akshaya-project-13

April 28, 2023

#

Google App store analysis

0.1 Google App Store Data Analysis

Project Title : Google App Store Data Analysis

Technologies : Data Science

Domain : Technology

Project Difficulties level : Intermediate

0.2 Author : Akshaya L

[334]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
import re
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
from sklearn import preprocessing
from sklearn.metrics import accuracy_score
from sklearn.feature_extraction.text import CountVectorizer
```

0.3 Basic analysis

First let's do some basic analysis to discover what our data looks like

[335]:
```python
df = pd.read_csv('./googleplaystore.csv')
df.head()
```

[335]:
```
                                                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
```

We can already notice that we have a lot of categorical columns: Category, Installs, Type, Content Rating, Genres, Current Ver and Android Ver.

We can also notice the App column that contains the App name, meaning that all its values will most likely be unique.

[336]:
```
df.duplicated().sum()
```

[336]:
```
483
```

We see that we have duplicated rows, let's drop them to not impact our analysis

[337]:
```
df.shape
```

[337]:
```
(10841, 13)
```

[338]:
```
df.drop_duplicates(inplace=True)
df.shape
```

[338]:
```
(10358, 13)
```

[339]:
```
df.describe()
```

[339]:
```
          Rating
count  8893.000000
mean      4.189542
std       0.545452
min       1.000000
25%       4.000000
50%       4.300000
75%       4.500000
max      19.000000
```

Rating has a max of 19, while the play store rates from 1 to 5. We have invalid data to remove here.

[340]:
```python
df.sort_values(by=['Rating'], ascending=False).head()
```

[340]:
```
                                           App      Category  Rating Reviews  \
10472  Life Made WI-Fi Touchscreen Photo Frame           1.9    19.0    3.0M
5139                              Chenoweth AH       MEDICAL     5.0       1
6851                            BV Mobile Apps  PRODUCTIVITY     5.0       3
6807                                 Jabbla BT         TOOLS     5.0       3
6816                                  BU Study        FAMILY     5.0       7

         Size Installs  Type     Price Content Rating             Genres  \
10472  1,000+     Free     0  Everyone            NaN  February 11, 2018
5139      27M     100+  Free         0       Everyone            Medical
6851     4.8M     100+  Free         0       Everyone       Productivity
6807      55k     100+  Free         0       Everyone              Tools
6816     5.6M      10+  Free         0       Everyone          Education

            Last Updated  Current Ver   Android Ver
10472             1.0.19   4.0 and up           NaN
5139       April 3, 2017  300000.0.78  4.0.3 and up
6851        June 5, 2018          2.0    4.2 and up
6807     October 6, 2014          1.0    4.2 and up
6816    December 7, 2017          1.0  4.0.3 and up
```

[341]:
```python
df.drop(10472, inplace=True);
```

[342]:
```python
df.shape
```

[342]:
```
(10357, 13)
```

[343]:
```python
df.dtypes
```

[343]:
```
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
```

We notice here that even if some columns like Reviews contains numeric values, they are encoded as objects.

[344]:
```
df.isnull().sum()
```

[344]:
```
App                  0
Category             0
Rating            1465
Reviews              0
Size                 0
Installs             0
Type                 1
Price                0
Content Rating       0
Genres               0
Last Updated         0
Current Ver          8
Android Ver          2
dtype: int64
```

Our data is pretty clean in term of NAN values, except for the Rating column which has 14% missing values

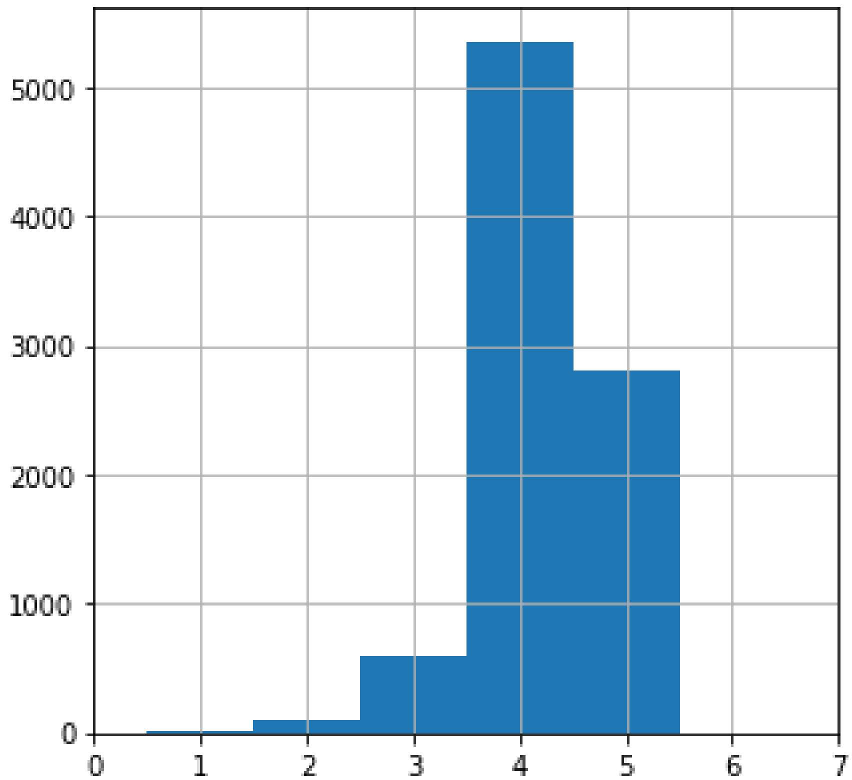
0.4 Cleaning up the data from any NAN values

[345]:
```
# The columns 'Type','Content Rating', 'Current Ver' and 'Android Ver' have so␣
↪few missing numbers
# that we can delete these rows
df_no_nan = df.dropna(axis=0, subset=['Type','Content Rating', 'Current Ver',␣
↪'Android Ver'])
```

[346]:
```
# In terms of Rating, it all depends on what information we want to extract␣
↪from our data. Let's see its distribution
plt.figure(figsize=(5,5))
bins = [0.5, 1.5, 2.5, 3.5, 4.5, 5.5, 6.5]
df['Rating'].hist(bins = bins);
```

```
plt.xlim(0,7)
```

[346]: (0, 7)



We have a left queued normal distribution with a maximum around 4.

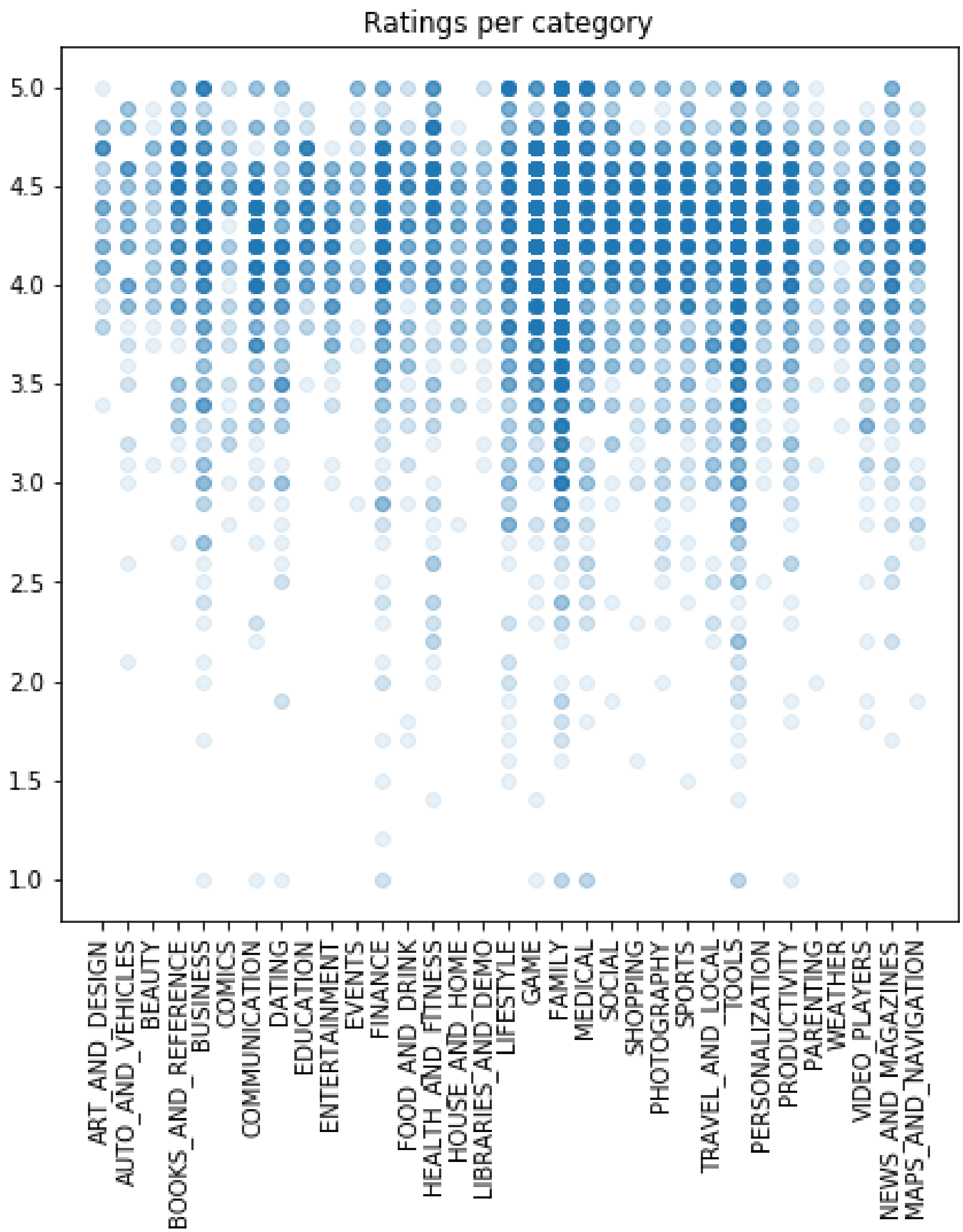
We can already say that our data shows that people who DO rate are mostly the one that are the most satisfied with the apps. Thus the missing values might indicate an app that is not good enough for people to take the time to rate it. In this condition we cannot afford to take the decision to replace a missing value by the mean (which is 4.19) since it would probably not represent the reality.

We could then choose to either remove the missing value rows or give them a value like 0 which is outside of the rating range but would probably be more realistic of the type of grade we would get if the value was existing.

[347]:
```
fill_nan = lambda col: col.fillna(0)
df_0_Rating = df_no_nan.apply(fill_nan, axis=0)

