TP1 GooglePlay #Web&Commerce

Analyse des facteurs de succès des applications mobiles

Le jeu de données a été construit par du **Web Scraping** sur plus de 10.000 applications afin d'analyser le marché de l'Android.

- Quelles sont les catégories proposant le plus d'applications sur le marché ?
- Est-ce que toutes les catégories d'applications suivent le même motif de performance d'évaluation ?
- Quelles sont les catégories d'applications les plus téléchargées sur le marché ?
- Quels sont les facteurs saillants permettant d'obtenir la meilleure note d'évaluation ?
- Quels sont les facteurs saillants permettant de devenir populaire (augmenter son nombre de téléchargement) ?

Pour répondre à ces questions non exhaustives, mettez en œuvre une fouille de données

- Nettoyage des données
- Exploration des données uni-variables, bi-variables
- Visualisation
- Analyse

In [1]:

```
#Librairies
import pandas as pd
import numpy as np
```

In [2]:

```
#Importation du fichier CSV
df = pd.read_csv('../input/googleplaystore.csv')
```

In [3]:

```
df.dtypes
```

Out[3]:

```
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
```

In [4]:

```
df.shape
```

Out[4]:

```
(10841, 13)
```

In [5]:

```
#Nombre de lignes
df.shape[0]
```

Out[5]:

```
10841
```

In [6]:

```
#Nombre de variables
df.shape[1]
```

Out[6]:

```
13
```

In [7]:

```
#Les 10 premières valeurs du vecteur App
df['App'][0:10]
```

Out[7]:

```
0       Photo Editor & Candy Camera & Grid & ScrapBook
1                                  Coloring book moana
2    U Launcher Lite – FREE Live Cool Themes, Hide ...
3                                Sketch - Draw & Paint
4                   Pixel Draw - Number Art Coloring Book
5                           Paper flowers instructions
6              Smoke Effect Photo Maker - Smoke Editor
7                                     Infinite Painter
8                                  Garden Coloring Book
9                         Kids Paint Free - Drawing Fun
Name: App, dtype: object
```

In [8]:

```
#Vecteur des noms de variables du dataset
df.columns
```

Out[8]:

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

In [9]:

```
#Equivalent de df['App'][0:10]
df[df.columns[0]][0:10]
```

Out[9]:

```
0       Photo Editor & Candy Camera & Grid & ScrapBook
1                                  Coloring book moana
2    U Launcher Lite – FREE Live Cool Themes, Hide ...
3                                Sketch - Draw & Paint
4                   Pixel Draw - Number Art Coloring Book
5                           Paper flowers instructions
6              Smoke Effect Photo Maker - Smoke Editor
7                                     Infinite Painter
8                                  Garden Coloring Book
9                       Kids Paint Free - Drawing Fun
Name: App, dtype: object
```

In [10]:

```
#Affiche un échantillon de 7 applications tiré au hasard
df.sample(7)
```

Out[10]:

	App	Category	Rating	Reviews	Size	Installs	Type	Pr
2756	AliExpress - Smarter Shopping, Better Living	SHOPPING	4.6	5916569	Varies with device	100,000,000+	Free	
431	Viber Messenger	COMMUNICATION	4.3	11334973	Varies with device	500,000,000+	Free	
9228	EC Security	LIFESTYLE	4.5	13	28M	1,000+	Free	
6913	Bw Events	EVENTS	4.4	16	1.7M	100+	Free	
6710	B R COACHINGS	FAMILY	NaN	1	9.2M	10+	Free	
11	Name Art Photo Editor - Focus n Filters	ART_AND_DESIGN	4.4	8788	12M	1,000,000+	Free	
6244	B y H Niños ES	BOOKS_AND_REFERENCE	4.6	53	16M	5,000+	Free	

Le dataset est composé de 13 variables. La description du dataframe semble indiquer qu'il n'y a qu'une seule variable quantitative (rating à float64). Pourtant, avec la connaissance métier, plusieurs autres variables peuvent être quantitatives.

