Author: Bo Yan

Table of Contents

DSE 200 FINAL PROJECT

Fall 2020

Due Date: December 4th, 2020

The final project is comprised of two parts:

- **Part I** is a set of coding questions that require the *numpy* library to analyze the provided dataset.
- **Part II** is a guided project for you to build your own end-to-end analysis using Python, especially using what you learned on Python *IO*, *pandas*, *matplotlib* and *scilitlearn* libraries.

Deliverables: Submit both parts as one notebook via Canvas by midnight on the due date above along with clear instructions on how to download the datasets you used for Part II and reproduce your results. The notebook should be organized with a clear table of contents on top *(see example in the Pylaski notebook at https://github.com/words-sdsc/wifire/blob/master/pylaski.ipynb)* and links to the parts/steps outlined. Don't forget to add your name on top as the author of the notebook.

PART I: 20%

Preliminaries

In [1]:
```
import numpy as np
```

1.1 Preliminaries

Use numpy to load `breast_cancer.npy` into a numpy matrix. Print the dataset's shape and the first 5 rows.

Output required:

- Tuple representing dataset's shape
- Matrix representing the first 5 rows

In [2]:
```
# For reference
column_names = ['Id','Radius','Texture','Perimeter','Concavity','Cancer_Stage']
stages_encoding = {'In-situ': 1, 'Early-stage': 2, 'Lymphatic': 3}
```

In [3]:
```
# Load data
data = np.load('breast_cancer.npy')

# Your code here

print("Data shape:\n", data.shape)
print("\nFirst 5 rows of the data:\n", data[:5, :])
```

```
Data shape:
 (120, 6)

First 5 rows of the data:
 [[1.  5.1 3.5 1.4 0.2 1. ]
 [2.  4.9 3.  1.4 0.2 1. ]
 [3.  4.7 3.2 1.3 0.2 1. ]
 [4.  4.6 3.1 1.5 0.2 1. ]
 [5.  5.  3.6 1.4 0.2 1. ]]
```

1.2 Transformations

The first column is the id of the patient, which isn't relevant for our purposes. Remove that column from the matrix by creating a new matrix composed of the rest of the columns.
As usual, print the shape of the resulting dataset and the first 5 rows.

Output required:

- Tuple representing dataset's shape
- Matrix representing the first 5 rows

In [4]:
```
new_data = data.copy()
new_data = np.delete(new_data, 0, 1)
print("New Data shape:\n", new_data.shape)
print("\nFirst 5 rows of the new data:\n", new_data[:5, :])
```

```
New Data shape:
 (120, 5)

First 5 rows of the new data:
 [[5.1 3.5 1.4 0.2 1. ]
 [4.9 3.  1.4 0.2 1. ]
 [4.7 3.2 1.3 0.2 1. ]
 [4.6 3.1 1.5 0.2 1. ]
 [5.  3.6 1.4 0.2 1. ]]
```

1.3 Summary Statistics

Note: Don't worry about the order in which you display the values in this section. Display them in whatever order/grouping makes most sense to you

a) Print the means and standard deviations of each column.

Output required:

- Floats representing the standard deviation of each column
- Floats representing the mean of each column

In [5]:
```
print("The standard deviation of each column:\n", new_data.std(axis = 0))
print("\nThe mean of each column:\n", new_data.mean(axis = 0))
```

```
The standard deviation of each column:
 [0.76756786 0.46343824 1.69690711 0.70496454 0.72168784]

The mean of each column:
 [5.6525 3.065  3.3275 0.995  1.75  ]
```

b) Print the minimum and maximum values of each column

Output required:

- Floats representing the minimum value found in each column
- Floats representing the maximum value found in each column

In [6]:
```
print("The minimum value of each column:\n", new_data.min(axis = 0))
print("\nThe maximum value of each column:\n", new_data.max(axis = 0))
```

```
The minimum value of each column:
 [4.3 2.  1.  0.1 1. ]

The maximum value of each column:
 [7.7 4.4 6.9 2.5 3. ]
```

c) Calculate the Cancer_stage-wise means and standard deviations.
Report these values with respect to the actual *name* of the stage, for which you must refer to 1.1

Output required:

- For each of the 3 cancer stages in the dataset:
  - Floats representing the standard deviation of each column for this stage
  - Floats representing the mean of each column for this stage

In [7]:
```
import warnings
warnings.filterwarnings(action = 'ignore', category = DeprecationWarning)

# sorted the last column
def sortFifth(val):
    return val[4]

new_data.tolist().sort(key = sortFifth)

# calculate count of each stage
num1 = 0
num2 = 0
num3 = 0

for i in range(0,121):
    if(new_data[i-1:i, 4:] == [1.]):
        num1 += 1
    elif(new_data[i-1:i, 4:] == [2.]):
        num2 += 1
    elif(new_data[i-1:i, 4:] == [3.]):
        num3 += 1

In_situ = new_data[0:num1, 0:4]
print("The mean of each column for In-situ stage:\n", In_situ.mean(axis = 0))
print("\nThe standard deviation of each column for In-situ stage:\n", In_situ.std(axis = 0))

Early_stage = new_data[num1:(num1 + num2), 0:4]
print("\nThe mean of each column for Early stage:\n", Early_stage.mean(axis = 0))
print("\nThe standard deviation of each column for Early stage:\n", Early_stage.std(axis = 0))

Lymphatic = new_data[(num1 + num2):120, 0:4]
print("\nThe mean of each column for Lymphatic stage:\n",Lymphatic.mean(axis = 0))
print("\nThe standard deviation of each column for Lymphatic stage:\n", Lymphatic.std(axis = 0))
```

```
The mean of each column for In-situ stage:
 [5.006 3.418 1.464 0.244]

The standard deviation of each column for In-situ stage:
 [0.34894699 0.37719491 0.17176728 0.10613199]

The mean of each column for Early stage:
 [5.936 2.77  4.26  1.326]

The standard deviation of each column for Early stage:
 [0.51098337 0.31064449 0.46518813 0.19576517]

The mean of each column for Lymphatic stage:
 [6.56  2.92  5.655 2.045]

The standard deviation of each column for Lymphatic stage:
 [0.71302174 0.37629775 0.62407932 0.26734809]
```

PART II : 80%

This project is culmination of all you've learned in this course! You should expect to spend **24-32 total hours** on the project. Be sure to read all of the items below before starting.

There are a number of steps outlined below, but is critical that you do not view this as an entirely linear process. Remember that the science component in data science is the creation of a hypothesis based on exploration and testing of that hypothesis through analysis. You may need to go through many of these steps multiple times before you arrive at meaningful hypothesis or conclusions.

Step 1: Find a dataset or datasets

Based on your interest, identify a dataset which you will want to examine. You will find a starting point for where you can find open datasets at the end of this notebook, but feel free to use other datasets you have access to and can publicly share results about.

This step may take some time, as you'll likely look at a number of datasets before you find one (or more) which holds promising data for the kinds of questions you want to ask. You are expected to use at least two interconnected datasets, e.g., two tables in one database or a combination of datasets which you can merge in some meaningful way.

In [8]:
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import time
import datetime
import re
import seaborn as sns
import nltk
from math import sqrt
from math import pi
from nltk.corpus import stopwords
import warnings
warnings.filterwarnings("ignore")
from sklearn.metrics import accuracy_score, mean_squared_error,mean_absolute_error,r2_score
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import Ridge, Lasso
from sklearn import metrics
from sklearn import tree
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn import preprocessing
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.feature_extraction.text import CountVectorizer
from kneed import KneeLocator, DataGenerator
from itertools import cycle, islice
from pandas.plotting import parallel_coordinates
from wordcloud import WordCloud

%matplotlib inline
```

Instructions on how to download the datasets and reproduce the results

How to download the datasets?

Go to Kaggle by clicking the following link to download the Google Play Store Apps Datasets.

https://www.kaggle.com/lava18/google-play-store-apps

How to reproduce the results?

A. Put this notebook and datasets in one folder.

```
Folder name: Final Project
```

B. In this folder, it contains the following items.

```
a) Final Project_Bo Yan.ipynb

b) breast_cancer.npy

c) googleplaystore.csv

d) googleplaystore_user_reviews.csv

e) Final Project_Bo Yan.pdf
```

C. Run the Kernel to get the results. It takes 1~2 minutes to run all cells.

Get the data and inspect it

In [9]:
```
# Load googleplaystore_user_reviews data, print shape and head of the dataframe.
df_google_play_store_user_reviews = pd.read_csv("./googleplaystore_user_reviews.csv")
print("Shape of DataFrame for Google Play Store User Reviews data:\n", df_google_play_store_user_reviews.sha
print("\nHead of DataFrame for Google Play Store User Reviews data:\n", df_google_play_store_user_reviews.he

# Load googleplaystore data, print shape and head of the dataframe.
df_google_play_store = pd.read_csv("./googleplaystore.csv")
print("\nShape of DataFrame for Google Play Store data:\n", df_google_play_store.shape)
print("\nHead of DataFrame for Google Play Store data:\n", df_google_play_store.head())
```

```
Shape of DataFrame for Google Play Store User Reviews data:
 (64295, 5)

Head of DataFrame for Google Play Store User Reviews data:
                      App                                  Translated_Review  \
0  10 Best Foods for You  I like eat delicious food. That's I'm cooking ...
1  10 Best Foods for You    This help eating healthy exercise regular basis
2  10 Best Foods for You                                                NaN
3  10 Best Foods for You         Works great especially going grocery store
4  10 Best Foods for You                                       Best idea us

  Sentiment  Sentiment_Polarity  Sentiment_Subjectivity
0  Positive                1.00                0.533333
1  Positive                0.25                0.288462
2       NaN                 NaN                     NaN
3  Positive                0.40                0.875000
4  Positive                1.00                0.300000

Shape of DataFrame for Google Play Store data:
 (10841, 13)

Head of DataFrame for Google Play Store data:
                                                  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
```

In [10]:
```
print(df_google_play_store_user_reviews.columns)
```

```
Index(['App', 'Translated_Review', 'Sentiment', 'Sentiment_Polarity',
       'Sentiment_Subjectivity'],
      dtype='object')
```

In [11]:
```
print(df_google_play_store_user_reviews['App'].unique())
```

```
['10 Best Foods for You' '104 找工作 - 找工作 找打工 找兼職 履歷健檢 履歷診療室' '11st' ...
 'Hotwire Hotel & Car Rental App' 'Housing-Real Estate & Property'
 'Houzz Interior Design Ideas']
```

In [12]:
```
df_google_play_store_user_reviews.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 64295 entries, 0 to 64294
Data columns (total 5 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   App                     64295 non-null  object
 1   Translated_Review       37427 non-null  object
 2   Sentiment               37432 non-null  object
 3   Sentiment_Polarity      37432 non-null  float64
 4   Sentiment_Subjectivity  37432 non-null  float64
dtypes: float64(2), object(3)
memory usage: 2.5+ MB
```

In [13]:
```
print(df_google_play_store.columns)
```

```
Index(['App', 'Category', 'Rating', 'Reviews', 'Size', 'Installs', 'Type',
       'Price', 'Content Rating', 'Genres', 'Last Updated', 'Current Ver',
       'Android Ver'],
      dtype='object')
```

In [14]:
```
print(df_google_play_store['Category'].unique())
```

```
['ART_AND_DESIGN' 'AUTO_AND_VEHICLES' 'BEAUTY' 'BOOKS_AND_REFERENCE'
 'BUSINESS' 'COMICS' 'COMMUNICATION' 'DATING' 'EDUCATION' 'ENTERTAINMENT'
 'EVENTS' 'FINANCE' 'FOOD_AND_DRINK' 'HEALTH_AND_FITNESS' 'HOUSE_AND_HOME'
 'LIBRARIES_AND_DEMO' 'LIFESTYLE' 'GAME' 'FAMILY' 'MEDICAL' 'SOCIAL'
 'SHOPPING' 'PHOTOGRAPHY' 'SPORTS' 'TRAVEL_AND_LOCAL' 'TOOLS'
 'PERSONALIZATION' 'PRODUCTIVITY' 'PARENTING' 'WEATHER' 'VIDEO_PLAYERS'
 'NEWS_AND_MAGAZINES' 'MAPS_AND_NAVIGATION' '1.9']
```

In [15]:
```
print(df_google_play_store['Type'].unique())
```

```
['Free' 'Paid' nan '0']
```

In [16]:
```
print(df_google_play_store['Content Rating'].unique())
```

```
['Everyone' 'Teen' 'Everyone 10+' 'Mature 17+' 'Adults only 18+' 'Unrated'
 nan]
```

In [17]:
```
print(df_google_play_store['Genres'].unique())
```

```
['Art & Design' 'Art & Design;Pretend Play' 'Art & Design;Creativity'
 'Art & Design;Action & Adventure' 'Auto & Vehicles' 'Beauty'
 'Books & Reference' 'Business' 'Comics' 'Comics;Creativity'
 'Communication' 'Dating' 'Education;Education' 'Education'
 'Education;Creativity' 'Education;Music & Video'
 'Education;Action & Adventure' 'Education;Pretend Play'
 'Education;Brain Games' 'Entertainment' 'Entertainment;Music & Video'
 'Entertainment;Brain Games' 'Entertainment;Creativity' 'Events' 'Finance'
 'Food & Drink' 'Health & Fitness' 'House & Home' 'Libraries & Demo'
 'Lifestyle' 'Lifestyle;Pretend Play' 'Adventure;Action & Adventure'
 'Arcade' 'Casual' 'Card' 'Casual;Pretend Play' 'Action' 'Strategy'
 'Puzzle' 'Sports' 'Music' 'Word' 'Racing' 'Casual;Creativity'
 'Casual;Action & Adventure' 'Simulation' 'Adventure' 'Board' 'Trivia'
 'Role Playing' 'Simulation;Education' 'Action;Action & Adventure'
 'Casual;Brain Games' 'Simulation;Action & Adventure'
 'Educational;Creativity' 'Puzzle;Brain Games' 'Educational;Education'
 'Card;Brain Games' 'Educational;Brain Games' 'Educational;Pretend Play'
 'Entertainment;Education' 'Casual;Education' 'Music;Music & Video'
 'Racing;Action & Adventure' 'Arcade;Pretend Play'
 'Role Playing;Action & Adventure' 'Simulation;Pretend Play'
 'Puzzle;Creativity' 'Sports;Action & Adventure'
 'Educational;Action & Adventure' 'Arcade;Action & Adventure'
 'Entertainment;Action & Adventure' 'Puzzle;Action & Adventure'
 'Strategy;Action & Adventure' 'Music & Audio;Music & Video'
 'Health & Fitness;Education' 'Adventure;Education' 'Board;Brain Games'
 'Board;Action & Adventure' 'Board;Pretend Play' 'Casual;Music & Video'
 'Role Playing;Pretend Play' 'Entertainment;Pretend Play'
 'Video Players & Editors;Creativity' 'Card;Action & Adventure' 'Medical'
 'Social' 'Shopping' 'Photography' 'Travel & Local'
 'Travel & Local;Action & Adventure' 'Tools' 'Tools;Education'
 'Personalization' 'Productivity' 'Parenting' 'Parenting;Music & Video'
 'Parenting;Education' 'Parenting;Brain Games' 'Weather'
 'Video Players & Editors' 'Video Players & Editors;Music & Video'
 'News & Magazines' 'Maps & Navigation'
 'Health & Fitness;Action & Adventure' 'Educational' 'Casino'
 'Adventure;Brain Games' 'Trivia;Education' 'Lifestyle;Education'
 'Books & Reference;Creativity' 'Books & Reference;Education'
 'Puzzle;Education' 'Role Playing;Education' 'Role Playing;Brain Games'
 'Strategy;Education' 'Racing;Pretend Play' 'Communication;Creativity'
 'February 11, 2018' 'Strategy;Creativity']
```

In [18]:
```
df_google_play_store.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
```

Step 2: Explore the datasets

In this step, you should explore what is present in the data and how the data is organized. You'll need to determine what common features allow you to merge the datasets.

You are expected to answer the following questions using the *pandas* library and markdown cells to describe your actions:

- Are there quality issues in the dataset (noisy, missing data, etc.)?
- What will you need to do to clean and/or transform the raw data for analysis?

You are also expected to use the *matplotlib* library to visually explore the datasets and explain your findings, specifically,

- How are the data distributed?
- What are some common trends?
- What are the relationships between variables in your datasets?

Data Quality Issues and Data Cleaning

1. Identify data quality issues in googleplaystore_user_reviews and do data cleaning.

In [19]:
```
# Find column with most missing values
for col in df_google_play_store_user_reviews:
    print(col, len(df_google_play_store_user_reviews[df_google_play_store_user_reviews[col].isnull()]))

# Drop unused columns
df_google_play_store_user_reviews = df_google_play_store_user_reviews.drop(["Translated_Review","Sentiment"]

