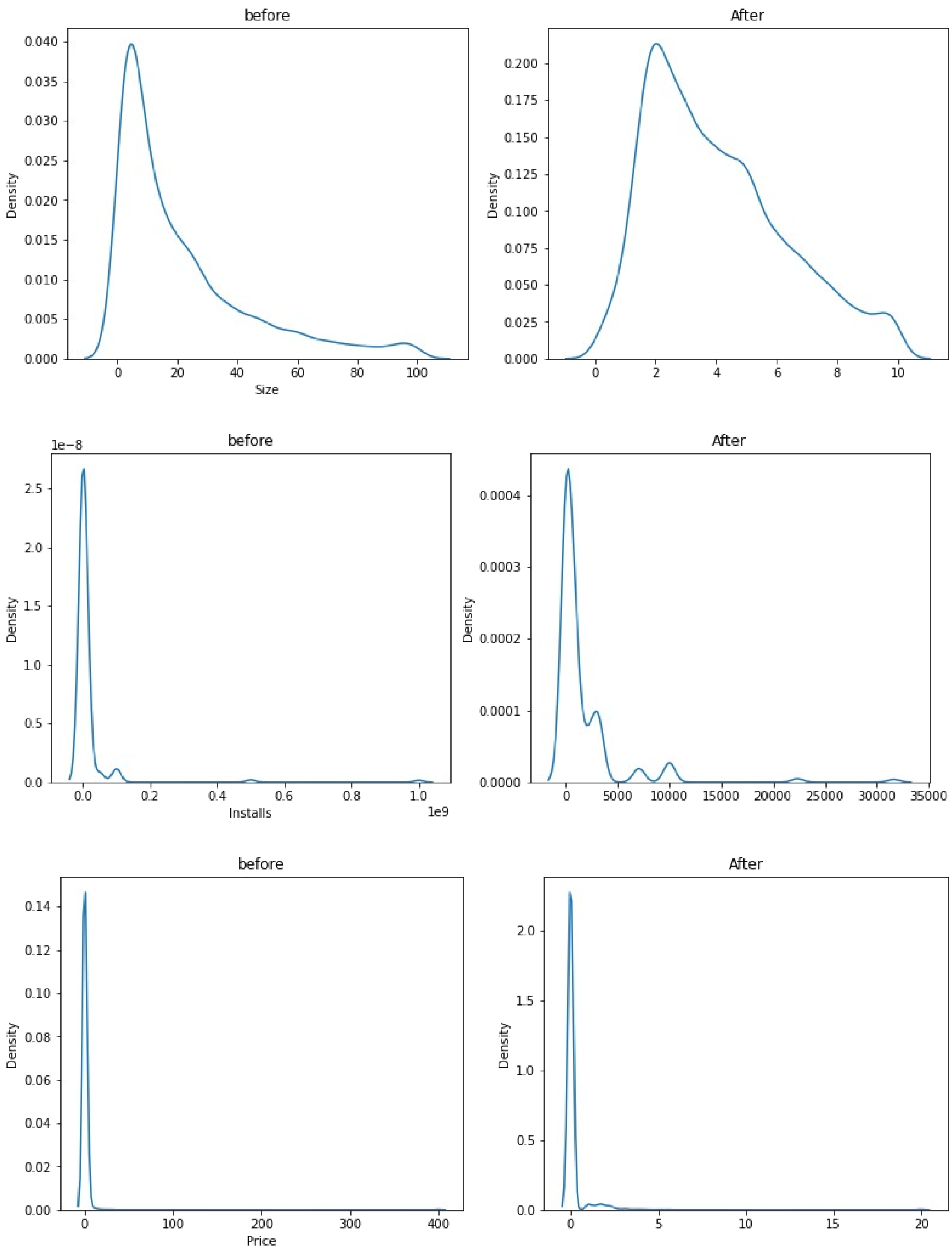
Square-Root Transformation

Here the x will replace by the square root(x). It is weaker than the Log Transformation.

The main advantage of square root transformation is, it can be applied to zero values.

```
log_transformer = FunctionTransformer(np.sqrt)
for col in res_list:
    X = np.array(data[col])
    Y = log_transformer.transform(X)
    plots(data,col,Y)
```



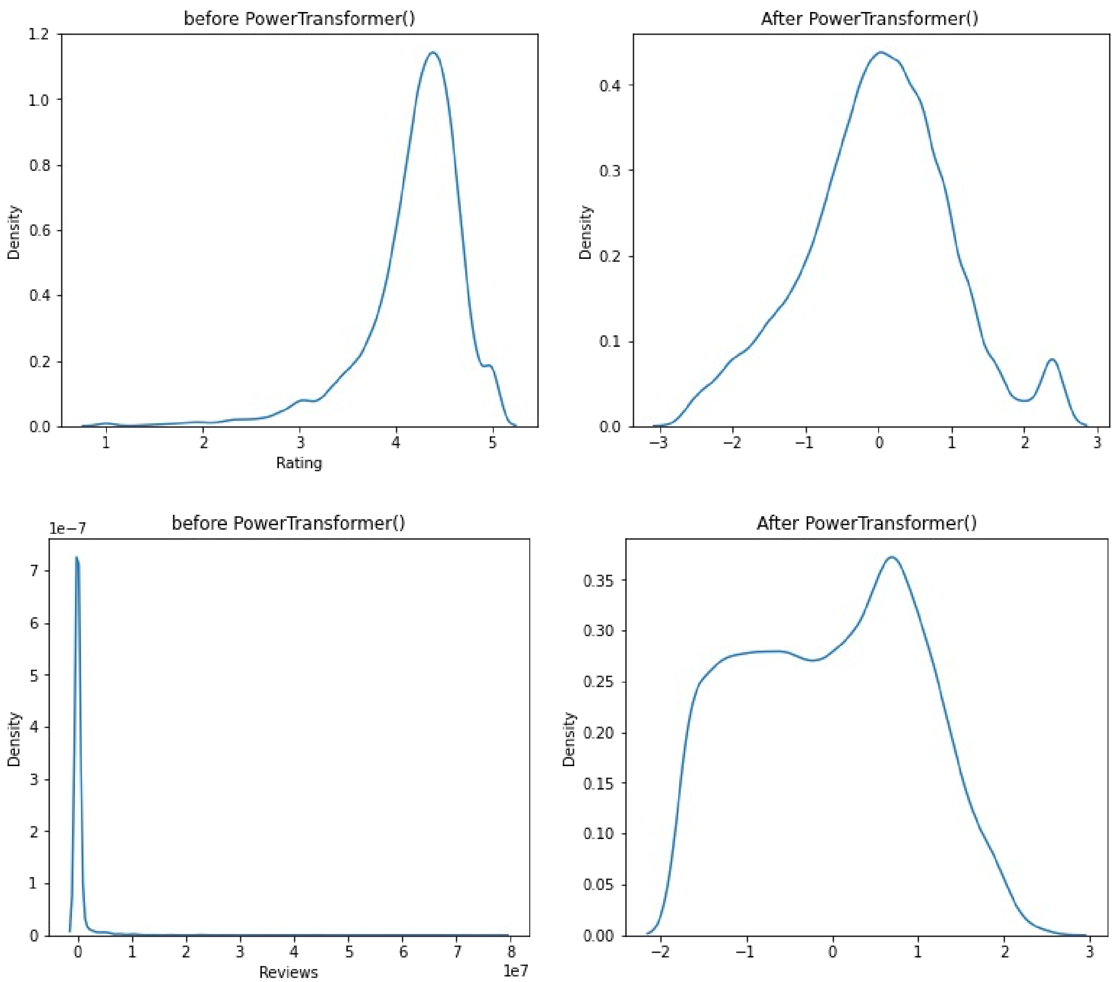


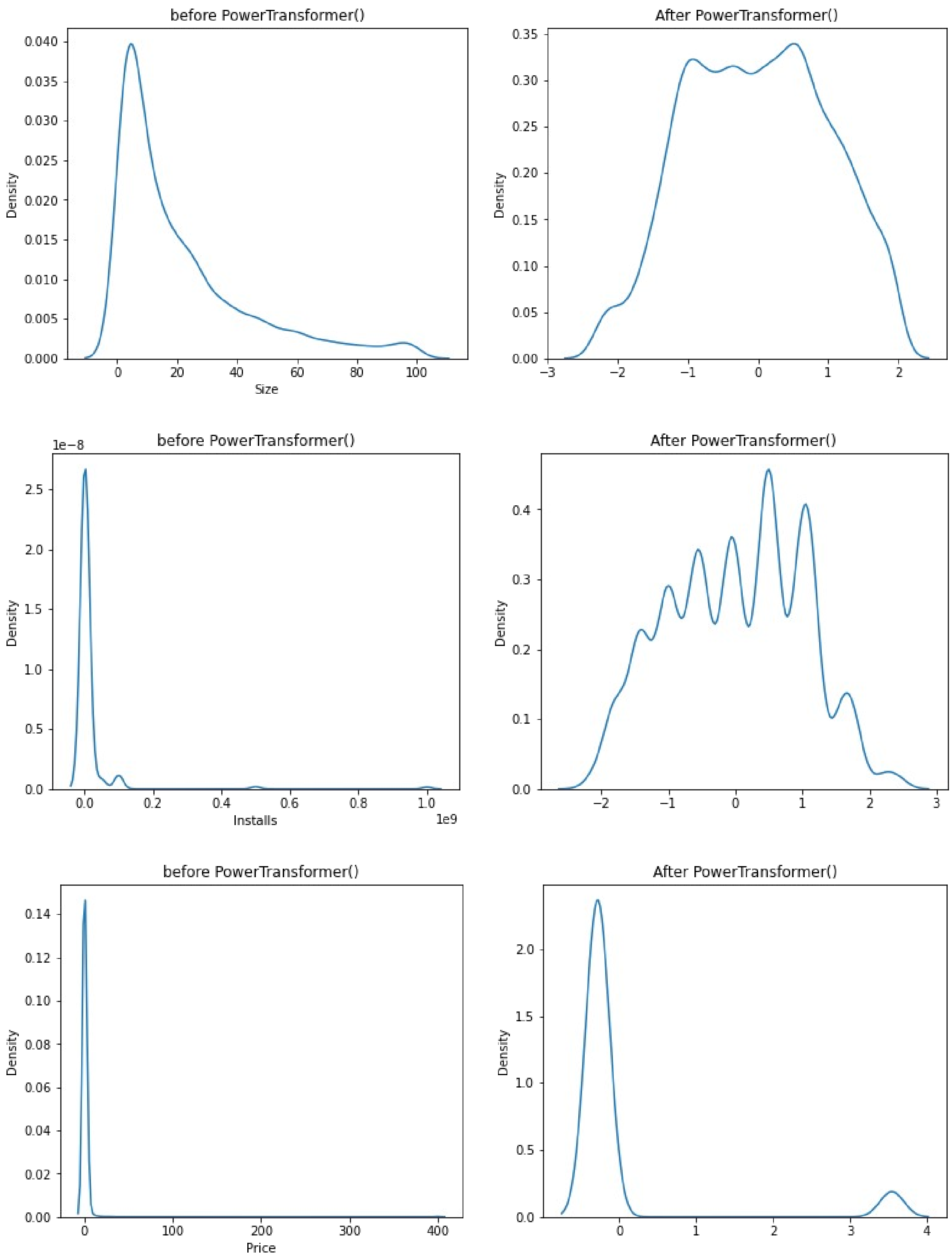
Insights

For our data Square root transformation is not working properly. Only sight change in Size distribution can be observed

Yeo-Johnson Transformation¶

```
def power_plots(data,var,t):
    plt.figure(figsize=(13,5))
    plt.subplot(121)
    sns.kdeplot(data[var])
    plt.title('before ' + str(t))
    plt.subplot(122)
    p1 = t.fit_transform(data[[var]]).flatten()
    sns.kdeplot(p1)
    plt.title('After ' + str(t))
for col in res_list:
    power_plots(data,col,PowerTransformer())
```





Insights

For our data Power transformation is working properly.

