```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

[2]

```
apps_df=pd.read_csv('googleplaystore.csv')
print(apps_df.shape)
apps_df.head()
```

```
(10841, 13)
```

[3]

```
apps_df.tail()
```

[4]

```
#droping the na values from the dataframe:
apps_df=apps_df.dropna()
apps_df.shape
```

```
(9360, 13)
```

[5]

```
apps_df.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
dtype: int64
```

[6]

```
apps_df.dtypes
```

```
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
```

Data type correction

[7]

```
#transforming the data that are in Mb to Kb then removing the unit the unit and ch
anging the data type to float:
for elmnt in range(len(apps_df.loc[:,'Size'])):
    value=apps_df.iloc[elmnt,4]
    if value[-1]=='M':
        value=value.replace('M','')
        new_value=float(value)*1000
        apps_df.iloc[elmnt,4]=new_value
    elif value[-1]=='K':
        new_value=float(value.replace('K',''))
        apps_df.iloc[elmnt,4]=new_value
    else:
        apps_df.iloc[elmnt,4]=np.nan
apps_df.loc[:,'Size']=apps_df.loc[:,'Size'].astype('float')
apps_df=apps_df.dropna(axis=0)
[8]
#changing the datatype of the reviews-column to float
apps_df.loc[:,'Reviews']=apps_df.loc[:,'Reviews'].astype('float')
[9]
#removing the + and , from the install-colmn and changing the data type to int:
for cell in range(len(apps_df.loc[:,'Installs'])):
    cell_value=apps_df.iloc[cell,5]
    if cell_value[-1]=="+":
        cell_value=cell_value.replace('+','')
        cell_value=cell_value.replace(',','')
        new_cell_value=int(cell_value)
        apps_df.iloc[cell,5]=new_cell_value
apps_df.loc[:,'Installs']=apps_df.loc[:,'Installs'].astype('int')
[10]
#removing dollar sign from the price-column and changing the data type to float:
for price in range(len(apps_df.loc[:,'Price'])):
    price_value=apps_df.iloc[price,7]
    if price_value[0]=='$':
        price_value=price_value.replace('$','')
        new_price=float(price_value)
        apps_df.iloc[price,7]=new_price
    if price_value=='Everyone':
        apps_df=apps_df.drop(apps_df.iloc[price,7],axis=0)
```

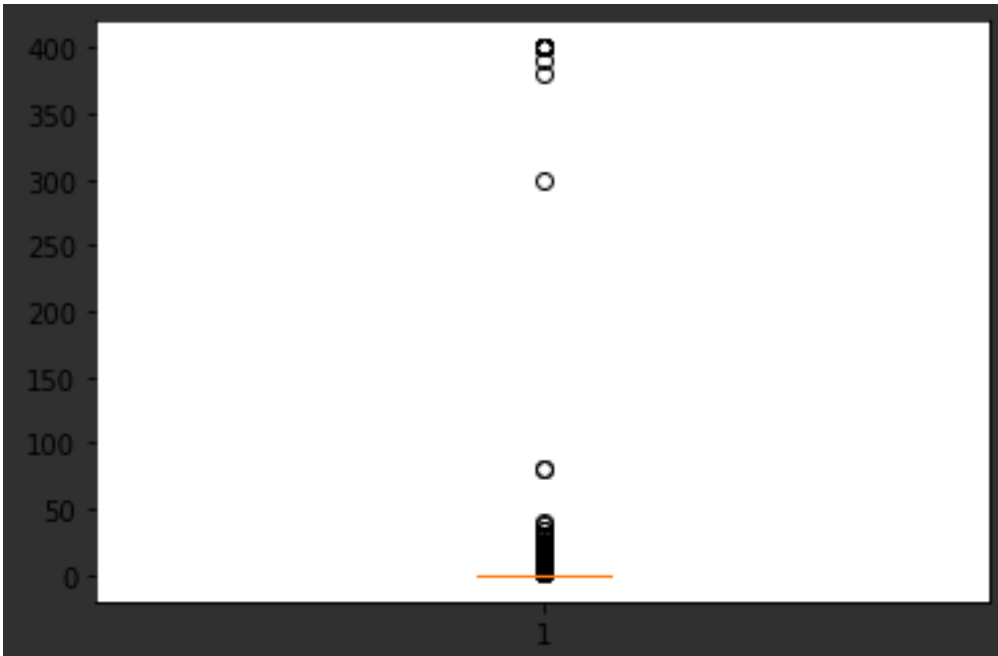
Sanity checks

```
[11]
#removing rating values that are less than 1 and greater than 5:
for rate in range(apps_df.shape[0]):
    if 1>apps_df.iloc[rate,2]>5:
        apps_df=apps_df.drop(apps_df.iloc[rate,2],axis=0)

[12]
#making sure that there is no rating values that are less than 1 or greater than 5
:
max_rating=max(apps_df.iloc[:,2])
```

```
min_rating=min(apps_df.iloc[:,2])
print(max_rating, min_rating)
5.0 1.0

[13]
#removing rows if the review value is greater than the install value:
for value in range(apps_df.shape[0]):
    if apps_df.iloc[value,3]>apps_df.iloc[value,3]:
        apps_df=apps_df.drop(index=[value])

[14]
#making sure that there is no Reviews-value greater than an Install-value
for value in range(apps_df.shape[0]):
    if apps_df.iloc[value,3]>apps_df.iloc[value,3]:
        print('there is a mistake in the code')

[15]
#droping any free apps with price >0 assigned to them:
for app in range(apps_df.shape[0]):
    if apps_df.iloc[app,6]=='Free':
        if int(apps_df.iloc[app,7])>0:
            apps_df=apps_df.drop(apps_df.iloc[app,6],axis=0)
[16]
#making sure there is no Type-free apps  that coast money:
for app in range(apps_df.shape[0]):
    if apps_df.iloc[app,6]=='Free':
        if int(apps_df.iloc[app,7])>0:
            print('there is free apps that coast money')
```

Performing univariate analysis

```
[17]
apps_df.dtypes
App                object
Category           object
Rating            float64
Reviews           float64
Size              float64
Installs            int32
Type               object
Price              object
Content Rating     object
Genres             object
Last Updated       object
Current Ver        object
Android Ver        object
dtype: object
[18]
#Boxplot for Price
apps_df.loc[:,'Price']=apps_df.loc[:,'Price'].astype('float')