# Drop any rows with NaN
df_google_play_store_user_reviews = df_google_play_store_user_reviews.dropna(axis = 0)

# Drop duplicated rows
df_google_play_store_user_reviews = df_google_play_store_user_reviews.drop_duplicates()

df_google_play_store_user_reviews
```

```
App 0
Translated_Review 26868
Sentiment 26863
Sentiment_Polarity 26863
Sentiment_Subjectivity 26863
```

Out[19]:

	App	Sentiment_Polarity	Sentiment_Subjectivity
0	10 Best Foods for You	1.000000	0.533333
1	10 Best Foods for You	0.250000	0.288462
3	10 Best Foods for You	0.400000	0.875000
4	10 Best Foods for You	1.000000	0.300000
6	10 Best Foods for You	0.600000	0.900000
...	...	...	...
64222	Housing-Real Estate & Property	0.173333	0.486667
64223	Housing-Real Estate & Property	0.225000	0.447222
64226	Housing-Real Estate & Property	-0.287500	0.250000
64227	Housing-Real Estate & Property	0.800000	1.000000
64230	Housing-Real Estate & Property	-0.316667	0.400000

25077 rows × 3 columns

In [20]:
```
# Reorganize df_google_play_store_user_reviews with unique list of App names.
df_new_google_play_store_user_reviews = pd.DataFrame(columns = ["App_name", "Sentiment_Polarity", "Sentiment

label_list = df_google_play_store_user_reviews["App"].to_list()
unique_list = df_google_play_store_user_reviews["App"].unique()

for i in range(0,len(unique_list)):
    num = 0
    sum_Sentiment_Polarity = 0
    sum_Sentiment_Subjectivity = 0
    for j in range(0, len(label_list)):
        if label_list[j] == unique_list[i]:
            num += 1
            sum_Sentiment_Polarity += df_google_play_store_user_reviews["Sentiment_Polarity"].to_list()[j]
            sum_Sentiment_Subjectivity += df_google_play_store_user_reviews["Sentiment_Subjectivity"].to_lis
    df_new_google_play_store_user_reviews = df_new_google_play_store_user_reviews.append({'App_name' : uniqu

df_new_google_play_store_user_reviews = df_new_google_play_store_user_reviews.drop_duplicates("App_name", ke

print("Shape of the cleaned google play store user reviews data:\n", df_new_google_play_store_user_reviews.s
print("\nHead of the cleaned google play store reviews data:\n", df_new_google_play_store_user_reviews.head(
```

```
Shape of the cleaned google play store user reviews data:
 (865, 3)

Head of the cleaned google play store reviews data:
                                App_name  Sentiment_Polarity  \
0                 10 Best Foods for You            0.469704
1  104 找工作 - 找工作 找打工 找兼職 履歷健檢 履歷診療室            0.410952
2                                  11st            0.156958
3            1800 Contacts - Lens Store            0.325910
4      1LINE – One Line with One Touch            0.224449

   Sentiment_Subjectivity
0                0.559517
1                0.557805
2                0.512354
3                0.603456
4                0.604413
```

1. Identify data quality issues in googleplaystore and do data cleaning.

In [21]:
```
# Find column with most missing values
for col in df_google_play_store:
    print(col, len(df_google_play_store[df_google_play_store[col].isnull()]))

# Drop duplicated rows
df_google_play_store = df_google_play_store.drop_duplicates("App", keep='last')

# Drop any rows with NaN
df_google_play_store = df_google_play_store.dropna(axis=0)

# Remove "+" and "," from the Installs attribute
df_google_play_store.Installs = df_google_play_store.Installs.str.replace('+','')
df_google_play_store.Installs = df_google_play_store.Installs.str.replace(',','')

# Convert Reviews and Installs to number.
df_google_play_store["Reviews"] = df_google_play_store["Reviews"].apply(pd.to_numeric, downcast='float', err
df_google_play_store["Installs"] = df_google_play_store["Installs"].apply(pd.to_numeric, downcast='float', e

# Column naming (convert column nemw to Snake_case)
df_google_play_store["Content_Rating"] = df_google_play_store["Content Rating"]
df_google_play_store["Last_Updated"] = df_google_play_store["Last Updated"]
df_google_play_store["Current_Ver"] = df_google_play_store["Current Ver"]
df_google_play_store["Android_Ver"] = df_google_play_store["Android Ver"]

print("\nShape of the cleaned google play store data:\n", df_google_play_store.shape)
print("\nHead of the cleaned google play store data:\n", df_google_play_store.head())
```

```
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

Shape of the cleaned google play store data:
 (8190, 17)

Head of the cleaned google play store data:
                                                  App        Category  Rating  \
0     Photo Editor & Candy Camera & Grid & ScrapBook  ART_AND_DESIGN     4.1
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
5                         Paper flowers instructions  ART_AND_DESIGN     4.4

    Reviews  Size    Installs  Type Price Content Rating  \
0     159.0   19M     10000.0  Free     0       Everyone
2   87510.0  8.7M   5000000.0  Free     0       Everyone
3  215644.0   25M  50000000.0  Free     0           Teen
4     967.0  2.8M    100000.0  Free     0       Everyone
5     167.0  5.6M     50000.0  Free     0       Everyone

                    Genres     Last Updated         Current Ver   Android Ver  \
0             Art & Design  January 7, 2018               1.0.0  4.0.3 and up
2             Art & Design   August 1, 2018               1.2.4  4.0.3 and up
3             Art & Design     June 8, 2018  Varies with device    4.2 and up
4  Art & Design;Creativity    June 20, 2018                 1.1    4.4 and up
5             Art & Design   March 26, 2017                 1.0    2.3 and up

  Content_Rating     Last_Updated         Current_Ver   Android_Ver
0       Everyone  January 7, 2018               1.0.0  4.0.3 and up
2       Everyone   August 1, 2018               1.2.4  4.0.3 and up
3           Teen     June 8, 2018  Varies with device    4.2 and up
4       Everyone    June 20, 2018                 1.1    4.4 and up
5       Everyone   March 26, 2017                 1.0    2.3 and up
```

In [22]:
```
# Verify Rating attribute in the correct range
df_google_play_store.Rating.describe()
```

Out[22]:
```
count    8190.000000
mean        4.173297
std         0.536335
min         1.000000
25%         4.000000
50%         4.300000
75%         4.500000
max         5.000000
Name: Rating, dtype: float64
```

Conclusions:

Through exploring the data, there are several data qualities issues that needs to be cleaned.

1. Missing values (drop)
2. Duplicated rows (drop)
3. Column naming (convert to Snake_case)
4. Data type (convert String to Numeric/int, Numeric to float)
5. Unused columns (drop)
6. Incorrect charactors (replace or drop)

The following three data quality issues are specifically for performing NLP analysis. They are not used here but used in Perform Analysis section.

1. Non-letter characters in Translated_Reviews (replace with space)
2. Upper case (convert to lower case)
3. Irrelevant words (remove)

Common Features

Based on the above dataframes, the common feature for these two dataframes is App (name).

In [23]:
```
# Merge these two dataframes together.

df_merge = df_new_google_play_store_user_reviews.merge(df_google_play_store,
                how = "inner",
                left_on = "App_name",
                right_on = "App")
df_merge = df_merge.drop(["App", "Last Updated", "Current Ver", "Android Ver", "Content Rating"],1)
df_merge = df_merge.drop_duplicates("App_name", keep = 'last')
df_merge = df_merge.dropna(axis = 0)

print(df_merge.shape)
print(df_merge.head())
```

```
(816, 15)
                                          App_name  Sentiment_Polarity  \
0                            10 Best Foods for You            0.469704
1                                             11st            0.156958
2                       1800 Contacts - Lens Store            0.325910
3                  1LINE – One Line with One Touch            0.224449
4  2018Emoji Keyboard 😂 Emoticons Lite -sticker&gif            0.416806

   Sentiment_Subjectivity            Category  Rating   Reviews  \
0                0.559517  HEALTH_AND_FITNESS     4.0    2490.0
1                0.512354            SHOPPING     3.8   48732.0
2                0.603456             MEDICAL     4.7   23160.0
3                0.604413                GAME     4.6  214878.0
4                0.550417     PERSONALIZATION     4.2  115773.0

                 Size    Installs  Type Price            Genres  \
0                3.8M    500000.0  Free     0  Health & Fitness
1                 20M  10000000.0  Free     0          Shopping
2                 26M   1000000.0  Free     0           Medical
3                 10M  10000000.0  Free     0             Board
4  Varies with device  10000000.0  Free     0   Personalization

  Content_Rating       Last_Updated         Current_Ver   Android_Ver
0   Everyone 10+  February 17, 2017                 1.9  2.3.3 and up
1       Everyone      July 31, 2018               7.8.1    4.0 and up
2       Everyone      July 27, 2018               7.4.1    5.0 and up
3       Everyone     August 3, 2018               2.0.4    4.1 and up
4       Everyone       May 22, 2018  Varies with device    4.1 and up
```

Conclusions:

The common feature is that both datasets have App(name) column, which can be used to merge together for further analysis.

Data Distribution

Highly skewed distribution: If the skewness value is less than −1 or greater than +1.

Moderately skewed distribution: If the skewness value is between −1 and −½ or between +½ and +1.

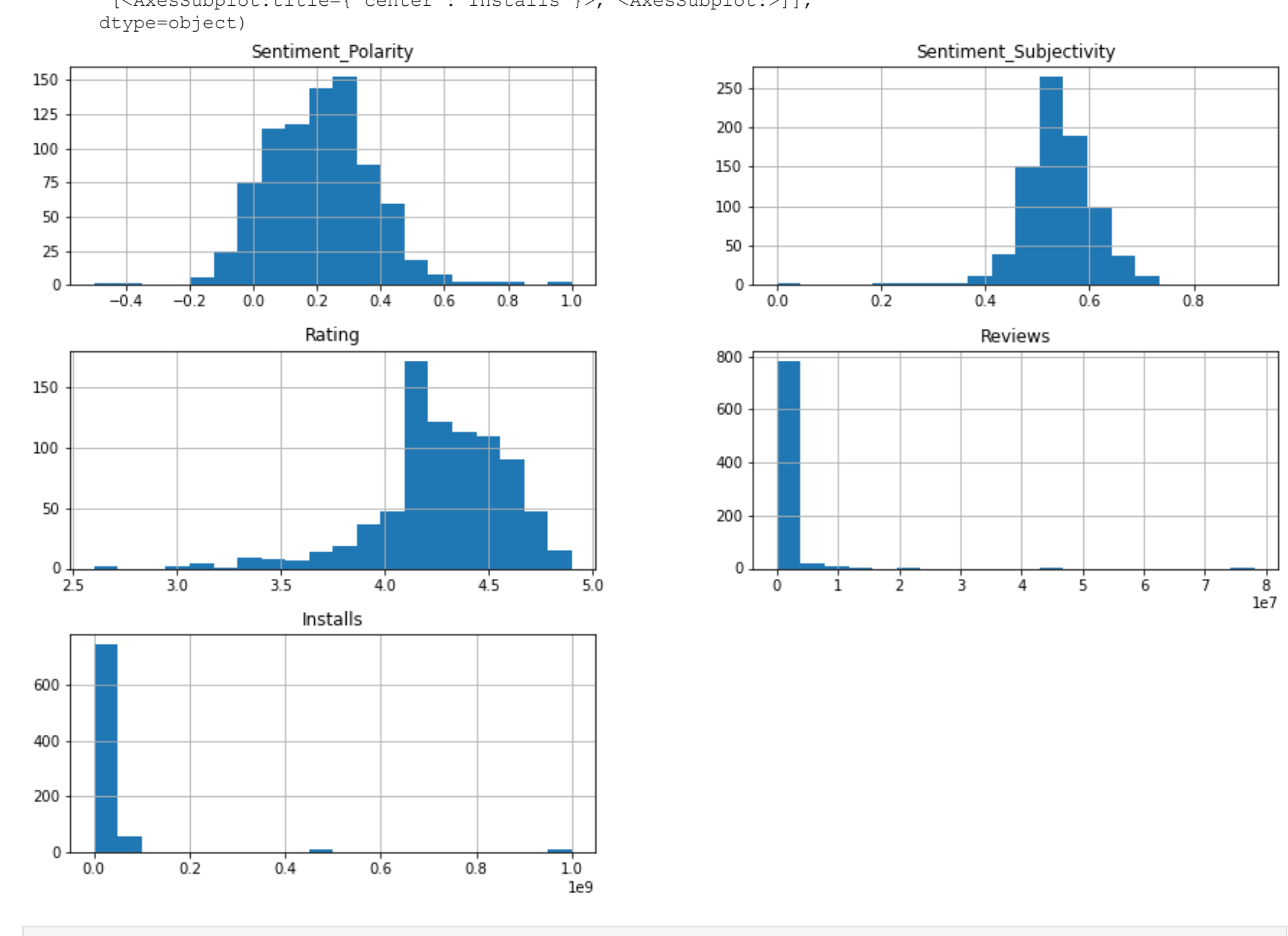
Approximately symmetric distribution: If the skewness value is between −½ and +½.

In [24]:
```
print(df_merge.skew())
```