OUTLIERS DETECTION & REMOVAL approaches

Identifying outliers with visualization

Z-score method

Interquartile Range Method(IQR) method

Compare Skewness

```
data.head()

                                                 App        Category
Rating  \
0     Photo Editor & Candy Camera & Grid & ScrapBook  ART_AND_DESIGN
4.1
1                                Coloring book moana  ART_AND_DESIGN
3.9
2  U Launcher Lite – FREE Live Cool Themes, Hide ...  ART_AND_DESIGN
4.7
3                              Sketch - Draw & Paint  ART_AND_DESIGN
4.5
4              Pixel Draw - Number Art Coloring Book  ART_AND_DESIGN
4.3

   Reviews  Size  Installs  Type  Price Content Rating  \
0      159  19.0     10000  Free    0.0       Everyone
1      967  14.0    500000  Free    0.0       Everyone
2    87510   8.7   5000000  Free    0.0       Everyone
3   215644  25.0  50000000  Free    0.0           Teen
4      967   2.8    100000  Free    0.0       Everyone

                      Genres Last Updated         Current Ver
Android Ver  \
0               Art & Design   2018-01-07               1.0.0  4.0.3
and up
1  Art & Design;Pretend Play   2018-01-15               2.0.0  4.0.3
and up
2               Art & Design   2018-08-01               1.2.4  4.0.3
and up
3               Art & Design   2018-06-08  Varies with device    4.2
and up
4    Art & Design;Creativity   2018-06-20                 1.1    4.4
and up

   Day  Month  Year
0    7      1  2018
1   15      1  2018
2    1      8  2018
3    8      6  2018
4   20      6  2018
```

```
data.shape
```

```
(10840, 16)
```

```
num_features = [col for col in data.columns if data[col].dtype != 'O']
num_data = data[num_features]
num_data.head()
```

	Rating	Reviews	Size	Installs	Price	Day	Month	Year
0	4.1	159	19.0	10000	0.0	7	1	2018
1	3.9	967	14.0	500000	0.0	15	1	2018
2	4.7	87510	8.7	5000000	0.0	1	8	2018
3	4.5	215644	25.0	50000000	0.0	8	6	2018
4	4.3	967	2.8	100000	0.0	20	6	2018

Z-score method

Z-score:

The number of standard deviations away from the mean that a particular observation is.

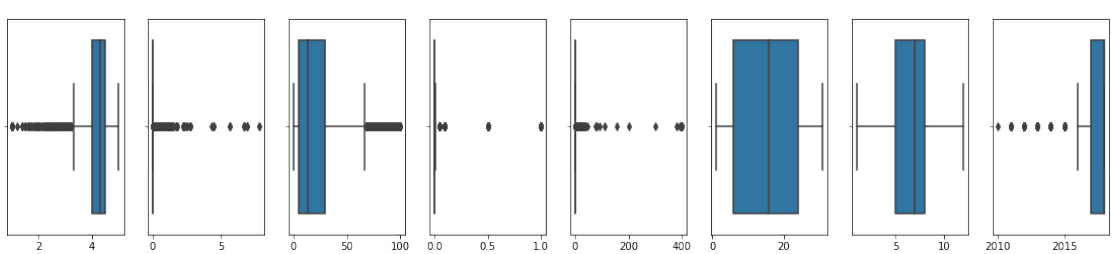
A negative Z-score means an observation is below the mean.

while a positive Z-score means means it above the mean.

The further away from 0 the Z-Score is, the further away from the mean your observation is.

```
# Function to detect outliers
def outlier_thresholds(dataframe, variable):
    quartile1 = dataframe[variable].quantile(0.10)
    quartile3 = dataframe[variable].quantile(0.90)
    interquantile_range = quartile3 - quartile1
    up_limit = quartile3 + 1.5 * interquantile_range
    low_limit = quartile1 - 1.5 * interquantile_range
    return low_limit, up_limit
```

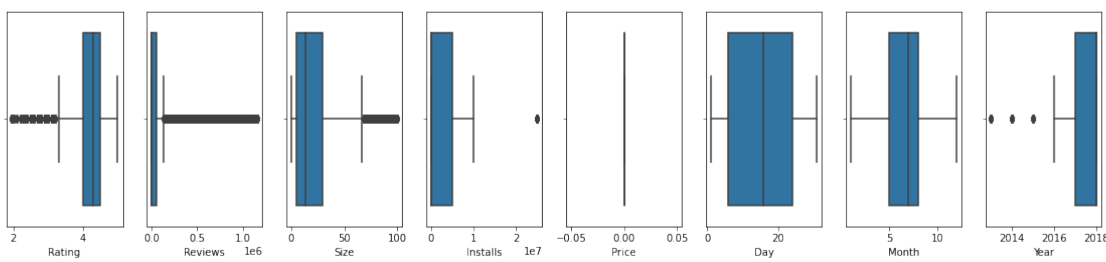
```
plt.figure(figsize=(22,18))
for i,col in enumerate(num_data.columns):
    plt.subplot(4,9,i+1)
    sns.boxplot(num_data[col])
```



Insights

Except Day and month feature we have outliers in all other features

```
## function to remove outliers
def replace_with_thresholds(dataframe, numeric_columns):
    for variable in numeric_columns:
        low_limit, up_limit = outlier_thresholds(dataframe, variable)
        dataframe.loc[(dataframe[variable] < low_limit), variable] = 
low_limit
        dataframe.loc[(dataframe[variable] > up_limit), variable] = 
up_limit

replace_with_thresholds(num_data, num_data.columns)

plt.figure(figsize=(22,18))
for i,col in enumerate(num_data.columns):
    plt.subplot(4,9,i+1)
    sns.boxplot(num_data[col])
```

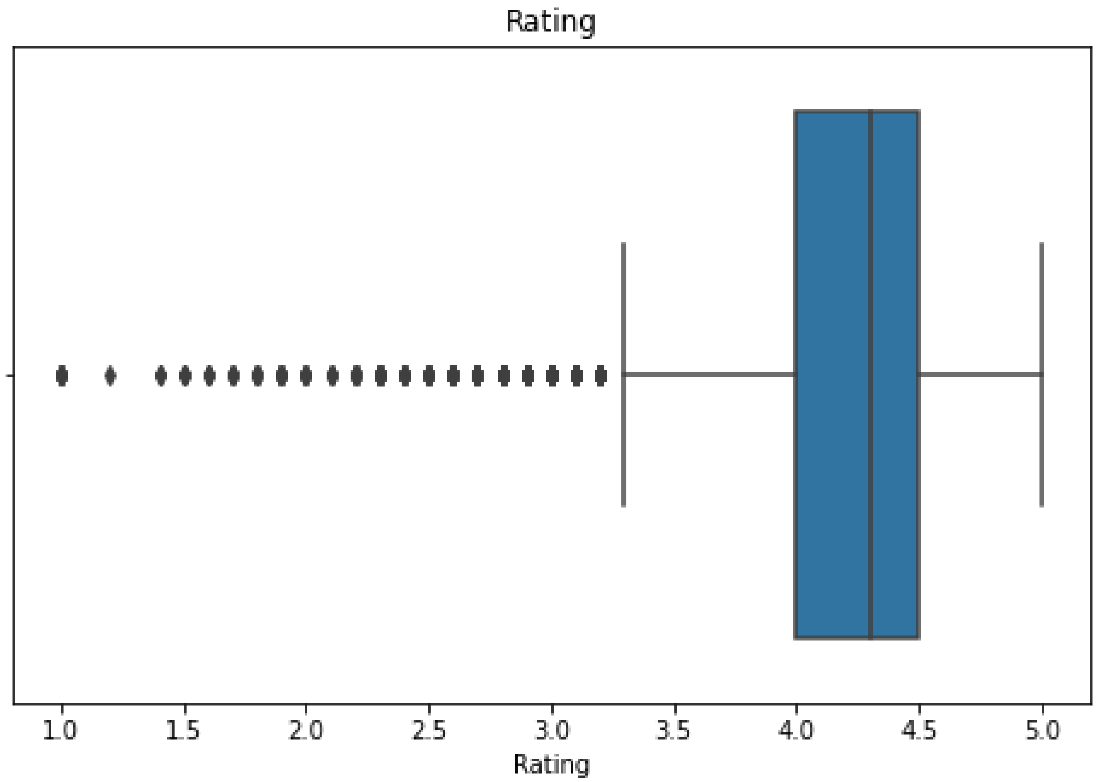


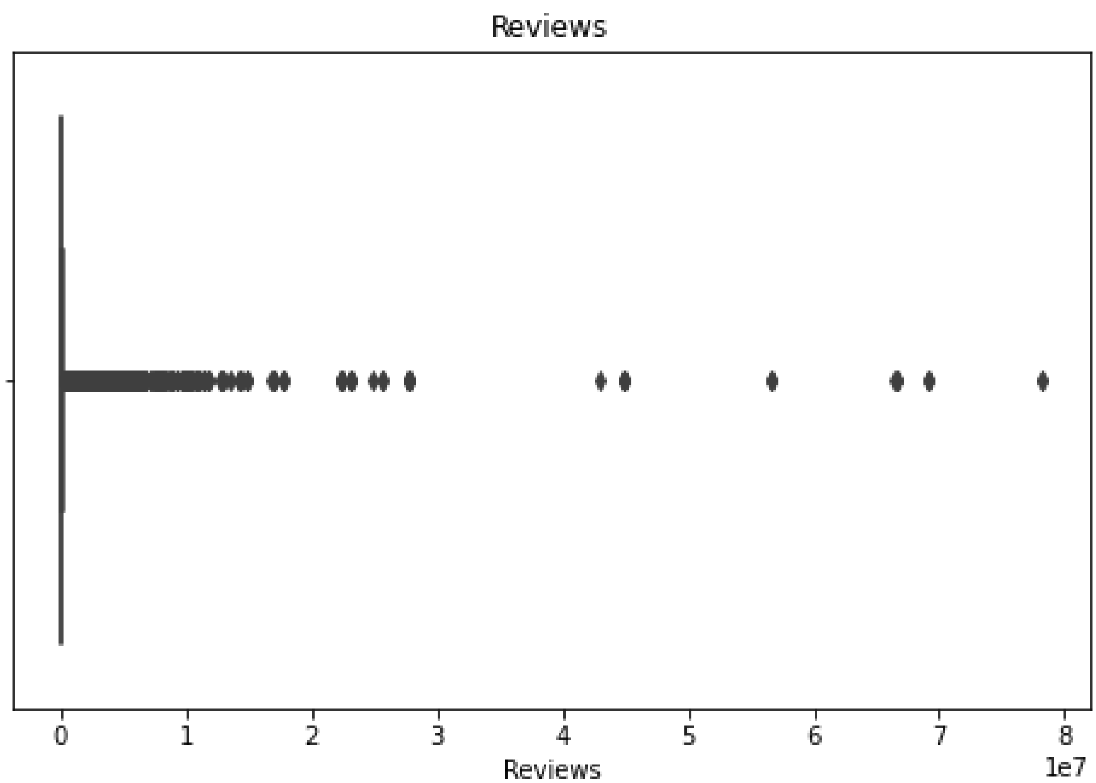
Insights

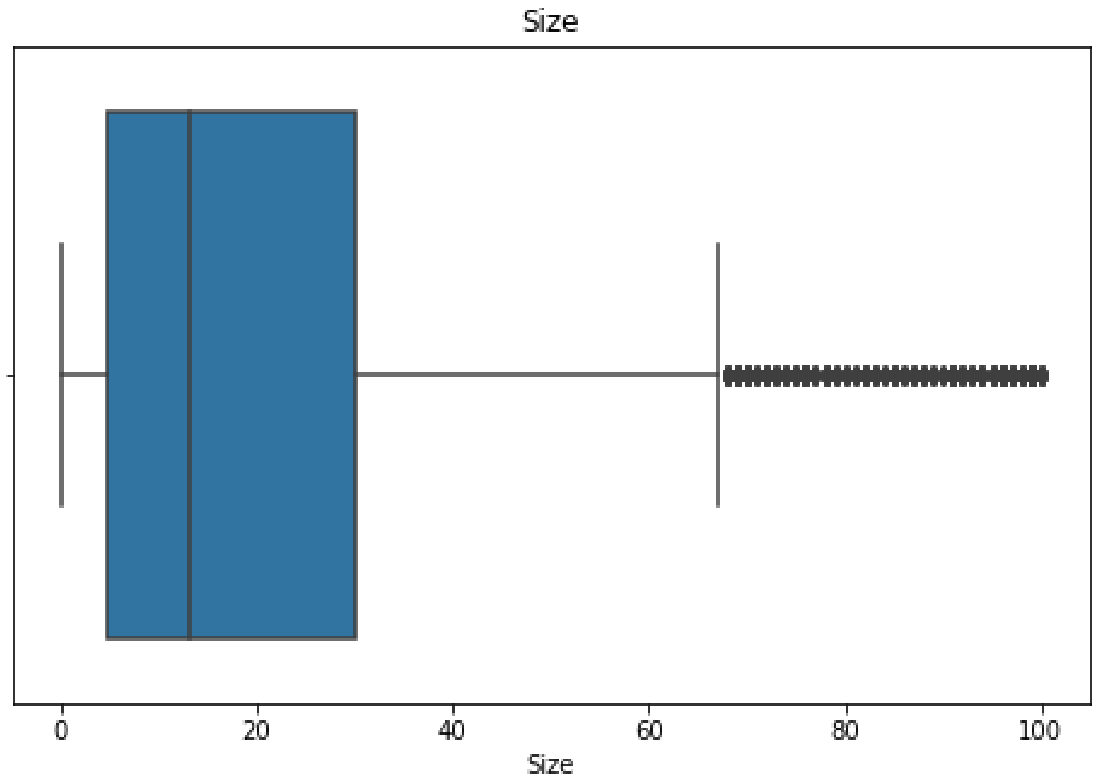
As we can see from above boxplots outliers are not removed properly