plt.boxplot(apps_df.loc[:,'Price']);
```

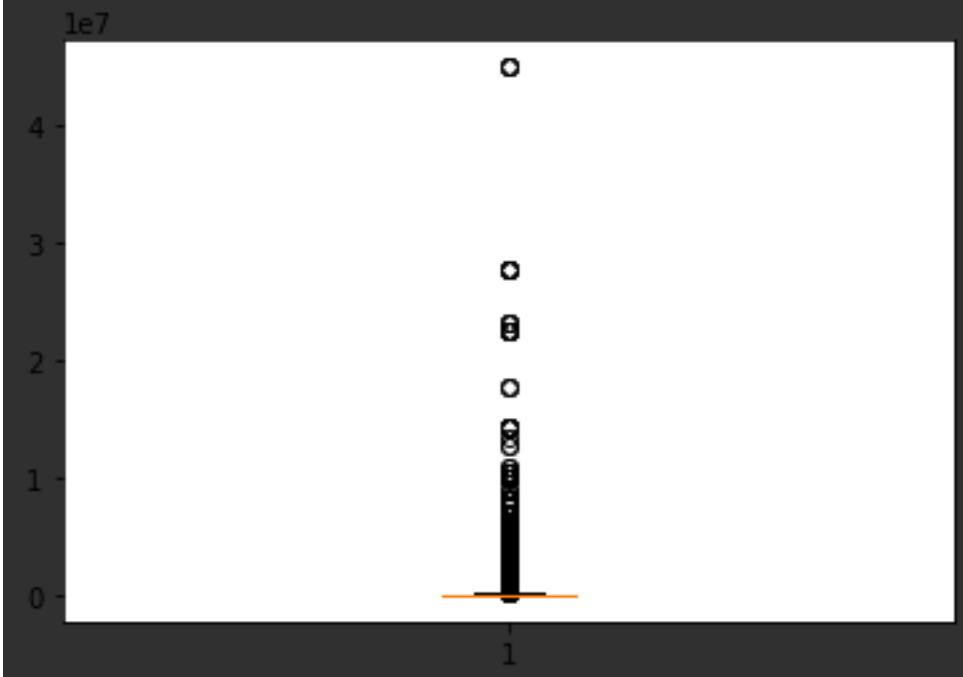


[19]

```
#The above plot have a huge amonut of outlier data, when the avearge apps coasts n
othing there are few apps that coast
#between 300 and 400 dollars
```

[20]

```
#box plot for reviews:
plt.boxplot(apps_df.loc[:,'Reviews']);
```

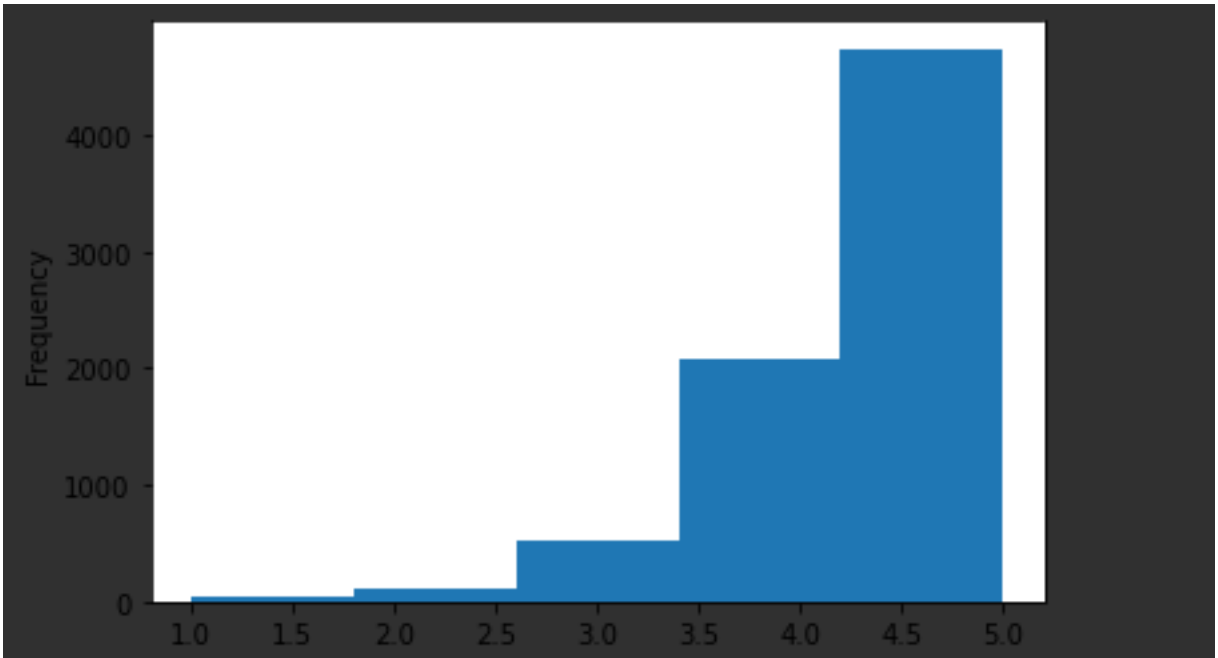


[21]

```
# the above boxplot shows that there are unlogical amount of reviews to an app.
```

[22]

```
# histogram for the column Rating
apps_df.loc[:,'Rating'].plot(kind='hist',bins=5,xlabel='Rating');
```

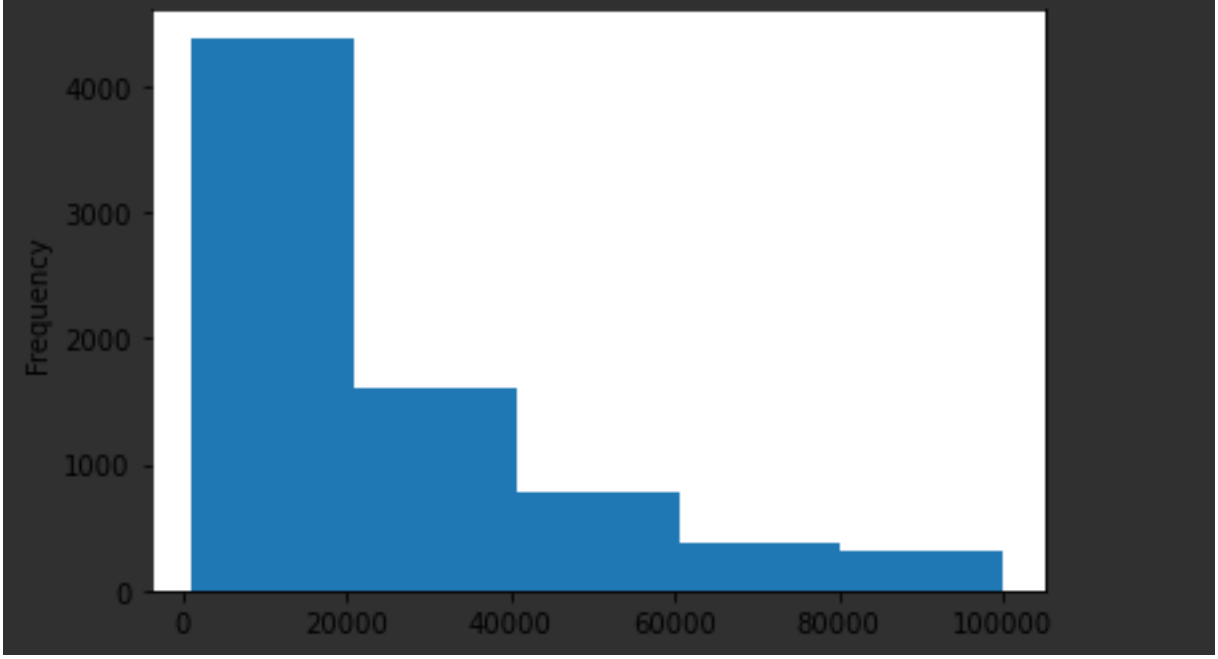


[23]