df_no_nan = df_no_nan.dropna(axis=0, subset=['Rating'])
```

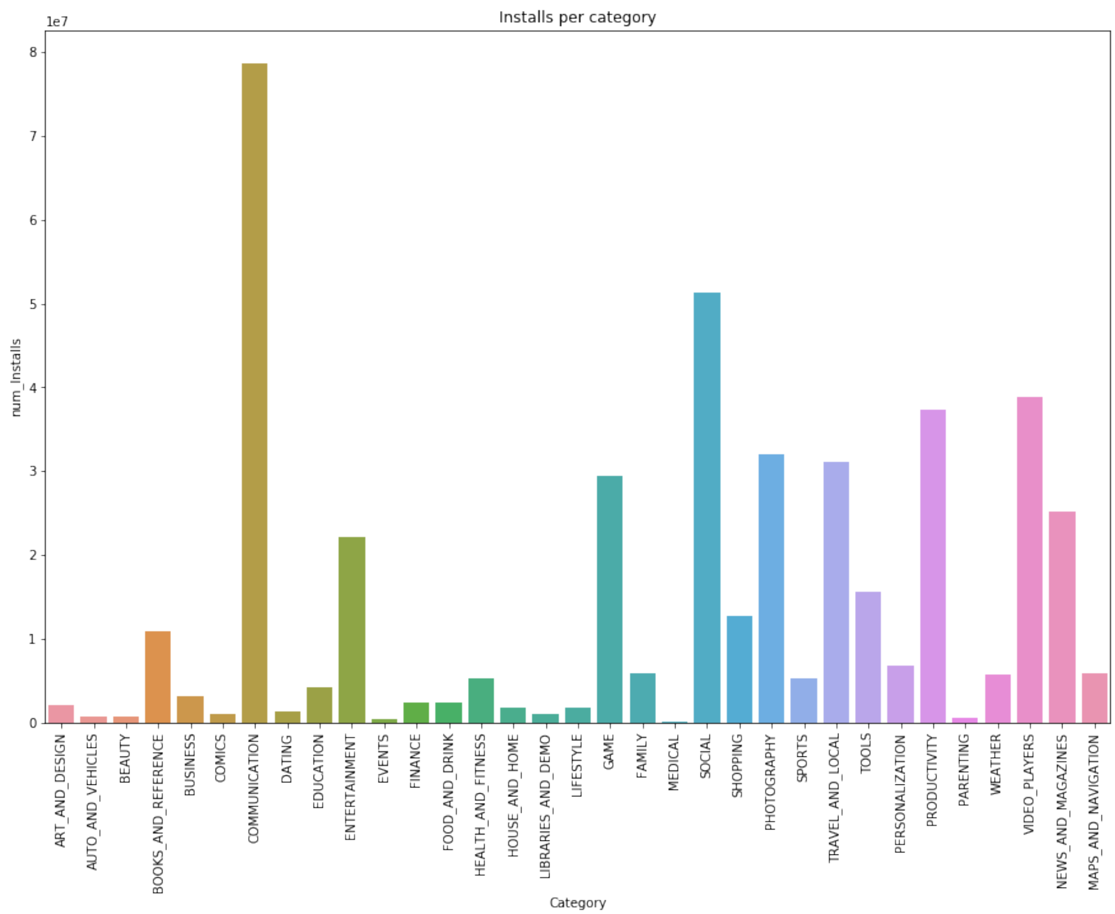
0.5 Data analysis

[348]:
```
plt.figure(figsize=(7,7))
plt.scatter(df_no_nan['Category'], df_no_nan['Rating'], alpha =.1);
plt.title('Ratings per category');
plt.xticks(rotation = 90);
```



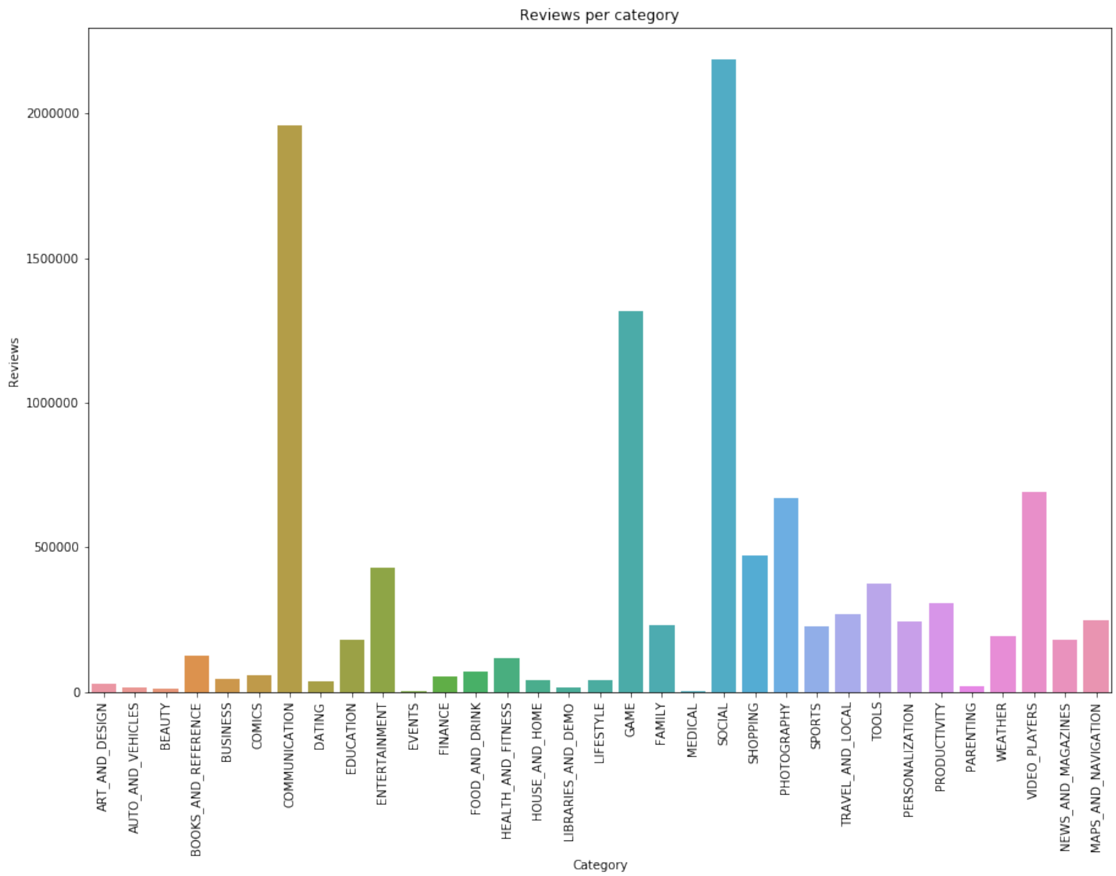
We see here that the rating is mostly concentrated between 4 and 4.5, except for categories like family which have a larger range because they have more values.

[349]:
```
df_no_nan['num_Installs'] = df_no_nan['Installs'].apply({'10,000+':10000,
↪'500,000+':500000, '5,000,000+':5000000,'50,000,000+':50000000, '100,000+':
↪100000, '50,000+':50000, '1,000,000+':1000000, '10,000,000+':10000000,
↪'5,000+':5000, '100,000,000+':100000000, '1,000,000,000+':1000000000,
↪'1,000+':1000, '500,000,000+':500000000, '50+':50, '100+':100, '500+':500,
↪'10+':10, '1+':1, '5+':5, '0+':0 }.get)

sns.barplot(x='Category', y='num_Installs', data=df_no_nan, ci = None);
plt.title('Installs per category');
plt.xticks(rotation = 90);
```



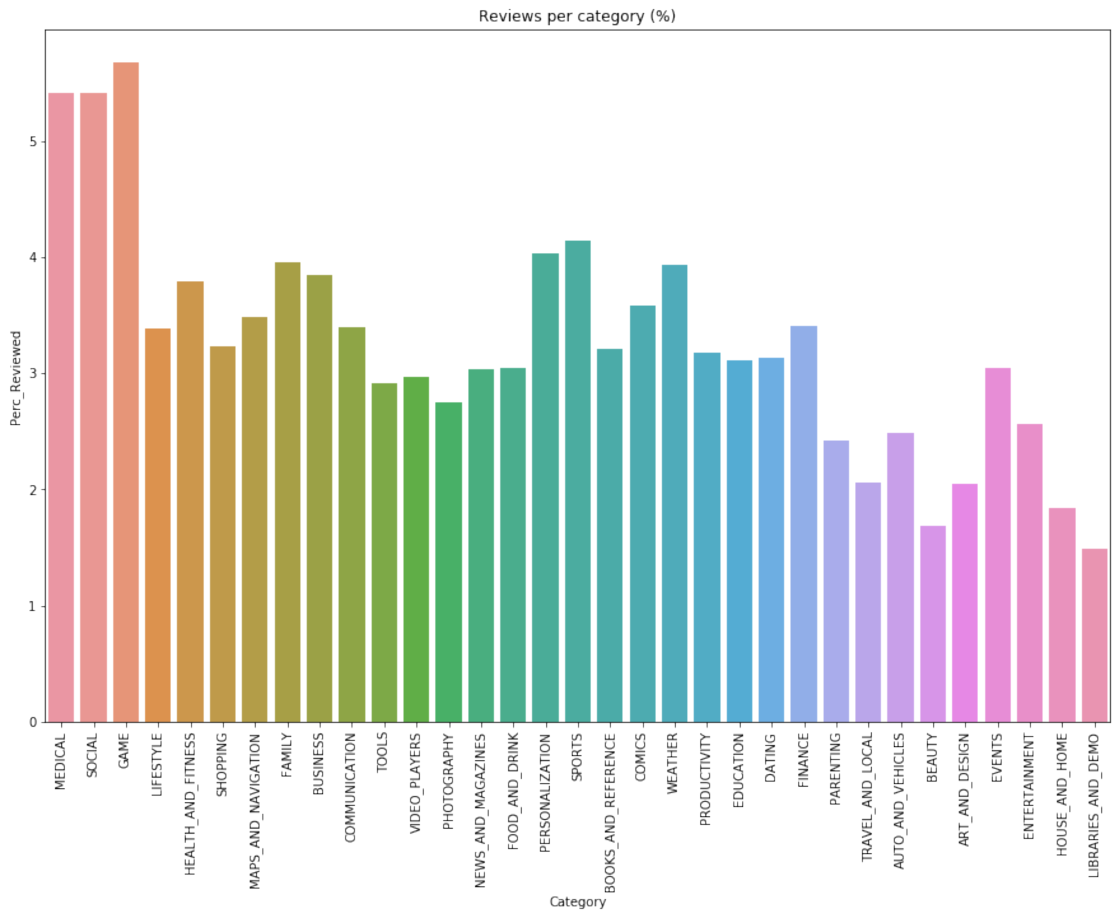
We can see here that the communication apps are much more installed than any other categories

[350]:
```
df_no_nan['Reviews'] = df_no_nan['Reviews'].astype('float64')