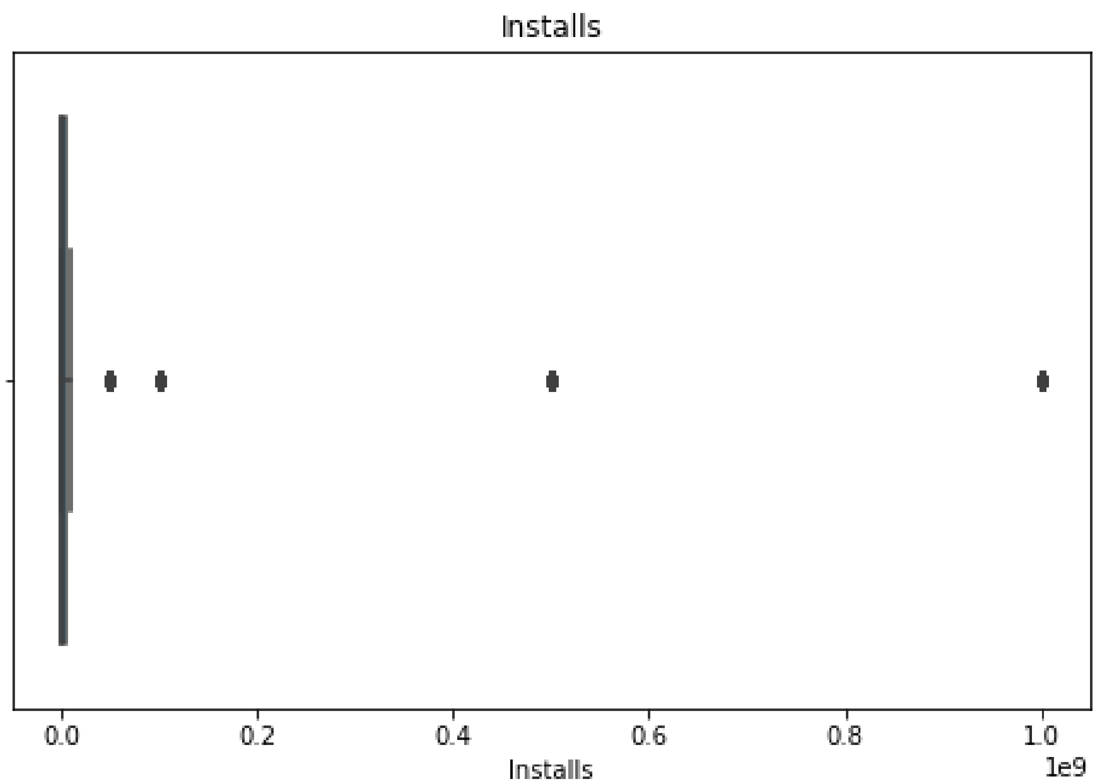
Interquartile Range Method(IQR) method

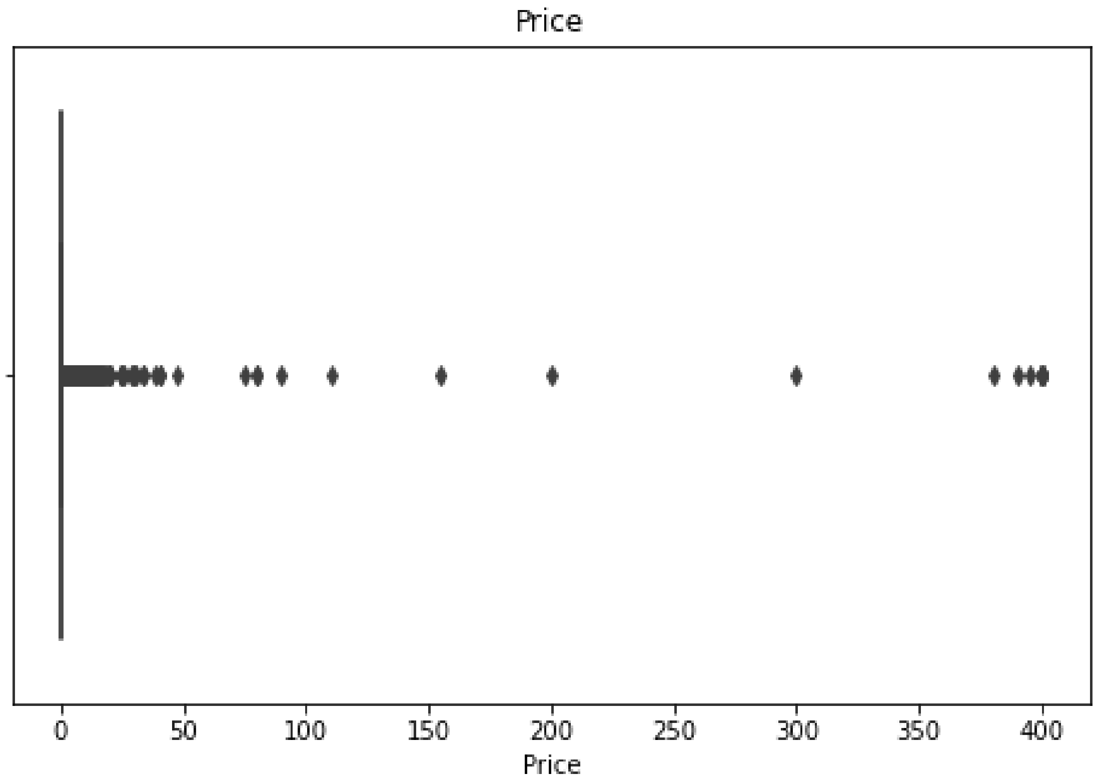
```
for col in num_data.columns:
    plt.figure(figsize=(8,5))
    sns.boxplot(data[col])
    plt.title(col)
```