```
#the above histogram shows a  data skewed to the left,thus the data is negativle d
istributed
#meaning as the rating value increases the amount of people rating increases aswel
l.
```

[24]

```
#histogram for the size column:

for x in range(len(apps_df.loc[:,'Size'])):
    if apps_df.iloc[x,4]=='Varies with device':
        apps_df.iloc[x,4]=np.nan
    else:
        continue

apps_df.loc[:,'Size'].plot(kind='hist',bins=5,xlabel='App Size');
```



[25]
```
# the above histgram has a right skewed data-
distribution, thus the data is positively distributed
#meaning that, the number of apps are decrease as the size of data increase
```

Outlier treatment:

[26]
```
#checking the values that are more than 200$:
apps_df.loc[:,'Price']=apps_df.loc[:,'Price'].astype('float')
apps_price_lst=apps_df.loc[:,'Price'].tolist()
prices_greater_200=[]
for price in apps_price_lst:
    if price>200:
        prices_greater_200.append(price)

print(apps_df.loc[:,'Price'].unique())
print(prices_greater_200)
```
```
[  0.     4.99   6.99   7.99   3.99   5.99   2.99   1.99   9.99   0.99
   9.     5.49  10.    24.99  11.99  79.99  16.99  14.99  29.99  12.99
   3.49   7.49   1.5   19.99  15.99  33.99  39.99   2.49   4.49   1.7
   1.49   3.88 399.99  17.99 400.     3.02   1.76   4.84   4.77   1.61
   1.59 299.99 379.99  37.99  18.99 389.99   8.49   1.75  14.     2.
   3.08   2.59  19.4   15.46   8.99   3.04  13.99   4.29   3.28   4.6
   1.    10.99   2.9    1.97   2.56   1.2 ]
[399.99, 399.99, 400.0, 399.99, 399.99, 299.99, 399.99, 379.99, 399.99, 399.99,
399.99, 389.99, 399.99, 399.99]
```

[27]
```
#finding if the value 200 is really a high value by finding the upper and lower li
mits:
percent_tile=np.percentile(apps_df.Price,[25,75])
print(percent_tile)
IQR=percent_tile[1]-percent_tile[0]
lower_limit=percent_tile[0]-(1.5*IQR)
upper_limit=percent_tile[1]+(1.5*IQR)
print('The lower limit is ',lower_limit)
print('The upper limit is ',upper_limit)
```
```
[0. 0.]
The lower limit is  0.0
The upper limit is  0.0
```

[28]
```
# as we can see that 200$ is way greater than the upper limit which is 0$, we cann
 say that the value 200 is an outlier
```
[29]
```
#another way to find the percentiles
price_sum=sum(apps_df.Price)
percentile_25=(25/price_sum)*100
percentile_75=(75/price_sum)*100
LL=percentile_25-(1.5*IQR)
UL=percentile_25+(1.5*IQR)
```

```
print('The lower limit is ',LL)
print('The upper limit is ',UL)
The lower limit is  0.3072932991623213
The upper limit is  0.3072932991623213

[30]
#dropping values above 200
apps_df.loc[:,'outlier_or_not']=np.where((apps_df.loc[:,'Price']>=200),'outlier','
not')
print(apps_df.loc[:,'outlier_or_not'].unique())
outlier_df=apps_df[~(apps_df.loc[:,'outlier_or_not']=='outlier')]
outlier_df.head()
['not' 'outlier']

[31]
#droping apps with more than 2 million reviews:
outlier_df.loc[:,'Reviews>2M']=np.where((outlier_df.loc[:,'Reviews']>=2000000),'ou
tlier','not')
df_apps=outlier_df[~(outlier_df.loc[:,'Reviews>2M']=='outlier')]
df_apps.head()
C:\Users\Heat\anaconda3\lib\site-packages\pandas\core\indexing.py:1596:
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: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self.obj[key] = _infer_fill_value(value)
C:\Users\Heat\anaconda3\lib\site-packages\pandas\core\indexing.py:1745:
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: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  isetter(ilocs[0], value)

[32]
#making sure there are no reviews more than 2M:
max_review_value=max(df_apps.loc[:,'Reviews'])
print(max_review_value)
1986068.0

[33]
# calculating the different percentiles - 10, 25, 50, 70, 90, 95, 99 in the instal
l column:
perct=np.percentile(df_apps.loc[:,'Installs'],[10,25,50,70,90,95,99])
perct
array([1.e+03, 1.e+04, 1.e+05, 1.e+06, 1.e+07, 1.e+07, 5.e+07])
[34]
#finding out outliers:
percentiles=np.percentile(df_apps.loc[:,'Installs'],[25,75])
iqr=percentiles[1]-percentiles[0]
```

```
installs_lowe_limit=percentiles[0]-(1.5*iqr)
installs_upper_limit=percentiles[1]+(1.5*iqr)
print('The lower limit is ',installs_lowe_limit)
print('The upper limit is ',installs_upper_limit)
```

```
The lower limit is  -1475000.0
The upper limit is  2485000.0
```

[35]

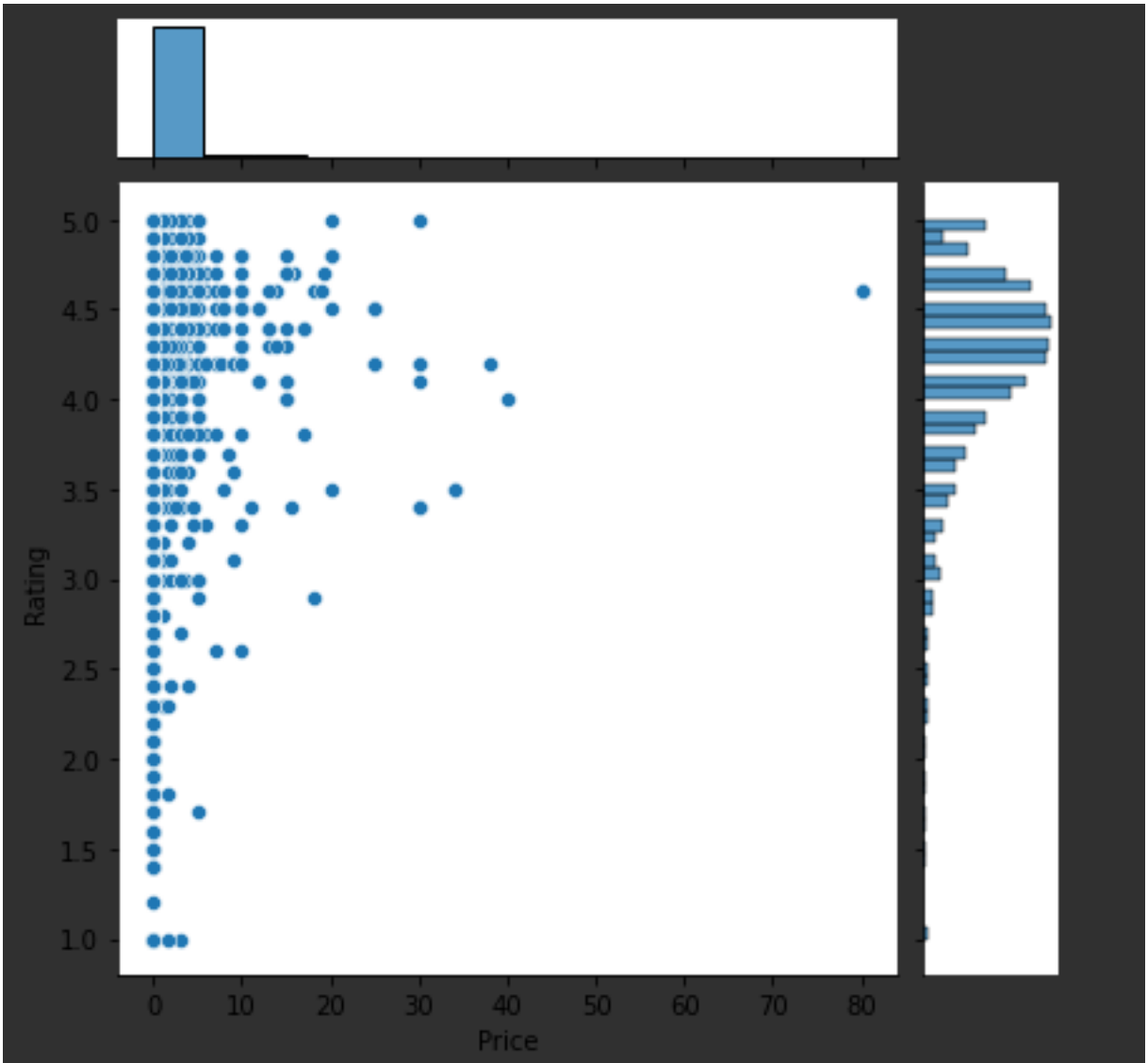
```
#removing outliers:
df_apps.loc[:,'install_outliers']=np.where(((df_apps.loc[:,'Installs']>=installs_u
pper_limit)|(df_apps.loc[:,'Installs']<=installs_lowe_limit)),'outlier','not')
new_df=df_apps[~(df_apps.loc[:,'install_outliers']=='outlier')]
new_df.head()
```