1 variable id : app

5 variables quantitatives : rating (le nombre d'étoile), reviews (le nombre d'évaluation), installs, price, size

7 variables descriptives discrètes : category, type (2 modalités), content rating, genres, last updated, current ver, android ver

Etape 1 : Nettoyer le dataset

In [11]:

```
#Est-ce qu'il y a des valeurs manquantes ?
df.isnull().any()
```

Out[11]:

```
App               False
Category          False
Rating             True
Reviews           False
Size              False
Installs          False
Type               True
Price             False
Content Rating     True
Genres            False
Last Updated      False
Current Ver        True
Android Ver        True
dtype: bool
```

In [12]:

```
#Connaître le ratio de valeurs manquantes
df.isnull().sum() / len(df)
```

Out[12]:

```
App               0.000000
Category          0.000000
Rating            0.135965
Reviews           0.000000
Size              0.000000
Installs          0.000000
Type              0.000092
Price             0.000000
Content Rating    0.000092
Genres            0.000000
Last Updated      0.000000
Current Ver       0.000738
Android Ver       0.000277
dtype: float64
```

Vous remarquerez que l'application de l'index 10472 est aberrante dans la mesure où :

- son évaluation est de 19 étoiles
- son nombre de review est de 3.0M
- sa taille est de 1,000+
- son nombre d'installation est Free
- etc.

En effet, les variables descriptives semblent être décalées pour cet individu. Nous identifions deux possibilités : (i) tout décaler mais nous n'aurons pas la catégorie ou (ii) supprimer cet individu.

In [13]:

```
#Donnée abérante
df.loc[10472]
```

Out[13]:

```
App               Life Made WI-Fi Touchscreen Photo Frame
Category                                              1.9
Rating                                                 19
Reviews                                              3.0M
Size                                               1,000+
Installs                                             Free
Type                                                    0
Price                                            Everyone
Content Rating                                        NaN
Genres                                  February 11, 2018
Last Updated                                       1.0.19
Current Ver                                    4.0 and up
Android Ver                                           NaN
Name: 10472, dtype: object
```

In [14]:

```
#Supprimer l'individu aberrant
#df = df.drop(10472)

#retire les tuples ayant la modalité 'Free'
df = df[df['Installs'] != 'Free']
#En effet, la modalité "Free" est normalement dans la variable "Type"
```

In [15]:

```
#Exemple de doublons
df.loc[df.App=='Tiny Scanner - PDF Scanner App']
```

Out[15]:

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres
231	Tiny Scanner - PDF Scanner App	BUSINESS	4.7	286897	39M	10,000,000+	Free	0	Everyone	Business
287	Tiny Scanner - PDF Scanner App	BUSINESS	4.7	286897	39M	10,000,000+	Free	0	Everyone	Business

Il y a des applications en doublons, par exemple "Tiny Scanner - PDF Scanner App" est présent à la ligne 231 et 287.

In [16]:

```
#Nombre de lignes avant le drop_duplicates
df.shape[0]
```

Out[16]:

```
10840
```

In [17]:

```
#Return DataFrame with duplicate rows removed, optionally only considering certain columns.
df.drop_duplicates(subset='App', inplace=True)
```

In [18]:

```
#Nombre de lignes après le drop_duplicates
df.shape[0]
```

Out[18]:

```
9659
```

1) Pour la variable 'Installs'

In [19]:

```
#affiche toutes les modalités
np.unique(df['Installs'])
```

Out[19]:

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

In [20]:

```
#retirer les '+' et les ','
df['Installs'] = df['Installs'].apply(lambda x: x.replace('+', ''))
df['Installs'] = df['Installs'].apply(lambda x: x.replace(',', ''))

#convertir en integer
df['Installs'] = df['Installs'].apply(lambda x: int(x)) #Bous pouvez essayez d'exécuter cet

print(type(df['Installs'].values))
```

```
<class 'numpy.ndarray'>
```