```
Sentiment_Polarity         0.273335
Sentiment_Subjectivity    -1.647470
Rating                    -1.254675
Reviews                   14.915484
Installs                   7.527610
dtype: float64
```

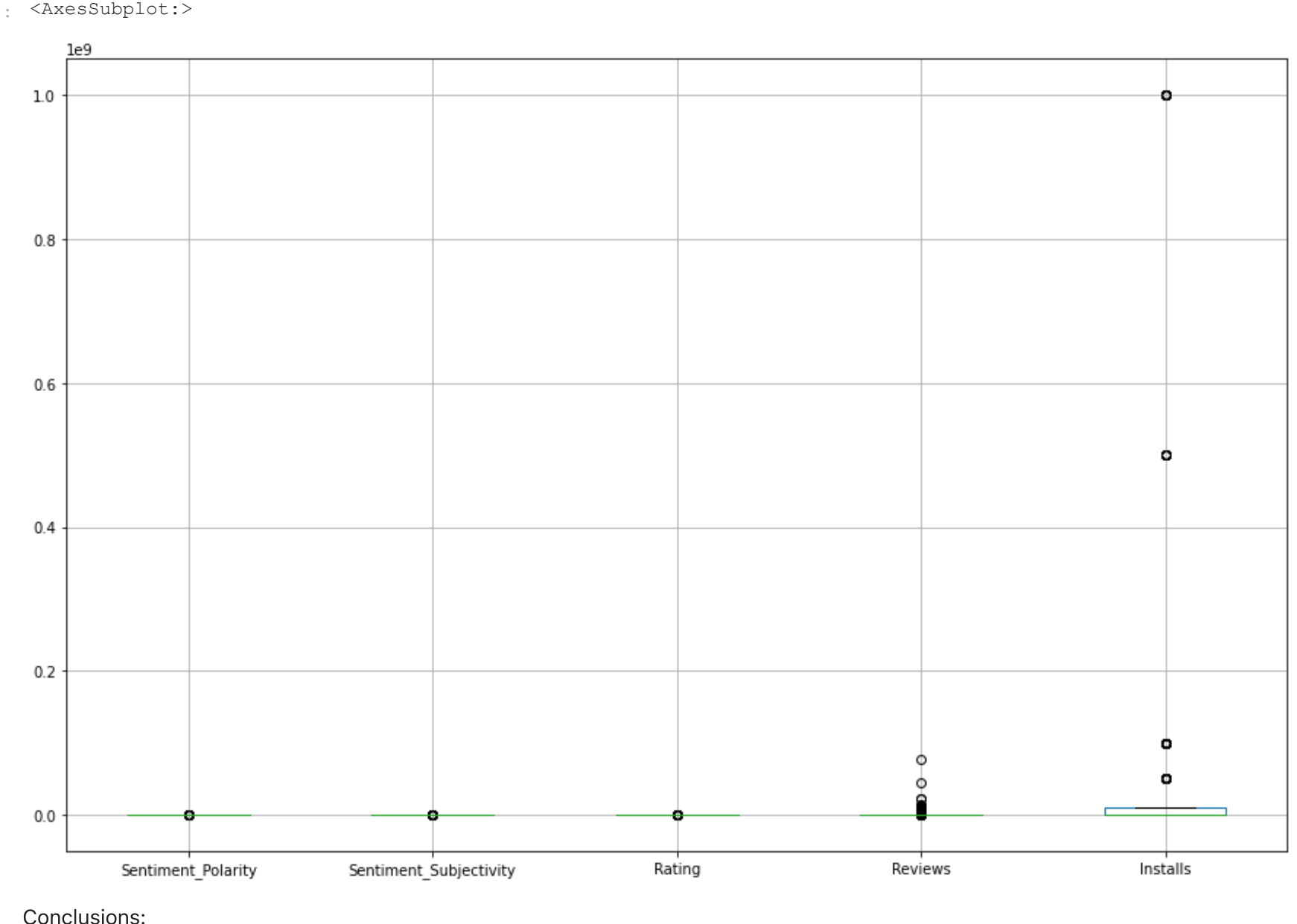
In [25]:
```
df_merge.hist(figsize = (15,10), bins = 20)
```

Out[25]:
```
array([[<AxesSubplot:title={'center':'Sentiment_Polarity'}>,
        <AxesSubplot:title={'center':'Sentiment_Subjectivity'}>],
       [<AxesSubplot:title={'center':'Rating'}>,
        <AxesSubplot:title={'center':'Reviews'}>],
       [<AxesSubplot:title={'center':'Installs'}>, <AxesSubplot:>]],
      dtype=object)
```



In [26]:
```
df_merge.boxplot(figsize = (15,10))
```

Out[26]:
```
<AxesSubplot:>
```



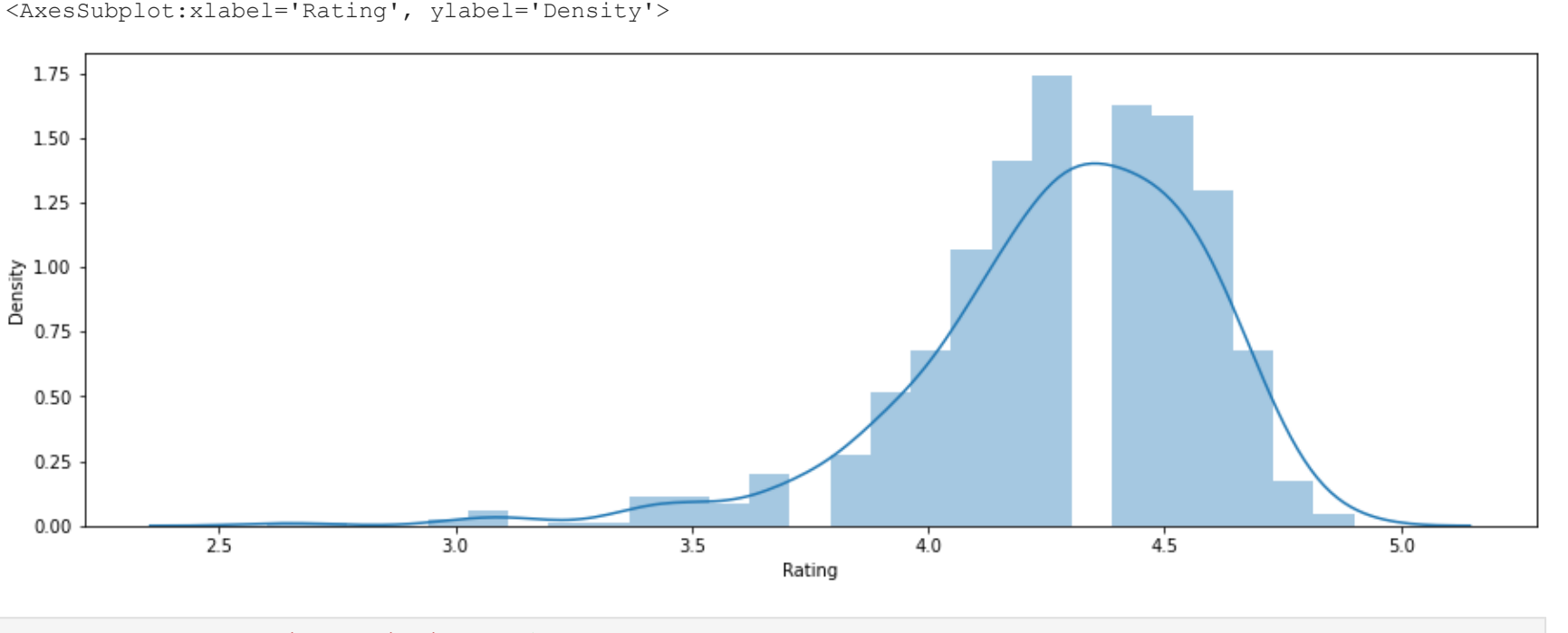
Conclusions:

1. Highly skewed distribution: Sentiment_Subjectivity, Rating, Reviews, Installs.
2. Approximately symmetric distribution: Sentiment_Polarity.
3. Positive skewness values such as Sentiment_Polarity, Reviews, and Installs indicate asymmetry in the distribution and the tail is larger towards the right hand side of the distribution.
4. Negative skewness values such as Sentiment_Subjectivity and Rating indicate asymmetry in the distribution and the tail is larger towards the left hand side of the distribution.
5. Both Installs and Reviews attributes have lots of outliers.

Common Trends

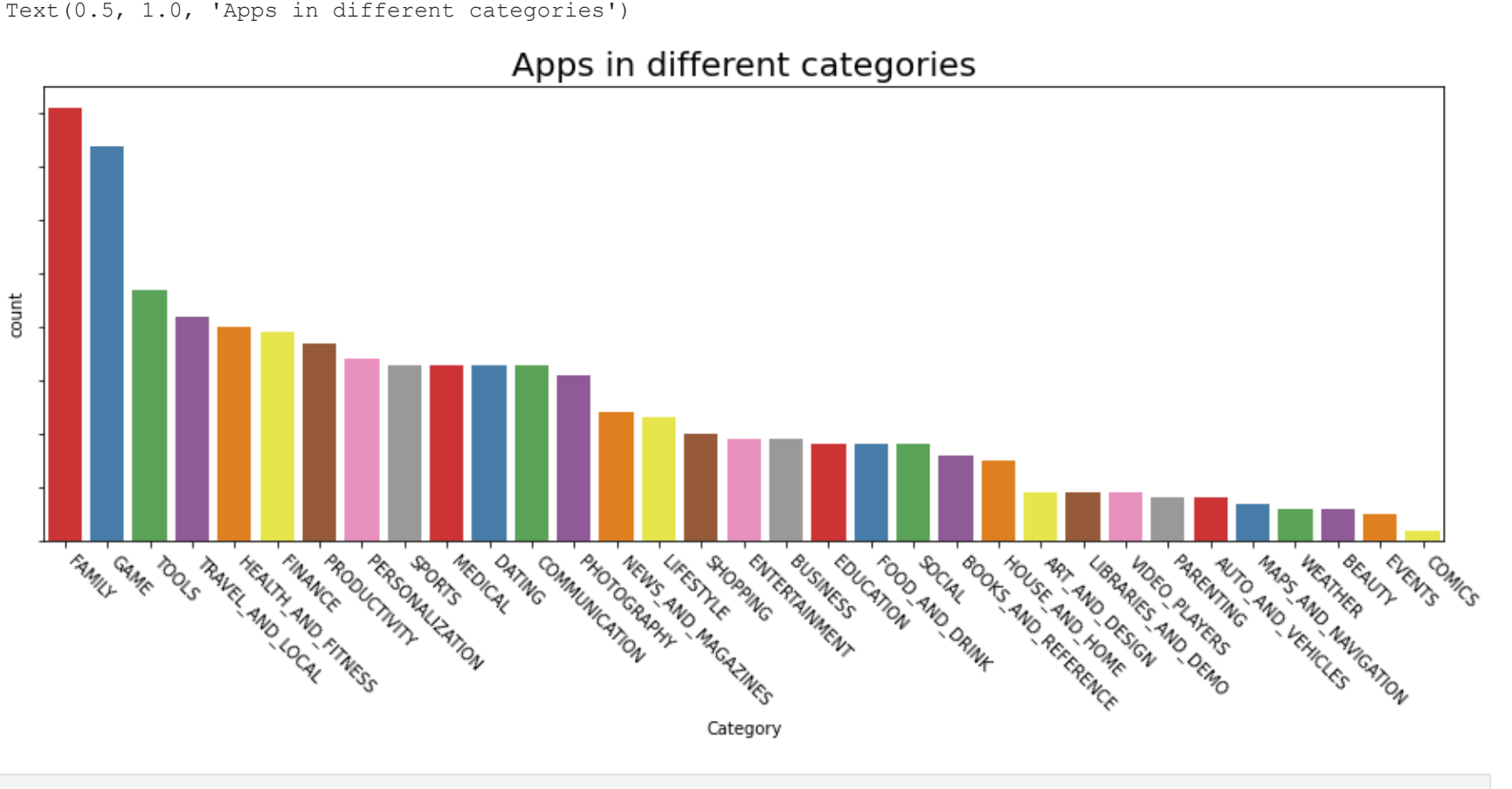
In [27]:
```
plt.figure(figsize = (15,5))
sns.distplot(df_merge["Rating"])
```

Out[27]: <AxesSubplot:xlabel='Rating', ylabel='Density'>



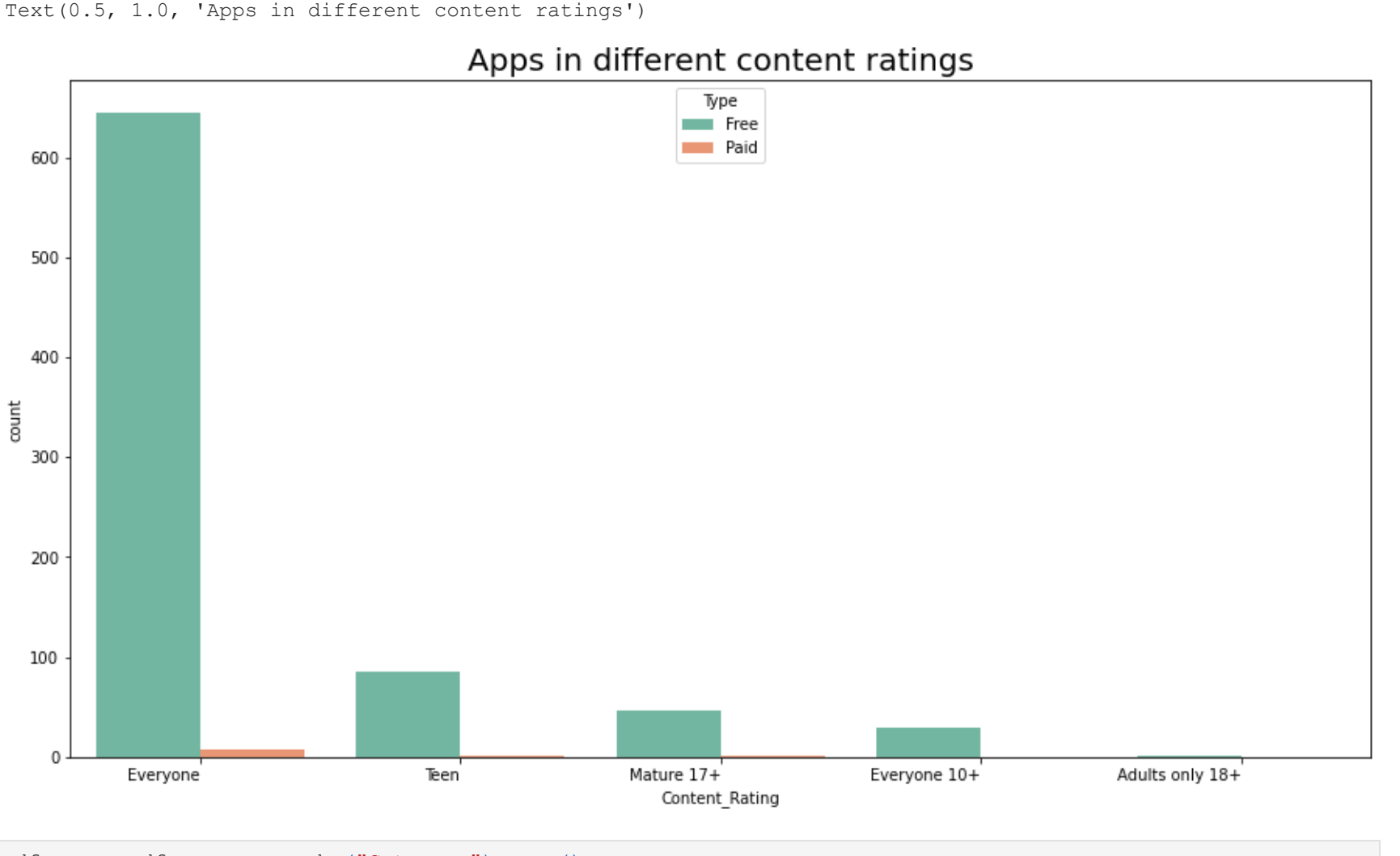
In [28]:
```
sns.set_context({"figure.figsize": (15, 5)})
cp = sns.countplot(x = "Category",data = df_merge, palette = "Set1",order = df_merge['Category'].value_count
cp.set_xticklabels(cp.get_xticklabels(), rotation = -45, ha = "left")
cp.set_yticklabels(cp.get_yticklabels(), rotation = 0, ha = "right")
plt.title('Apps in different categories',size = 20)
```

Out[28]: Text(0.5, 1.0, 'Apps in different categories')



In [29]:
```
sns.set_context({"figure.figsize": (15, 8)})
ctpt = sns.countplot(x = "Content_Rating",hue = 'Type',data = df_merge, palette = "Set2",order = df_merge['C
ctpt.set_xticklabels(ctpt.get_xticklabels(), rotation = 0, ha = "right")
plt.title('Apps in different content ratings',size = 20)
```

Out[29]: Text(0.5, 1.0, 'Apps in different content ratings')



In [30]:
```
df_mean = df_merge.groupby("Category").mean()
max_rating = df_mean['Rating'].idxmax()
min_rating = df_mean['Rating'].idxmin()
mean_rating_max = df_mean.loc[df_mean['Rating'].idxmax()]
mean_rating_min = df_mean.loc[df_mean['Rating'].idxmin()]

print("Max rating category:", max_rating, "\nMean rating:", mean_rating_max['Rating'])
print("\nMin rating category:", min_rating, "\nMean rating:", mean_rating_min['Rating'])
```