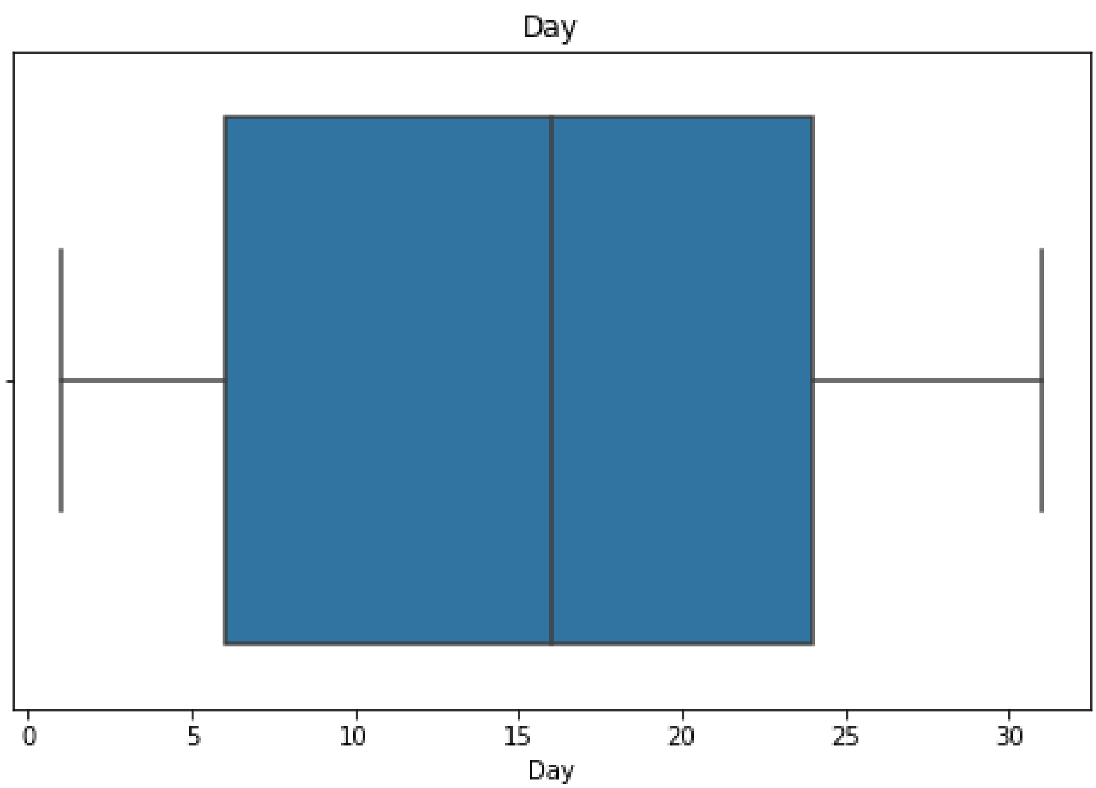

Rating
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Rating

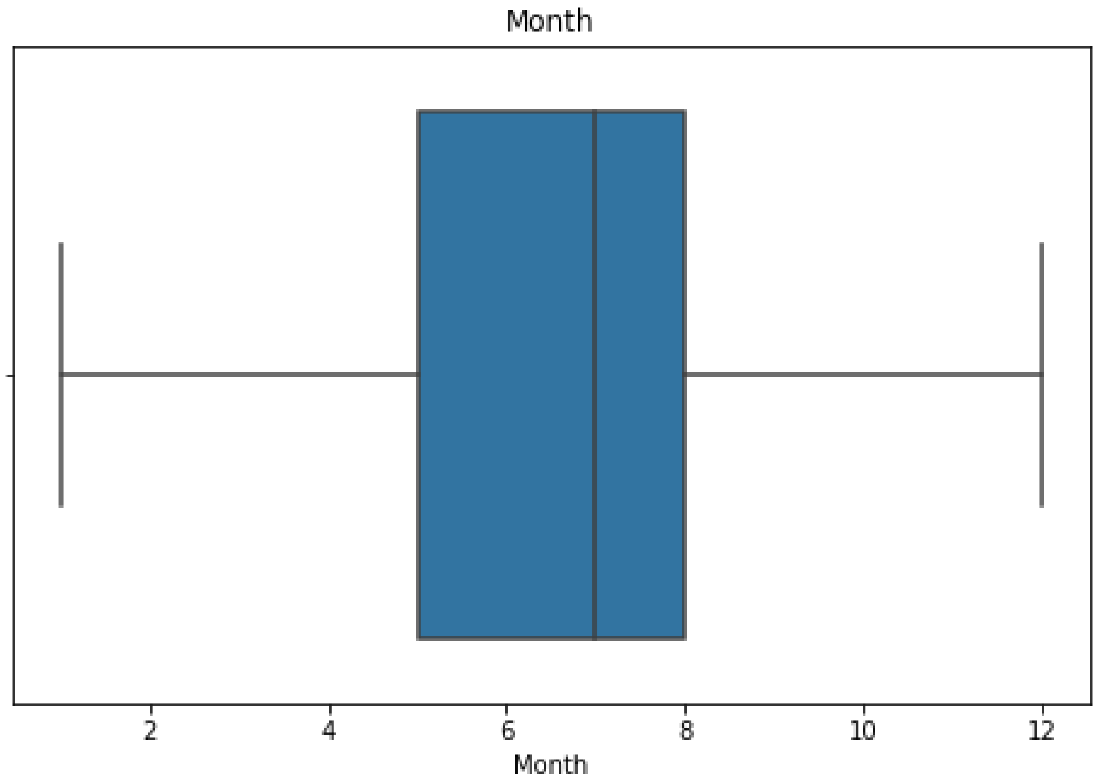

Reviews
0
1
2
3
4
5
6
7
8
1e7
Reviews

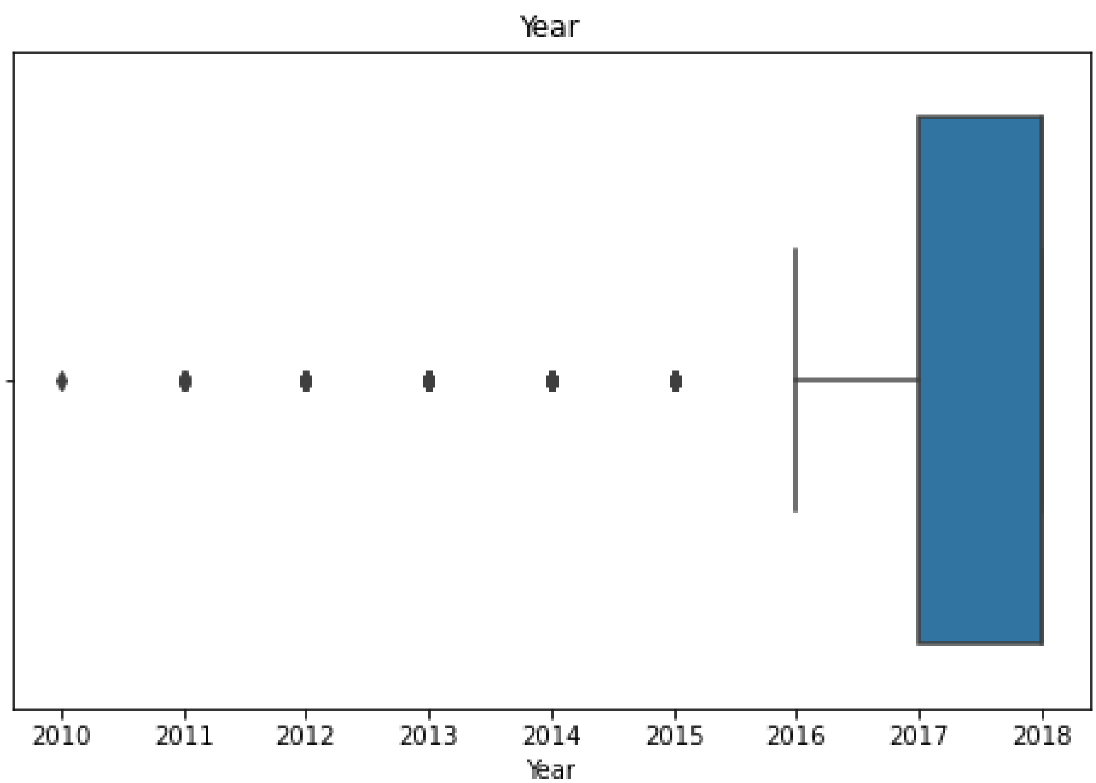

Size
0
20
40
60
80
100
Size


Installs
0.0
0.2
0.4
0.6
0.8
1.0
Installs
1e9


Price
0
50
100
150
200
250
300
350
400
Price


Day
0
5
10
15
20
25
30
Day






```
df1 = data.copy()
def remove_outliers_IQR(col):
```

```
    # Finding the IQR
    percentile25 = df1[col].quantile(0.25)
    percentile75 = df1[col].quantile(0.75)
    print("percentile25",percentile25)
    print("percentile75",percentile75)
    iqr = percentile75 - percentile25
    upper_limit = percentile75 + 1.5 * iqr
    lower_limit = percentile25 - 1.5 * iqr
    print("Upper limit",upper_limit)
    print("Lower limit",lower_limit)
    df1[col] = np.where(df1[col]>upper_limit, upper_limit,
np.where(df1[col]<lower_limit,lower_limit,df1[col]))
    return df1[df1[col] > upper_limit]
```

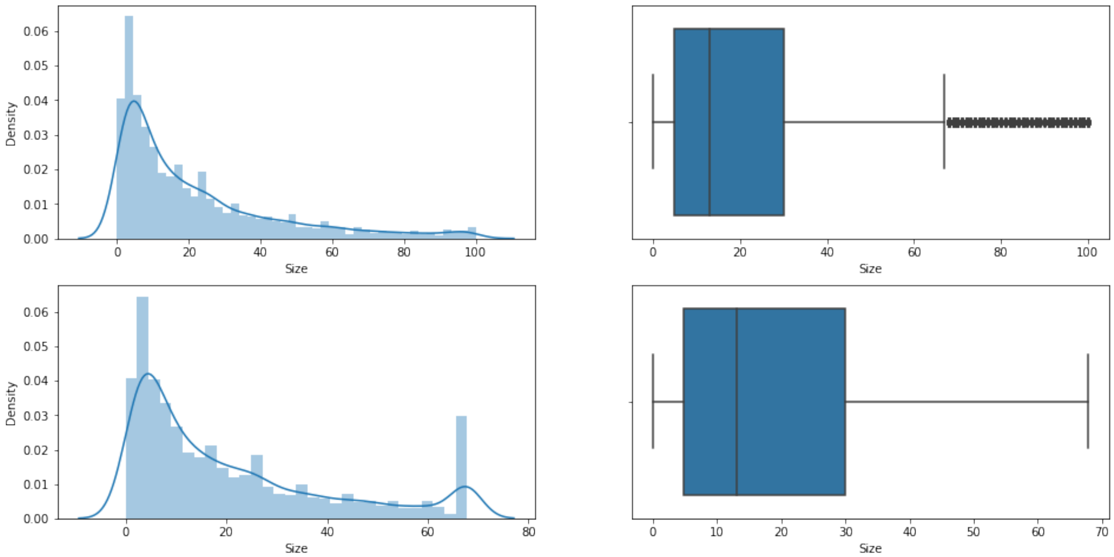
```
remove_outliers_IQR('Size')
```

```
percentile25 4.8
percentile75 30.0
Upper limit 67.8
Lower limit -33.0
```

```
Empty DataFrame
Columns: [App, Category, Rating, Reviews, Size, Installs, Type, Price,
Content Rating, Genres, Last Updated, Current Ver, Android Ver, Day,
Month, Year]
Index: []
```

```
def create_comparison_plot(data,df1,column):
    # Comparing
    plt.figure(figsize=(16,8))
    plt.subplot(2,2,1)
    sns.distplot(data[column])

    plt.subplot(2,2,2)
    sns.boxplot(data[column])

    plt.subplot(2,2,3)
    sns.distplot(df1[column])

    plt.subplot(2,2,4)
    sns.boxplot(df1[column])

    plt.show()
```

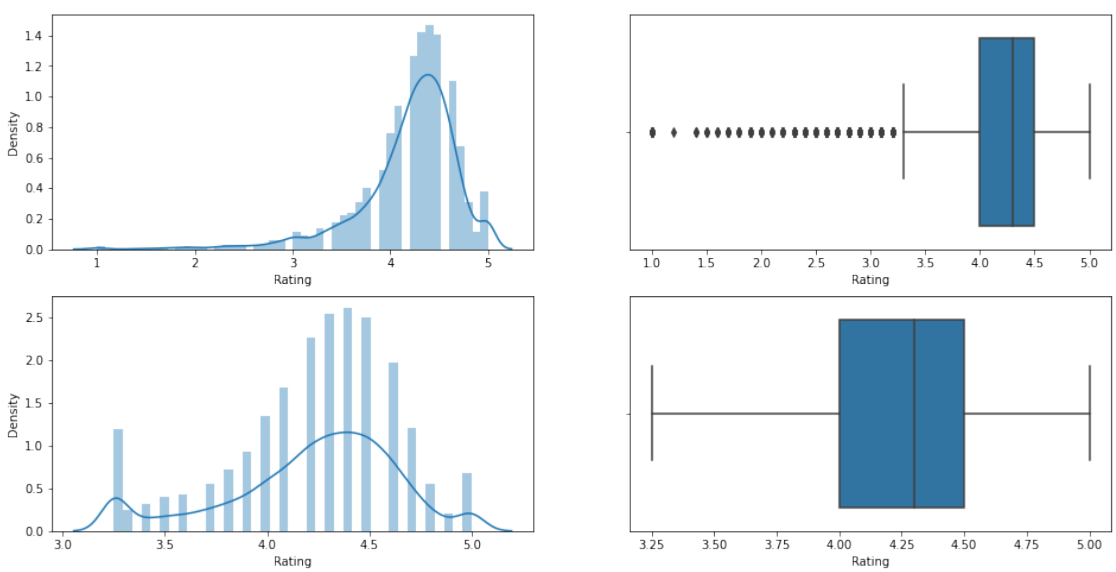
```
create_comparison_plot(data,df1,"Size")
```




```
remove_outliers_IQR('Rating')
```

```
percentile25 4.0
percentile75 4.5
Upper limit 5.25
Lower limit 3.25

Empty DataFrame
Columns: [App, Category, Rating, Reviews, Size, Installs, Type, Price,
Content Rating, Genres, Last Updated, Current Ver, Android Ver, Day,
Month, Year]
Index: []
```

```
create_comparison_plot(data,df1,"Rating")
```

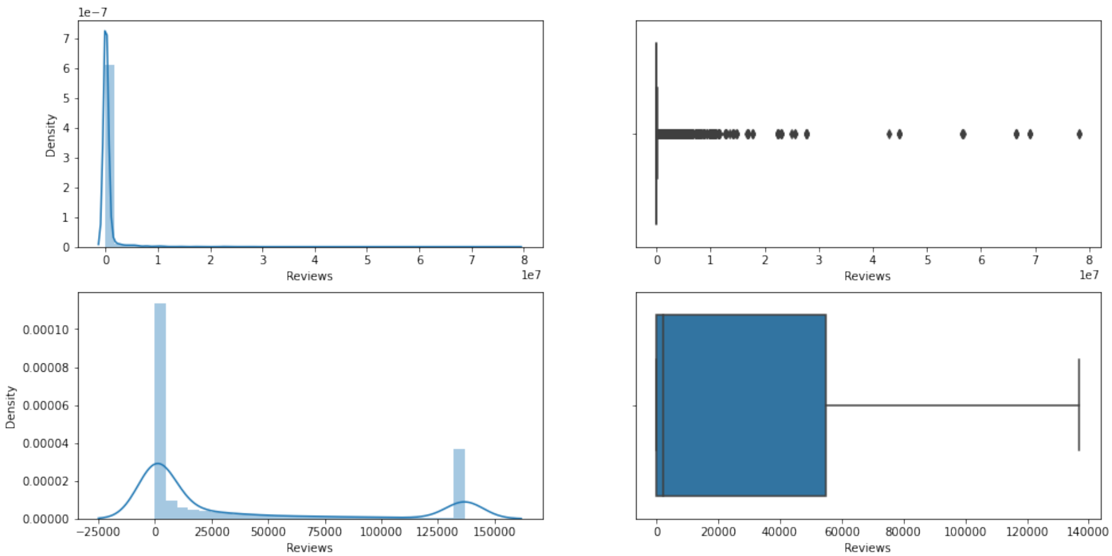



```
remove_outliers_IQR('Reviews')
```

percentile25 38.0
percentile75 54775.5
Upper limit 136881.75
Lower limit -82068.25

Empty DataFrame
Columns: [App, Category, Rating, Reviews, Size, Installs, Type, Price, Content Rating, Genres, Last Updated, Current Ver, Android Ver, Day, Month, Year]
Index: []

```
create_comparison_plot(data,df1,"Reviews")
```

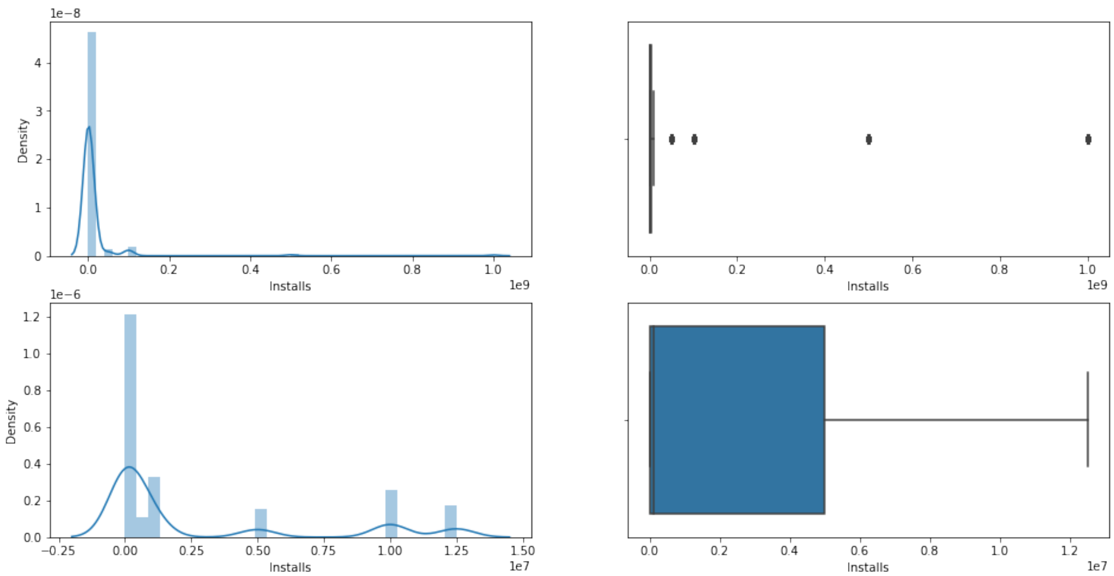



```
remove_outliers_IQR('Installs')
```

percentile25 1000.0
percentile75 5000000.0
Upper limit 12498500.0
Lower limit -7497500.0

Empty DataFrame
Columns: [App, Category, Rating, Reviews, Size, Installs, Type, Price, Content Rating, Genres, Last Updated, Current Ver, Android Ver, Day, Month, Year]
Index: []