In [21]:

```
df.dtypes
```

Out[21]:

```
App                object
Category           object
Rating            float64
Reviews            object
Size               object
Installs            int64
Type               object
Price              object
Content Rating     object
Genres             object
Last Updated       object
Current Ver        object
Android Ver        object
dtype: object
```

In [22]:

```
#Convertir en float
df['Installs'] = df['Installs'].apply(lambda x: float(x))
```

In [23]:

```
df.dtypes
```

Out[23]:

```
App                object
Category           object
Rating            float64
Reviews            object
Size               object
Installs          float64
Type               object
Price              object
Content Rating     object
Genres             object
Last Updated       object
Current Ver        object
Android Ver        object
dtype: object
```

In [24]:

```
#Est-ce qu'il y a des données manquantes ?
any(pd.isna(df['Installs']))
```

Out[24]:

```
False
```

2) Pour la variable 'Size'

In [25]:

```
#Afficher toutes les modalités
np.unique(df['Size'])
```

Out[25]:

```
array(['1.0M', '1.1M', '1.2M', '1.3M', '1.4M', '1.5M', '1.6M', '1.7M',
       '1.8M', '1.9M', '10.0M', '100M', '1020k', '103k', '108k', '10M',
       '116k', '118k', '11M', '11k', '121k', '122k', '12M', '13M', '141k',
       '143k', '144k', '14M', '14k', '153k', '154k', '157k', '15M',
       '160k', '161k', '164k', '169k', '16M', '170k', '172k', '173k',
       '175k', '176k', '17M', '17k', '186k', '18M', '18k', '190k', '191k',
       '192k', '193k', '196k', '19M', '2.0M', '2.1M', '2.2M', '2.3M',
       '2.4M', '2.5M', '2.6M', '2.7M', '2.8M', '2.9M', '200k', '201k',
       '203k', '206k', '208k', '209k', '20M', '20k', '210k', '219k',
       '21M', '220k', '221k', '222k', '226k', '228k', '22M', '232k',
       '234k', '237k', '238k', '239k', '23M', '23k', '240k', '241k',
       '243k', '245k', '246k', '24M', '24k', '251k', '253k', '257k',
       '259k', '25M', '25k', '266k', '269k', '26M', '26k', '270k', '27M',
       '27k', '280k', '283k', '288k', '28M', '28k', '292k', '293k', '29M',
       '29k', '3.0M', '3.1M', '3.2M', '3.3M', '3.4M', '3.5M', '3.6M',
       '3.7M', '3.8M', '3.9M', '306k', '308k', '309k', '30M', '313k',
       '314k', '317k', '318k', '319k', '31M', '322k', '323k', '329k',
       '32M', '334k', '335k', '33M', '33k', '34M', '34k', '350k', '351k',
       '353k', '35M', '364k', '36M', '371k', '373k', '375k', '376k',
       '378k', '37M', '383k', '387k', '38M', '39M', '39k', '4.0M', '4.1M',