sns.barplot(x='Category', y='Reviews', data=df_no_nan, ci = None);
plt.title('Reviews per category');
plt.xticks(rotation = 90);
```



The social apps are the most reviewed, closely followed by the communicaion apps.

[351]:
```
df_no_nan['Perc_Reviewed'] = df_no_nan['Reviews']/df_no_nan['num_Installs']*100;
sns.barplot(x='Category', y='Perc_Reviewed', data=df_no_nan.
  ↪sort_values(by='Perc_Reviewed', ascending=False), ci = None);
plt.title('Reviews per category (%)');
plt.xticks(rotation = 90);
```



The number of reviews per category is biaised by the number of installations per category. If a category is more installed, it is expected to be more reviewed. To remove this biais we plotted the percentage of reviewed and now we see that the games are much likely to be reviewed than any other apps. Though they are closely followed by the social and medical apps.

[352]:
```
df_number_per_category = df_no_nan.groupby('Category')['App'].nunique()
df_number_per_category = df_number_per_category.to_frame()
df_number_per_category['Category'] = df_number_per_category.index
df_number_per_category.columns=['Count', 'Category']
df_number_per_category['Percentage'] = df_number_per_category['Count']/
 ↪df_number_per_category.shape[0]
df_number_per_category.sort_values(by='Percentage', ascending=False).head(10)
```

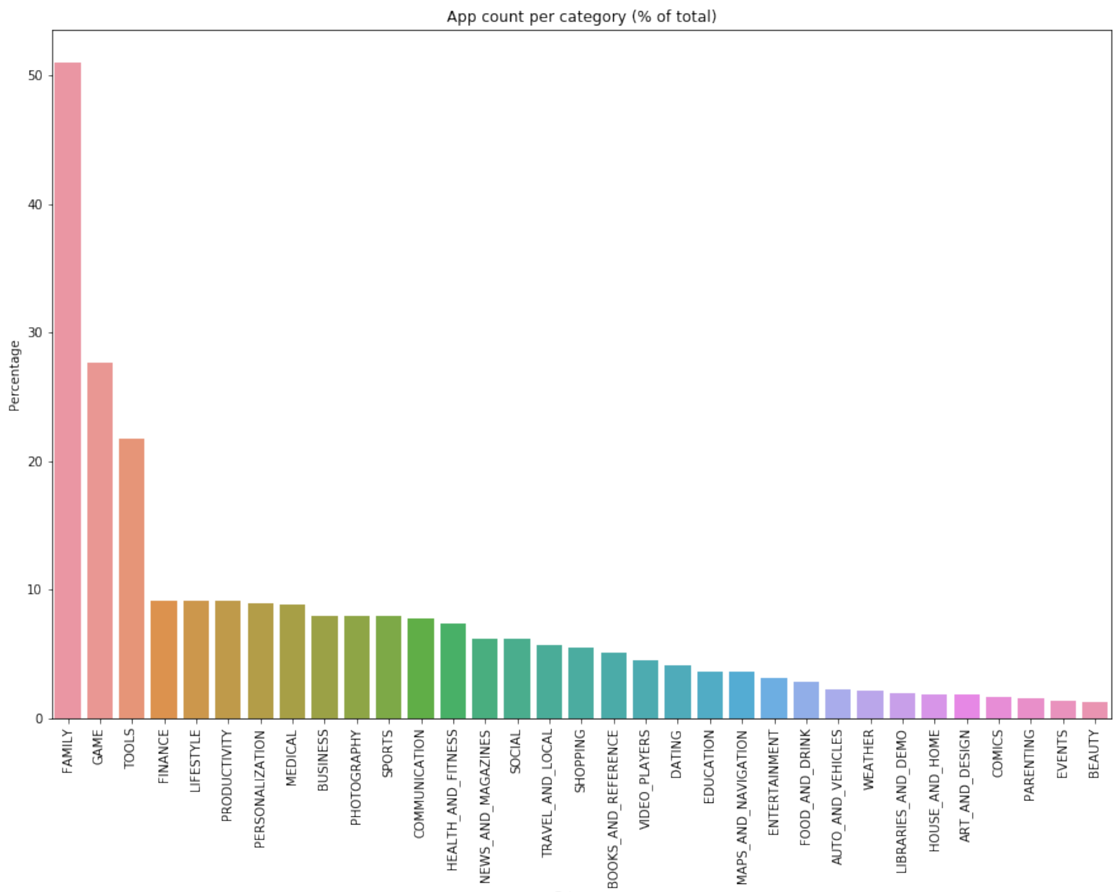
[352]:

	Count	Category	Percentage
Category			
FAMILY	1683	FAMILY	51.000000
GAME	913	GAME	27.666667
TOOLS	719	TOOLS	21.787879
FINANCE	302	FINANCE	9.151515

```
LIFESTYLE          301        LIFESTYLE    9.121212
PRODUCTIVITY       301     PRODUCTIVITY    9.121212
PERSONALIZATION    296  PERSONALIZATION    8.969697
MEDICAL            291          MEDICAL    8.818182
BUSINESS           263         BUSINESS    7.969697
PHOTOGRAPHY        263      PHOTOGRAPHY    7.969697
```

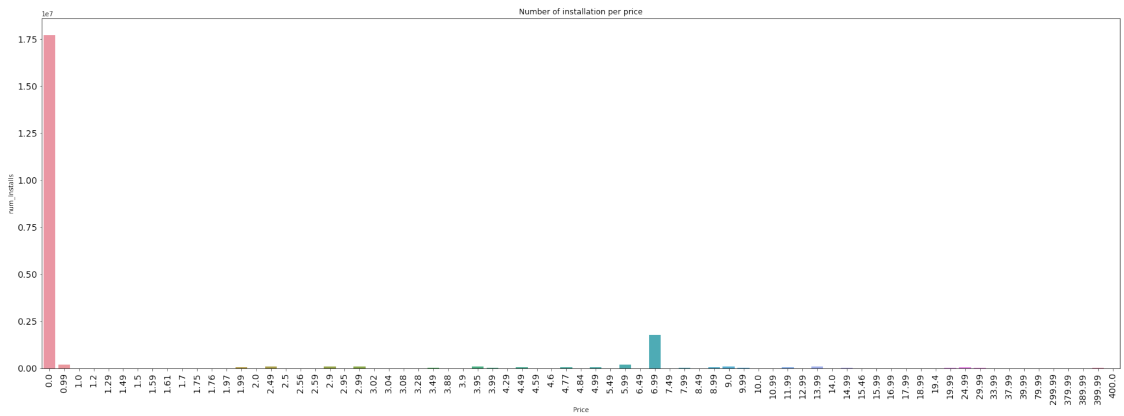
Here we can see that 51% of the apps in the app store are classified as family while 27% are games and 21% are tools. Let's not forget that a lot of apps are classified in several categoris and so the total percentage WILL BE higher than 100%

[353]:
```python
sns.barplot(x='Category', y='Percentage', data=df_number_per_category.
 ↪sort_values(by='Percentage', ascending=False), ci = None);
plt.title('App count per category (% of total)');
plt.xticks(rotation = 90);
```



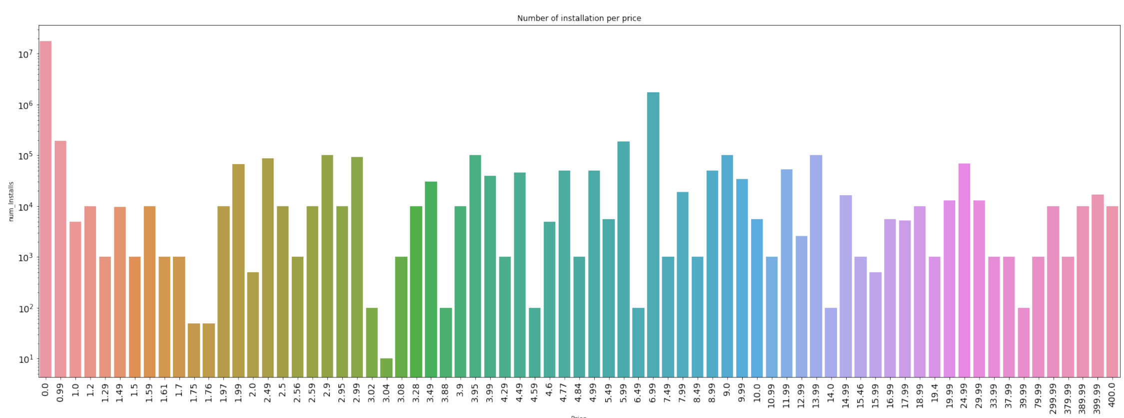
[354]:
```python
df_no_nan['Price'] = df_no_nan['Price'].apply(lambda x : x.replace('$', ''))
df_no_nan['Price'] = df_no_nan['Price'].astype('float64')
plt.figure(figsize=(30, 10))
```