```
C:\Users\Heat\anaconda3\lib\site-packages\pandas\core\indexing.py:1596:
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: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self.obj[key] = _infer_fill_value(value)
C:\Users\Heat\anaconda3\lib\site-packages\pandas\core\indexing.py:1745:
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: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  isetter(ilocs[0], value)
```

Bivariate analysis

[36]

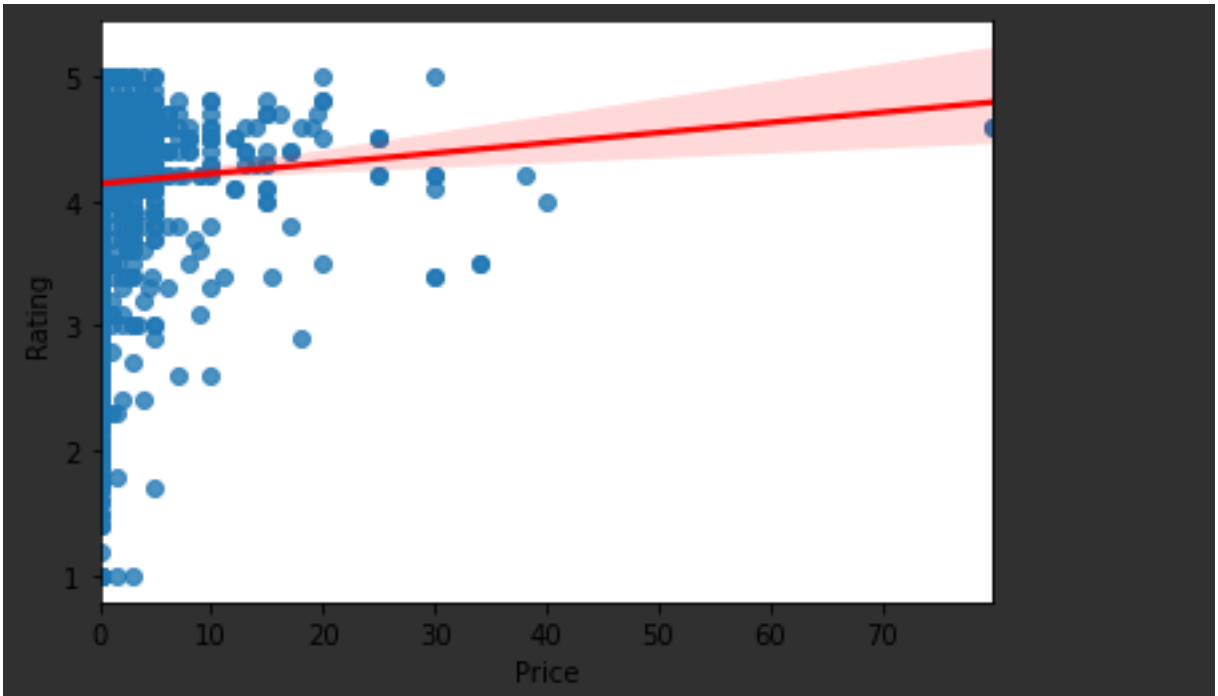
```
#scatter plot/joinplot for Rating vs. Price
sns.jointplot(data=new_df,x='Price',y='Rating',kind='scatter');
```



[37]