       '4.2M', '4.3M', '4.4M', '4.5M', '4.6M', '4.7M', '4.8M', '4.9M',
       '400k', '404k', '40M', '411k', '412k', '414k', '417k', '41M',
       '41k', '420k', '421k', '429k', '42M', '430k', '437k', '43M',
       '442k', '444k', '44M', '44k', '454k', '458k', '459k', '45M', '45k',
       '460k', '467k', '46M', '470k', '473k', '475k', '478k', '47M',
       '485k', '48M', '48k', '496k', '498k', '499k', '49M', '5.0M',
       '5.1M', '5.2M', '5.3M', '5.4M', '5.5M', '5.6M', '5.7M', '5.8M',
       '5.9M', '500k', '506k', '50M', '50k', '511k', '514k', '516k',
       '518k', '51M', '51k', '523k', '525k', '526k', '52M', '53M', '540k',
       '544k', '545k', '549k', '54M', '54k', '551k', '552k', '554k',
       '556k', '55M', '55k', '562k', '569k', '56M', '57M', '582k', '585k',
       '58M', '58k', '592k', '597k', '598k', '59M', '6.0M', '6.1M',
       '6.2M', '6.3M', '6.4M', '6.5M', '6.6M', '6.7M', '6.8M', '6.9M',
       '600k', '601k', '608k', '609k', '60M', '613k', '619k', '61M',
       '61k', '624k', '626k', '629k', '62M', '636k', '63M', '642k',
       '643k', '647k', '64M', '655k', '656k', '658k', '65M', '663k',
       '66M', '676k', '67M', '67k', '683k', '688k', '68M', '691k', '695k',
       '696k', '69M', '7.0M', '7.1M', '7.2M', '7.3M', '7.4M', '7.5M',
       '7.6M', '7.7M', '7.8M', '7.9M', '704k', '705k', '70M', '70k',
       '713k', '714k', '716k', '717k', '71M', '720k', '721k', '728k',
       '72M', '72k', '730k', '73M', '73k', '743k', '746k', '749k', '74M',
       '74k', '754k', '756k', '75M', '76M', '772k', '775k', '778k',
       '779k', '77M', '780k', '782k', '784k', '785k', '787k', '78M',
       '78k', '79M', '79k', '8.0M', '8.1M', '8.2M', '8.3M', '8.4M',
       '8.5M', '8.5k', '8.6M', '8.7M', '8.8M', '8.9M', '801k', '809k',
       '80M', '811k', '812k', '816k', '818k', '81M', '81k', '82M', '82k',
       '837k', '83M', '840k', '842k', '847k', '84M', '853k', '857k',
       '85M', '860k', '861k', '862k', '865k', '86M', '872k', '874k',
       '879k', '87M', '881k', '885k', '887k', '88M', '892k', '898k',
       '899k', '89M', '89k', '9.0M', '9.1M', '9.2M', '9.3M', '9.4M',
       '9.5M', '9.6M', '9.7M', '9.8M', '9.9M', '902k', '903k', '904k',
       '90M', '913k', '914k', '916k', '91M', '91k', '920k', '921k',
       '924k', '92M', '930k', '939k', '93M', '93k', '940k', '942k',
       '948k', '94M', '951k', '953k', '954k', '957k', '95M', '961k',
```

```
       '963k', '965k', '96M', '970k', '975k', '976k', '97M', '97k',
       '980k', '981k', '982k', '986k', '98M', '992k', '994k', '99M',
       'Varies with device'], dtype=object)
```