```
plt.title('Number of installation per price');
plt.xticks(rotation = 90, fontsize=14);
plt.yticks(fontsize=14);
sns.barplot(x='Price', y='num_Installs', data=df_no_nan, ci = None);
```



Here we can clearly see that the free apps are by far the most installed. After that, the apps paid 6.99$ are the most installed

[355]:
```
plt.figure(figsize=(30, 10))
plt.title('Number of installation per price');
plt.xticks(rotation = 90, fontsize=14);
plt.yticks(fontsize=14);
plt.yscale('log')
sns.barplot(x='Price', y='num_Installs', data=df_no_nan, ci = None);
```



[356]:
```
df_no_nan['cumul_Installs_per_price'] = df_no_nan.
  ↪groupby(['Price'])['num_Installs'].apply(lambda x: x.cumsum())
```

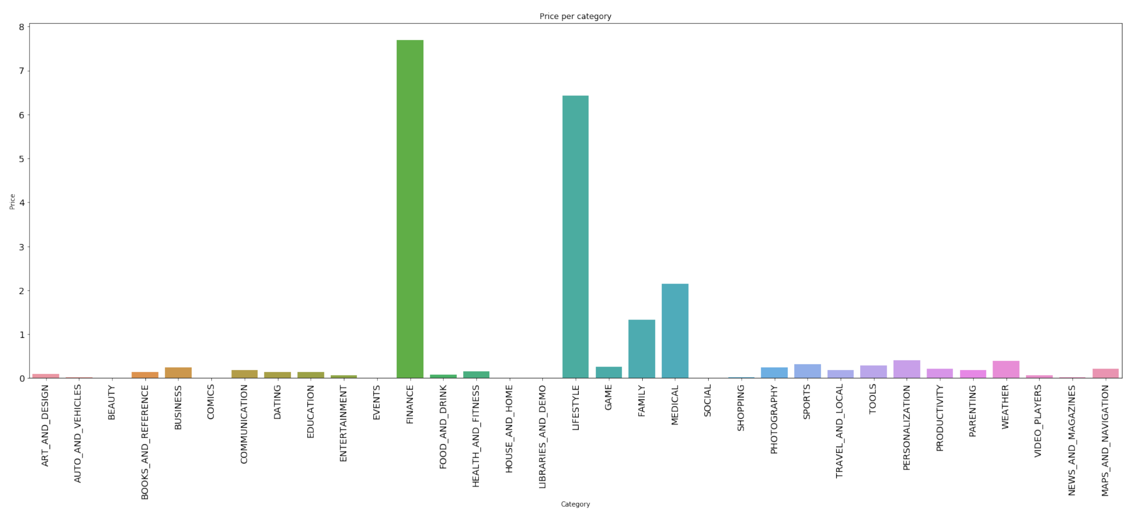
```
df_no_nan_cumsum = df_no_nan.groupby(['Price']).agg({'cumul_Installs_per_price': 'sum'})
df_no_nan_cumsum['percentage']=df_no_nan_cumsum['cumul_Installs_per_price']/df_no_nan_cumsum['cumul_Installs_per_price'].sum()*100
df_no_nan_cumsum.sort_values(by='percentage', ascending = False).head()
```

[356]:

Price	cumul_Installs_per_price	percentage
0.00	835889058796914	99.999711
0.99	1407569111	0.000168
2.99	481841680	0.000058
6.99	185293300	0.000022
4.99	102784560	0.000012

More than 99.99% of the installed apps are free!

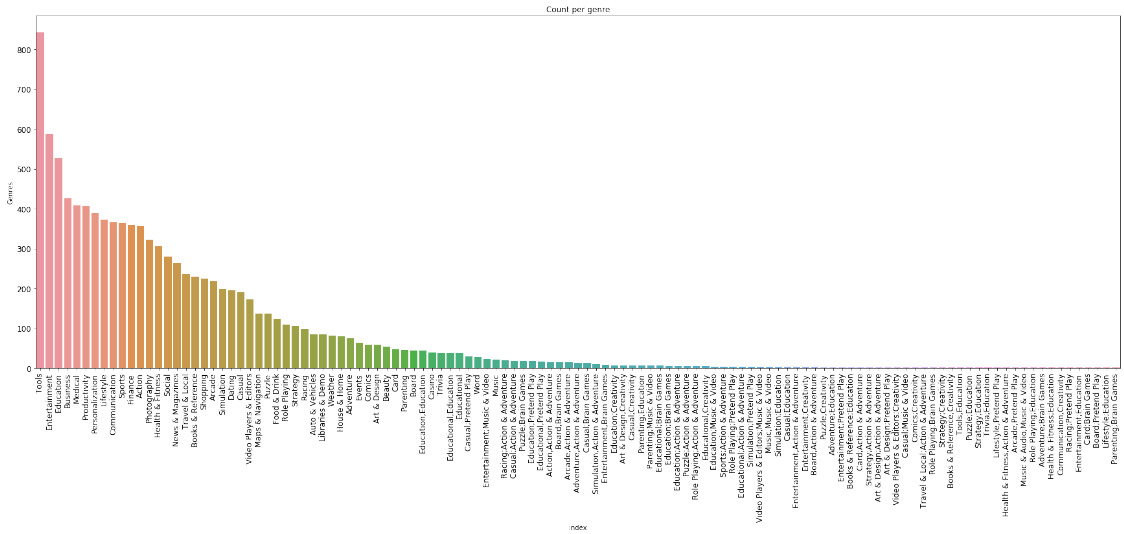
[357]:
```
plt.figure(figsize=(30, 10))
plt.title('Price per category');
plt.xticks(rotation = 90, fontsize=14);
plt.yticks(fontsize=14);
sns.barplot(x='Category', y='Price', data=df_no_nan, ci = None);
```



The price of the apps clearly depends on the category: the financial apps are sold at a much higher price than most of the other ones.

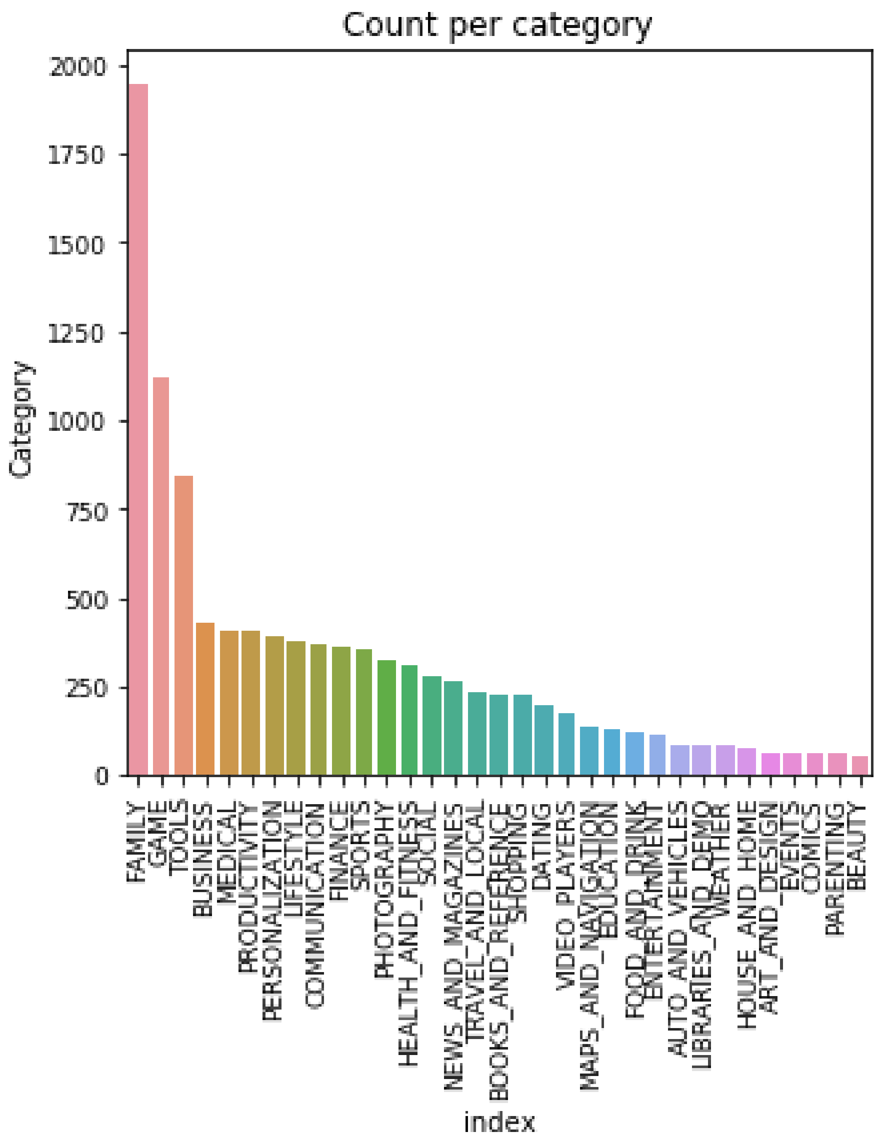
[358]:
```
plt.figure(figsize=(30, 10))
plt.title('Count per genre');
plt.xticks(rotation = 90, fontsize=12);
plt.yticks(fontsize=12)
```

```
sns.barplot(y=df['Genres'].value_counts().reset_index()['Genres'],␣
 ↪x=df['Genres'].value_counts().reset_index()[:]['index']);
```



The apps with a genre tools, entertainment and education represent the most part of the play store.

[359]:
```
plt.figure(figsize=(5, 5))
plt.title('Count per category');
plt.xticks(rotation = 90, fontsize=9);
plt.yticks(fontsize=9);
sns.barplot(y=df['Category'].value_counts().reset_index()['Category'],␣
 ↪x=df['Category'].value_counts().reset_index()['index']);
```

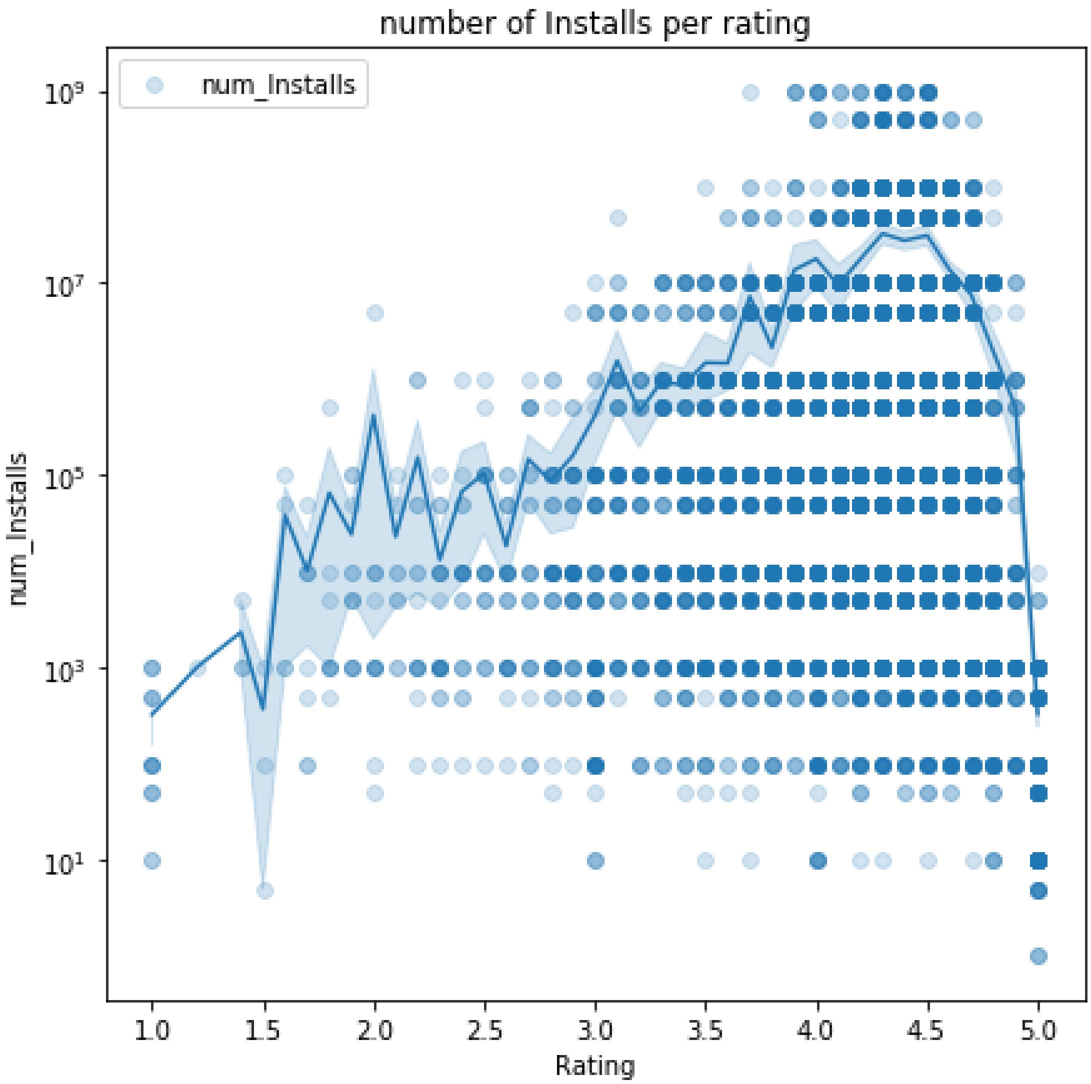


The categories the most represented is the app store are family, game and tools

[360]:
```
plt.figure(figsize=(7,7))
plt.title('number of Installs per rating')
plt.scatter( x=df_no_nan['Rating'], y=df_no_nan['num_Installs'], alpha = 0.2)
sns.lineplot(x="Rating", y="num_Installs", data=df_no_nan)
plt.yscale('log')
```

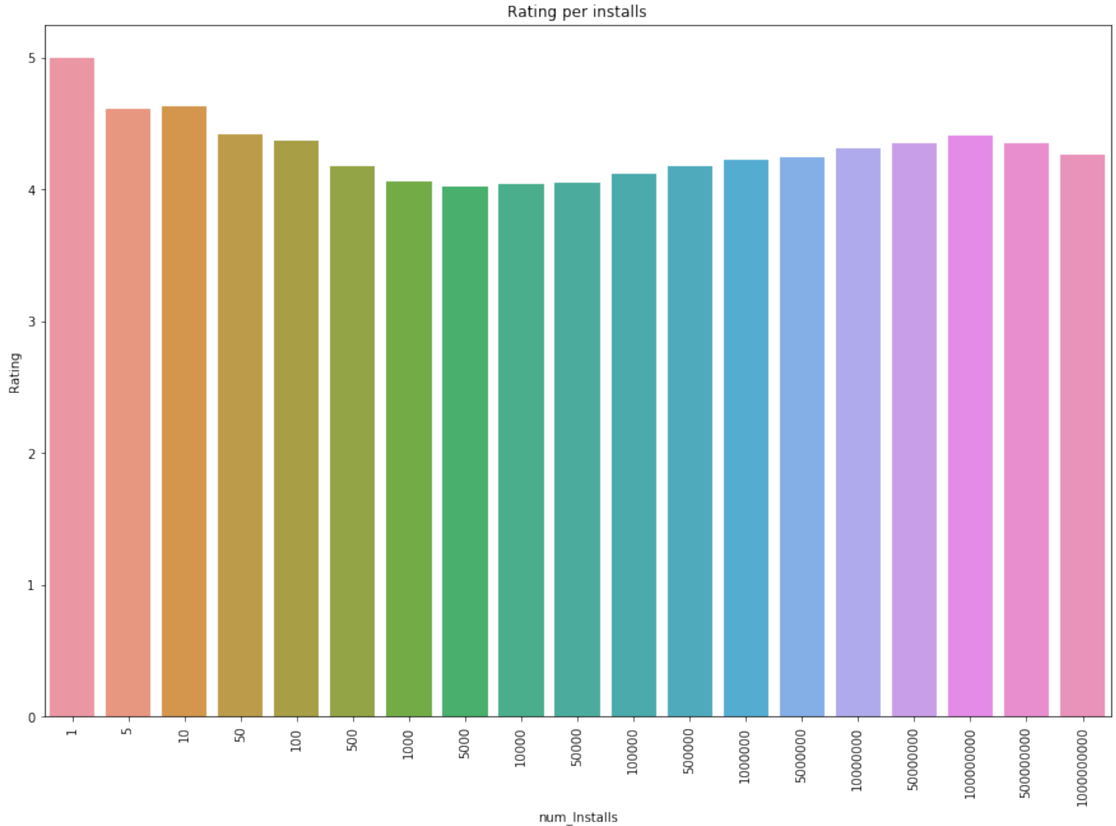
```
C:\Users\Soizic\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713:
FutureWarning: Using a non-tuple sequence for multidimensional indexing is
deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will
be interpreted as an array index, `arr[np.array(seq)]`, which will result either
```