```
sns.regplot(data=new_df,x=new_df.loc[:,'Price'],y=new_df.loc[:,'Rating'],line_kws=
{"color": "red"})
```

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

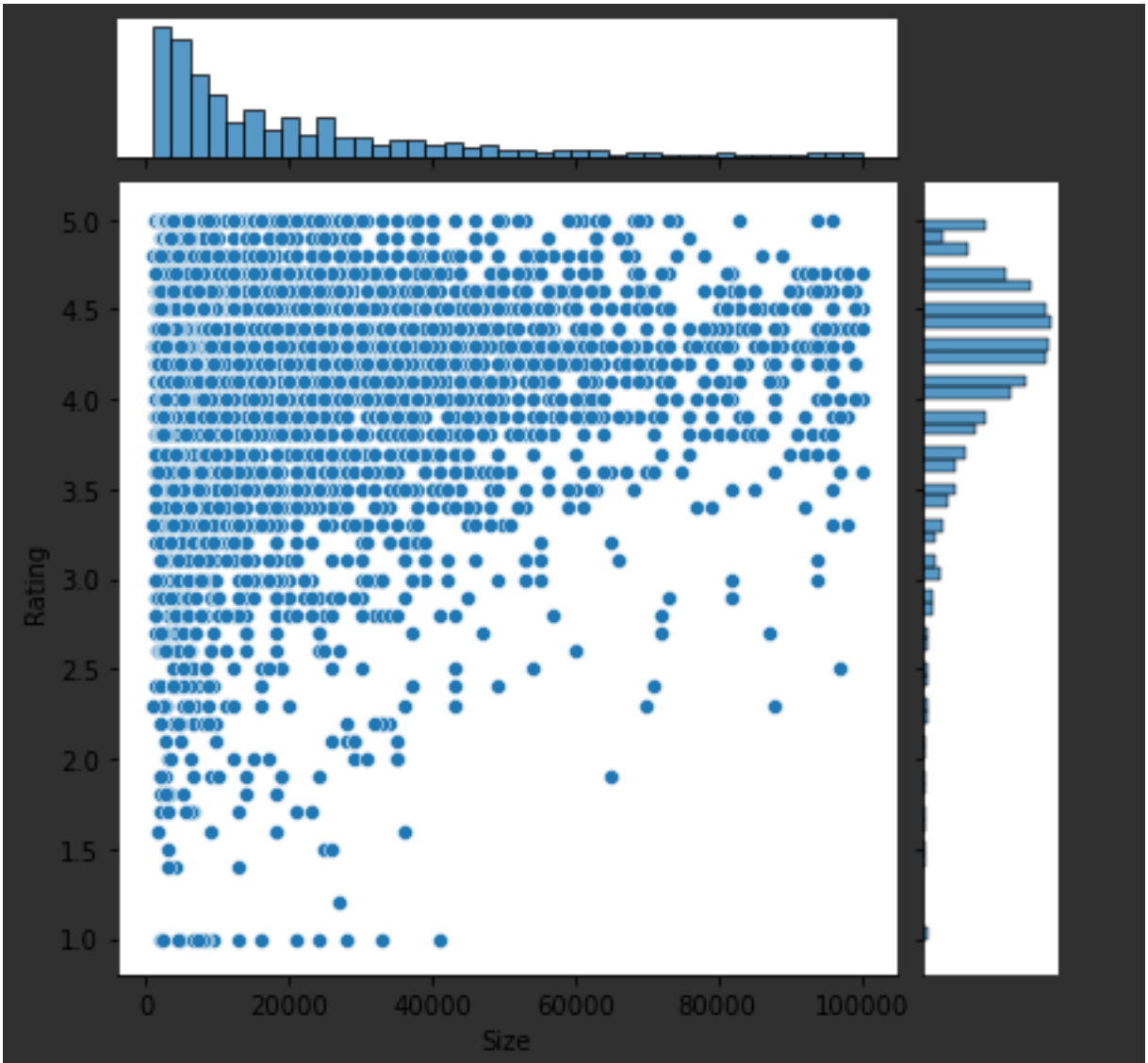


[38]

```
#the above plot shows a slight positive relation btween the rating of the app and
its price.meaning that a high price
#will asure you that the app will have a high rating but lower price dosen't mean
that the app will not have a high rating.
```

[39]

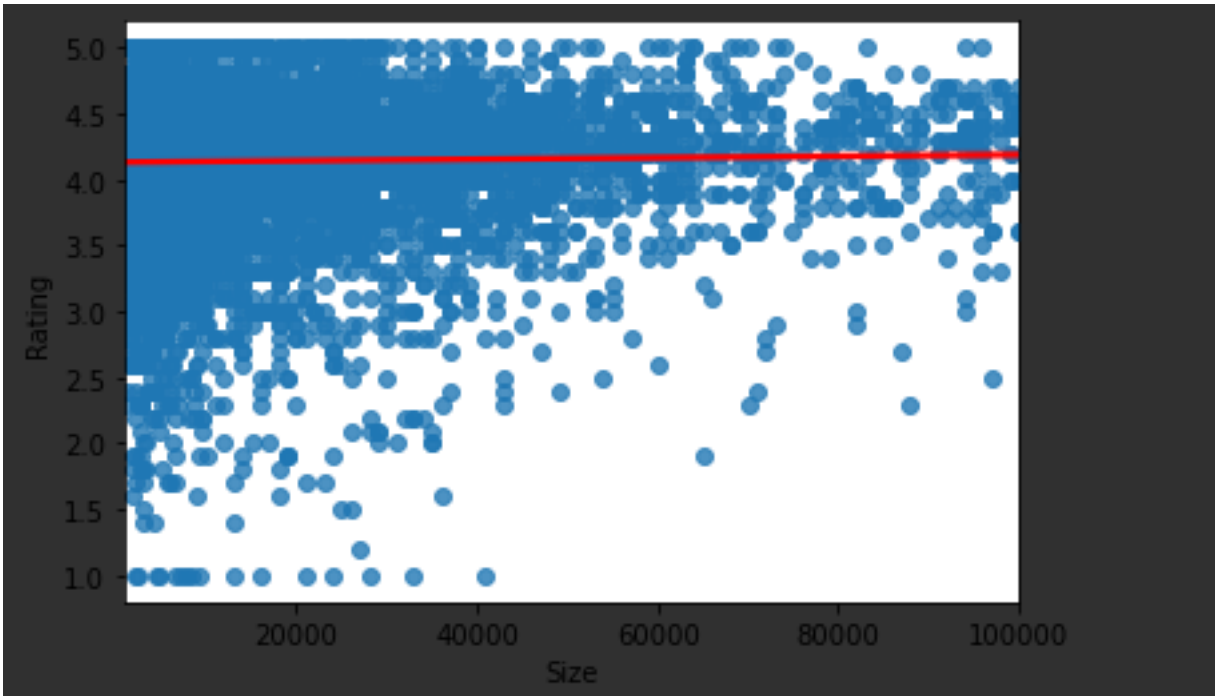
```
#scatter plot/joinplot for Rating vs. Size:
sns.jointplot(data=new_df,x='Size',y='Rating',kind='scatter');
```



[40]