In [26]:

```
#Est-ce qu'il existe au moins un tuple qui a la variable Size à la modalité 'Varies with de
any(df['Size']=='Varies with device')
```

Out[26]:

```
True
```

In [27]:

```
#Remplacer la modalité 'Varies with device' par 'NaN'
df['Size'] = df['Size'].apply(lambda x: str(x).replace('Varies with device', 'NaN'))

#Afficher toutes les modalités
np.unique(df['Size'])
```

Out[27]:

```
array(['1.0M', '1.1M', '1.2M', '1.3M', '1.4M', '1.5M', '1.6M', '1.7M',
       '1.8M', '1.9M', '10.0M', '100M', '1020k', '103k', '108k', '10M',
       '116k', '118k', '11M', '11k', '121k', '122k', '12M', '13M', '141k',
       '143k', '144k', '14M', '14k', '153k', '154k', '157k', '15M',
       '160k', '161k', '164k', '169k', '16M', '170k', '172k', '173k',
       '175k', '176k', '17M', '17k', '186k', '18M', '18k', '190k', '191k',
       '192k', '193k', '196k', '19M', '2.0M', '2.1M', '2.2M', '2.3M',
       '2.4M', '2.5M', '2.6M', '2.7M', '2.8M', '2.9M', '200k', '201k',
       '203k', '206k', '208k', '209k', '20M', '20k', '210k', '219k',
       '21M', '220k', '221k', '222k', '226k', '228k', '22M', '232k',
       '234k', '237k', '238k', '239k', '23M', '23k', '240k', '241k',
       '243k', '245k', '246k', '24M', '24k', '251k', '253k', '257k',
       '259k', '25M', '25k', '266k', '269k', '26M', '26k', '270k', '27M',
       '27k', '280k', '283k', '288k', '28M', '28k', '292k', '293k', '29M',
       '29k', '3.0M', '3.1M', '3.2M', '3.3M', '3.4M', '3.5M', '3.6M',
       '3.7M', '3.8M', '3.9M', '306k', '308k', '309k', '30M', '313k',
       '314k', '317k', '318k', '319k', '31M', '322k', '323k', '329k',
       '32M', '334k', '335k', '33M', '33k', '34M', '34k', '350k', '351k',
       '353k', '35M', '364k', '36M', '371k', '373k', '375k', '376k',
       '378k', '37M', '383k', '387k', '38M', '39M', '39k', '4.0M', '4.1M',
       '4.2M', '4.3M', '4.4M', '4.5M', '4.6M', '4.7M', '4.8M', '4.9M',
       '400k', '404k', '40M', '411k', '412k', '414k', '417k', '41M',
       '41k', '420k', '421k', '429k', '42M', '430k', '437k', '43M',
       '442k', '444k', '44M', '44k', '454k', '458k', '459k', '45M', '45k',
       '460k', '467k', '46M', '470k', '473k', '475k', '478k', '47M',
       '485k', '48M', '48k', '496k', '498k', '499k', '49M', '5.0M',
       '5.1M', '5.2M', '5.3M', '5.4M', '5.5M', '5.6M', '5.7M', '5.8M',
       '5.9M', '500k', '506k', '50M', '50k', '511k', '514k', '516k',
       '518k', '51M', '51k', '523k', '525k', '526k', '52M', '53M', '540k',
       '544k', '545k', '549k', '54M', '54k', '551k', '552k', '554k',
       '556k', '55M', '55k', '562k', '569k', '56M', '57M', '582k', '585k',
       '58M', '58k', '592k', '597k', '598k', '59M', '6.0M', '6.1M',
       '6.2M', '6.3M', '6.4M', '6.5M', '6.6M', '6.7M', '6.8M', '6.9M',
       '600k', '601k', '608k', '609k', '60M', '613k', '619k', '61M',
       '61k', '624k', '626k', '629k', '62M', '636k', '63M', '642k',
       '643k', '647k', '64M', '655k', '656k', '658k', '65M', '663k',
       '66M', '676k', '67M', '67k', '683k', '688k', '68M', '691k', '695k',
       '696k', '69M', '7.0M', '7.1M', '7.2M', '7.3M', '7.4M', '7.5M',
       '7.6M', '7.7M', '7.8M', '7.9M', '704k', '705k', '70M', '70k',
       '713k', '714k', '716k', '717k', '71M', '720k', '721k', '728k',
       '72M', '72k', '730k', '73M', '73k', '743k', '746k', '749k', '74M',
       '74k', '754k', '756k', '75M', '76M', '772k', '775k', '778k',
       '779k', '77M', '780k', '782k', '784k', '785k', '787k', '78M',
       '78k', '79M', '79k', '8.0M', '8.1M', '8.2M', '8.3M', '8.4M',
       '8.5M', '8.5k', '8.6M', '8.7M', '8.8M', '8.9M', '801k', '809k',
       '80M', '811k', '812k', '816k', '818k', '81M', '81k', '82M', '82k',
       '837k', '83M', '840k', '842k', '847k', '84M', '853k', '857k',
       '85M', '860k', '861k', '862k', '865k', '86M', '872k', '874k',
       '879k', '87M', '881k', '885k', '887k', '88M', '892k', '898k',
       '899k', '89M', '89k', '9.0M', '9.1M', '9.2M', '9.3M', '9.4M',
       '9.5M', '9.6M', '9.7M', '9.8M', '9.9M', '902k', '903k', '904k',
```

```
'90M', '913k', '914k', '916k', '91M', '91k', '920k', '921k',
'924k', '92M', '930k', '939k', '93M', '93k', '940k', '942k',
'948k', '94M', '951k', '953k', '954k', '957k', '95M', '961k',
'963k', '965k', '96M', '970k', '975k', '976k', '97M', '97k',
'980k', '981k', '982k', '986k', '98M', '992k', '994k', '99M',
'NaN'], dtype=object)
```