```
in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

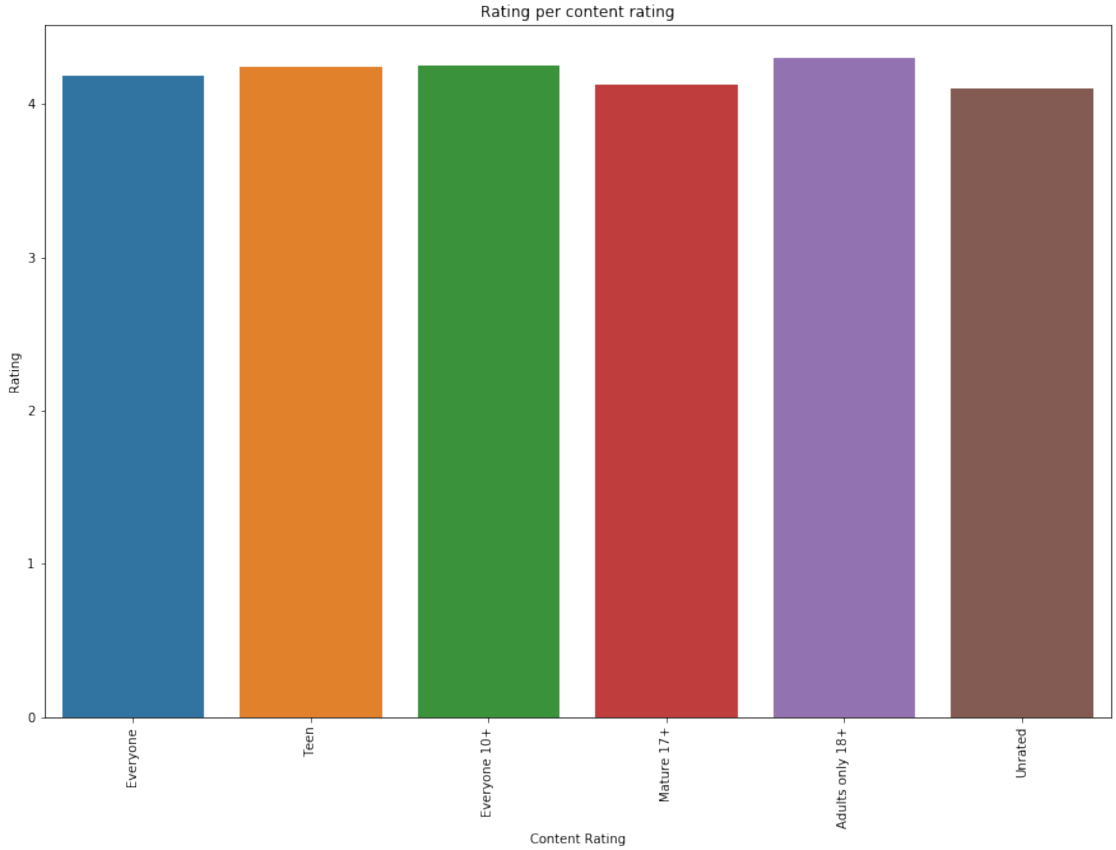


We can see here that the number of installs and the rating are proportional until a rating of approximately 4.5, After that the number of installs drops considerably, probably meaning that the apps with a rating greater than 4.5 are mostly installed by friends and family

[361]:
```
sns.barplot(x='num_Installs', y='Rating', data=df_no_nan, ci = None);
plt.title('Rating per installs');
plt.xticks(rotation = 90);
```

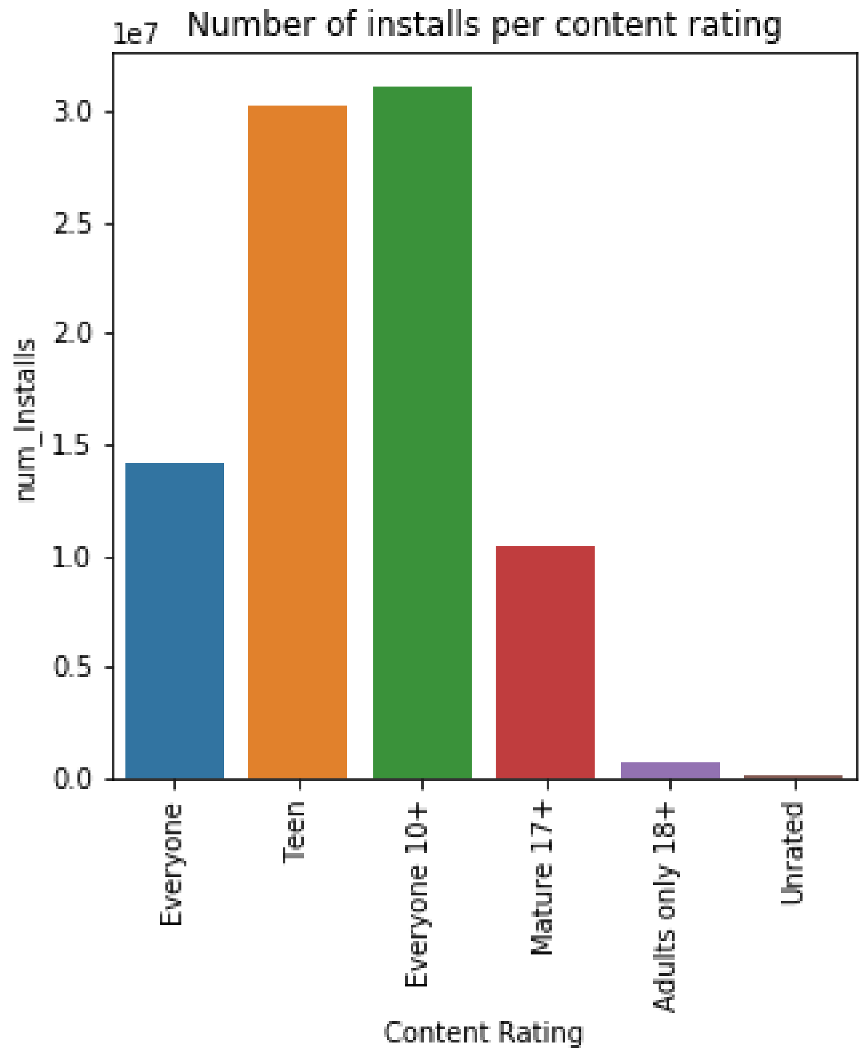


[362]:
```
sns.barplot(x='Content Rating', y='Rating', data=df_no_nan, ci = None);
plt.title('Rating per content rating');
plt.xticks(rotation = 90);
```



The content rating of an app does not impact much its rating in the play store

[363]:
```
plt.figure(figsize=(5,5))
plt.title('Number of installs per content rating')
sns.barplot(y="num_Installs", x="Content Rating", data=df_no_nan, ci=None)
plt.xticks(rotation = 90);
```