```
Max rating category: AUTO_AND_VEHICLES
Mean rating: 4.575

Min rating category: ENTERTAINMENT
Mean rating: 4.015789473684211
```

Conclusions:

1. Most of the Ratings are in the range of 4.0 to 4.7.
2. Most of the Apps belong to the category of Family.
3. Category of Comics has the least Apps.
4. Most of the Apps belongs to the content rating of Everyone.
5. Most of the Apps belongs to the type of Free.
6. AUTO_AND_VEHICLES category has the max Rating.
7. ENTERTAINMENT category has the min Rating.

Variable Relationships

Perfect: If the value is near ± 1, then it said to be a perfect correlation: as one variable increases, the other variable tends to also increase (if positive) or decrease (if negative).

High degree: If the coefficient value lies between ± 0.50 and ± 1, then it is said to be a strong correlation.

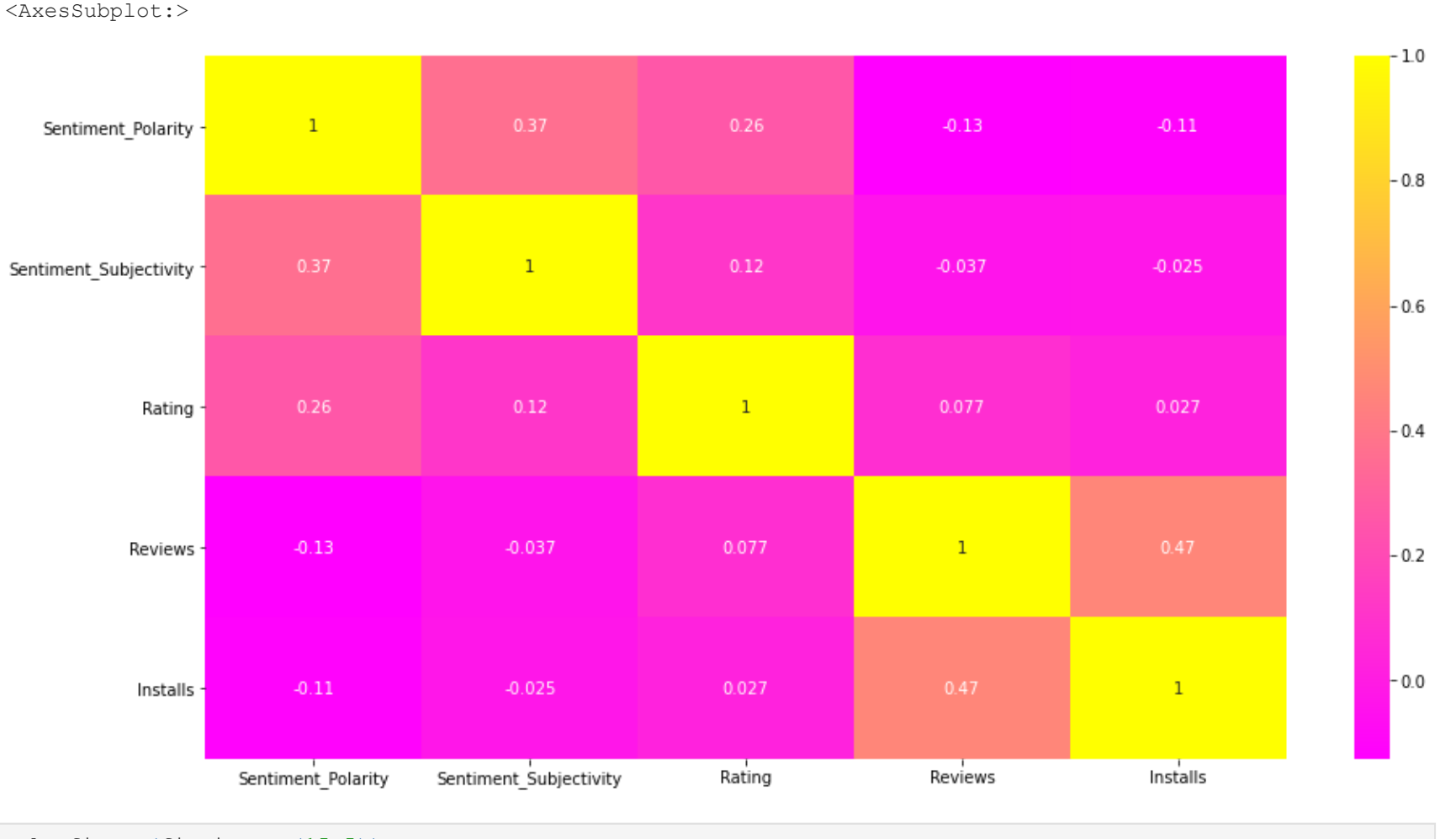
Moderate degree: If the value lies between ± 0.30 and ± 0.49, then it is said to be a medium correlation.

Low degree: When the value lies below + .29, then it is said to be a small correlation.

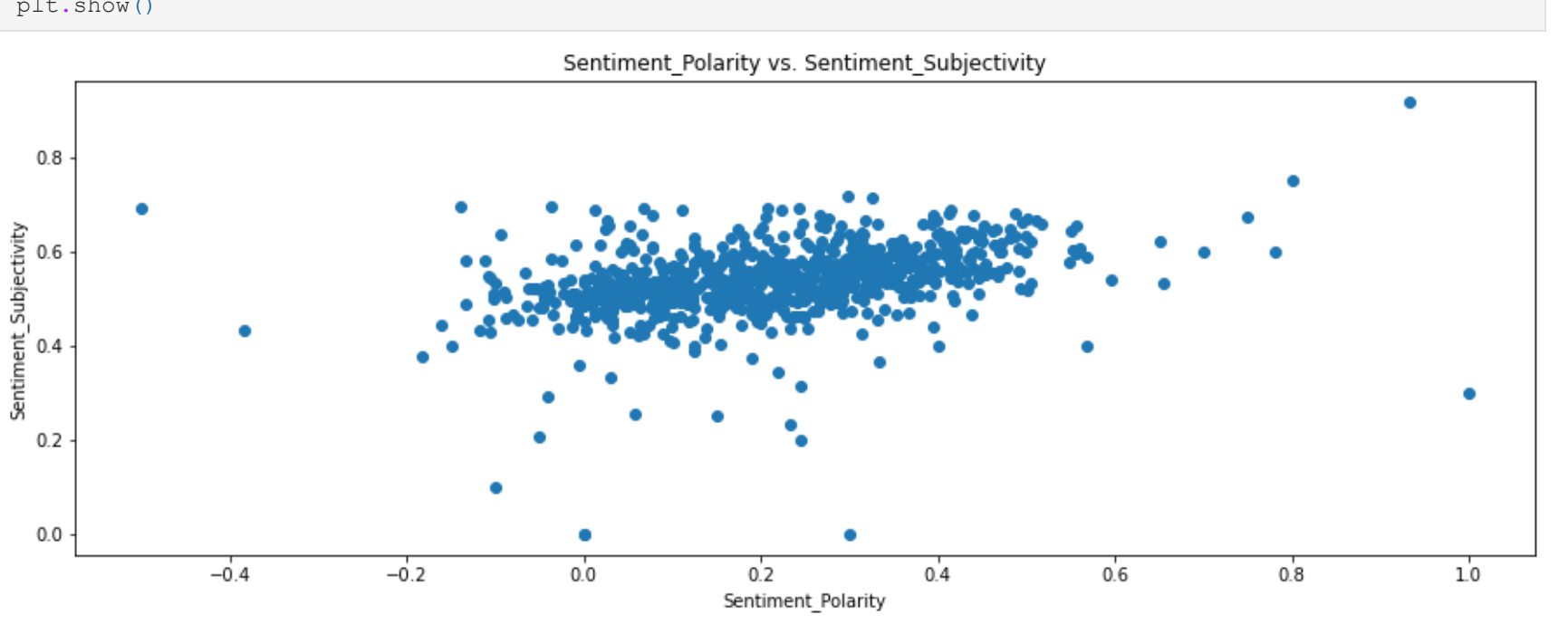
No correlation: When the value is zero.

In [31]:
```
sns.heatmap(df_merge.corr(),annot = True,cmap = "spring")
```

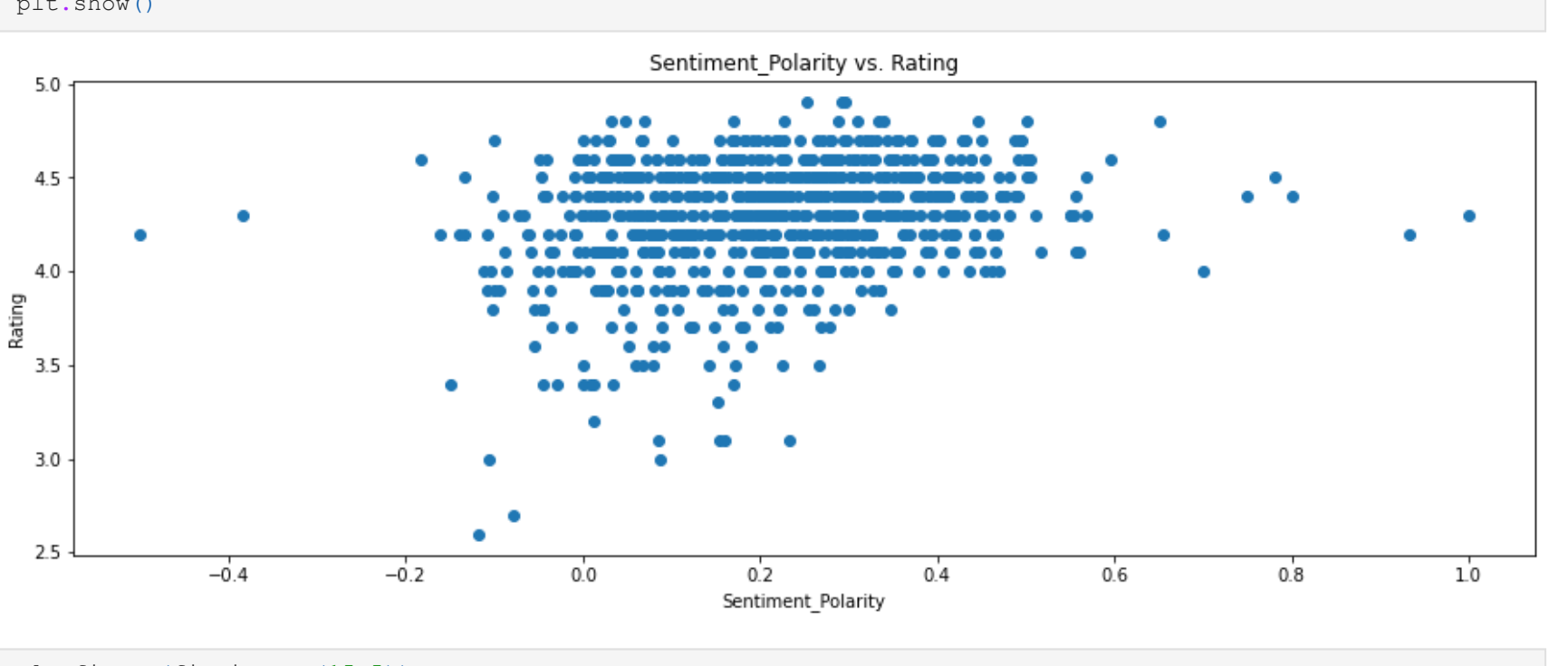
Out[31]: <AxesSubplot:>



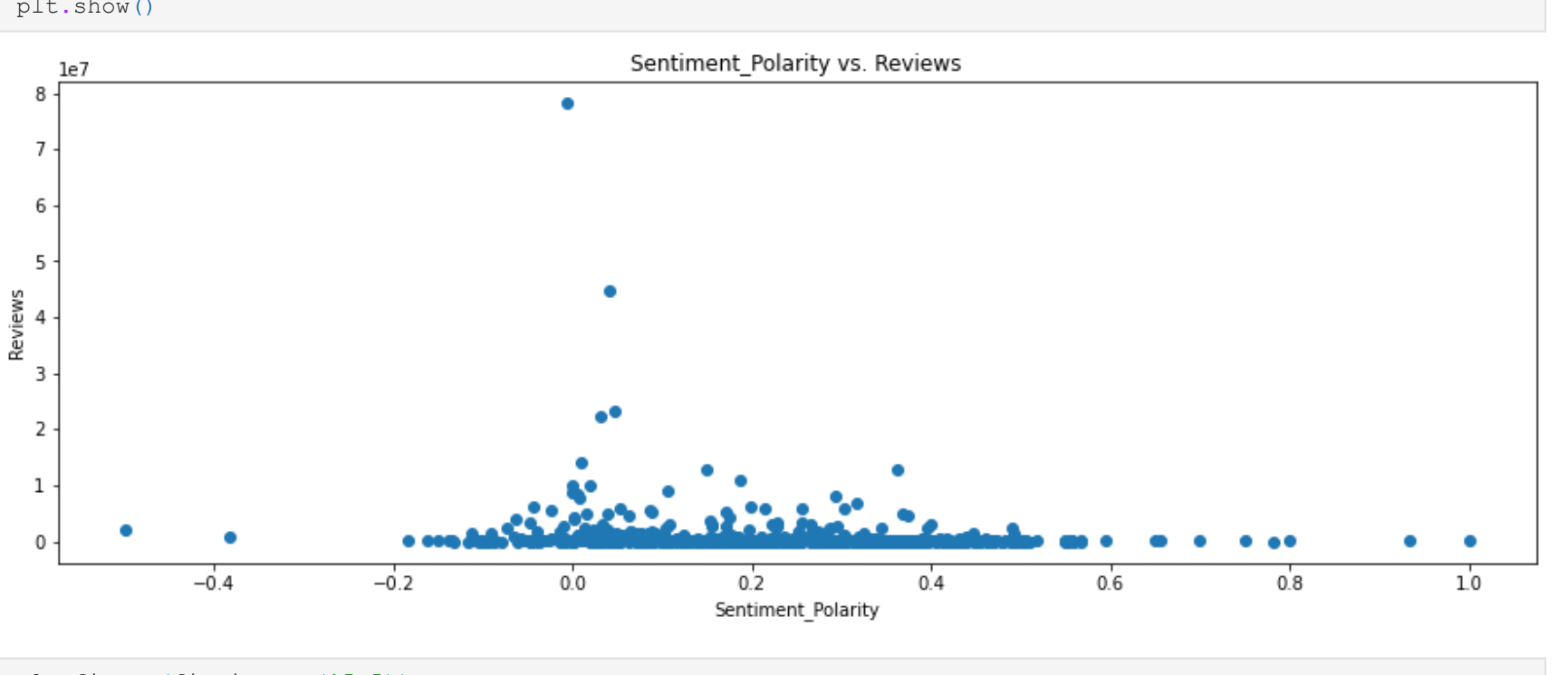
In [32]:
```
plt.figure(figsize = (15,5))
plt.scatter(x = df_merge['Sentiment_Polarity'], y = df_merge['Sentiment_Subjectivity'])
plt.title('Sentiment_Polarity vs. Sentiment_Subjectivity')
plt.xlabel('Sentiment_Polarity')
plt.ylabel('Sentiment_Subjectivity')
plt.show()
```



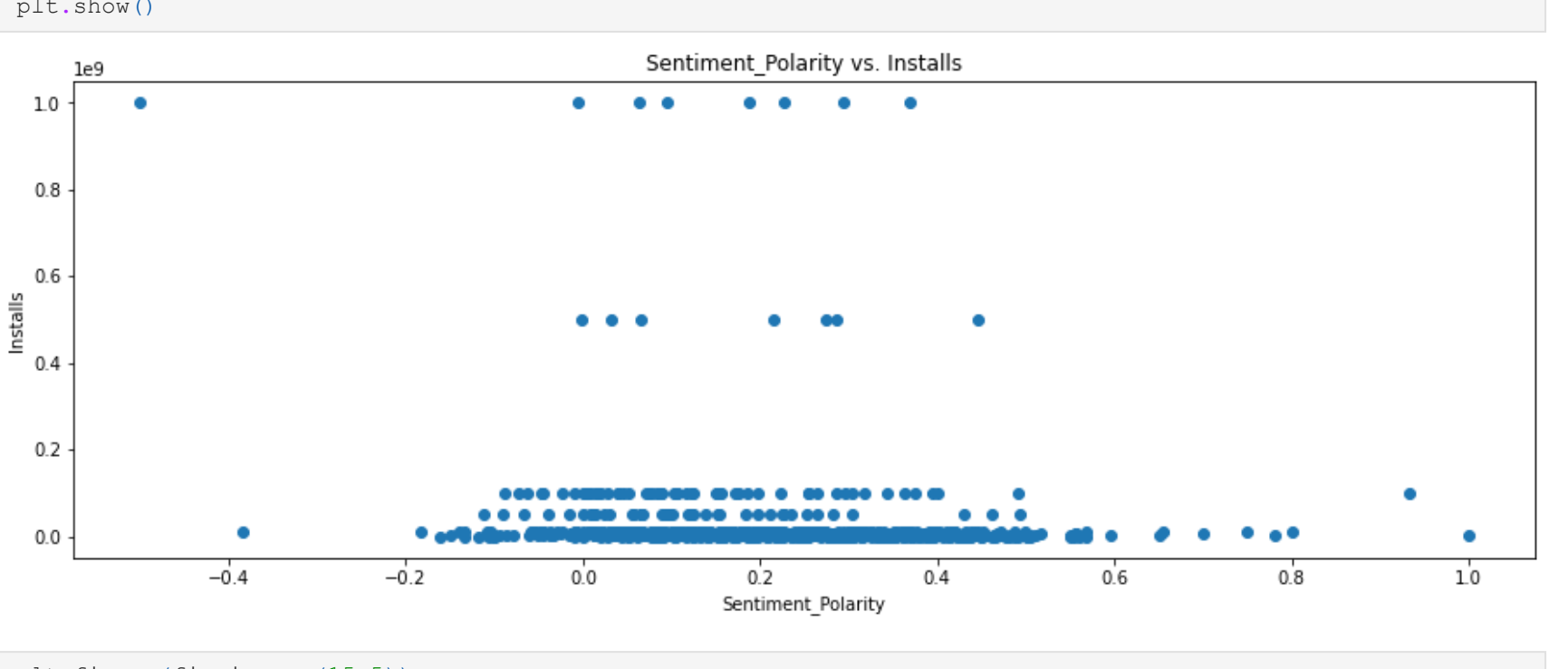
In [33]:
```
plt.figure(figsize = (15,5))
plt.scatter(x = df_merge['Sentiment_Polarity'], y = df_merge['Rating'])
plt.title('Sentiment_Polarity vs. Rating')
plt.xlabel('Sentiment_Polarity')
plt.ylabel('Rating')
plt.show()
```



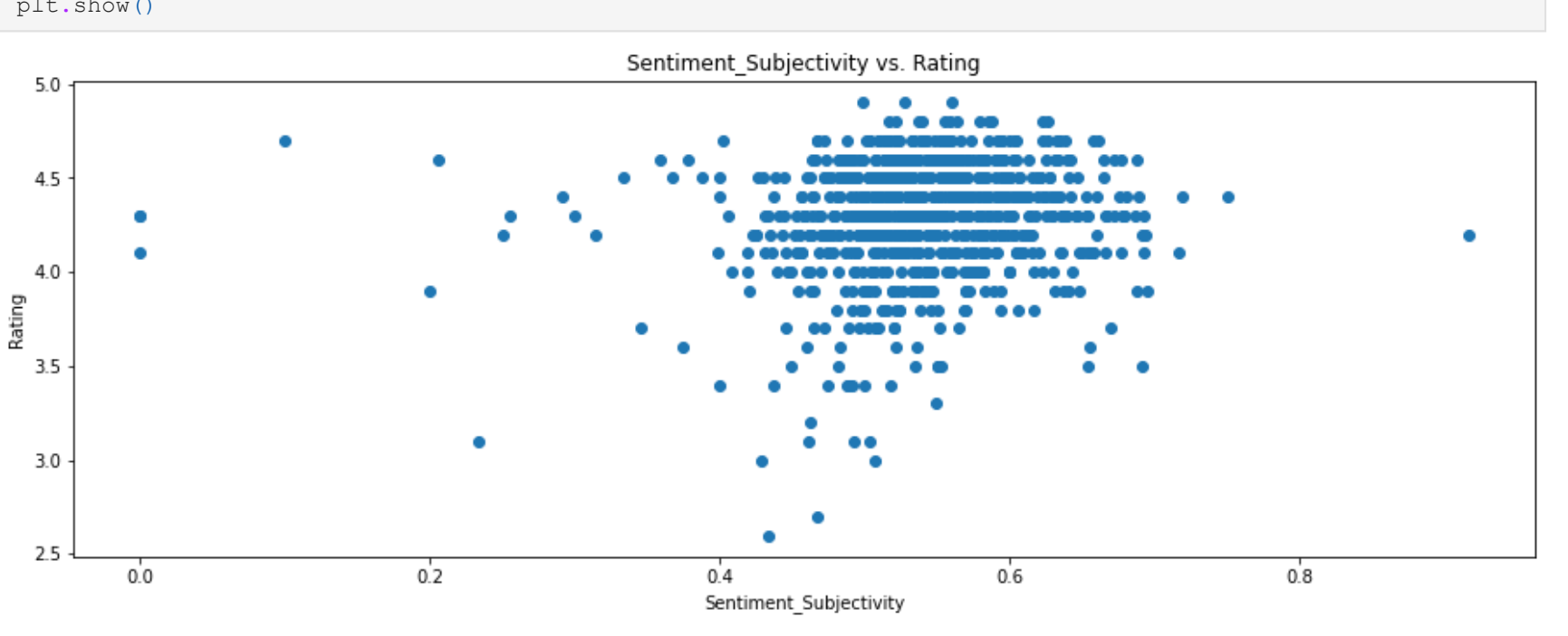
In [34]:
```
plt.figure(figsize = (15,5))
plt.scatter(x = df_merge['Sentiment_Polarity'], y = df_merge['Reviews'])
plt.title('Sentiment_Polarity vs. Reviews')
plt.xlabel('Sentiment_Polarity')
plt.ylabel('Reviews')
plt.show()
```



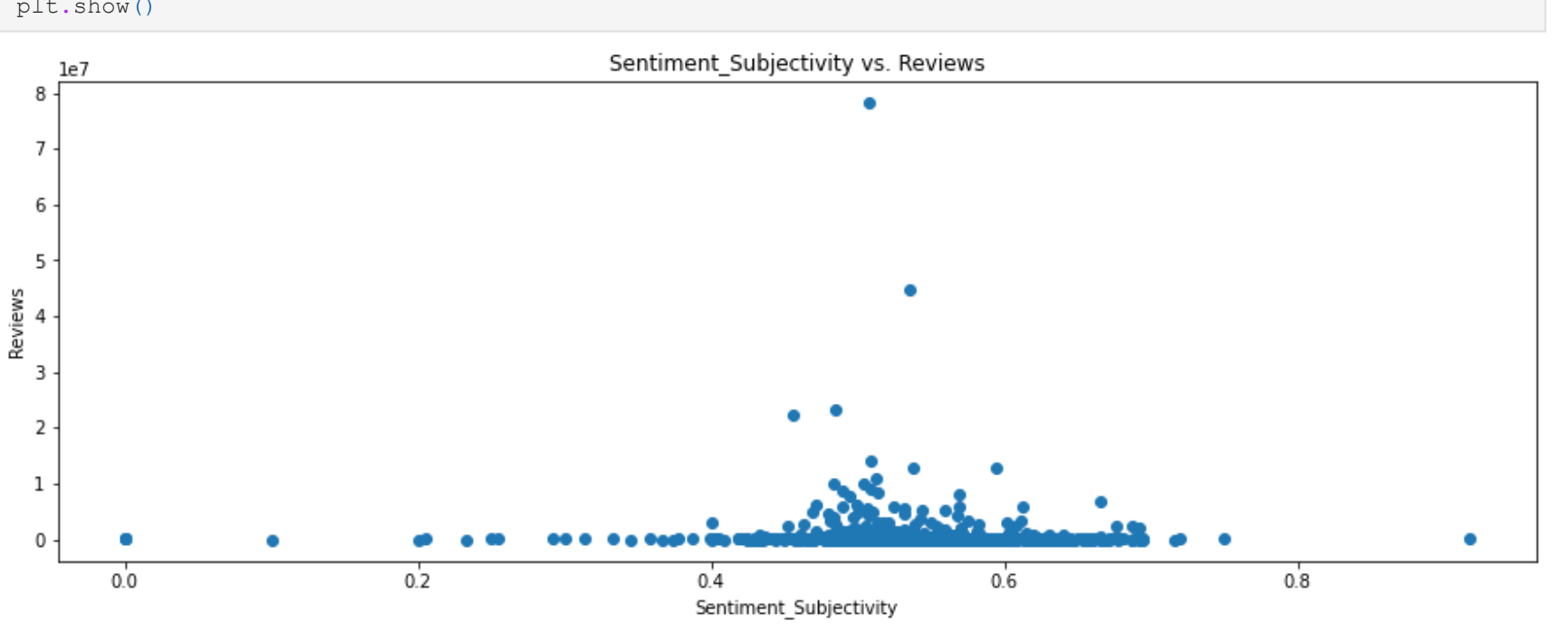
In [35]:
```
plt.figure(figsize = (15,5))
plt.scatter(x = df_merge['Sentiment_Polarity'], y = df_merge['Installs'])
plt.title('Sentiment_Polarity vs. Installs')
plt.xlabel('Sentiment_Polarity')
plt.ylabel('Installs')
plt.show()
```



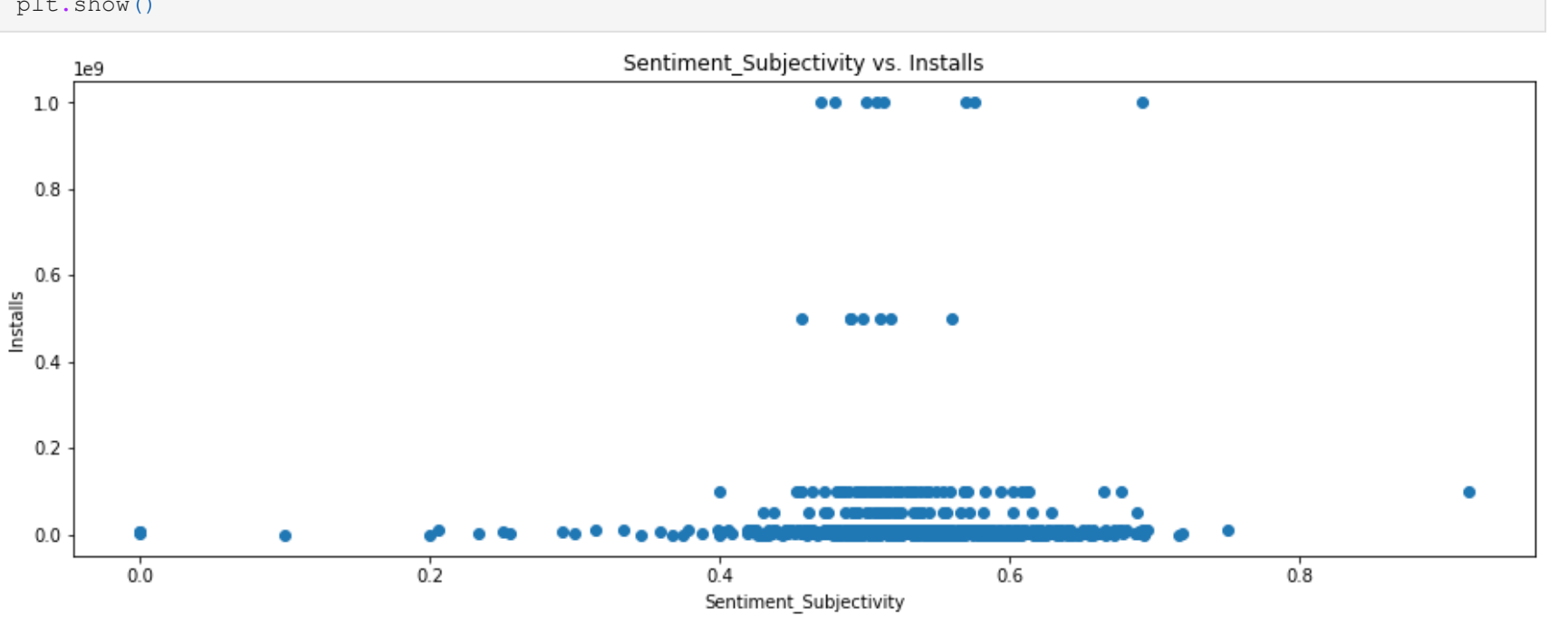
In [36]:
```
plt.figure(figsize = (15,5))
plt.scatter(x = df_merge['Sentiment_Subjectivity'], y = df_merge['Rating'])
plt.title('Sentiment_Subjectivity vs. Rating')
plt.xlabel('Sentiment_Subjectivity')
plt.ylabel('Rating')
plt.show()
```



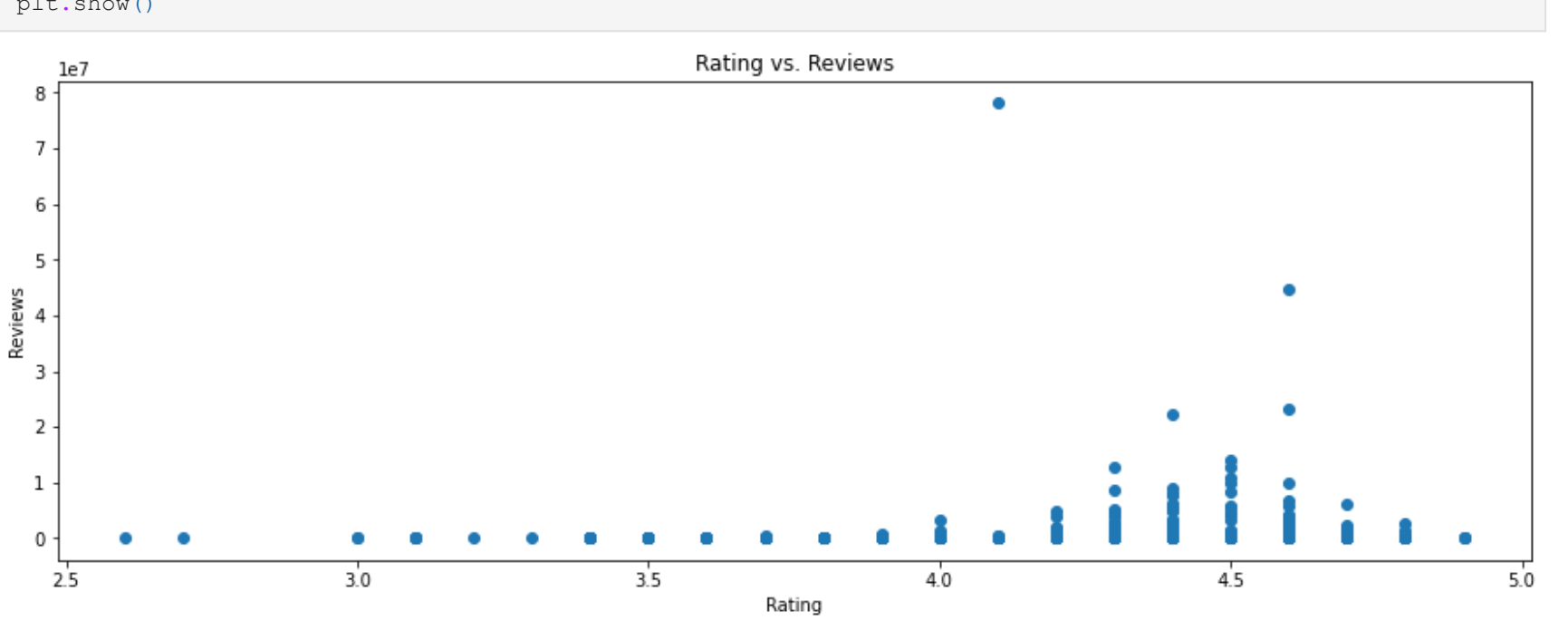
In [37]:
```
plt.figure(figsize = (15,5))
plt.scatter(x = df_merge['Sentiment_Subjectivity'], y = df_merge['Reviews'])
plt.title('Sentiment_Subjectivity vs. Reviews')
plt.xlabel('Sentiment_Subjectivity')
plt.ylabel('Reviews')
plt.show()
```



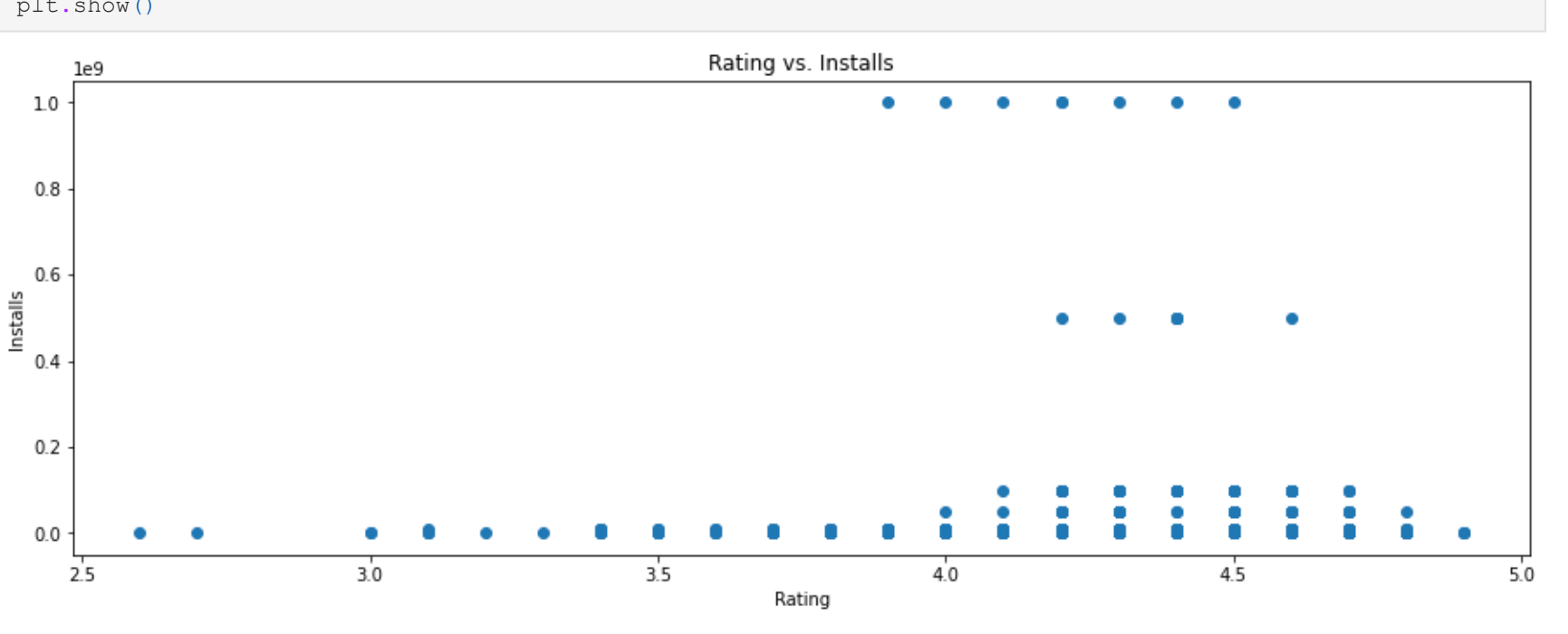
In [38]:
```
plt.figure(figsize = (15,5))
plt.scatter(x = df_merge['Sentiment_Subjectivity'], y = df_merge['Installs'])
plt.title('Sentiment_Subjectivity vs. Installs')
plt.xlabel('Sentiment_Subjectivity')
plt.ylabel('Installs')
plt.show()
```



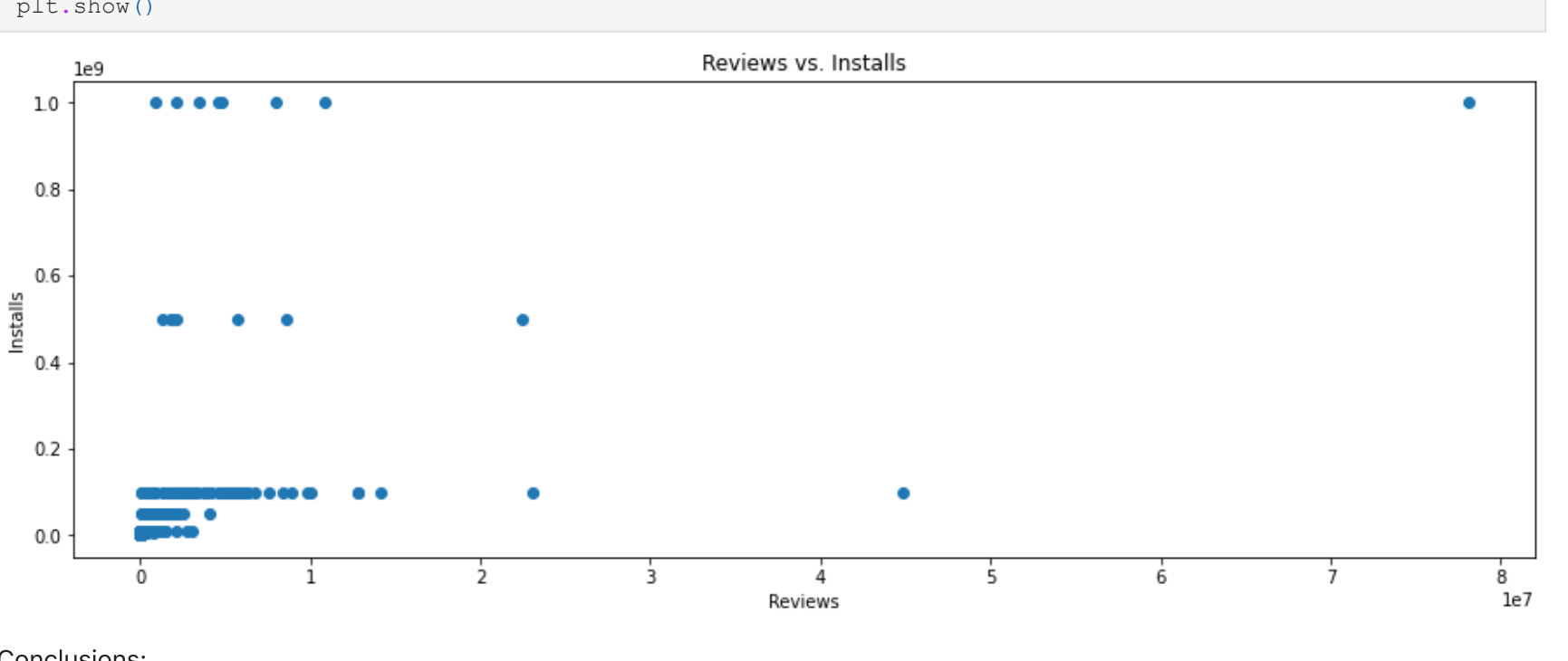
In [39]:
```
plt.figure(figsize = (15,5))
plt.scatter(x = df_merge['Rating'], y = df_merge['Reviews'])
plt.title('Rating vs. Reviews')
plt.xlabel('Rating')
plt.ylabel('Reviews')
plt.show()
```



In [40]:
```
plt.figure(figsize = (15,5))
plt.scatter(x = df_merge['Rating'], y = df_merge['Installs'])
plt.title('Rating vs. Installs')
plt.xlabel('Rating')
plt.ylabel('Installs')
plt.show()
```



In [41]:
```
plt.figure(figsize = (15,5))
plt.scatter(x = df_merge['Reviews'], y = df_merge['Installs'])
plt.title('Reviews vs. Installs')
plt.xlabel('Reviews')
plt.ylabel('Installs')
plt.show()
```



Conclusions:

1. The relationship between Sentiment_Polarity and Sentiment_Subjectivity is positively medium correlation.
2. The relationship between Sentiment_Polarity and Rating is positively small correlation.
3. The relationship between Sentiment_Polarity and Reviews is negatively small correlation.
4. The relationship between Sentiment_Polarity and Installs is negatively small correlation.
5. The relationship between Sentiment_Subjectivity and Rating is positively small correlation.
6. The relationship between Sentiment_Subjectivity and Reviews is negatively small correlation.
7. The relationship between Sentiment_Subjectivity and Installs is negatively small correlation.
8. The relationship between Rating and Reviews is positively small correlation.
9. The relationship between Rating and Installs is positively small correlation.
10. The relationship between Reviews and Installs is positively medium correlation.

Step 3: Identify 1-3 research questions and perform analysis

Now that you have a better understanding of the data, you will want to form a research question which is interesting to you. The research question should be broad enough to be of interest to a reader but narrow enough that the question can be answered with the data. Some examples:

- **Too Narrow:** What is the GDP of the U.S. for 2011? This is just asking for a fact or a single data point.
- **Too Broad:** What is the primary reason for global poverty? This could be a Ph.D. thesis and would still be way too broad. What data will you use to answer this question? Even if a single dataset offered an answer, would it be defendable given the variety of datasets out there?
- **Good:** Can you use simple sentiment analysis on comments about movies in a movie database to predict its box office earnings? If you have, or can obtain, data on a variety of movies and you have their box office earnings, this is a question which you can potentially answer well.

Remember, this course is for learning Python. You will not be graded on the complexity, accuracy or performance of your analytical methods. However, you are expected to use a Python library, e.g., *scikitlearn*, successfully to generate results and explain why you picked the methods you used.

Research Questions

1. Can we predict App Ratings for Apps on the Google Play Store?
2. What categories of Apps that a marketing strategy should targeted on the Google Play Store Apps?
3. Can we use NLP to train the Google Play Store dataset to predict reviews?

Perform Analysis

Question 1

1. Can we predict App Ratings for Apps on the Google Play Store?

In [42]:
```
data = df_merge.copy()
data = data.drop(["App_name", "Category", "Size", "Type", "Price", "Genres", "Content_Rating", "Last_Updated
```

In [43]:
```
features = ['Sentiment_Polarity', 'Sentiment_Subjectivity', 'Reviews', 'Installs']
X = data[features]
y = data['Rating']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 10)
```

Linear Regression

In [44]:
```
# Linear Regression: Fit a model to the training set
regressor = LinearRegression()
regressor.fit(X_train, y_train)