In [28]:

```
#Convertir la taille en MB

#---pour les M---
#Retirer 'M'
df['Size'] = df['Size'].apply(lambda x: str(x).replace('M', ''))
df['Size'] = df['Size'].apply(lambda x: str(x).replace(',', ''))

#---Pour les k---
#retirer k et ramener à l'échelle M
df['Size'] = df['Size'].apply(lambda x: float(str(x).replace('k', '')) / 1000 if 'k' in str
```

In [29]:

```
df.dtypes
```

Out[29]:

```
App                object
Category           object
Rating            float64
Reviews            object
Size               object
Installs          float64
Type               object
Price              object
Content Rating     object
Genres             object
Last Updated       object
Current Ver        object
Android Ver        object
dtype: object
```

In [30]:

```
#Convernir en float
df['Size'] = df['Size'].apply(lambda x: float(x))
```

In [31]:

```
df.dtypes
```

Out[31]:

```
App                object
Category           object
Rating            float64
Reviews            object
Size              float64
Installs          float64
Type               object
Price              object
Content Rating     object
Genres             object
Last Updated       object
Current Ver        object
Android Ver        object
dtype: object
```

In [32]:

```
#Librairie
from matplotlib import pyplot as plt
```

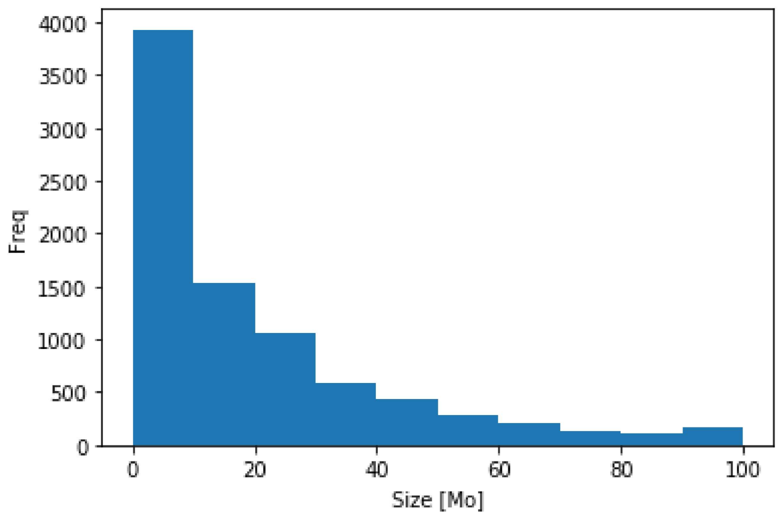
In [33]:

```
#Histogramme
plt.hist(df['Size'])
#plt.hist(df['Size'],range=(df['Size'].min(), df['Size'].max())) #pour la version de python

plt.xlabel("Size [Mo]")
plt.ylabel("Freq")
```

```
d:\python\envs\myenvcnn\lib\site-packages\numpy\lib\histograms.py:824: Runti
meWarning: invalid value encountered in greater_equal
  keep = (tmp_a >= first_edge)
d:\python\envs\myenvcnn\lib\site-packages\numpy\lib\histograms.py:825: Runti
meWarning: invalid value encountered in less_equal
  keep &= (tmp_a <= last_edge)
```

Out[33]:

```
Text(0, 0.5, 'Freq')
```



3) Pour la variable 'Price'