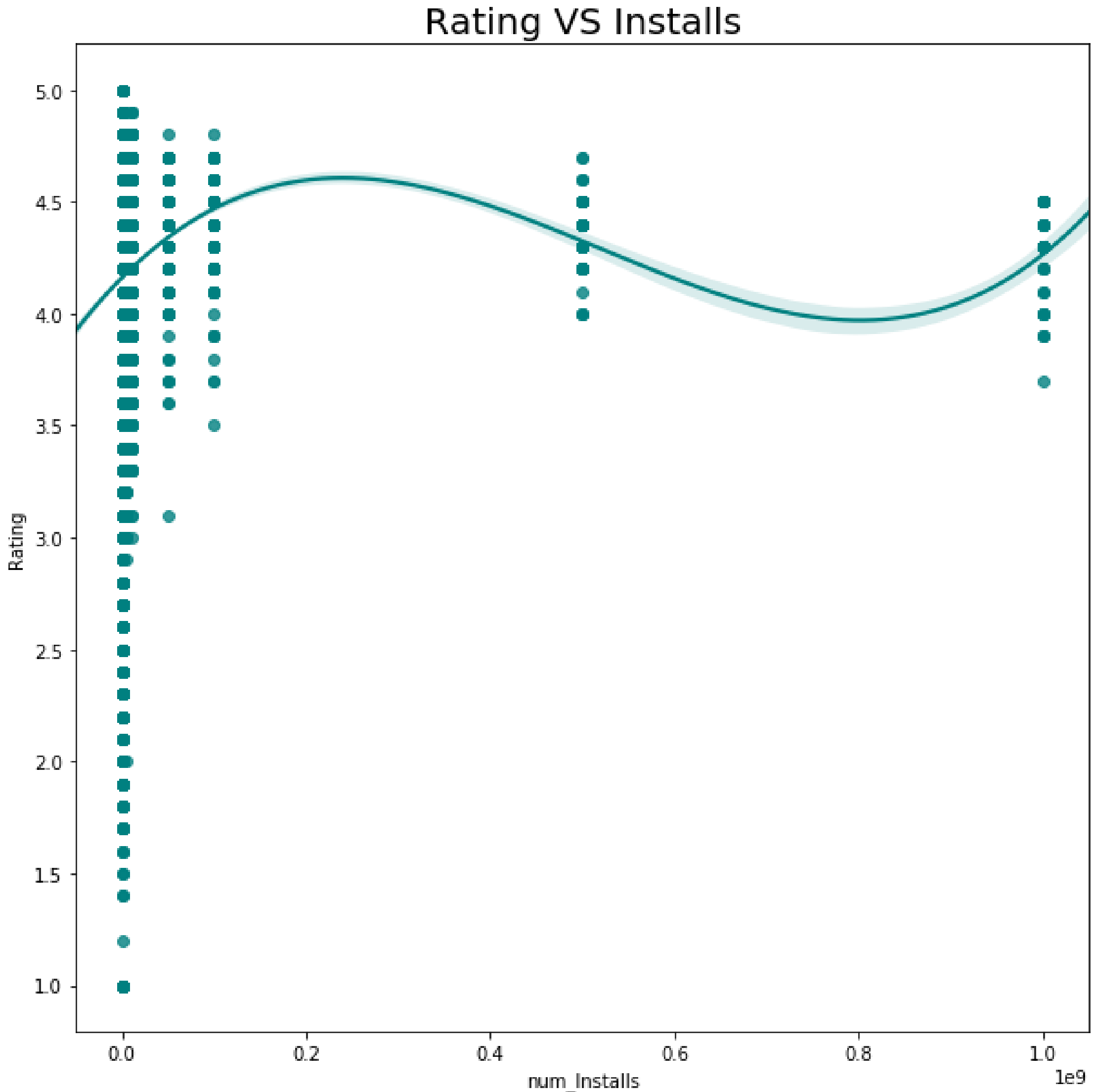
Though the apps targetting teenagers are much more installed than the other ones.

0.6 Modeling

[364]:
```
plt.figure(figsize = (10,10))
sns.regplot(x="num_Installs", y="Rating", color = 'teal',data=df_no_nan,␣
  ↪order=3);
plt.title('Rating VS Installs',size = 20)
```

```
C:\Users\Soizic\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713:
FutureWarning: Using a non-tuple sequence for multidimensional indexing is
deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will
be interpreted as an array index, `arr[np.array(seq)]`, which will result either
in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

[364]: Text(0.5,1,'Rating VS Installs')

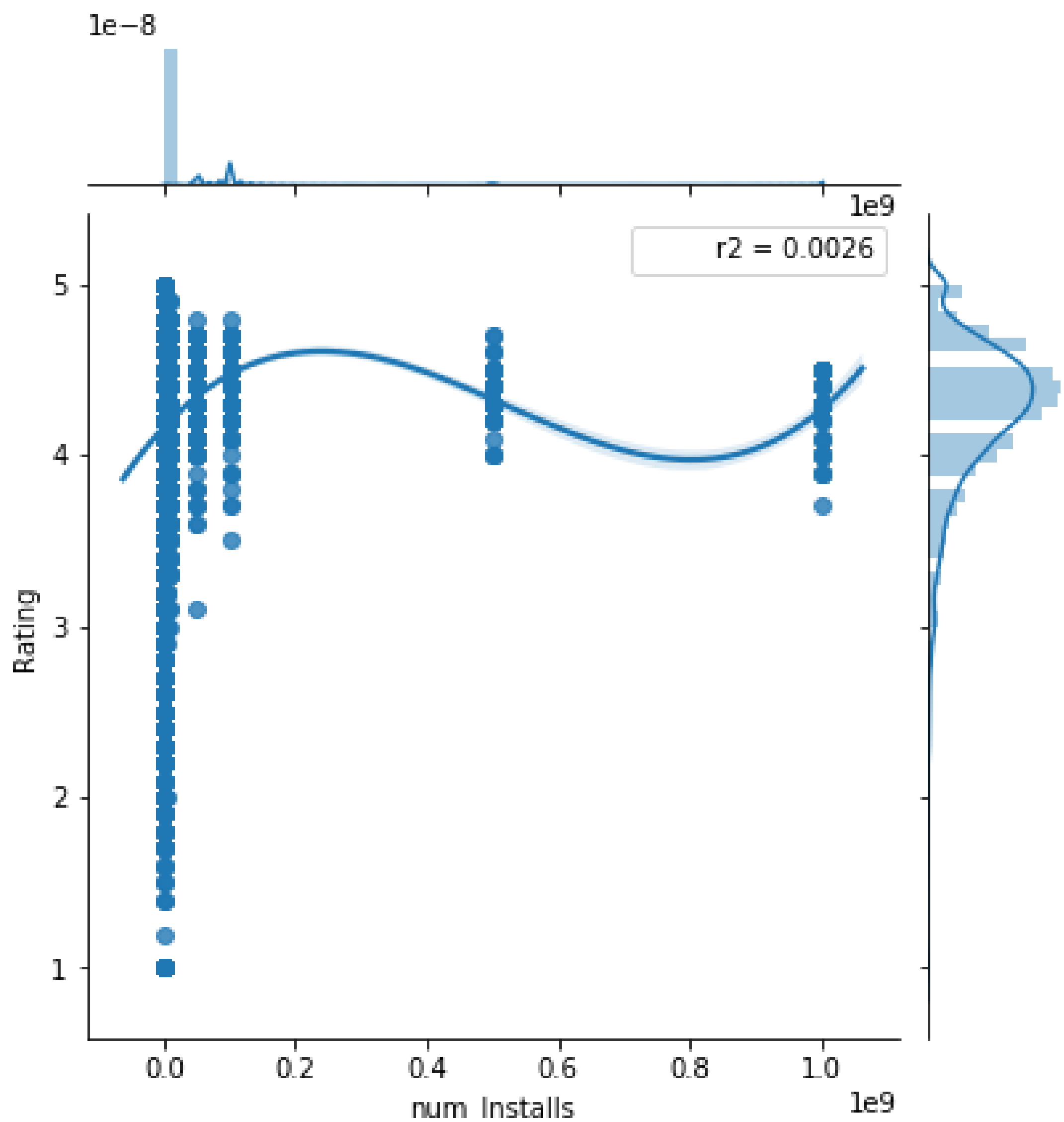


[365]:
```
def r2(x, y):
    return stats.pearsonr(x, y)[0] ** 2
sns.jointplot(df_no_nan['num_Installs'], df_no_nan['Rating'], kind="reg",␣
↪stat_func=r2, order=3)
```

```
C:\Users\Soizic\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713:
FutureWarning: Using a non-tuple sequence for multidimensional indexing is
deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will
be interpreted as an array index, `arr[np.array(seq)]`, which will result either
in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

```
C:\Users\Soizic\Anaconda3\lib\site-packages\seaborn\axisgrid.py:1847:
UserWarning: JointGrid annotation is deprecated and will be removed in a future
release.
  warnings.warn(UserWarning(msg))
```

[365]: <seaborn.axisgrid.JointGrid at 0x1842df9e0b8>



With an r2 close to 0, we can definitely conclude that trying to approximate the correlation between ratings and the number of installs with a polynomial regression of rank 3 is not the right approach

[ ]:

[366]:
```
df_no_nan.head()
```

[366]:
```
                                                 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     159.0   19M      10,000+  Free    0.0       Everyone
1     967.0   14M     500,000+  Free    0.0       Everyone
2   87510.0  8.7M   5,000,000+  Free    0.0       Everyone
3  215644.0   25M  50,000,000+  Free    0.0           Teen
4     967.0  2.8M     100,000+  Free    0.0       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  num_Installs  Perc_Reviewed  cumul_Installs_per_price
0  4.0.3 and up         10000       1.590000                     10000
1  4.0.3 and up        500000       0.193400                    510000
2  4.0.3 and up       5000000       1.750200                   5510000
3    4.2 and up      50000000       0.431288                  55510000
4    4.4 and up        100000       0.967000                  55610000
```

[367]:
```
df_no_nan.drop(['App','Installs','Type','Last Updated','Current Ver','Android␣
↪Ver'], axis=1, inplace = True)
```

[368]:
```
df_no_nan['Size'].unique()
```

[368]:
```
array(['19M', '14M', '8.7M', '25M', '2.8M', '5.6M', '29M', '33M', '3.1M',
       '28M', '12M', '20M', '21M', '37M', '5.5M', '17M', '39M', '31M',
       '4.2M', '23M', '6.0M', '6.1M', '4.6M', '9.2M', '5.2M', '11M',
       '24M', 'Varies with device', '9.4M', '15M', '10M', '1.2M', '26M',
       '8.0M', '7.9M', '56M', '57M', '35M', '54M', '201k', '3.6M', '5.7M',
       '8.6M', '2.4M', '27M', '2.7M', '2.5M', '7.0M', '16M', '3.4M',
       '8.9M', '3.9M', '2.9M', '38M', '32M', '5.4M', '18M', '1.1M',
       '2.2M', '4.5M', '9.8M', '52M', '9.0M', '6.7M', '30M', '2.6M',
       '7.1M', '22M', '6.4M', '3.2M', '8.2M', '4.9M', '9.5M', '5.0M',
       '5.9M', '13M', '73M', '6.8M', '3.5M', '4.0M', '2.3M', '2.1M',
       '42M', '9.1M', '55M', '23k', '7.3M', '6.5M', '1.5M', '7.5M', '51M',
       '41M', '48M', '8.5M', '46M', '8.3M', '4.3M', '4.7M', '3.3M', '40M',
       '7.8M', '8.8M', '6.6M', '5.1M', '61M', '66M', '79k', '8.4M',
       '3.7M', '118k', '44M', '695k', '1.6M', '6.2M', '53M', '1.4M',
```

```
'3.0M', '7.2M', '5.8M', '3.8M', '9.6M', '45M', '63M', '49M', '77M',
'4.4M', '70M', '9.3M', '8.1M', '36M', '6.9M', '7.4M', '84M', '97M',
'2.0M', '1.9M', '1.8M', '5.3M', '47M', '556k', '526k', '76M',
'7.6M', '59M', '9.7M', '78M', '72M', '43M', '7.7M', '6.3M', '334k',
'93M', '65M', '79M', '100M', '58M', '50M', '68M', '64M', '34M',
'67M', '60M', '94M', '9.9M', '232k', '99M', '624k', '95M', '8.5k',
'41k', '292k', '80M', '1.7M', '10.0M', '74M', '62M', '69M', '75M',
'98M', '85M', '82M', '96M', '87M', '71M', '86M', '91M', '81M',
'92M', '83M', '88M', '704k', '862k', '899k', '378k', '4.8M',
'266k', '375k', '1.3M', '975k', '980k', '4.1M', '89M', '696k',
'544k', '525k', '920k', '779k', '853k', '720k', '713k', '772k',
'318k', '58k', '241k', '196k', '857k', '51k', '953k', '865k',
'251k', '930k', '540k', '313k', '746k', '203k', '26k', '314k',
'239k', '371k', '220k', '730k', '756k', '91k', '293k', '17k',
'74k', '14k', '317k', '78k', '924k', '818k', '81k', '939k', '169k',
'45k', '965k', '90M', '545k', '61k', '283k', '655k', '714k', '93k',
'872k', '121k', '322k', '976k', '206k', '954k', '444k', '717k',
'210k', '609k', '308k', '306k', '175k', '350k', '383k', '454k',
'1.0M', '70k', '812k', '442k', '842k', '417k', '412k', '459k',
'478k', '335k', '782k', '721k', '430k', '429k', '192k', '460k',
'728k', '496k', '816k', '414k', '506k', '887k', '613k', '778k',
'683k', '592k', '186k', '840k', '647k', '373k', '437k', '598k',
'716k', '585k', '982k', '219k', '55k', '323k', '691k', '511k',
'951k', '963k', '25k', '554k', '351k', '27k', '82k', '208k',
'551k', '29k', '103k', '116k', '153k', '209k', '499k', '173k',
'597k', '809k', '122k', '411k', '400k', '801k', '787k', '50k',
'643k', '986k', '516k', '837k', '780k', '20k', '498k', '600k',
'656k', '221k', '228k', '176k', '34k', '259k', '164k', '458k',
'629k', '28k', '288k', '775k', '785k', '636k', '916k', '994k',
'309k', '485k', '914k', '903k', '608k', '500k', '54k', '562k',
'847k', '948k', '811k', '270k', '48k', '523k', '784k', '280k',
'24k', '892k', '154k', '18k', '33k', '860k', '364k', '387k',
'626k', '161k', '879k', '39k', '170k', '141k', '160k', '144k',
'143k', '190k', '376k', '193k', '473k', '246k', '73k', '253k',
'957k', '420k', '72k', '404k', '470k', '226k', '240k', '89k',
'234k', '257k', '861k', '467k', '676k', '552k', '582k', '619k'],
dtype=object)
```