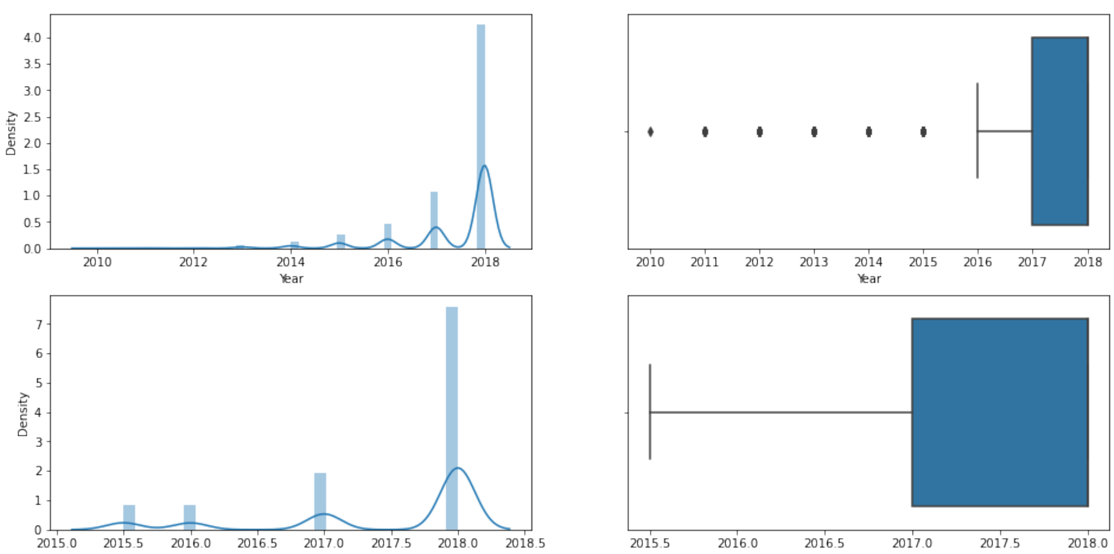
```
create_comparison_plot(data,df1,"Installs")
```




```
remove_outliers_IQR('Year')
```

```
percentile25 2017.0
percentile75 2018.0
Upper limit 2019.5
Lower limit 2015.5

Empty DataFrame
Columns: [App, Category, Rating, Reviews, Size, Installs, Type, Price,
Content Rating, Genres, Last Updated, Current Ver, Android Ver, Day,
Month, Year]
Index: []
```

```
create_comparison_plot(data,df1,"Year")
```



Compare Skewness

```
data.skew()
```

```
Rating      -1.831695
Reviews     16.449584
Size         1.556132
Installs     9.572067
Price       23.707392
Day         -0.002569
Month       -0.114442
Year        -2.288293
dtype: float64
```

```
df1.skew()
```

```
Rating      -0.674792
Reviews      1.197882
Size         1.138549
Installs     1.384312
Price       23.707392
Day         -0.002569
Month       -0.114442
Year        -1.371946
dtype: float64
```

Insights

Skewness is reduced after we have removed ouliers using IQR Method

Model Training

Importing Required Packages

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
from sklearn.preprocessing import PowerTransformer
from sklearn.preprocessing import FunctionTransformer
warnings.filterwarnings("ignore")

from sklearn.preprocessing import OneHotEncoder
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.compose import make_column_transformer
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.preprocessing import OrdinalEncoder
from sklearn.model_selection import train_test_split
from sklearn import metrics


#core import for hyperparamter tuning
from sklearn.model_selection import RandomizedSearchCV
%matplotlib inline
```

```
df1.head()
```

```
                                                 App        Category
Rating  \
0     Photo Editor & Candy Camera & Grid & ScrapBook  ART_AND_DESIGN
4.1
1                                Coloring book moana  ART_AND_DESIGN
3.9
2  U Launcher Lite – FREE Live Cool Themes, Hide ...  ART_AND_DESIGN
4.7
3                              Sketch - Draw & Paint  ART_AND_DESIGN
4.5
4              Pixel Draw - Number Art Coloring Book  ART_AND_DESIGN
4.3

     Reviews  Size    Installs  Type  Price Content Rating  \
0     159.00  19.0     10000.0  Free    0.0       Everyone
1     967.00  14.0    500000.0  Free    0.0       Everyone
2   87510.00   8.7   5000000.0  Free    0.0       Everyone
3  136881.75  25.0  12498500.0  Free    0.0           Teen
4     967.00   2.8    100000.0  Free    0.0       Everyone

                      Genres Last Updated         Current Ver
Android Ver  \
0               Art & Design   2018-01-07               1.0.0  4.0.3
and up
1  Art & Design;Pretend Play   2018-01-15               2.0.0  4.0.3
and up
2               Art & Design   2018-08-01               1.2.4  4.0.3
and up
3               Art & Design   2018-06-08  Varies with device    4.2
and up
4    Art & Design;Creativity   2018-06-20                 1.1    4.4
and up

   Day  Month    Year
0    7      1  2018.0
1   15      1  2018.0
2    1      8  2018.0
3    8      6  2018.0
4   20      6  2018.0
```

Dropping columns that do not contribute numerically to the Regression Model

```
df1.drop(columns=['Current Ver','Android Ver','App','Last
Updated'],inplace=True)
```

Encoding categorical values

```
df1=pd.get_dummies(df1,columns=['Type','Content
Rating'],drop_first=True)
```

Splitting our mathematical feature columns and assigning it to 'X'¶

```
X=df1.drop(columns=['Category','Rating','Genres'],axis=1)
```

Splitting our target variable 'Rating' and assigning it to 'y'

```
y=df1['Rating']
```

Train Test Split

```
from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test=train_test_split(
X,y,test_size=0.33,random_state=10)

X_train.shape,y_train.shape

((7262, 13), (7262,))

X_test.shape,y_test.shape

((3578, 13), (3578,))
```

Independent Train Dataset

```
X_train

        Reviews    Size    Installs  Price  Day  Month    Year
Type_Paid  \
9365       4.00   3.900       100.0    0.0   24      4  2018.0
0
1772   70226.00  38.000   5000000.0    0.0   25      7  2018.0
0
3790   49259.00   6.300   5000000.0    0.0    2      8  2018.0
0
4688  136881.75  40.000  12498500.0    0.0    4      8  2018.0
0
1130   16808.00  22.000   1000000.0    0.0    1      5  2018.0
0
...         ...     ...         ...    ...  ...    ...     ...
...
9372   13388.00   7.100    500000.0    0.0   24      8  2015.5
0
7291       1.00   2.300       100.0    0.0   24      1  2017.0
0
1344   77563.00  39.000  10000000.0    0.0   23      7  2018.0
0
```

```
7293       6.00   1.100      1000.0    0.0    2      8  2018.0
0
1289  136881.75   0.421  10000000.0    0.0   11      7  2018.0
0

      Content Rating_Everyone  Content Rating_Everyone 10+  \
9365                        1                            0
1772                        1                            0
3790                        0                            1
4688                        0                            0
1130                        1                            0
...                       ...                          ...
9372                        1                            0
7291                        1                            0
1344                        1                            0
7293                        1                            0
1289                        1                            0

      Content Rating_Mature 17+  Content Rating_Teen  Content
Rating_Unrated
9365                          0                    0
0
1772                          0                    0
0
3790                          0                    0
0
4688                          0                    1
0
1130                          0                    0
0
...                         ...                  ...
...
9372                          0                    0
0
7291                          0                    0
0
1344                          0                    0
0
7293                          0                    0
0
1289                          0                    0
0

[7262 rows x 13 columns]
```

Independent Test Dataset

```
X_test

         Reviews   Size    Installs  Price  Day  Month    Year
Type_Paid  \
```

```
212       6903.00  14.0   1000000.0   0.00   10      7  2018.0
0
6547     33661.00  60.0    100000.0   2.99    3      8  2018.0
1
2378        63.00  25.0     10000.0   9.99    3     10  2016.0
1
5744       249.00   3.0     50000.0   0.00   17      3  2018.0
0
3793     78154.00  12.0   1000000.0   0.00    3      8  2018.0
0
...           ...   ...         ...    ...  ...    ...     ...         .
..
10226     1060.00   3.2    100000.0   0.00   20     12  2017.0
0
3684   136881.75  16.0  10000000.0   0.00   28      6  2018.0
0
7124       916.00  12.0    100000.0   0.00   13      7  2018.0
0
7679         6.00   7.0      1000.0   0.00   30      7  2016.0
0
3631     11118.00   9.7   1000000.0   0.00   26      4  2018.0
0

       Content Rating_Everyone  Content Rating_Everyone 10+  \
212                          1                            0
6547                         1                            0
2378                         1                            0
5744                         1                            0
3793                         0                            1
...                        ...                          ...
10226                        1                            0
3684                         0                            0
7124                         0                            0
7679                         1                            0
3631                         1                            0

       Content Rating_Mature 17+  Content Rating_Teen  Content
Rating_Unrated
212                            0                    0
0
6547                           0                    0
0
2378                           0                    0
0
5744                           0                    0
0
3793                           0                    0
0
...                          ...                  ...
...
```

```
10226                          0                    0
0
3684                           0                    1
0
7124                           1                    0
0
7679                           0                    0
0
3631                           0                    0
0

[3578 rows x 13 columns]
```

Dependent Train Dataset

```
y_train
```

```
9365    5.0
1772    4.2
3790    4.1
4688    4.6
1130    4.5
       ...
9372    3.8
7291    5.0
1344    4.6
7293    4.1
1289    4.5
Name: Rating, Length: 7262, dtype: float64
```

Dependent Test Dataset

```
y_test
```

```
212      4.1
6547     4.4
2378     4.5
5744     4.2
3793     4.2
        ...
10226    4.5
3684     4.3
7124     3.7
7679     4.6
3631     4.7
Name: Rating, Length: 3578, dtype: float64
```

Standardizing or Feature Scaling

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()  ## Initialising
```

```
scaler
```

```
StandardScaler()
```

```
X_train=scaler.fit_transform(X_train)
```

```
X_test=scaler.transform(X_test)
```

```
X_train
```

```
array([[-0.659545  , -0.83521377, -0.61732116, ..., -0.2171454 ,
        -0.35431972, -0.01659765],
       [ 0.66410973,  0.89327213,  0.54716349, ..., -0.2171454 ,
        -0.35431972, -0.01659765],
       [ 0.26889073, -0.71356081,  0.54716349, ..., -0.2171454 ,
        -0.35431972, -0.01659765],
       ...,
       [ 0.80240905,  0.94396086,  1.71167144, ..., -0.2171454 ,
        -0.35431972, -0.01659765],
       [-0.6595073 , -0.97714223, -0.61711155, ..., -0.2171454 ,
        -0.35431972, -0.01659765],
       [ 1.92054217, -1.01155989,  1.71167144, ..., -0.2171454 ,
        -0.35431972, -0.01659765]])
```

```
X_test
```

```
array([[-0.5295018 , -0.32325754, -0.38444286, ..., -0.2171454 ,
        -0.35431972, -0.01659765],
       [-0.02512492,  2.00842432, -0.59405429, ..., -0.2171454 ,
        -0.35431972, -0.01659765],
       [-0.65843288,  0.23431856, -0.61501544, ..., -0.2171454 ,
        -0.35431972, -0.01659765],
       ...,
       [-0.64235419, -0.42463501, -0.59405429, ...,  4.60520912,
        -0.35431972, -0.01659765],
       [-0.6595073 , -0.67807869, -0.61711155, ..., -0.2171454 ,
        -0.35431972, -0.01659765],
       [-0.45005085, -0.5412191 , -0.38444286, ..., -0.2171454 ,
        -0.35431972, -0.01659765]])
```

Linear Regression Model implementation

```
from sklearn.linear_model import LinearRegression
```

```
regression=LinearRegression()


regression.fit(X_train,y_train)
```

```
LinearRegression()
```

Coefficient

```
print(regression.coef_)
```

```
[ 0.15967828 -0.0026349  -0.08973467 -0.02113514 -0.00238562
0.01186021
  0.04949104  0.03884574 -0.01094163 -0.00859019 -0.02671545 -
0.01660199
  0.00386846]
```