# Perform Prediction using Linear Regression Model
y_prediction = regressor.predict(X_test)
y_prediction
```

Out[44]:
```
array([4.25553971, 4.2286547 , 4.35424716, 4.42727049, 4.2310485 ,
       4.47165403, 4.20162923, 4.22941127, 4.26399847, 4.36486569,
       4.29283582, 4.26043216, 4.35375674, 4.26607465, 4.26183732,
       4.3501017 , 4.26569116, 4.21022499, 4.21005825, 4.35021077,
       4.29058251, 4.29007471, 4.41226582, 4.42758018, 4.27196247,
       4.31872617, 4.12082212, 4.16649117, 4.33784356, 4.19151303,
       4.30394538, 4.25234116, 4.24943659, 4.32597446, 4.2220585 ,
       4.27392197, 4.28084079, 4.28537028, 4.21624821, 4.30617891,
       4.36314113, 4.32125533, 4.08119403, 4.2320141 , 4.35066559,
       4.24557476, 4.23544962, 4.34311737, 4.3551423 , 4.23127484,
       4.14702748, 4.18976119, 4.36730847, 4.1984759 , 4.26095526,
       4.04913432, 4.28661551, 4.31589894, 4.21820463, 4.1020365 ,
       4.45089407, 4.2653815 , 4.20791059, 4.26458727, 4.19584345,
       4.33501264, 4.34918202, 4.29963793, 4.17576956, 4.3185688 ,
       4.26144271, 4.16086531, 4.23012029, 4.23562605, 4.28548397,
       4.4004992 , 4.35054748, 4.20659393, 4.36145847, 4.63218555,
       4.13309353, 4.40727004, 4.14487009, 4.17235882, 4.42831679,
       4.1695952 , 4.22040364, 4.16214687, 4.28540037, 4.3481893 ,
       4.23731881, 4.12157931, 4.50564498, 4.2643636 , 4.42813465,
       4.22314492, 4.37363853, 4.15302733, 4.4124037 , 4.16837848,
       4.3289064 , 4.26002448, 4.37534786, 4.39864578, 4.25981085,
       4.25978715, 4.3125213 , 4.15998625, 4.24763972, 4.22592215,
       4.30673529, 4.35962063, 4.27135825, 4.18086682, 4.26895179,
       4.26664462, 4.14774059, 4.34883314, 4.1462979 , 4.37294756,
       4.2140739 , 4.34431082, 4.31610005, 4.11393964, 4.17993962,
       4.33634793, 4.23946261, 4.22564516, 4.2073505 , 4.22214667,
       4.39884863, 4.14612144, 4.21516356, 4.36494586, 4.37756224,
       4.21713998, 4.3402378 , 4.28675235, 4.26749107, 4.20464702,
       4.27596925, 4.30919371, 4.09260822, 4.3797093 , 4.23623904,
       4.2287755 , 4.19187434, 4.40320139, 4.31741042, 4.21975965,
       4.07195389, 4.47418179, 4.38179984, 4.18871833, 4.25762449,
       4.38054068, 4.33688574, 4.20773796, 4.29672162, 4.35288683,
       4.43336102, 4.26845025, 4.17113241, 4.18391909])
```

In [45]:
```
# The mean of the expected target value in test set
y_test.describe()
```

Out[45]:
```
count    164.000000
mean       4.259756
std        0.328544
min        2.600000
25%        4.100000
50%        4.300000
75%        4.500000
max        4.900000
Name: Rating, dtype: float64
```

In [46]:
```
# Evaluate Linear Regression Accuracy using Root Mean Square Error
RMSE_linear_regression = sqrt(mean_squared_error(y_true = y_test, y_pred = y_prediction))
print(RMSE_linear_regression)
```