[369]:
```
df_no_nan['Size'] = df_no_nan['Size'].apply(lambda x : x.replace('k', '000'))
df_no_nan['Size'] = df_no_nan['Size'].apply(lambda x : x.replace('M', '000000'))
df_no_nan['Size'] = df_no_nan['Size'].apply(lambda x : re.sub(r"\..", "", x))
df_no_nan['Size'] = df_no_nan['Size'].apply(lambda x : x.replace('Varies with␣
↪device', 'NAN'))
df_no_nan['Size'] = df_no_nan['Size'].astype('float64')
df_no_nan = df_no_nan.dropna(subset=['Size'])
df_no_nan['Size'].unique()
```

```
[369]: array([1.90e+07, 1.40e+07, 8.00e+06, 2.50e+07, 2.00e+06, 5.00e+06,
           2.90e+07, 3.30e+07, 3.00e+06, 2.80e+07, 1.20e+07, 2.00e+07,
           2.10e+07, 3.70e+07, 1.70e+07, 3.90e+07, 3.10e+07, 4.00e+06,
           2.30e+07, 6.00e+06, 9.00e+06, 1.10e+07, 2.40e+07, 1.50e+07,
           1.00e+07, 1.00e+06, 2.60e+07, 7.00e+06, 5.60e+07, 5.70e+07,
           3.50e+07, 5.40e+07, 2.01e+05, 2.70e+07, 1.60e+07, 3.80e+07,
           3.20e+07, 1.80e+07, 5.20e+07, 3.00e+07, 2.20e+07, 1.30e+07,
           7.30e+07, 4.20e+07, 5.50e+07, 2.30e+04, 5.10e+07, 4.10e+07,
           4.80e+07, 4.60e+07, 4.00e+07, 6.10e+07, 6.60e+07, 7.90e+04,
           1.18e+05, 4.40e+07, 6.95e+05, 5.30e+07, 4.50e+07, 6.30e+07,
           4.90e+07, 7.70e+07, 7.00e+07, 3.60e+07, 8.40e+07, 9.70e+07,
           4.70e+07, 5.56e+05, 5.26e+05, 7.60e+07, 5.90e+07, 7.80e+07,
           7.20e+07, 4.30e+07, 3.34e+05, 9.30e+07, 6.50e+07, 7.90e+07,
           1.00e+08, 5.80e+07, 5.00e+07, 6.80e+07, 6.40e+07, 3.40e+07,
           6.70e+07, 6.00e+07, 9.40e+07, 2.32e+05, 9.90e+07, 6.24e+05,
           9.50e+07, 8.00e+03, 4.10e+04, 2.92e+05, 8.00e+07, 7.40e+07,
           6.20e+07, 6.90e+07, 7.50e+07, 9.80e+07, 8.50e+07, 8.20e+07,
           9.60e+07, 8.70e+07, 7.10e+07, 8.60e+07, 9.10e+07, 8.10e+07,
           9.20e+07, 8.30e+07, 8.80e+07, 7.04e+05, 8.62e+05, 8.99e+05,
           3.78e+05, 2.66e+05, 3.75e+05, 9.75e+05, 9.80e+05, 8.90e+07,
           6.96e+05, 5.44e+05, 5.25e+05, 9.20e+05, 7.79e+05, 8.53e+05,
           7.20e+05, 7.13e+05, 7.72e+05, 3.18e+05, 5.80e+04, 2.41e+05,
           1.96e+05, 8.57e+05, 5.10e+04, 9.53e+05, 8.65e+05, 2.51e+05,
           9.30e+05, 5.40e+05, 3.13e+05, 7.46e+05, 2.03e+05, 2.60e+04,
           3.14e+05, 2.39e+05, 3.71e+05, 2.20e+05, 7.30e+05, 7.56e+05,
           9.10e+04, 2.93e+05, 1.70e+04, 7.40e+04, 1.40e+04, 3.17e+05,
           7.80e+04, 9.24e+05, 8.18e+05, 8.10e+04, 9.39e+05, 1.69e+05,
           4.50e+04, 9.65e+05, 9.00e+07, 5.45e+05, 6.10e+04, 2.83e+05,
           6.55e+05, 7.14e+05, 9.30e+04, 8.72e+05, 1.21e+05, 3.22e+05,
           9.76e+05, 2.06e+05, 9.54e+05, 4.44e+05, 7.17e+05, 2.10e+05,
           6.09e+05, 3.08e+05, 3.06e+05, 1.75e+05, 3.50e+05, 3.83e+05,
           4.54e+05, 7.00e+04, 8.12e+05, 4.42e+05, 8.42e+05, 4.17e+05,
           4.12e+05, 4.59e+05, 4.78e+05, 3.35e+05, 7.82e+05, 7.21e+05,
           4.30e+05, 4.29e+05, 1.92e+05, 4.60e+05, 7.28e+05, 4.96e+05,
           8.16e+05, 4.14e+05, 5.06e+05, 8.87e+05, 6.13e+05, 7.78e+05,
           6.83e+05, 5.92e+05, 1.86e+05, 8.40e+05, 6.47e+05, 3.73e+05,
           4.37e+05, 5.98e+05, 7.16e+05, 5.85e+05, 9.82e+05, 2.19e+05,
           5.50e+04, 3.23e+05, 6.91e+05, 5.11e+05, 9.51e+05, 9.63e+05,
           2.50e+04, 5.54e+05, 3.51e+05, 2.70e+04, 8.20e+04, 2.08e+05,
           5.51e+05, 2.90e+04, 1.03e+05, 1.16e+05, 1.53e+05, 2.09e+05,
           4.99e+05, 1.73e+05, 5.97e+05, 8.09e+05, 1.22e+05, 4.11e+05,
           4.00e+05, 8.01e+05, 7.87e+05, 5.00e+04, 6.43e+05, 9.86e+05,
           5.16e+05, 8.37e+05, 7.80e+05, 2.00e+04, 4.98e+05, 6.00e+05,
           6.56e+05, 2.21e+05, 2.28e+05, 1.76e+05, 3.40e+04, 2.59e+05,
           1.64e+05, 4.58e+05, 6.29e+05, 2.80e+04, 2.88e+05, 7.75e+05,
           7.85e+05, 6.36e+05, 9.16e+05, 9.94e+05, 3.09e+05, 4.85e+05,
           9.14e+05, 9.03e+05, 6.08e+05, 5.00e+05, 5.40e+04, 5.62e+05,
```

```
        8.47e+05, 9.48e+05, 8.11e+05, 2.70e+05, 4.80e+04, 5.23e+05,
        7.84e+05, 2.80e+05, 2.40e+04, 8.92e+05, 1.54e+05, 1.80e+04,
        3.30e+04, 8.60e+05, 3.64e+05, 3.87e+05, 6.26e+05, 1.61e+05,
        8.79e+05, 3.90e+04, 1.70e+05, 1.41e+05, 1.60e+05, 1.44e+05,
        1.43e+05, 1.90e+05, 3.76e+05, 1.93e+05, 4.73e+05, 2.46e+05,
        7.30e+04, 2.53e+05, 9.57e+05, 4.20e+05, 7.20e+04, 4.04e+05,
        4.70e+05, 2.26e+05, 2.40e+05, 8.90e+04, 2.34e+05, 2.57e+05,
        8.61e+05, 4.67e+05, 6.76e+05, 5.52e+05, 5.82e+05, 6.19e+05])
```

[370]:
```
df_no_nan.select_dtypes(include=['object']).columns
```

[370]:
```
Index(['Category', 'Content Rating', 'Genres'], dtype='object')
```

[371]:
```
df_no_nan['Reviews'] = df_no_nan['Reviews'].astype('float64')
```

[372]:
```
cat_vars = df_no_nan.select_dtypes(include=['object']).columns
for col in cat_vars:
    df_no_nan = pd.concat([df_no_nan.drop([col], axis=1), pd.
 ↪get_dummies(df_no_nan[col], prefix=col)], axis=1)
```

[373]:
```
df_no_nan.head()
```

[373]:
```
   Rating   Reviews        Size  Price  num_Installs  Perc_Reviewed  \
0     4.1     159.0  19000000.0    0.0         10000       1.590000
1     3.9     967.0  14000000.0    0.0        500000       0.193400
2     4.7   87510.0   8000000.0    0.0       5000000       1.750200
3     4.5  215644.0  25000000.0    0.0      50000000       0.431288
4     4.3     967.0   2000000.0    0.0        100000       0.967000

   cumul_Installs_per_price  Category_ART_AND_DESIGN  \
0                     10000                        1
1                    510000                        1
2                   5510000                        1
3                  55510000                        1
4                  55610000                        1

   Category_AUTO_AND_VEHICLES  Category_BEAUTY  …       \
0                           0                0  …
1                           0                0  …
2                           0                0  …
3                           0                0  …
4                           0                0  …

   Genres_Strategy;Education  Genres_Tools  Genres_Travel & Local  \
0                          0             0                      0
1                          0             0                      0
2                          0             0                      0
```

```
3                           0             0                       0
4                           0             0                       0

   Genres_Travel & Local;Action & Adventure  Genres_Trivia  \
0                                         0              0
1                                         0              0
2                                         0              0
3                                         0              0
4                                         0              0

   Genres_Video Players & Editors  Genres_Video Players & Editors;Creativity  \
0                               0                                          0
1                               0                                          0
2                               0                                          0
3                               0                                          0
4                               0                                          0

   Genres_Video Players & Editors;Music & Video  Genres_Weather  Genres_Word
0                                             0               0            0
1                                             0               0            0
2                                             0               0            0
3                                             0               0            0
4                                             0               0            0

[5 rows x 158 columns]
```