```
sns.regplot(data=new_df,x=new_df.loc[:,'Size'],y=new_df.loc[:,'Rating'],line_kws={
"color": "red"})
```

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

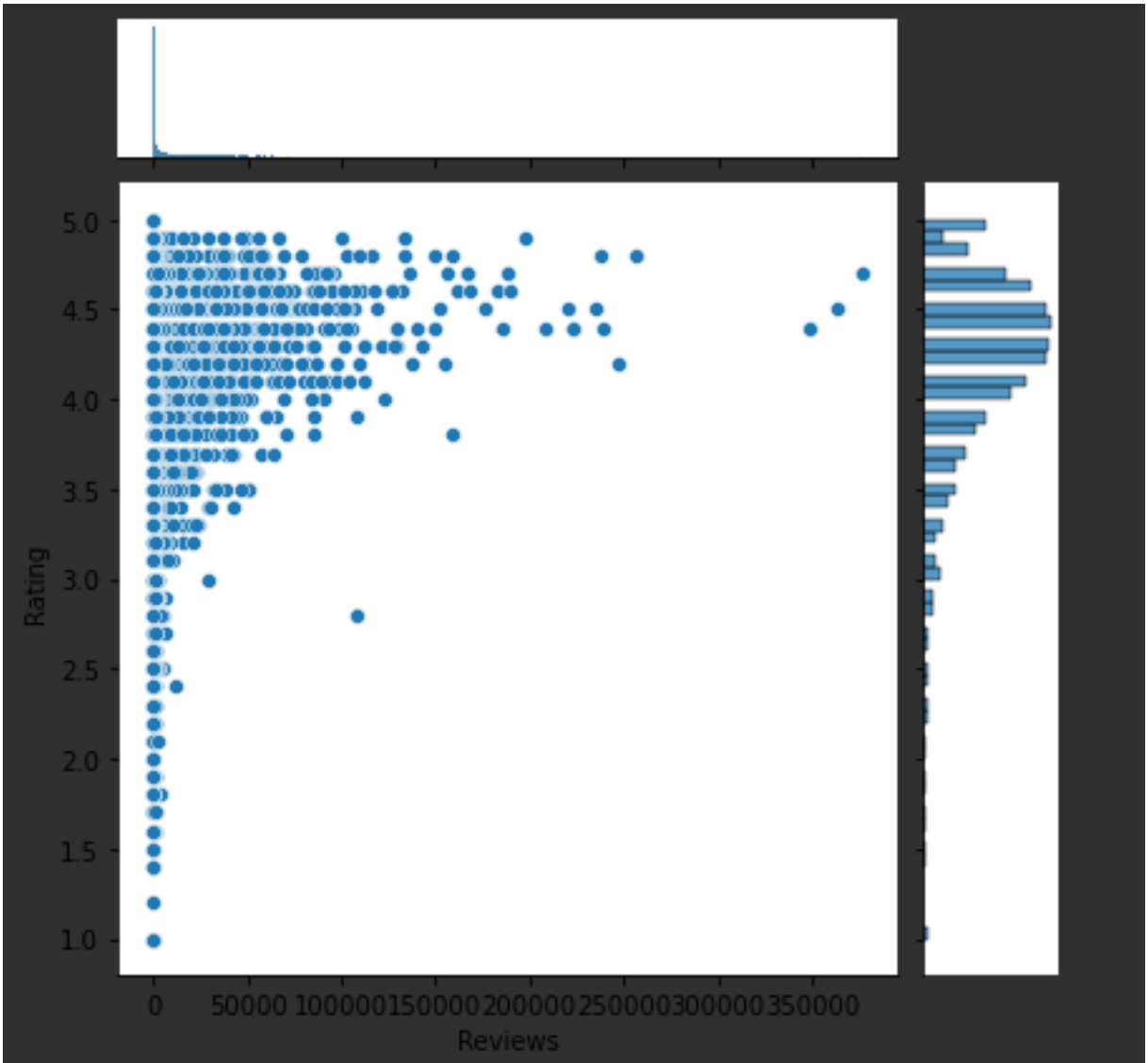


[41]

```
#the above plots say that apps with higher size will have higher rating however a
small app
#does not mean a low rate all the time
```

[42]

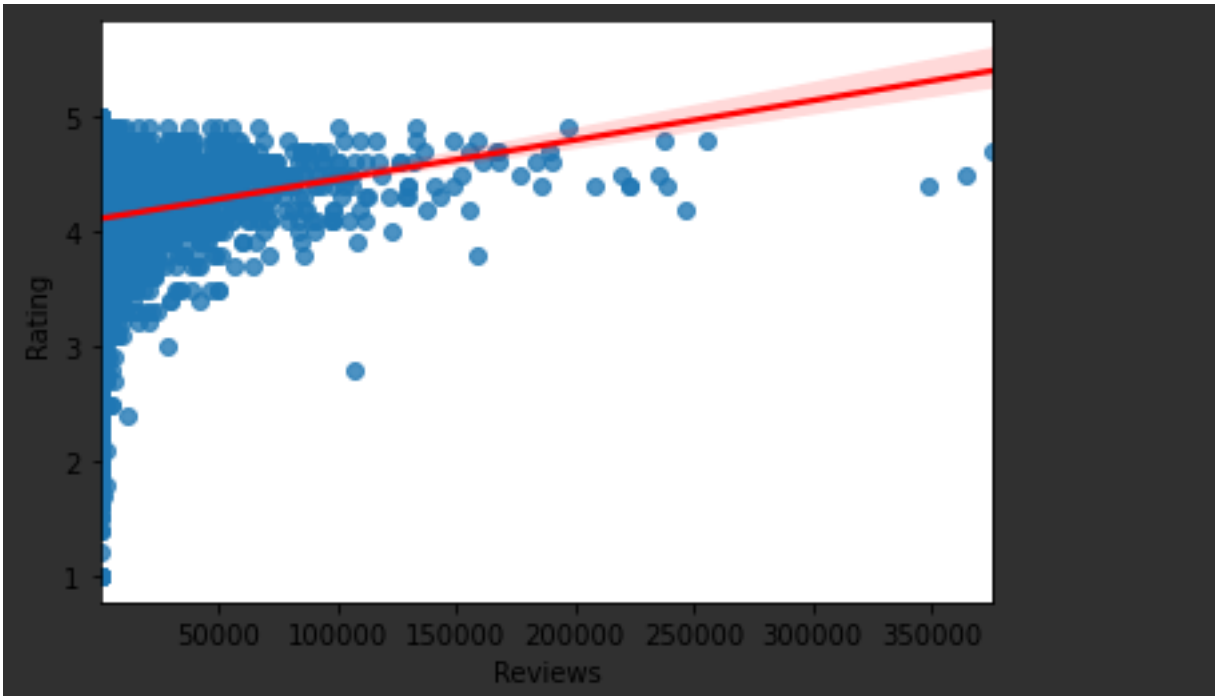
```
#scatter plot/joinplot for Rating vs. Reviews
sns.jointplot(data=new_df,x='Reviews',y='Rating',kind='scatter');
```



[43]

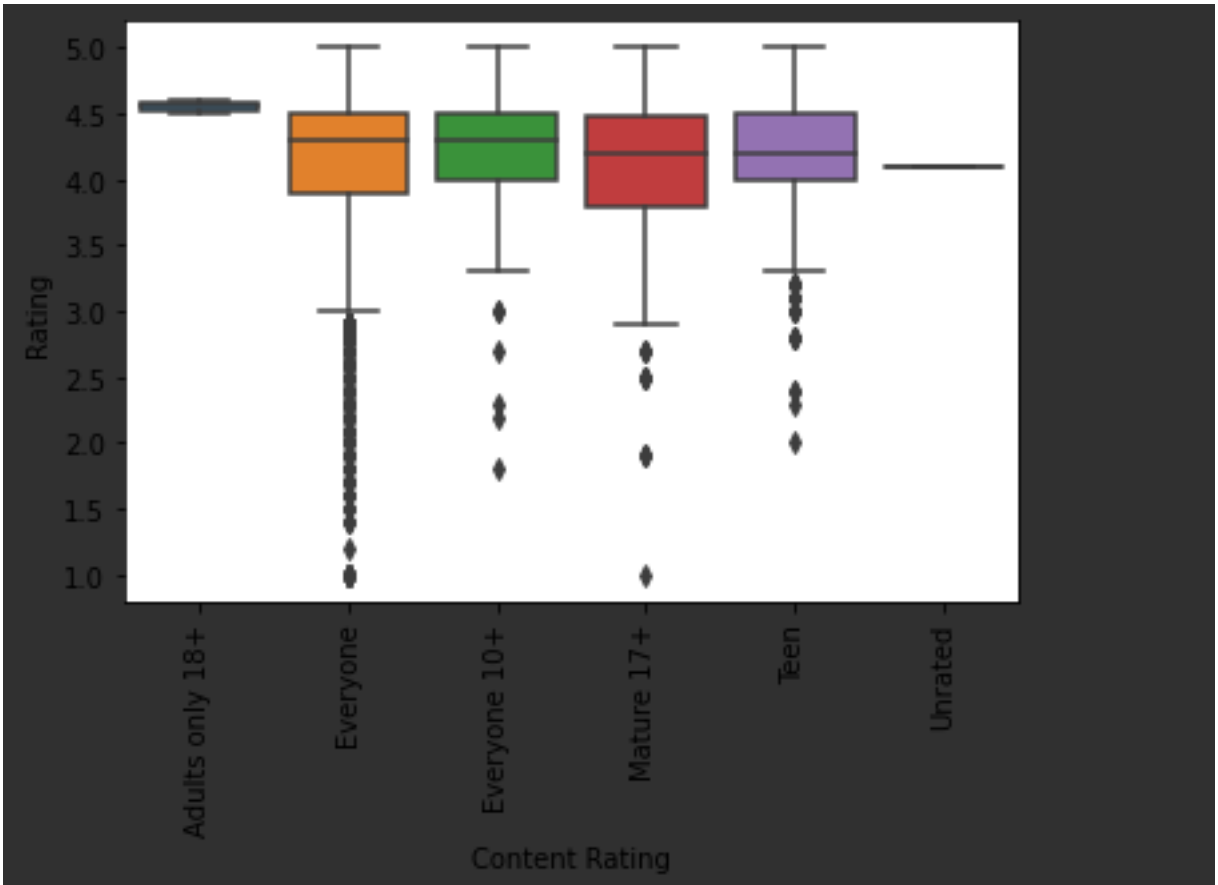
```
sns.regplot(data=new_df,x=new_df.loc[:,'Reviews'],y=new_df.loc[:,'Rating'],line_kws={"color": "red"})
```

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



[44]
```
#the above plots shows that as the number of reviews increase the chance that the
app will have a high rating increase.
```
[45]
```
#boxplot for Rating vs. Content Rating
new_df['Content Rating']=new_df['Content Rating'].astype('category')
fig,ax=plt.subplots()
box_plot=sns_plot=sns.boxplot(data=new_df,x='Content Rating',y='Rating',ax=ax,orde
r=new_df.groupby('Content Rating')['Rating'].describe().index)
box_plot.set_xticklabels(sns_plot.get_xticklabels(), rotation=90)
```