Intercept

```
print(regression.intercept_)
```

```
4.219533186450011
```

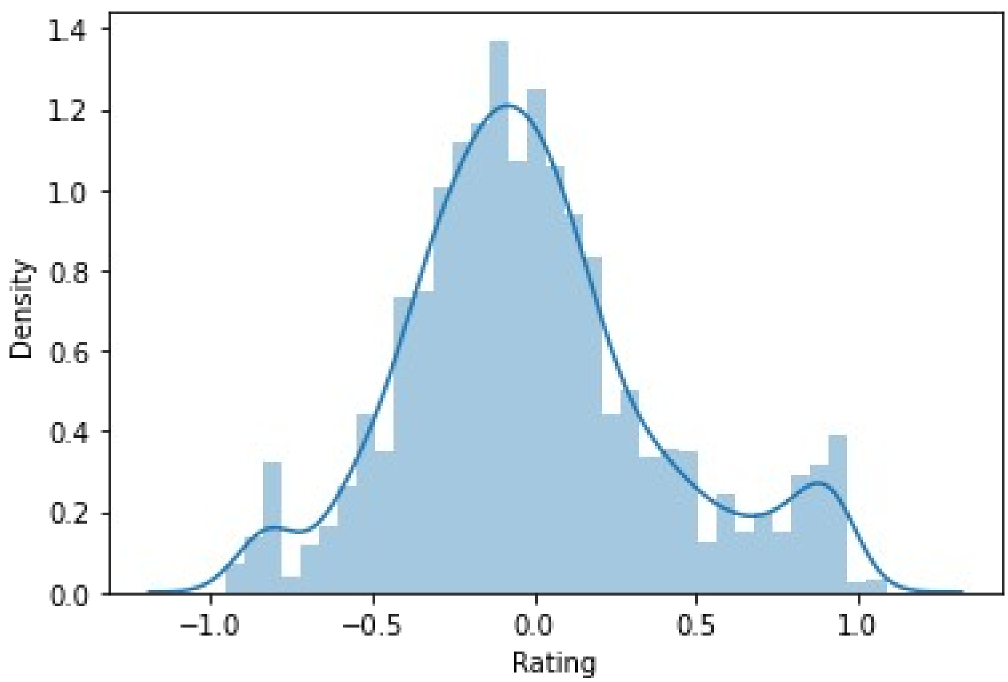
Prediction

```
reg_pred=regression.predict(X_test)
reg_pred
```

```
array([4.20331757, 4.44773901, 4.23380747, ..., 4.10243904,
4.08007892,
       4.19880965])
```

```
import seaborn as sns
sns.distplot(reg_pred-y_test)
```

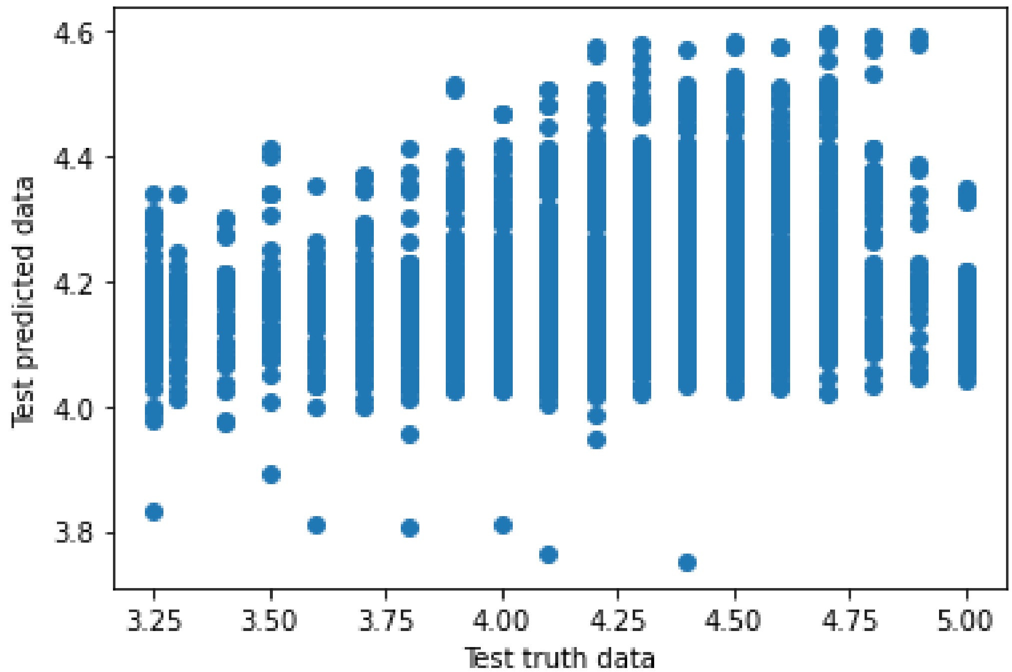
```
<AxesSubplot:xlabel='Rating', ylabel='Density'>
```



Assumption of Linear Regression

```
plt.scatter(y_test,reg_pred)
plt.xlabel("Test truth data")
plt.ylabel('Test predicted data')
```

```
Text(0, 0.5, 'Test predicted data')
```



Residuals

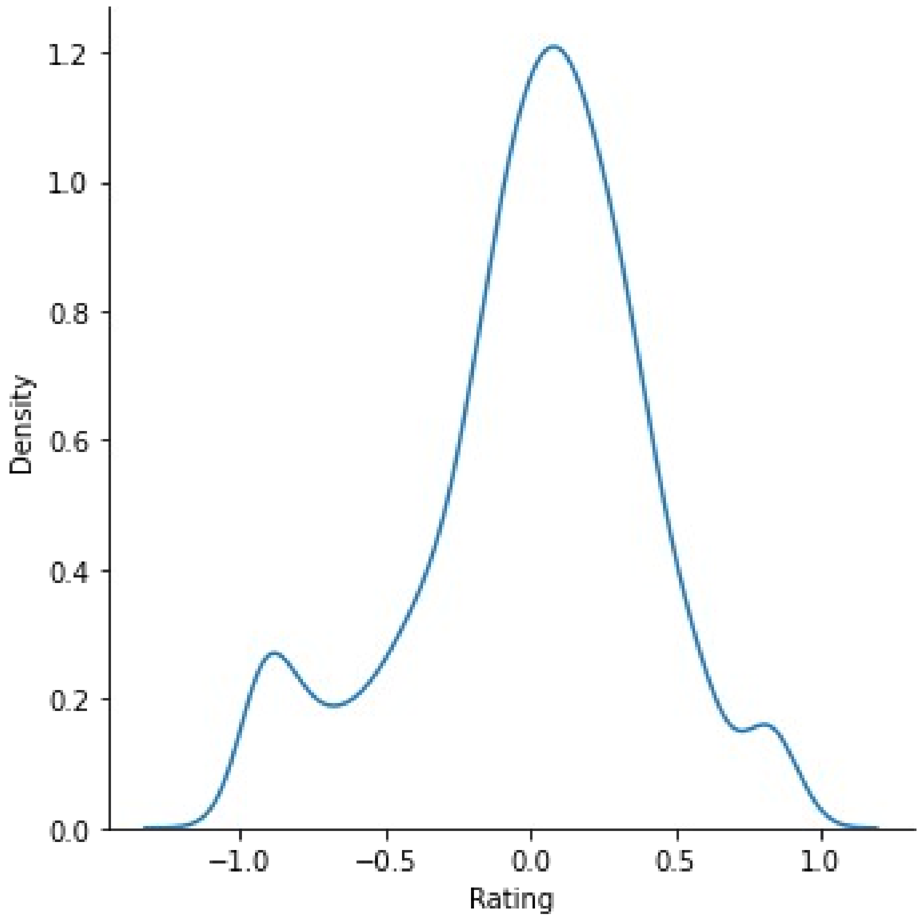
```
residual=y_test-reg_pred
```

```
residual
```

```
212      -0.103318
6547     -0.047739
2378      0.266193
5744      0.015494
3793     -0.206625
            ...
10226     0.333218
3684     -0.071993
7124     -0.402439
7679      0.519921
3631      0.501190
Name: Rating, Length: 3578, dtype: float64
```

```
sns.displot(residual,kind='kde')
```

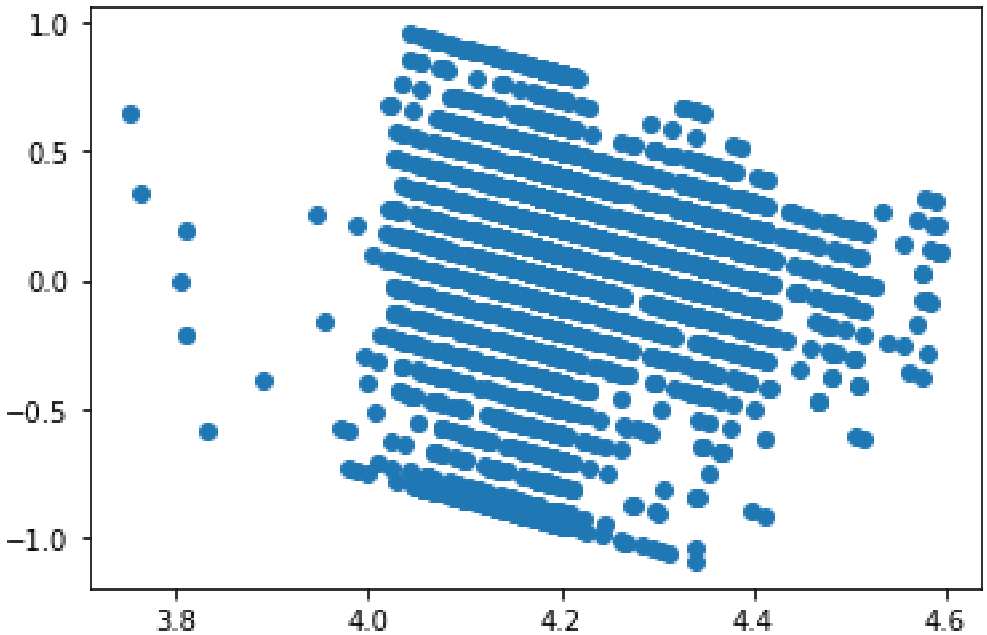
```
<seaborn.axisgrid.FacetGrid at 0x206f66528b0>
```



Scatterplot with Prediction and Residual

```
plt.scatter(reg_pred,residual)
```

```
<matplotlib.collections.PathCollection at 0x206f6be8d00>
```



Performance Metrics

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
print(mean_squared_error(y_test,reg_pred))
print(mean_absolute_error(y_test,reg_pred))
print(np.sqrt(mean_squared_error(y_test,reg_pred)))
```

```
0.1596600883629503
0.3065417776053013
0.3995748845497553
```

Rsquare

```
from sklearn.metrics import r2_score
score=r2_score(y_test,reg_pred)
print(score)
```

```
0.07621120029538742
```

Adjusted Rsquare

```
1-(1-score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
```

```
0.0728416002964648
```

Ridge Regression Algorithm

```
from sklearn.linear_model import Ridge
```

```
ridge = Ridge()
```

```
ridge
```

```
Ridge()
```

```
ridge.fit(X_train,y_train)
```

```
Ridge()
```

```
## Coefficient

print(ridge.coef_)
```

```
[ 0.15947485 -0.00262608 -0.08954044 -0.02113098 -0.00238785
0.01186046
  0.04948439  0.03883846 -0.0089889  -0.00765137 -0.02568525 -
0.01504048
  0.0039495 ]
```

```
## Intercept

print(ridge.intercept_)
```

```
4.219533186450011
```

Prediction

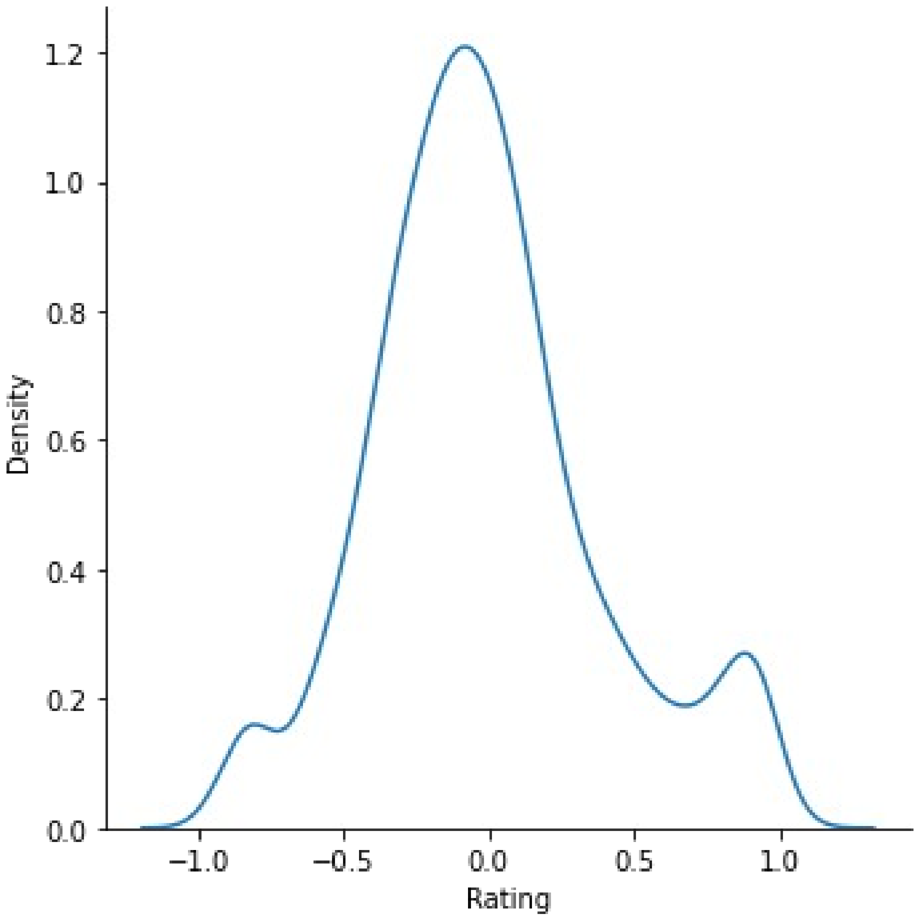
```
ridge_pred=ridge.predict(X_test)
```