```
0.3195341285398724
```

Decision Tree Regression

In [47]:
```
# Decision Tree Regression: Fit a new regression model to the training set
regressor = DecisionTreeRegressor(max_depth=10)
regressor.fit(X_train, y_train)

# Perform Prediction using Decision Tree Regressor
y_prediction = regressor.predict(X_test)
y_prediction
```

Out[47]:
```
array([3.5       , 4.6       , 4.1       , 4.53333333, 4.35      ,
       4.38      , 4.6       , 4.39375   , 4.5       , 4.35263158,
       4.41428571, 4.6       , 4.35263158, 4.53      , 4.5       ,
       4.1       , 4.3       , 4.7       , 4.2       , 4.6       ,
       4.33333333, 4.125     , 4.51333333, 4.53333333, 3.9       ,
       4.22608696, 4.21428571, 4.21428571, 4.35263158, 3.9       ,
       4.6       , 3.9       , 4.5       , 3.7       , 3.5       ,
       4.7       , 4.33333333, 4.        , 4.        , 4.53      ,
       4.6       , 3.8       , 3.4       , 4.41428571, 4.22608696,
       3.9       , 4.3       , 4.7       , 4.22608696, 4.56666667,
       3.6       , 4.5       , 4.35263158, 3.825     , 4.33333333,
       4.1       , 4.41428571, 4.35263158, 4.6       , 4.1       ,
       4.53333333, 4.275     , 3.9       , 4.1       , 3.99090909,
       4.22608696, 4.6       , 4.15      , 3.8       , 4.33333333,
       4.3       , 3.99090909, 4.53333333, 4.4       , 4.5       ,
       4.6       , 4.6       , 4.175     , 4.35263158, 3.1       ,
       4.        , 4.3       , 4.3       , 4.21428571, 4.1       ,
       4.3       , 4.6       , 4.3       , 4.2       , 4.35263158,
       3.875     , 3.9       , 4.38      , 3.5       , 4.51333333,
       4.16      , 4.08      , 4.3       , 4.4       , 3.4       ,
       4.2       , 4.35      , 4.08      , 4.3       , 4.5       ,
       4.5       , 4.15      , 4.2       , 3.9       , 4.2       ,
       4.33333333, 4.08      , 4.175     , 3.9       , 4.33333333,
       4.3       , 4.5       , 4.4       , 4.4       , 4.36666667,
       4.7       , 4.2       , 4.41428571, 4.4       , 4.21428571,
       4.15      , 4.6       , 4.21428571, 4.21428571, 4.53333333,
       4.58      , 4.17777778, 4.3       , 4.51333333, 4.15      ,
       3.1       , 4.6       , 4.425     , 4.3       , 4.56666667,
       4.175     , 3.5       , 3.        , 4.08      , 4.        ,
       4.1       , 4.5       , 4.46666667, 3.8       , 3.1       ,
       3.4       , 4.4       , 4.4       , 3.7       , 4.4       ,
       4.51333333, 4.33333333, 4.6       , 4.36666667, 4.2       ,
       4.3       , 4.275     , 3.99090909, 4.1       ])
```

In [48]:
```
# The mean of the expected target value in test set
y_test.describe()
```

Out[48]:
```
count    164.000000
mean       4.259756
std        0.328544
min        2.600000
25%        4.100000
50%        4.300000
75%        4.500000
max        4.900000
Name: Rating, dtype: float64
```

In [49]:
```
# Evaluate Decision Tree Regression Accuracy using Root Mean Square Error
RMSE_decision_tree = sqrt(mean_squared_error(y_true = y_test, y_pred = y_prediction))
print(RMSE_decision_tree)
```

```
0.38952532064781514
```

Random Forest Regression

In [50]:
```
# Random Forest Regression: Fit a new regression model to the training set
model = RandomForestRegressor(n_estimators = 200, n_jobs = -1, random_state = 10)
model.fit(X_train, y_train)

# Perform Prediction using Random Forest Regressor
y_prediction = model.predict(X_test)
y_prediction
```

Out[50]:
```
array([4.254 , 4.518 , 4.43  , 4.427 , 4.371 , 4.362 , 4.507 , 4.383 ,
       4.4405, 4.3075, 4.383 , 4.463 , 4.376 , 4.4635, 4.4775, 4.604 ,
       4.2665, 4.567 , 3.9445, 4.5375, 4.32  , 4.1065, 4.453 , 4.407 ,
       4.179 , 4.354 , 4.019 , 4.1015, 4.279 , 4.132 , 4.499 , 4.194 ,
       4.4285, 3.956 , 4.1365, 4.5405, 4.261 , 4.3645, 4.1815, 4.4955,
       4.4035, 4.435 , 3.9035, 4.216 , 4.331 , 4.1005, 4.3205, 4.1595,
       4.366 , 4.508 , 4.0065, 4.3755, 4.3095, 4.0245, 4.4565, 4.0625,
       4.1825, 4.245 , 4.4425, 4.0365, 4.4365, 4.2395, 3.457 , 4.149 ,
       4.128 , 4.2825, 4.4535, 4.27  , 4.026 , 4.2255, 4.321 , 4.0365,
       4.486 , 4.2745, 4.3145, 4.523 , 4.504 , 4.198 , 4.3225, 4.33  ,
       4.231 , 4.3285, 4.3345, 4.188 , 4.3335, 3.6865, 4.452 , 4.408 ,
       4.214 , 4.2805, 4.1605, 3.828 , 4.3915, 4.244 , 4.5425, 4.008 ,
       4.2295, 3.4515, 4.444 , 3.782 , 4.3665, 4.4695, 4.297 , 4.415 ,
       4.413 , 4.465 , 4.3185, 4.3495, 3.853 , 4.1145, 4.215 , 4.3135,
       4.299 , 4.0955, 4.316 , 4.454 , 3.814 , 4.3765, 4.1155, 4.445 ,
       4.533 , 4.5045, 4.484 , 4.1625, 4.1325, 4.278 , 4.1175, 4.2315,
       4.208 , 4.3975, 4.469 , 4.1755, 4.2415, 4.465 , 4.206 , 3.976 ,
       4.429 , 4.271 , 4.292 , 4.451 , 4.3445, 4.15  , 3.3545, 4.1605,
       3.901 , 4.3395, 4.328 , 4.48  , 4.1815, 3.9955, 3.5835, 4.4765,
       4.4705, 4.047 , 4.358 , 4.383 , 4.329 , 4.5225, 4.3125, 4.376 ,
       4.312 , 3.94  , 3.8645, 3.9825])
```

In [51]:
```
# The mean of the expected target value in test set
y_test.describe()
```

Out[51]:
```
count    164.000000
mean       4.259756
std        0.328544
min        2.600000
25%        4.100000
50%        4.300000
75%        4.500000
max        4.900000
Name: Rating, dtype: float64
```

In [52]:
```
# Evaluate Random Forest Regression Accuracy using Root Mean Square Error
RMSE_random_Forest = sqrt(mean_squared_error(y_true = y_test, y_pred = y_prediction))
print(RMSE_random_Forest)
```

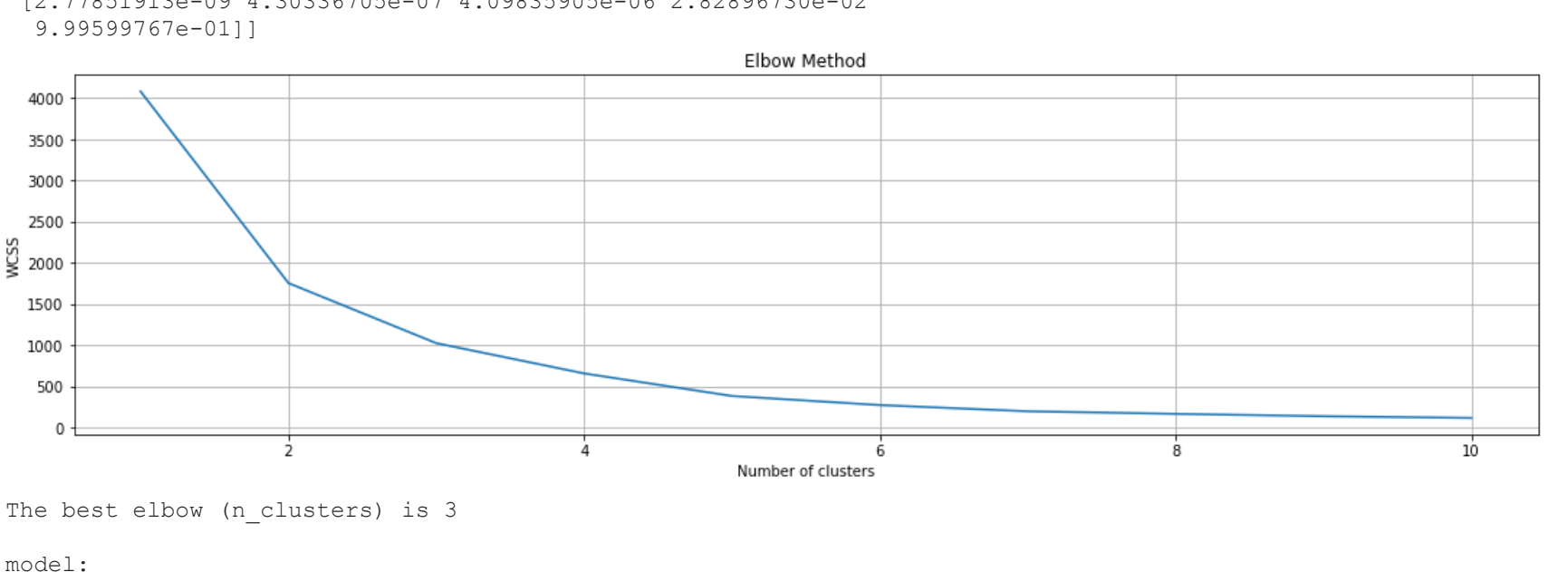
```
0.2928674208912836
```

Question 2

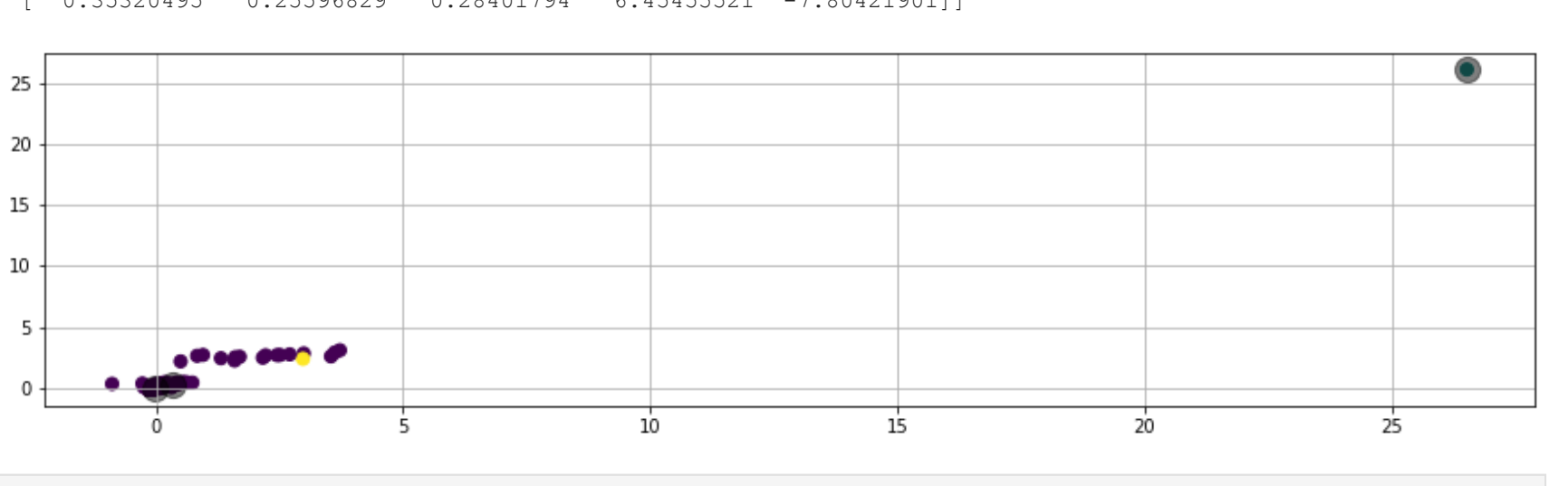
1. What categories of Apps that a marketing strategy should targeted on the Google Play Store Apps?

In [53]:
```
data_clustering = df_merge.copy()
data_clustering = data_clustering.drop(["App_name", "Category", "Size", "Type", "Price", "Genres", "Content_
normalized = preprocessing.normalize(data_clustering)
print("The normalized data for clustering:\n", normalized)