```
<ipython-input-45-e90d3cf841a3>:2: 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: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  new_df['Content Rating']=new_df['Content Rating'].astype('category')

[Text(0, 0, 'Adults only 18+'),
 Text(1, 0, 'Everyone'),
 Text(2, 0, 'Everyone 10+'),
 Text(3, 0, 'Mature 17+'),
 Text(4, 0, 'Teen'),
 Text(5, 0, 'Unrated')]
```



[46]

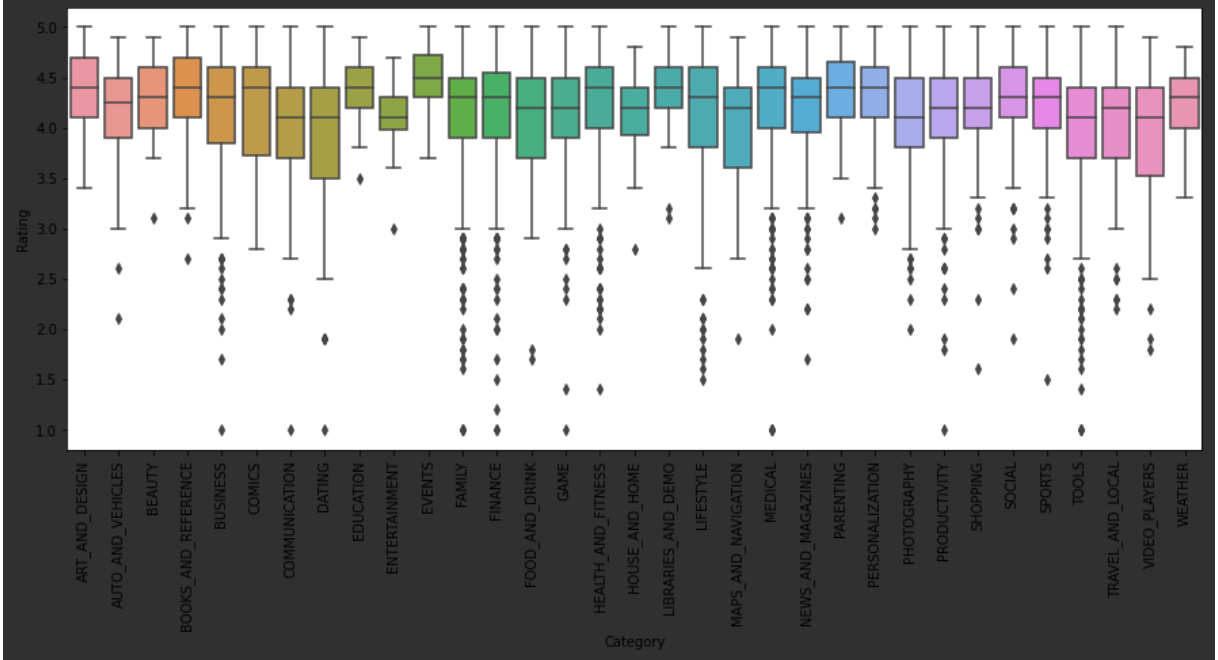
```
# the above boxplot shows that all the contents have a similar rating thus the con
tent have no significant influence on the
#app rating.
```

[47]

```
#boxplot for Ratings vs. Category
new_df.loc[:,'Category']=new_df.loc[:,'Category'].astype('category')
fig2,ax=plt.subplots(figsize=(15,6))
sns_plot=sns_plot=sns.boxplot(data=new_df,x='Category',y='Rating',ax=ax,order=new_
df.groupby('Category')['Rating'].describe().index);
sns_plot.set_xticklabels(sns_plot.get_xticklabels(), rotation=90)
```

```
C:\Users\Heat\anaconda3\lib\site-packages\pandas\core\indexing.py:1745:
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: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  isetter(ilocs[0], value)

[Text(0, 0, 'ART_AND_DESIGN'),
 Text(1, 0, 'AUTO_AND_VEHICLES'),
 Text(2, 0, 'BEAUTY'),
 Text(3, 0, 'BOOKS_AND_REFERENCE'),
 Text(4, 0, 'BUSINESS'),
 Text(5, 0, 'COMICS'),
 Text(6, 0, 'COMMUNICATION'),
```