```
ridge_pred
```

```
array([4.20334331, 4.44761661, 4.23381223, ..., 4.10247936,
4.08009416,
       4.19881328])
```

```
import seaborn as sns
sns.displot(ridge_pred-y_test,kind='kde')
```

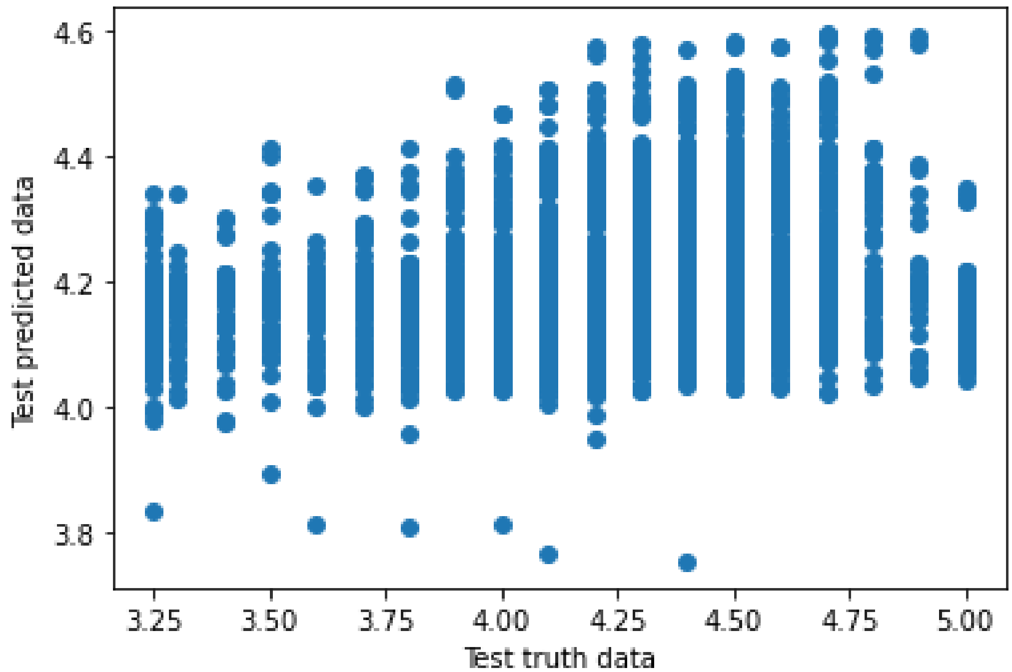
```
<seaborn.axisgrid.FacetGrid at 0x206f6c22850>
```



Assumption of Ridge Regression

```
plt.scatter(y_test,ridge_pred)
plt.xlabel("Test truth data")
plt.ylabel('Test predicted data')
```

```
Text(0, 0.5, 'Test predicted data')
```



Residual

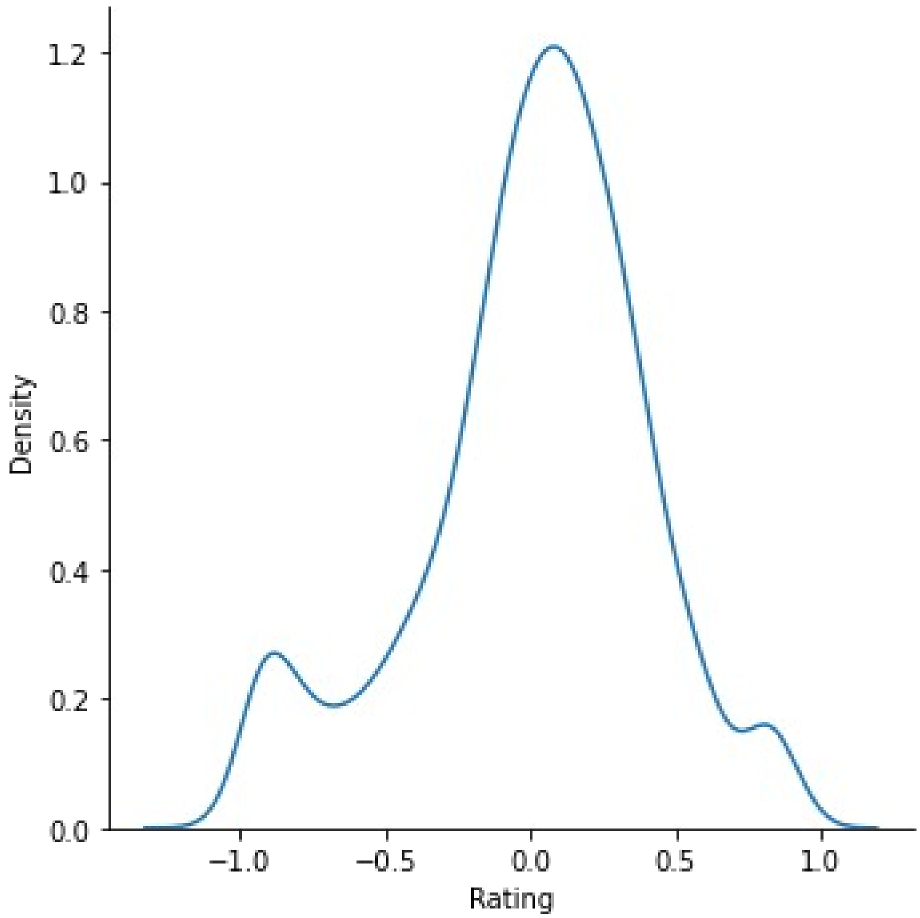
```
ridge_residual=y_test-ridge_pred
```

```
ridge_residual
```

```
212      -0.103343
6547     -0.047617
2378      0.266188
5744      0.015492
3793     -0.206406
            ...
10226     0.333209
3684     -0.071950
7124     -0.402479
7679      0.519906
3631      0.501187
Name: Rating, Length: 3578, dtype: float64
```

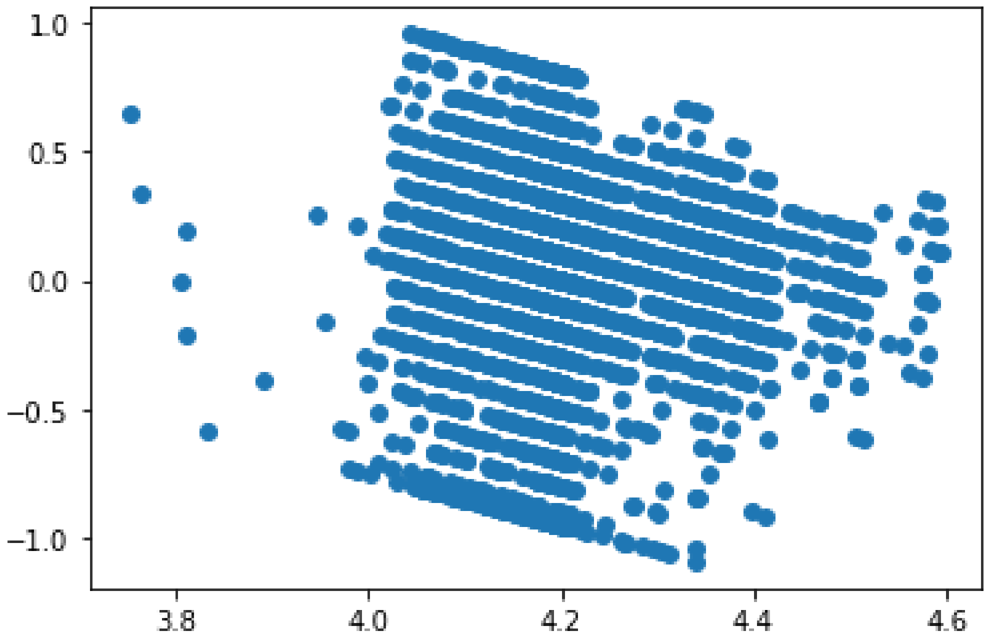
```
sns.displot(ridge_residual,kind='kde')
```

```
<seaborn.axisgrid.FacetGrid at 0x206f839eca0>
```



Scatterplot with residual and predicion

```
plt.scatter(ridge_pred,ridge_residual)

<matplotlib.collections.PathCollection at 0x206f8450760>
```



Performance Metrics

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
print(mean_squared_error(y_test,ridge_pred))
print(mean_absolute_error(y_test,ridge_pred))
print(np.sqrt(mean_squared_error(y_test,ridge_pred)))
```

```
0.1596622598617302
0.3065442234892713
0.3995776018018655
```

Rsquare

```
from sklearn.metrics import r2_score
ridge_score=r2_score(y_test,ridge_pred)
print(ridge_score)
```

```
0.07619863606426358
```

Adjusted Rsquare

```
1-(1-ridge_score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
```

```
0.07282899023621514
```

Lasso Regression

```
from sklearn.linear_model import Lasso
```

```
lasso=Lasso()
```

```
lasso
```

```
Lasso()
```

```
lasso.fit(X_train,y_train)
```

```
Lasso()
```

Coefficient and Intercept

```
print(lasso.coef_)
```

```
[ 0.  0.  0. -0. -0.  0.  0.  0. -0.  0. -0.  0.  0.]
```

```
print(lasso.intercept_)
```

```
4.219533186450014
```

Prediction

```
lasso_pred = lasso.predict(X_test)
```

```
lasso_pred
```

```
array([4.21953319, 4.21953319, 4.21953319, ..., 4.21953319,
4.21953319,
       4.21953319])
```

Performance Metrics

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
print(mean_squared_error(y_test,lasso_pred))
print(mean_absolute_error(y_test,lasso_pred))
print(np.sqrt(mean_squared_error(y_test,lasso_pred)))
```

```
0.172955293894834
0.3250473801848224
0.41587894139380754
```

Rsquare

```
from sklearn.metrics import r2_score
lasso_score=r2_score(y_test,lasso_pred)
print(lasso_score)
```

```
-0.0007144865563244451
```

Adjusted Rsquare

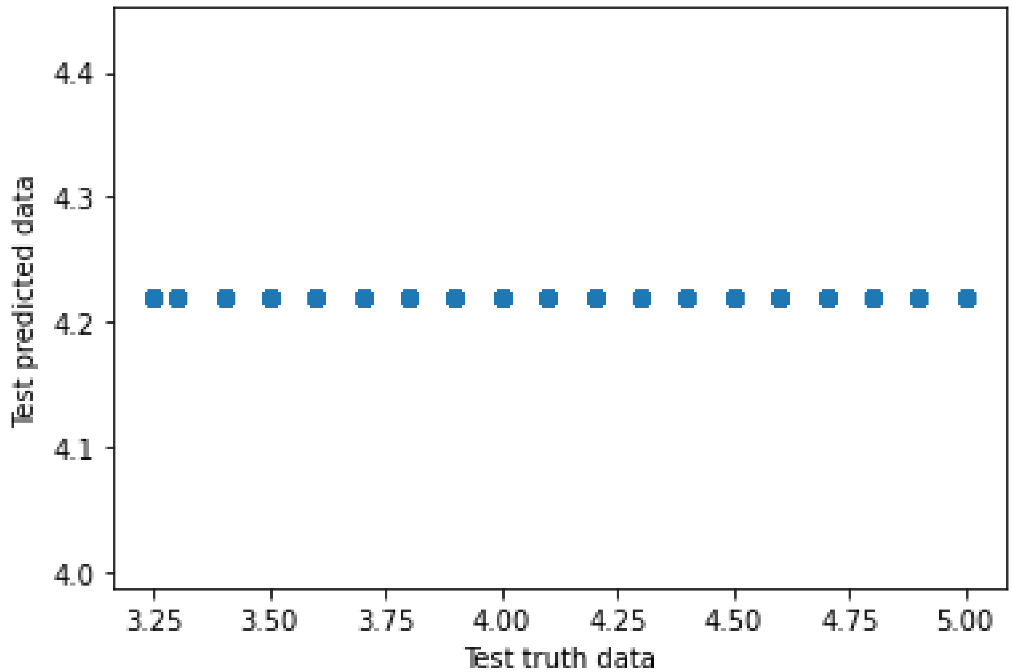
```
1-(1-lasso_score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
```

```
-0.004364679689105699
```

Assumption of Lasso Regression

```
plt.scatter(y_test,lasso_pred)
plt.xlabel("Test truth data")
plt.ylabel('Test predicted data')
```