[374]:
```
y = df_no_nan['Rating']
X = df_no_nan.drop(['Rating'], axis=1)
```

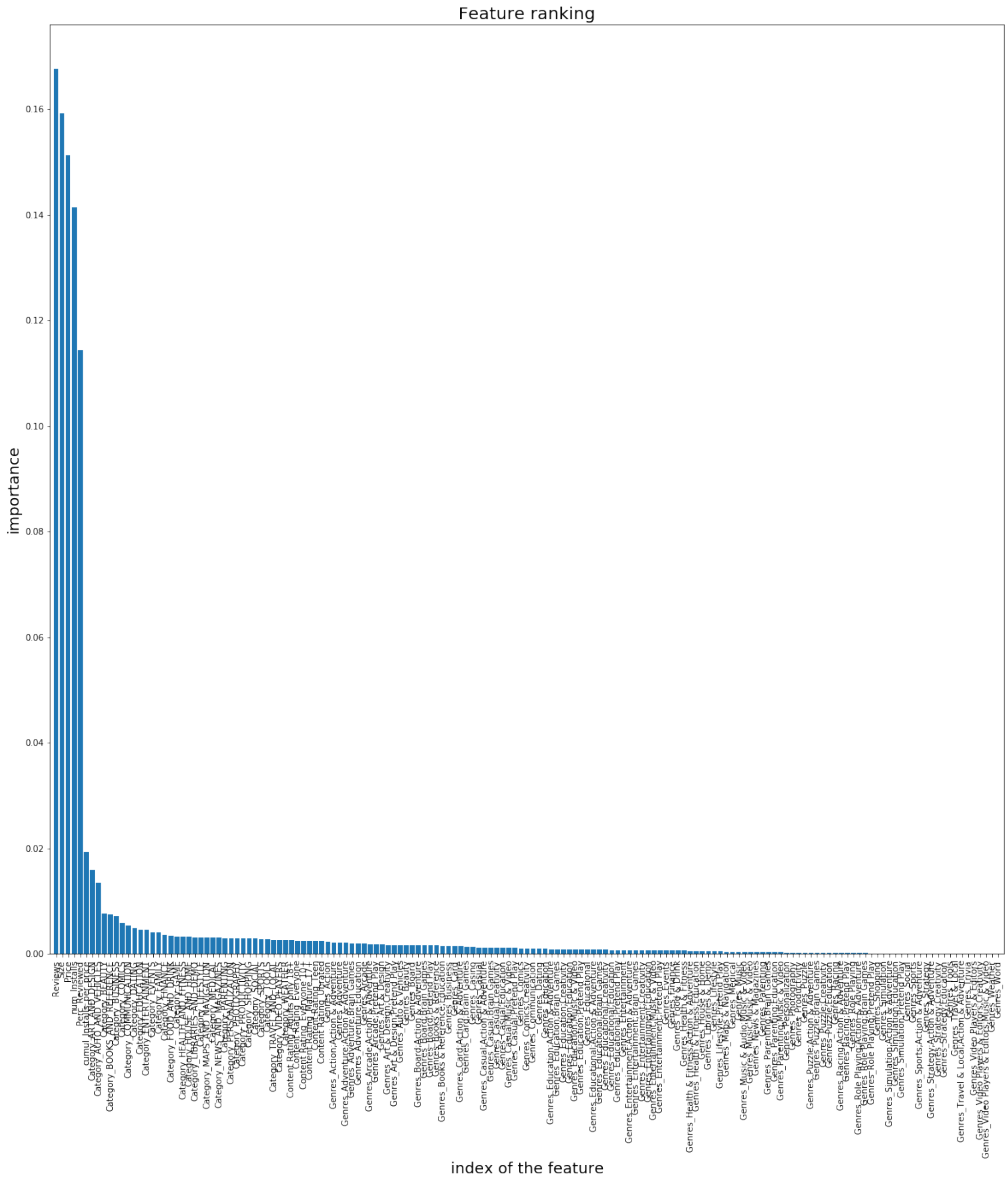
[375]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3,␣
 ↪random_state=42)

clf = RandomForestClassifier(n_estimators=100, max_depth=20,random_state=42)
lab_enc = preprocessing.LabelEncoder()
y_encoded = lab_enc.fit_transform(y_train)
clf.fit(X_train, y_encoded)

y_pred = clf.predict(X_test)
y_test_encoded = lab_enc.fit_transform(y_test)
accuracy_score(y_test_encoded, y_pred)
```

[375]:
```
0.10062893081761007
```

With an accuracy of 11% (the best I was able to achieve when playing with the parameters), we cannot really predict our rating based on the other information with have in the dataset.

[376]:

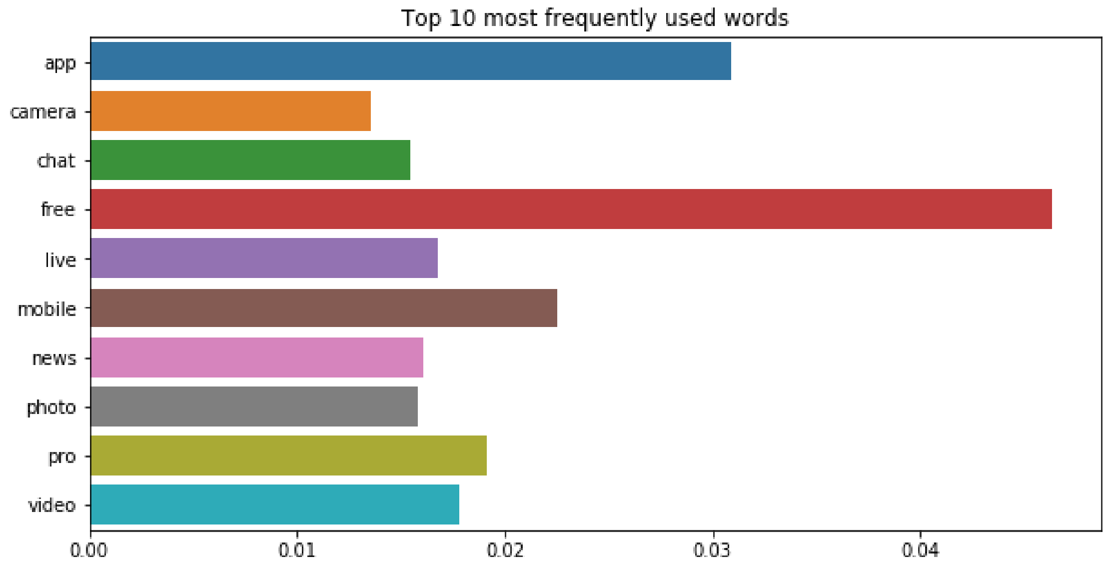
```
importances = clf.feature_importances_
indices = np.argsort(importances)[::-1]
feature_names = df_no_nan.drop(['Rating'], axis=1).columns
f, ax = plt.subplots(figsize=(20,20));
plt.title("Feature ranking", fontsize = 20);
plt.bar(range(X_train.shape[1]), importances[indices],align="center");
plt.xticks(range(X_train.shape[1]), indices);
plt.xlim([-1, X_train.shape[1]]);
plt.ylabel("importance", fontsize = 18);
plt.xlabel("index of the feature", fontsize = 18);
plt.xticks(range(X_train.shape[1]), feature_names, rotation=90);
```



The top 5 elements that help us predict with an accuracy of 11% are the number of reviews, the size of the app, the price, the number of intallations and the percentage reviewed, which makes sense: - the more an application is liked, the more it will be installed and reviewed - the size of the app might be linked to the graphics people tend to like more sofisticated designs.

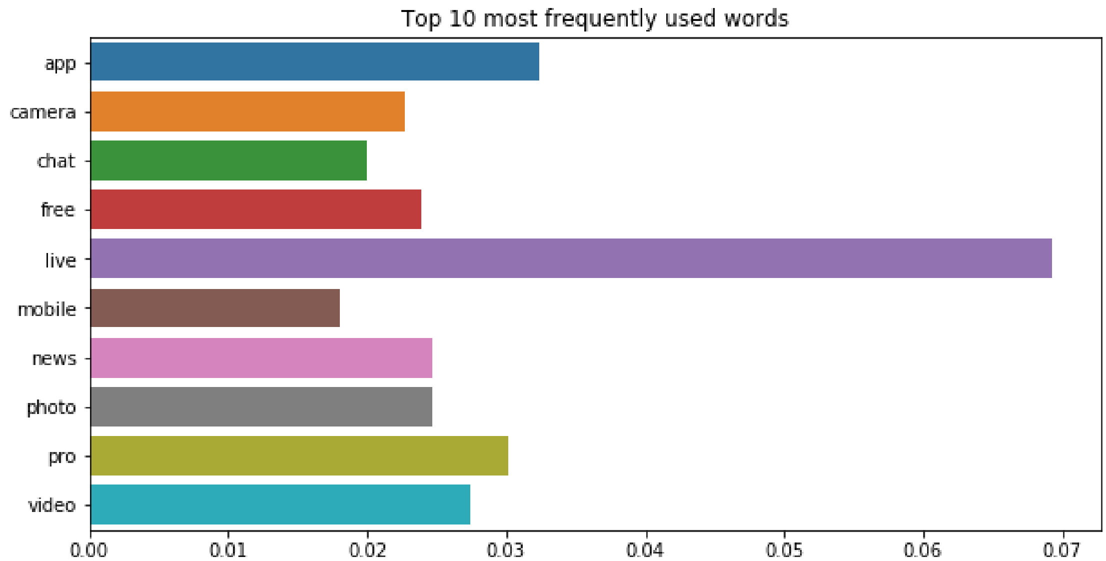
Let's see if we can find anything with the name of the app

[377]:
```
model = CountVectorizer(max_features=10, stop_words='english')
X = model.fit_transform(list(df['App']))

plt.figure(figsize=(10,5))
plt.title('Top 10 most frequently used words')
sns.barplot(x=X.toarray().mean(axis=0), y=vectorizer.get_feature_names());
```



More than 4% of the apps in the Google play store have the word free in their name! The 2 other most important keywords are app and mobile.

[378]:
```
df['num_Installs'] = df['Installs'].apply({'10,000+':10000, '500,000+':500000,␣
↪'5,000,000+':5000000,'50,000,000+':50000000, '100,000+':100000, '50,000+':
↪50000, '1,000,000+':1000000, '10,000,000+':10000000, '5,000+':5000,␣
↪'100,000,000+':100000000, '1,000,000,000+':1000000000, '1,000+':1000,␣
↪'500,000,000+':500000000, '50+':50, '100+':100, '500+':500, '10+':10, '1+':
↪1, '5+':5, '0+':0 }.get)
X = model.fit_transform(list(df[df['num_Installs']>=1000000]['App']))

plt.figure(figsize=(10,5))
plt.title('Top 10 most frequently used words')
sns.barplot(x=X.toarray().mean(axis=0), y=vectorizer.get_feature_names());
```



But if we look at only at the apps that have been installed 1,000,000+ times, then the most important keyword is "live".

```
[ ]:
```