```
 Text(7, 0, 'DATING'),
 Text(8, 0, 'EDUCATION'),
 Text(9, 0, 'ENTERTAINMENT'),
 Text(10, 0, 'EVENTS'),
 Text(11, 0, 'FAMILY'),
 Text(12, 0, 'FINANCE'),
 Text(13, 0, 'FOOD_AND_DRINK'),
 Text(14, 0, 'GAME'),
 Text(15, 0, 'HEALTH_AND_FITNESS'),
 Text(16, 0, 'HOUSE_AND_HOME'),
 Text(17, 0, 'LIBRARIES_AND_DEMO'),
 Text(18, 0, 'LIFESTYLE'),
 Text(19, 0, 'MAPS_AND_NAVIGATION'),
 Text(20, 0, 'MEDICAL'),
 Text(21, 0, 'NEWS_AND_MAGAZINES'),
 Text(22, 0, 'PARENTING'),
 Text(23, 0, 'PERSONALIZATION'),
 Text(24, 0, 'PHOTOGRAPHY'),
 Text(25, 0, 'PRODUCTIVITY'),
 Text(26, 0, 'SHOPPING'),
 Text(27, 0, 'SOCIAL'),
 Text(28, 0, 'SPORTS'),
 Text(29, 0, 'TOOLS'),
 Text(30, 0, 'TRAVEL_AND_LOCAL'),
 Text(31, 0, 'VIDEO_PLAYERS'),
 Text(32, 0, 'WEATHER')]
```



[48]

```
#the above boxplots shows that the category 'Events' have slightly better rating t
han the rest of the categories.
```

Data preprocessing

[49]

```
inp1=new_df.copy()
inp1.head()
```
[50]
```
#log transformation (np.log1p) to Reviews and Installs:

skwe_reviews_data=np.log1p(inp1.loc[:,'Reviews'])
skwe_installs_data=np.log1p(inp1.loc[:,'Installs'])
```
[51]
```
#droping columns App, Last Updated, Current Ver, Android Ver,outlier_or_not, Revie
ws>2M, install_outliers
inp1=inp1.drop(columns=['App','Last Updated','Current Ver','Android Ver','outlier_
or_not','Reviews>2M','install_outliers'],axis=1)
```
[52]
```
inp1.head()
```
[53]
```
#getting dummy columns for Category, Genres, and Content Rating:
inp2=pd.get_dummies(inp1,columns=['Category','Genres','Content Rating','Type'])

```
[54]
```
inp2.dtypes
```
```
Rating                         float64
Reviews                        float64
Size                           float64
Installs                         int32
Price                          float64
                                ...
Content Rating_Mature 17+        uint8
Content Rating_Teen              uint8
Content Rating_Unrated           uint8
Type_Free                        uint8
Type_Paid                        uint8
Length: 152, dtype: object
```
[55]
```
#Train test split  and apply 70-30 split:
df_test=inp2.loc[:,'Rating']
df_train=inp2.drop(columns=['Rating'],axis=1)

```
[56]
```
from sklearn.model_selection import train_test_split

X_train, y_train, X_test, y_test=train_test_split(df_train, df_test, test_size=0.3
, random_state=0)
```

Model building

[57]
```
from sklearn.linear_model import LinearRegression
rating_lr=LinearRegression().fit(df_train,df_test)
```
[58]
```
#The regression's intercept
rating_lr.intercept_
```
```
4.286105369846971
```
[59]
```
#the regression's coef.
rating_lr.coef_
```

```
array([ 3.37790265e-06, -4.38559764e-07,  3.43760299e-08, -1.05590180e-03,
       -1.22755058e-01, -2.84896891e-03,  7.48022150e-02,  8.01082168e-02,
       -1.04726771e-02,  2.12041453e-01, -9.12713944e-02, -1.23430507e-01,
        3.44365828e-02, -4.76202522e-02,  1.74409016e-01,  3.36335878e-02,
       -2.97832199e-02, -5.23063802e-02,  2.52133861e-01, -3.92333867e-03,
       -1.25269052e-02,  1.03925640e-01, -1.98480809e-02, -9.50783525e-02,
        1.73958474e-02, -1.55437207e-02, -2.25395740e-02,  7.12233202e-02,
       -7.87314640e-02, -3.40886077e-02,  6.38670704e-03,  3.43760968e-02,
       -9.24854711e-02, -9.52460458e-02, -5.83610381e-02, -1.09240343e-01,
        2.32288558e-02, -2.92718650e-01,  9.22858755e-02, -3.22507411e-01,
       -6.25458469e-02,  2.75523257e-01, -2.37987488e-01,  4.96786233e-02,
        2.73307911e-01,  3.53844969e-01,  4.34949625e-01, -1.88398774e-01,
       -2.84896896e-03,  7.48022150e-02, -2.29482392e-01, -1.94546476e-01,
        1.11994995e-01,  5.15844531e-01,  8.01082168e-02, -1.78225833e-02,
       -1.04726773e-02, -5.74079284e-01, -4.68096330e-01,  2.53384424e-01,
       -2.42951872e-01, -1.78239591e-01, -3.62200078e-01,  2.91790109e-01,
        4.24511600e-01,  2.49064734e-02, -6.58484634e-02, -2.59084937e-01,
        4.71126390e-01, -9.12713944e-02, -1.23430507e-01,  1.08904280e-01,
        7.99799493e-02,  6.29020822e-02,  1.32921014e-01,  1.08528855e-01,
        3.83183015e-02,  1.56438589e-01, -3.36155320e-01,  1.35303802e-01,
       -9.71970596e-02, -3.90167757e-01,  9.01830304e-02, -6.75919774e-02,
       -9.77071983e-02,  8.62207540e-03, -1.00239788e-01,  3.93427570e-01,
        1.93016656e-01, -1.31081981e-01, -4.81311873e-01,  1.74409016e-01,
       -2.97832199e-02, -5.23063802e-02, -3.92333867e-03, -3.55163268e-01,
        5.35123912e-01, -1.25269052e-02,  1.03925640e-01, -1.98480809e-02,
       -9.50783525e-02,  1.73958476e-02, -4.14628080e-01,  1.30218858e-01,
        2.70698784e-01, -1.55437208e-02,  2.63401914e-01, -3.41230025e-01,
       -2.63981057e-01,  3.19269594e-01,  7.12233200e-02, -7.87314640e-02,
       -3.40886078e-02,  7.87261396e-02,  1.60772327e-01,  8.75301812e-02,
        1.16496309e-01,  4.51126876e-01, -3.55879117e-01,  4.56115653e-02,
        3.37940783e-01, -8.35118204e-02,  3.19156226e-01, -3.01890111e-01,
        6.38670705e-03, -1.03562257e-01, -8.40398940e-02, -9.95041411e-03,
        1.46724549e-01,  3.43760968e-02,  1.21016830e-01, -1.93139211e-01,
       -2.40004526e-01,  3.06370753e-01, -1.96593691e-01,  3.41999596e-01,
       -9.52460459e-02, -9.49781184e-02,  3.66170803e-02, -3.59501417e-01,
       -1.09240343e-01,  2.32288557e-02, -1.65997039e-01,  2.08082417e-01,
       -9.18039396e-02, -8.62293056e-02, -6.68834408e-02, -4.03025955e-02,
        7.71368646e-02, -7.56683544e-02,  7.56683544e-02])
[60]
#making predictos
predicted_ratings=pd.DataFrame(rating_lr.predict(df_train),columns=['Predicted_Rat
ings'])
predicted_ratings
[64]
#Reporting R2 score on test set:
from sklearn.metrics import r2_score
r2_score(df_test,predicted_ratings)
0.07010247227466682
```