```
Text(0, 0.5, 'Test predicted data')
```



Residual

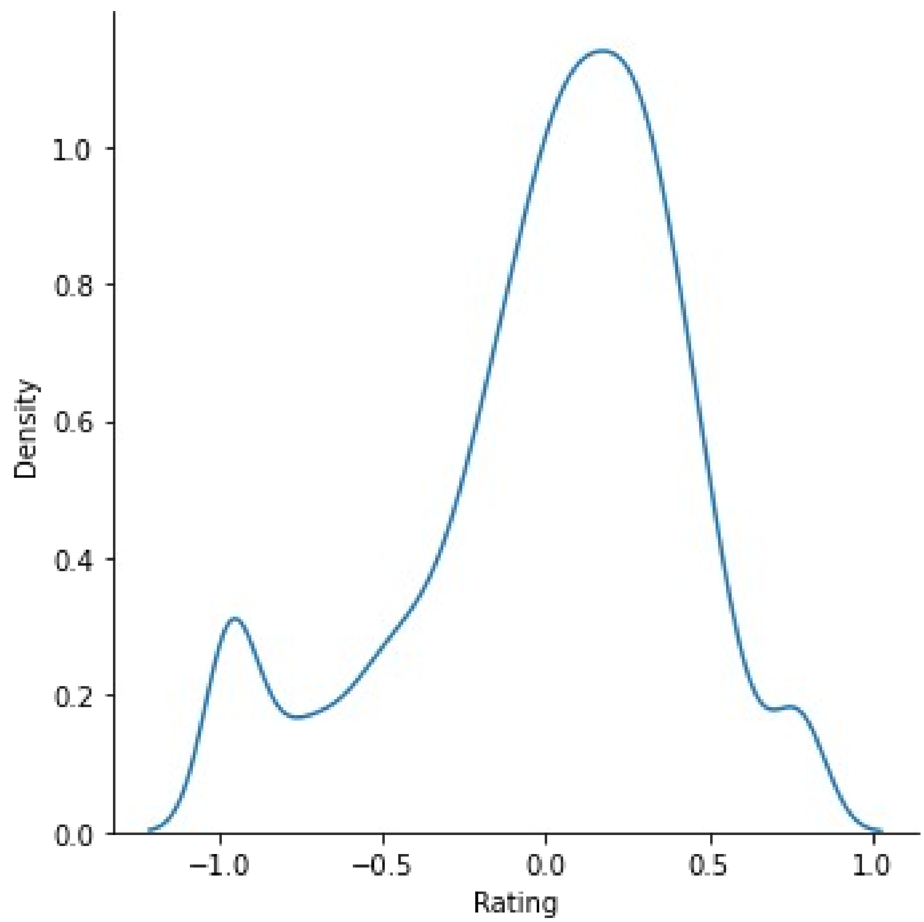
```
lasso_residual=y_test-lasso_pred
```

```
lasso_residual
```

```
212     -0.119533
6547     0.180467
2378     0.280467
5744    -0.019533
3793    -0.019533
           ...
10226    0.280467
3684     0.080467
7124    -0.519533
7679     0.380467
3631     0.480467
Name: Rating, Length: 3578, dtype: float64
```

```
sns.displot(lasso_residual,kind='kde')
```

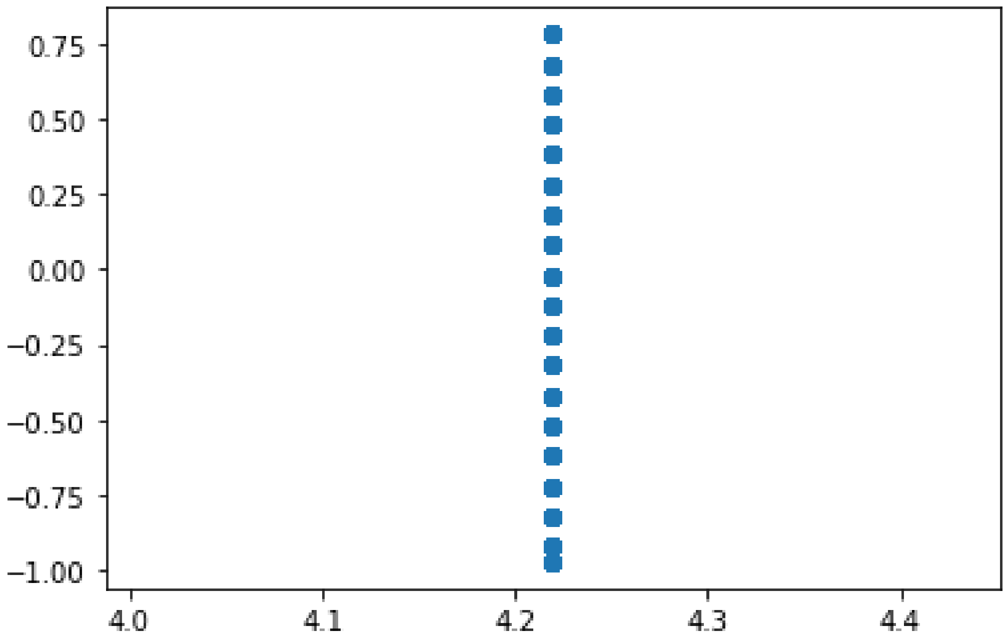
```
<seaborn.axisgrid.FacetGrid at 0x206f84e38b0>
```



Scatterplot with residual and prediction

```
plt.scatter(lasso_pred,lasso_residual)
```

```
<matplotlib.collections.PathCollection at 0x206f857afa0>
```



Performance Metrics

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
print(mean_squared_error(y_test,lasso_pred))
print(mean_absolute_error(y_test,lasso_pred))
print(np.sqrt(mean_squared_error(y_test,lasso_pred)))
```

```
0.172955293894834
0.3250473801848224
0.41587894139380754
```

Rsquare

```
from sklearn.metrics import r2_score
lasso_score=r2_score(y_test,lasso_pred)
print(lasso_score)
```

```
-0.0007144865563244451
```

Adjusted Rsquare

```
1-(1-lasso_score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
```

```
-0.004364679689105699
```

Elastic-Net Regression

```
from sklearn.linear_model import ElasticNet
```

```
elastic=ElasticNet()
```

```
elastic
```

```
ElasticNet()
```

```
elastic.fit(X_train,y_train)
```

```
ElasticNet()
```

Coefficient and Intercept

```
print(elastic.coef_)
```

```
[ 0.  0.  0. -0. -0.  0.  0.  0. -0.  0. -0.  0.  0.]
```

```
print(elastic.intercept_)
```

```
4.219533186450014
```

Prediction

```
elastic_pred = elastic.predict(X_test)
```

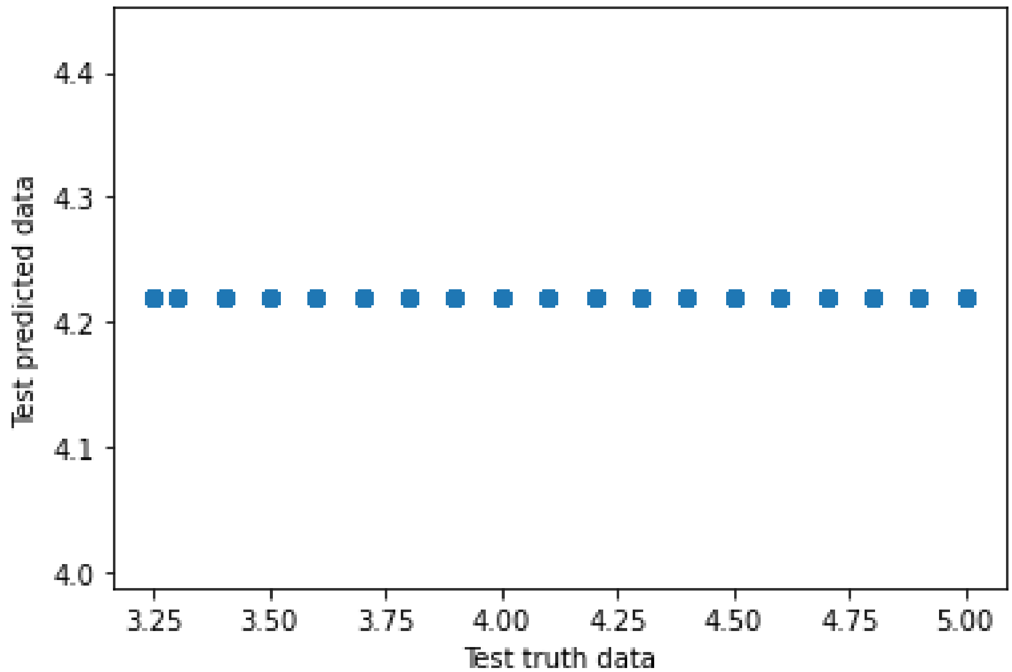
```
elastic_pred
```

```
array([4.21953319, 4.21953319, 4.21953319, ..., 4.21953319,
4.21953319,
       4.21953319])
```

Assumption of Elastic-Net Regression

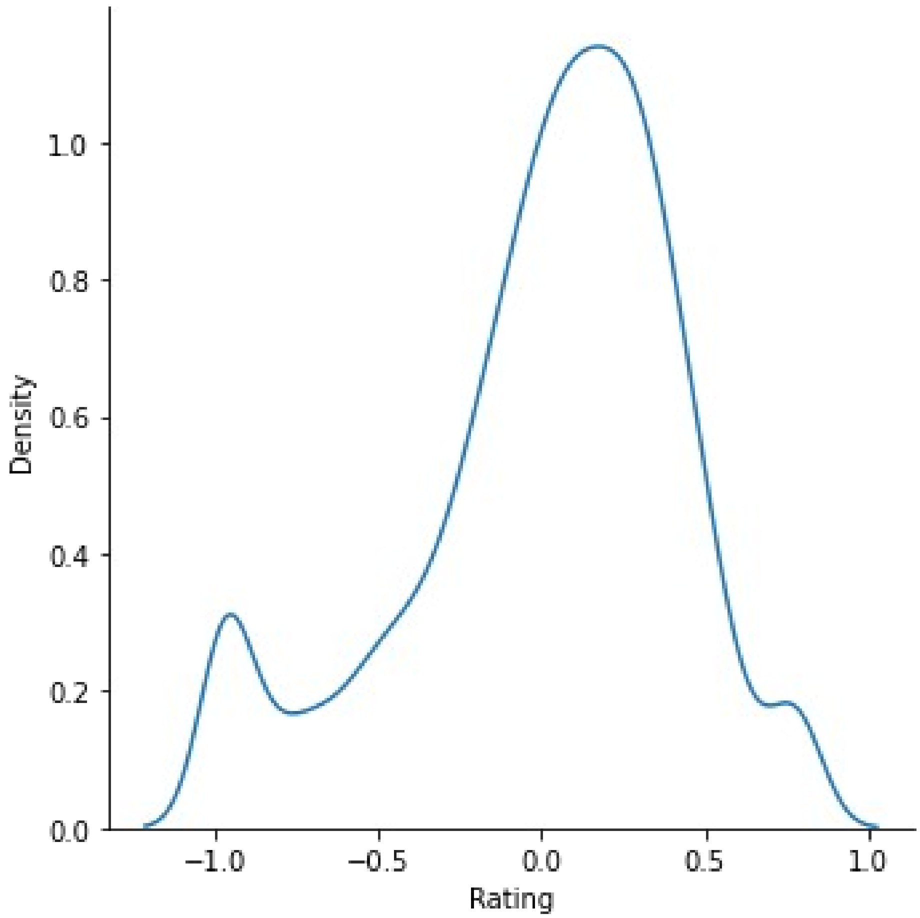
```
plt.scatter(y_test,elastic_pred)
plt.xlabel("Test truth data")
plt.ylabel('Test predicted data')
```

```
Text(0, 0.5, 'Test predicted data')
```



Residual

```
elastic_residual=y_test-elastic_pred

elastic_residual

212      -0.119533
6547      0.180467
2378      0.280467
5744     -0.019533
3793     -0.019533
            ...
10226     0.280467
3684      0.080467
7124     -0.519533
7679      0.380467
3631      0.480467
Name: Rating, Length: 3578, dtype: float64

sns.displot(elastic_residual,kind='kde')

<seaborn.axisgrid.FacetGrid at 0x206f8543910>
```



Performance Metrics

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
print(mean_squared_error(y_test,elastic_pred))
print(mean_absolute_error(y_test,elastic_pred))
print(np.sqrt(mean_squared_error(y_test,elastic_pred)))
```

```
0.172955293894834
0.3250473801848224
0.41587894139380754
```

Rsquare

```
from sklearn.metrics import r2_score
elastic_score=r2_score(y_test,elastic_pred)
print(elastic_score)
```

```
-0.0007144865563244451
```

Adjusted Rsquare

```
1-(1-elastic_score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
```

```
-0.004364679689105699
```