X_input = StandardScaler().fit_transform(normalized)

# plot the elbow method
wcss = []
for i in range(1, 11):
    kmeans_p = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)
    kmeans_p.fit(X_input)
    wcss.append(kmeans_p.inertia_)
plt.figure(figsize = (15,8))
plt.subplot(2, 1, 1)
plt.plot(range(1, 11), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.grid()
plt.tight_layout()
plt.show()

# find the best elbow
kl = KneeLocator(range(1, 11), wcss, curve = "convex", direction = "decreasing")
elbow = kl.elbow
print("The best elbow (n_clusters) is", elbow, "\n")

# print the model, fit the model to the normalized data
kmeans = KMeans(n_clusters = elbow)
model = kmeans.fit(X_input)
print("model:\n", model, "\n")

# print the resulting cluster centers
centers = model.cluster_centers_
print("Cluster centers:\n", centers, "\n")

predict = kmeans.predict(X_input)

plt.figure(figsize = (15,8))
plt.subplot(2, 1, 2)
plt.scatter(X_input[:, 0], X_input[:, 1], c = predict, s = 50, cmap = 'viridis')
plt.scatter(centers[:, 0], centers[:, 1], c = 'black', s = 200, alpha = 0.5);
plt.grid()
```

```
The normalized data for clustering:
 [[9.39397214e-07 1.11902073e-06 7.99990080e-06 4.97993825e-03
  9.99987600e-01]
 [1.56955674e-08 5.12348310e-08 3.79995488e-07 4.87314214e-03
  9.99988126e-01]
 [3.25822762e-07 6.03294662e-07 4.69874000e-06 2.31537912e-02
  9.99731915e-01]
 ...
 [3.93624188e-10 5.09290960e-09 4.29487722e-08 4.87982551e-02
  9.98808655e-01]
 [2.14867421e-07 5.46797850e-07 4.29977090e-06 1.03224500e-02
  9.99946722e-01]
 [2.77851913e-09 4.30336705e-07 4.09835905e-06 2.82896730e-02
  9.99599767e-01]]
```




```
The best elbow (n_clusters) is 3

model:
 KMeans(n_clusters=3)

Cluster centers:
 [[ -0.03587705  -0.03454877  -0.03492727  -0.06680897   0.08384201]
 [ 26.51622087  26.12362753  26.23310922   8.7997609  -13.11481286]
 [  0.35320495   0.25596829   0.28401794   6.45455521  -7.80421901]]
```



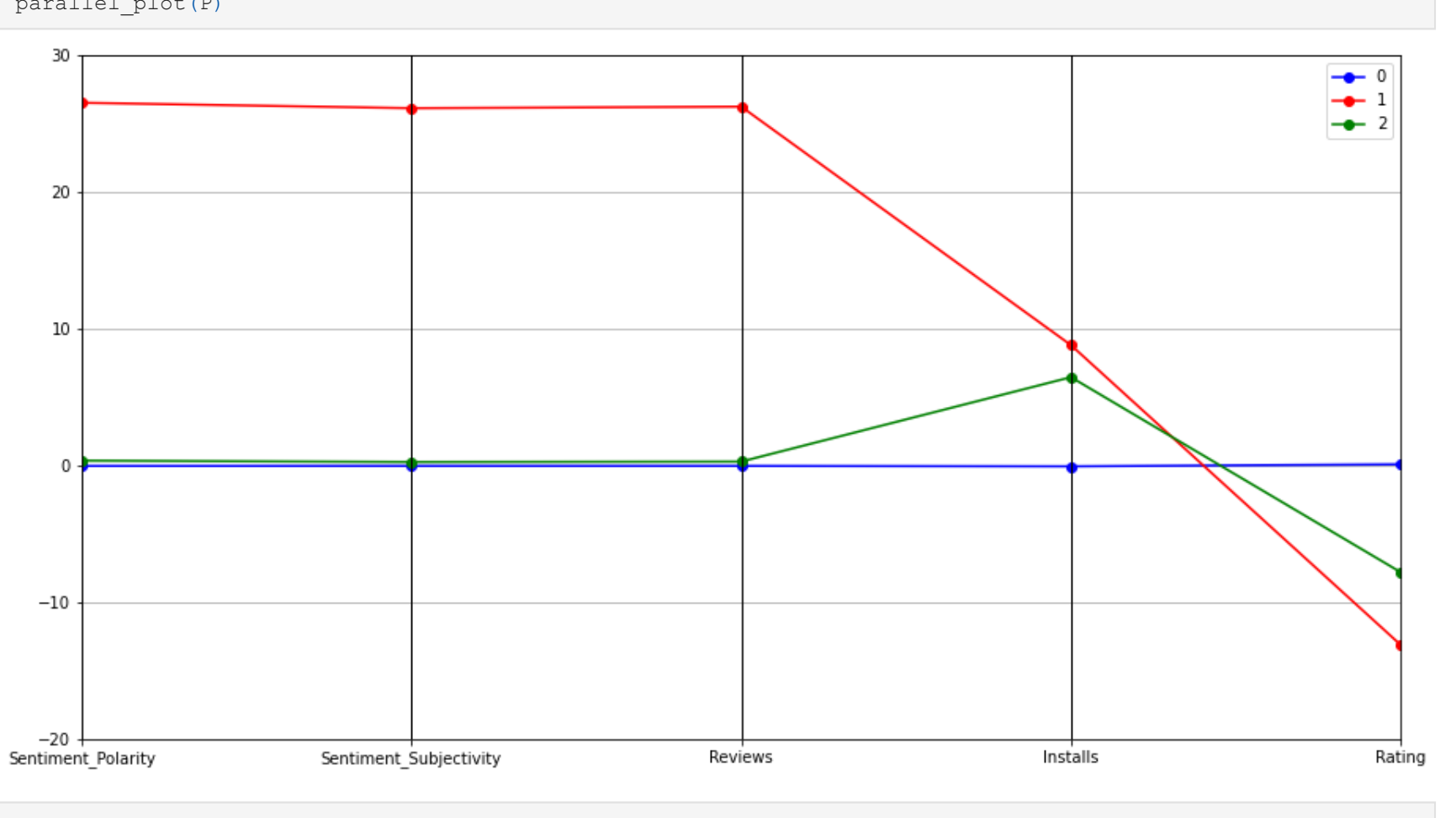
In [54]:
```
# Function that creates a DataFrame with a column for Cluster Number

def pd_centers(featuresUsed, centers):
        colNames = list(featuresUsed)
        colNames.append('Cluster')

        # Zip with a column called 'cluster' (index)
        Z = [np.append(A, index) for index, A in enumerate(centers)]

        # Convert to pandas data frame for plotting
        P = pd.DataFrame(Z, columns = colNames)
        P['Cluster'] = P['Cluster'].astype(int)
        return P
```

In [55]:
```
features = ['Sentiment_Polarity', 'Sentiment_Subjectivity', 'Reviews', 'Installs', 'Rating']

P = pd_centers(features, centers)
print(P)
```

```
   Sentiment_Polarity  Sentiment_Subjectivity    Reviews  Installs     Rating  \
0           -0.035877               -0.034549  -0.034927 -0.066809   0.083842
1           26.516221               26.123628  26.233109  8.799761 -13.114813
2            0.353205                0.255968   0.284018  6.454555  -7.804219

   Cluster
0        0
1        1
2        2
```

In [56]:
```
# Function that creates Parallel Plots

def parallel_plot(data):
        my_colors = list(islice(cycle(['b', 'r', 'g', 'y', 'k']), None, len(data)))
        plt.figure(figsize = (15,8)).gca().axes.set_ylim([-20,+30])
        parallel_coordinates(data, 'Cluster', color = my_colors, marker = 'o')
```

In [57]:
```
parallel_plot(P)
```



In [58]:
```
new_data = data_clustering.copy()

predict_data = kmeans.predict(X_input)
new_data['Cluster'] = predict_data
new_data['Category'] = df_merge.Category

new_data_shape = new_data.shape
print("DataFrame's shape:\n", new_data_shape, "\n")

new_data_head = new_data.head()
print("DataFrame's head:\n", new_data_head, "\n")
```

```
DataFrame's shape:
 (816, 7)

DataFrame's head:
    Sentiment_Polarity  Sentiment_Subjectivity  Rating   Reviews     Installs  \
0            0.469704                0.559517     4.0    2490.0     500000.0
1            0.156958                0.512354     3.8   48732.0   10000000.0
2            0.325910                0.603456     4.7   23160.0    1000000.0
3            0.224449                0.604413     4.6  214878.0   10000000.0
4            0.416806                0.550417     4.2  115773.0   10000000.0

   Cluster            Category
0        0  HEALTH_AND_FITNESS
1        0            SHOPPING
2        0             MEDICAL
```

```
3          0              GAME
4          0     PERSONALIZATION
```

In [59]:
```python
label_list = new_data["Cluster"].to_list()
unique_list = new_data["Cluster"].unique()

for i in range(0,len(unique_list)):
    counter = 0
    for j in range(0, len(label_list)):
        if label_list[j] == unique_list[i]:
            counter += 1
    print("Cluster ID =", unique_list[i], "Cluster Size =", counter)
```

```
Cluster ID = 0 Cluster Size = 808
Cluster ID = 2 Cluster Size = 7
Cluster ID = 1 Cluster Size = 1
```

In [60]:
```python
new_data[new_data.Cluster == 0].Category.describe()
```

Out[60]:
```
count        808
unique        33
top       FAMILY
freq          78
Name: Category, dtype: object
```

In [61]:
```python
new_data[new_data.Cluster == 1].Category.describe()
```

Out[61]:
```
count           1
unique          1
top       MEDICAL
freq            1
Name: Category, dtype: object
```

In [62]:
```python
new_data[new_data.Cluster == 2].Category.describe()
```

Out[62]:
```
count          7
unique         4
top       FAMILY
freq           3
Name: Category, dtype: object
```

Question 3

1. Can we use NLP to train the Google Play Store dataset to predict reviews?

In [63]:
```python
df = pd.read_csv("./googleplaystore_user_reviews.csv")

# Here I select the top 2000 items since it takes too long to run all items.
df_user_reviews = df.head(2000)
```

In [64]:
```python
# Create a new dataframe for NLP analysis.
df_nlp = pd.concat([df_user_reviews.App, df_user_reviews.Translated_Review, df_user_reviews.Sentiment], axis
df_nlp.head()
```

Out[64]:

	App	Translated_Review	Sentiment
0	10 Best Foods for You	I like eat delicious food. That's I'm cooking ...	Positive
1	10 Best Foods for You	This help eating healthy exercise regular basis	Positive
2	10 Best Foods for You	NaN	NaN
3	10 Best Foods for You	Works great especially going grocery store	Positive
4	10 Best Foods for You	Best idea us	Positive

In [65]:
```python
# Data cleaning
# Drop missing values
df_nlp.dropna(axis = 0,inplace = True)

# Drop duplicates rows
df_nlp.duplicated(subset = 'App').value_counts()

# Replace non-letter characters with space
first_text = df_nlp.Translated_Review[0]
text = re.sub("[^a-zA-Z]"," ",first_text)

# Convert upper case to lower case
text = text.lower()

# Review changes
print(df_nlp.Translated_Review[0])
print(text)
```

```
I like eat delicious food. That's I'm cooking food myself, case "10 Best Foods" helps lot, also "Best Before
(Shelf Life)"
i like eat delicious food  that s i m cooking food myself  case     best foods  helps lot  also  best before
  shelf life
```

In [66]:
```python
df_nlp.Sentiment.value_counts()
```

Out[66]:
```
Positive    934
Negative    241
Neutral     161
Name: Sentiment, dtype: int64
```

In [67]:
```python
# Converting review types to int form 0=Positive, 1=Negative, 2=Neutral.
df_nlp.Sentiment = [0 if i == "Positive" else 1 if i == "Negative" else 2 for i in df_nlp.Sentiment]
df_nlp.head(10)
```

Out[67]:

	App	Translated_Review	Sentiment
0	10 Best Foods for You	I like eat delicious food. That's I'm cooking ...	0
1	10 Best Foods for You	This help eating healthy exercise regular basis	0
3	10 Best Foods for You	Works great especially going grocery store	0
4	10 Best Foods for You	Best idea us	0
5	10 Best Foods for You	Best way	0
6	10 Best Foods for You	Amazing	0
8	10 Best Foods for You	Looking forward app,	2
9	10 Best Foods for You	It helpful site ! It help foods get !	2
10	10 Best Foods for You	good you.	0
11	10 Best Foods for You	Useful information The amount spelling errors ...	0

In [68]:
```python
# Remove irrelavent words
nltk.download("stopwords")
nltk.download("punkt")
```

```
[nltk_data] Downloading package stopwords to /Users/boyan/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /Users/boyan/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Out[68]: True

In [69]:
```python
# Separate all words
text = nltk.word_tokenize(text)
text
```

Out[69]:
```
['i',
 'like',
 'eat',
 'delicious',
 'food',
 'that',
 's',
 'i',
 'm',
 'cooking',
 'food',
 'myself',
 'case',
 'best',
 'foods',
 'helps',
 'lot',
 'also',
 'best',
 'before',
 'shelf',
 'life']
```

In [70]:
```python
#lemmatization
nltk.download('wordnet')
```

```
[nltk_data] Downloading package wordnet to /Users/boyan/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Out[70]: True

In [71]:
```python
# Convert words to their root format.
wnl = nltk.WordNetLemmatizer()
text = [wnl.lemmatize(i) for i in text]
text = " ".join(text)
text
```

Out[71]:
```
'i like eat delicious food that s i m cooking food myself case best food help lot also best before shelf lif
e'
```

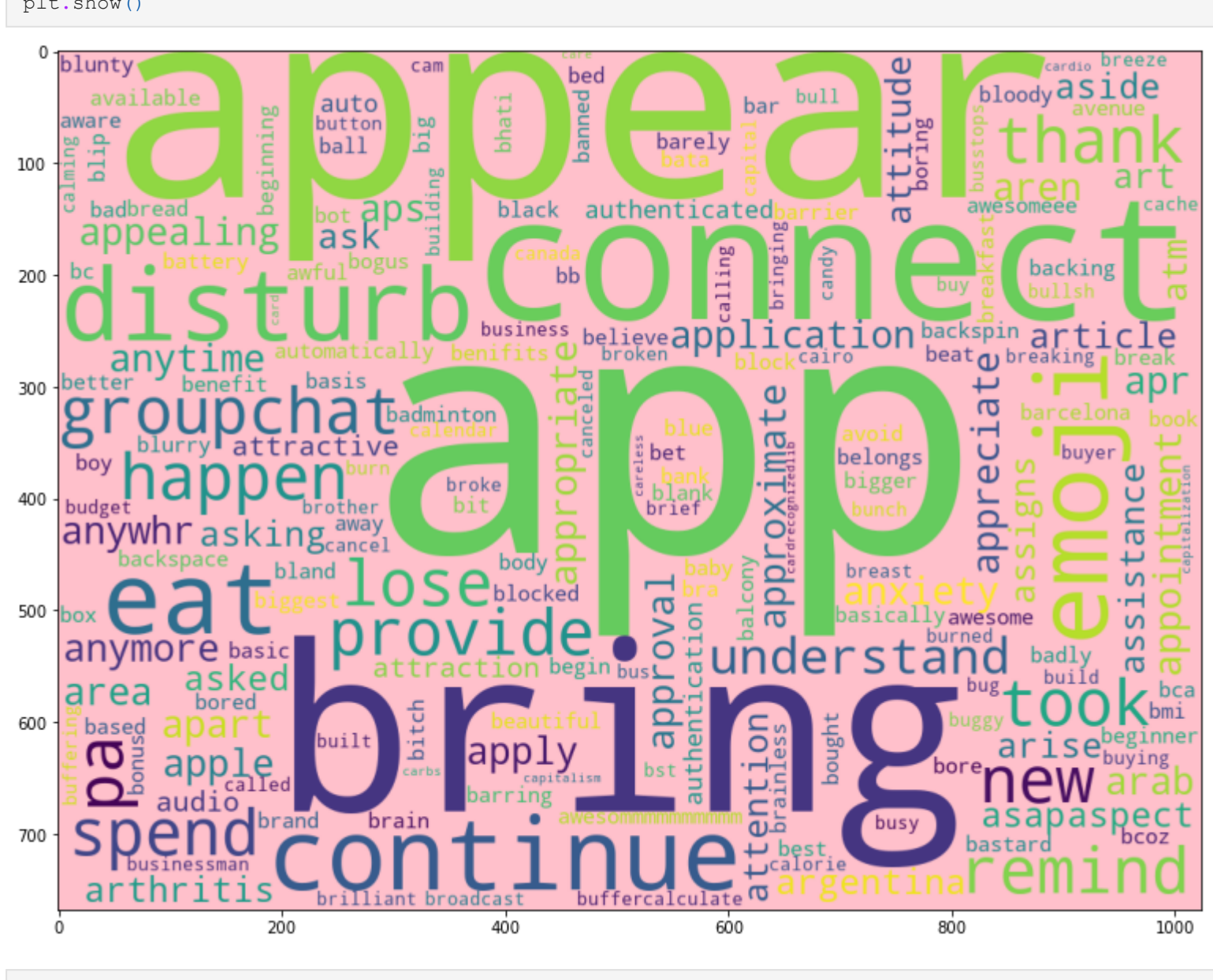
In [72]:
```python
text_list = []
for i in df_nlp.Translated_Review:
    text = re.sub("[^a-zA-Z]"," ",i)
    text = text.lower()
    text = nltk.word_tokenize(text)
    wnl = nltk.WordNetLemmatizer()
    text = [wnl.lemmatize(word) for word in text]
    text = " ".join(text)
    text_list.append(text)

text_list[:10]
```

Out[72]:
```
['i like eat delicious food that s i m cooking food myself case best food help lot also best before shelf li
fe',
 'this help eating healthy exercise regular basis',
 'work great especially going grocery store',
 'best idea u',
 'best way',
 'amazing',
 'looking forward app',
 'it helpful site it help food get',
 'good you',
 'useful information the amount spelling error question validity information shared once fixed star given']
```

In [73]:
```python
# Bag of words
max_features = 200000
count_vector = CountVectorizer(max_features = max_features, stop_words = "english")
sparce_matrix = count_vector.fit_transform(text_list).toarray()
all_words = count_vector.get_feature_names()
all_words[0:50]
```

Out[73]:
```
['aap',
 'abandonware',
 'abducted',
 'ability',
 'able',
 'abruptly',
 'absent',
 'absolutely',
 'abt',
 'abu',
 'acc',
 'accent',
 'accept',
 'access',
 'accidentally',
 'accidently',
 'accompany',
 'accomplished',
 'according',
 'account',
 'accountable',
 'accurate',
 'achieved',
 'achievement',
 'achieves',
 'acknowledging',
 'acoutered',
 'act',
 'acting',
 'action',
 'activate',
 'active',
 'actively',
 'activity',
 'actual',
 'actually',
 'actul',
 'ad',
 'add',
 'added',
 'addict',
 'addicted',
 'addiction',
 'addictive',
 'addition',
 'additional',
 'address',
 'admin',
 'adobe',
 'adorable']
```

In [74]:
```python
plt.subplots(figsize = (15,10))
wordcloud = WordCloud(background_color = "pink",width = 1024,height = 768).generate(" ".join(all_words[100:]
plt.imshow(wordcloud)
plt.show()
```



In [75]:
```python
# Classification
y = df_nlp.iloc[:,1].values
x = sparce_matrix
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.2,random_state = 0)
```

In [76]:
```python
#logistic Regression
logistic_regression = LogisticRegression()
logistic_regression.fit(x_train,y_train)
y_prediction = logistic_regression.predict(x_test)
cm = confusion_matrix(y_test, y_prediction)
accuracy_logistic = logistic_regression.score(x_test,y_test)
print(cm)
print("\nlogistic_regression accuracy: ", accuracy_logistic)
```

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

logistic_regression accuracy:  0.45149253731343286
```

In [77]:
```python
# Now Fitting Naive Bayes classifier to the Training set

classifier = GaussianNB()
classifier.fit(x_train, y_train)

# Predicting the Test set results
y_prediction = classifier.predict(x_test)

# Making the Confusion Matrix and find Accuracy
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_prediction)
accuracy_naive_bays = accuracy_score(y_test, y_prediction)

print(cm)
print("\nNaive Bayes Classifier accuracy: ", accuracy_naive_bays)
```

```
[[0 0 0 ... 0 0 0]
 [0 1 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 2 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 1]]

Naive Bayes Classifier accuracy:  0.5186567164179104
```

Step 4: Present your findings

In this step, you can begin to report your findings. What did you learn from the data and how do your findings help answer your research question? Use *matplotlib* visualizations to present these findings.

Remember: Rarely will a single data analysis conclusively answer a research question. Here, you need to identify possible limitations. For example, are your results limited to a certain area, city, or country? Are you making assumptions about the data which may, or may not, be valid (e.g., that students in one term are equally qualified as students in another)? Document these limitations in a few paragraphs.

Findings

In this notebook, pandas and numpy are used for data preprocessiong, and matplotlib, seaborn, sklearn, and mostly plotly are used for visualization. By analyzing the Google Play Store Apps datasets, different machine learning models are used to predict the app ratings, marketing strategies, and reviews.

What did I learn from the data and how do my findings help answer my research question?

1. EDA Findings

By exploring the data, the data quality issues such as missing values, duplicated rows, and unused columns are found. In order to analysis data precisely, I have cleaned the data. After cleaning the data, the two datasets are merged together by using their common features.

It is found that most of Apps ratings are within the range of 4.0 to 4.7 with freely available and belong to the Family category. Most of the content rating of these Apps are for Everyone. Family and Medical categories of Apps are popular while Comics category Apps are not so popular among these categories.

Most of the attributes among the data are positively correlated, such as reviews and installs, while some are negatively correlated, such as Sentiment_Polarity and Installs. Some attributes have outliers, such as Installs and Reviews.

Also, there are positive skewness values such as Sentiment_Polarity, Reviews, and Installs which indicate asymmetry in the distribution and the tail is larger towards the right hand side of the distribution. And negative skewness values such as Sentiment_Subjectivity and Rating indicate asymmetry in the distribution and the tail is larger towards the left hand side of the distribution. It is also found the highly skewed distributions such as Sentiment_Subjectivity, Rating, Reviews, and Installs, and approximately symmetric distribution such as Sentiment_Polarity.

You can also find these findings in each section under the Step 2: Explore the datasets.

2. Data Analysis Findings

Based on the EDA findings, three questions were brought up. By analyzing the datasets with different machine learning models, it is found that the RMSE for Linear Regression, Decision Tree Regression, and Random Forest Regression are very close, which means that the accuracy by using different models to predict App Ratings are very close. Please refer to the chart 1.

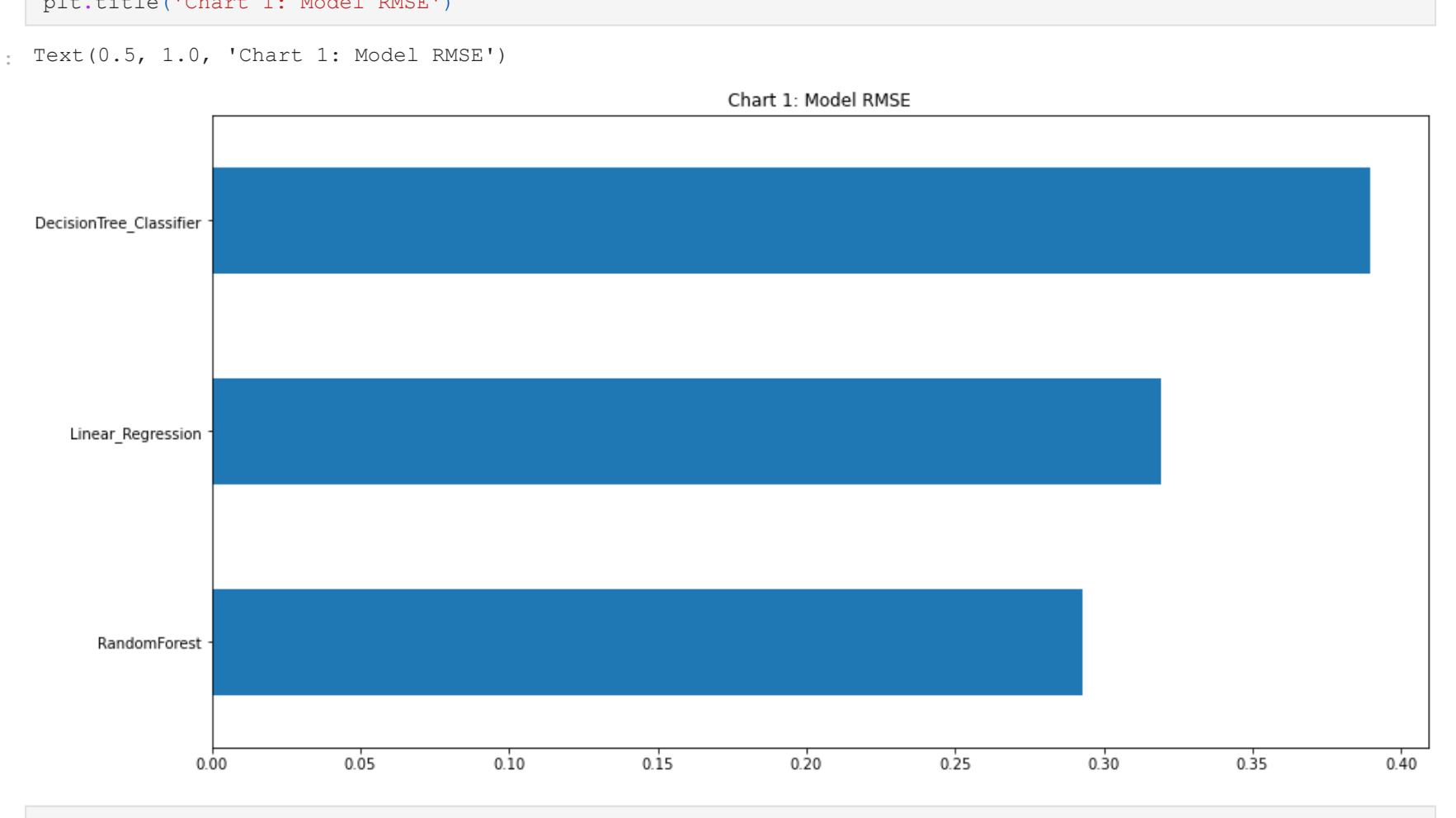
By using k-means clustering, it is found that most Apps are focus on Family and Medical categories, and the market might be saturated. New Apps in Comics category might be a potential market.

Through NLP sentiment analysis, it is found that customer reviews can be predicted, but we need to choose the right model. The Logistic regression and Naive Bays classifier I choose do not have a very high accuracy. Maybe I should try other models. Please refer to the chart 2.

3. All of the above findings help me better understand the datasets and lead me to think further when performing the analyzing.

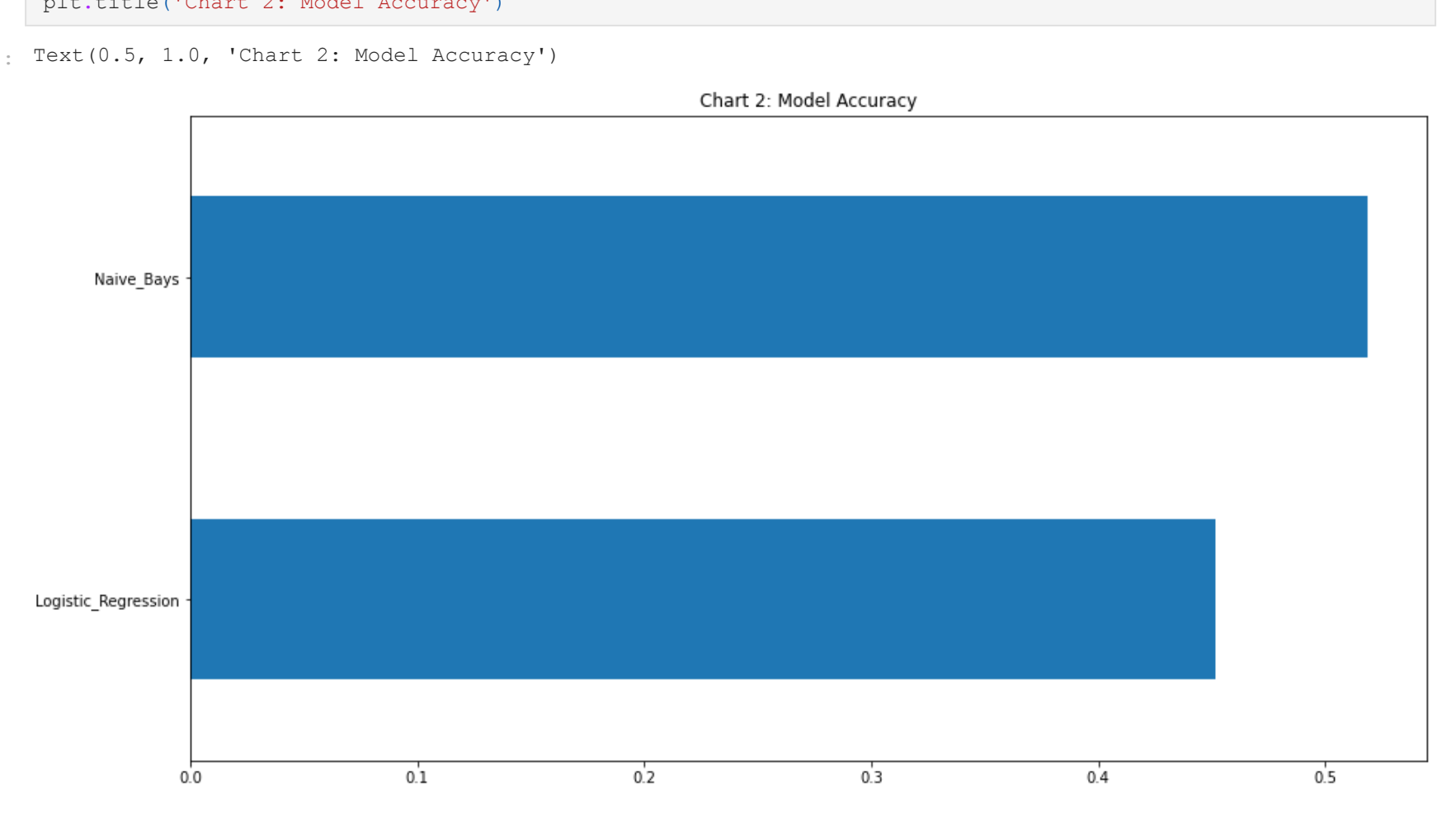
In [78]:
```python
model_accuracy = pd.Series(data = [RMSE_linear_regression,RMSE_decision_tree,RMSE_random_Forest],
        index = ['Linear_Regression','DecisionTree_Classifier','RandomForest'])
fig = plt.figure(figsize = (15,8))
model_accuracy.sort_values().plot.barh()
plt.title('Chart 1: Model RMSE')
```

Out[78]: Text(0.5, 1.0, 'Chart 1: Model RMSE')



In [79]:
```python
model_accuracy = pd.Series(data = [accuracy_logistic,accuracy_naive_bays],
        index = ['Logistic_Regression','Naive_Bays'])
fig = plt.figure(figsize = (15,8))
model_accuracy.sort_values().plot.barh()
plt.title('Chart 2: Model Accuracy')
```

Out[79]: Text(0.5, 1.0, 'Chart 2: Model Accuracy')



Limitations

There are limitations during the data exploration and analysis.

1. The data quality issues actually force me to drop lots of rows. If these rows can be filled with valuable values, that must be very helpful to get a more accurate results.
2. The validate data sizes between these two datasets are not good enough to match each other.
3. The performance issues between different machine learning models are hard to guess. I made assumptions when choosing these models.
4. Also, the results for all these questions are only limited to these specific datasets downloaded from Kaggle. There are no attributes that show these two datasets are collected on the same period.
5. For question 3, in order to get the results faster, I limit the dataset to only 2000 items. Otherwise, I need to wait for quite a while. So the computer CPU is also a limitation.

Where to find project datasets?

There are a surprising number of open datasets on the web. There are also a number of excellent websites which help keep track of where to find datasets. Rather than attempt to recreate the good work from these teams, here's where you can find there collections:

- UCI's Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets.php) - UC Irvine maintains a fantastic collection of datasets for machine learning, tagged by machine learning task (among other things).
- Kaggle (https://www.kaggle.com/datasets) - Online collection of datasets. Kaggle also has competitions for data mining and information about jobs in data science.
- KDnuggets Dataset (http://www.kdnuggets.com/datasets/index.html) - Another collection of datasets - most of the datasets are free. Within KDnuggets, there are links for government data (http://www.kdnuggets.com/datasets/government-local-public.html), Data APIs (http://www.kdnuggets.com/datasets/api-hub-marketplace-platform.html), and Data Mining Competitions (http://www.kdnuggets.com/competitions/index.html).
- There are also many sites that share open government and organizational datasets: US Government Data (https://www.data.gov/), UK Government Data (https://data.gov.uk/), Canada's Open Data Exchange (https://codx.ca/), World Health Organization (http://www.who.int/gho/en/), and the World Bank (http://data.worldbank.org/).

Remember, finding a dataset of interest and exploring it is most of your job for your final project. So expect this to take some time - and that's both perfectly normal and completely okay. Get started early and make sure to check if the dataset you find satisfies the project requirements.

The datasets I used for this final project are Google Play Store Apps from Kaggle. You can find the project datasets by clicking the following link to download the Google Play Store Apps Datasets.

https://www.kaggle.com/lava18/google-play-store-apps

There are two .csv files in the datasets.

1. googleplaystore.csv (1.3 MB)

1. googleplaystore_user_reviews.csv (7.31 MB)