In [34]:

```
np.unique(df['Price'])
```

Out[34]:

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

In [35]:

```
#retirer le caratère '$'
df['Price'] = df['Price'].apply(lambda x: str(x).replace('$', ''))

#convert en float
df['Price'] = df['Price'].apply(lambda x: float(x))
```

4) Pour la variable 'Reviews'

In [36]:

```
#Convert en int
df['Reviews'] = df['Reviews'].apply(lambda x: int(x))
```

In [37]:

```
df.dtypes
```

Out[37]:

```
App                object
Category           object
Rating            float64
Reviews             int64
Size              float64
Installs          float64
Type               object
Price             float64
Content Rating     object
Genres             object
Last Updated       object
Current Ver        object
Android Ver        object
dtype: object
```

5) Pour la variable 'Type'

In [38]:

```
#Est-ce qu'il y a des données manquantes ?
any(pd.isna(df['Type']))
```

Out[38]:

```
True
```

In [39]:

```
#Convertir les deux modalités en des variables catégorielles numériques discrètes : 0 - Fre
df['Type'] = pd.factorize(df['Type'])[0]
```

In [40]:

```
#Afficher les modalités
np.unique(df['Type'])
```

Out[40]:

```
array([-1,  0,  1], dtype=int64)
```

In [41]:

```
#Afficher les modalités
np.unique(df['Type'])
```

Out[41]:

```
array([-1,  0,  1], dtype=int64)
```

6) Pour la variable 'Rating'

In [42]:

```
df['Rating'][0:10]
```

Out[42]:

```
0    4.1
1    3.9
2    4.7
3    4.5
4    4.3
5    4.4
6    3.8
7    4.1
8    4.4
9    4.7
Name: Rating, dtype: float64
```

In [43]:

```
#Est-ce que l'évaluation est NaN
pd.isna(df['Rating'])
#retourne un booléen
```

Out[43]:

```
0        False
1        False
2        False
3        False
4        False
5        False
6        False
7        False
8        False
9        False
10       False
11       False
12       False
13       False
14       False
15       False
16       False
17       False
18       False
19       False
20       False
21       False
22       False
23        True
24       False
25       False
26       False
27       False
28       False
29       False
         ...
10811     True
10812    False
10813     True
10814    False
10815    False
10816     True
10817    False
10818     True
10819    False
10820    False
10821     True
10822     True
10823     True
10824     True
10825     True
10826    False
10827    False
10828    False
10829    False
10830    False
10831     True
10832    False
```

```
10833    False
10834    False
10835     True
10836    False
10837    False
10838     True
10839    False
10840    False
Name: Rating, Length: 9659, dtype: bool
```

In [44]:

```
#Est-ce qu'il y a au moins une app qui a son évaluation en NaN ?
any(pd.isna(df['Rating']))
```

Out[44]:

```
True
```

Il y a au moins une app qui a son évaluation en NaN

In [45]:

```
#afficher que les 10 premiers
pd.isna(df['Rating'])[0:10]
```

Out[45]:

```
0    False
1    False
2    False
3    False
4    False
5    False
6    False
7    False
8    False
9    False
Name: Rating, dtype: bool
```

In [46]:

```
#vecteur des index toutes les apps où l'évaluation est NaN
np.where(pd.isna(df['Rating']))[0]
```

Out[46]:

```
array([  23,  113,  123, ..., 9649, 9653, 9656], dtype=int64)
```

In [47]:

```
#Longueur du vecteur des index toutes les apps où l'évaluation est NaN
#Nombre de valeurs manquantes pour la variable Rating
len(np.where(pd.isna(df['Rating']))[0])
```

Out[47]:

```
1463
```

In [48]:

```
#Ratio de valeurs manquantes pour la variable Rating
len(np.where(pd.isna(df['Rating']))[0]) / df.shape[0]
```

Out[48]:

```
0.15146495496428203
```

In [49]:

```
#Retirer les tuples contenant des NaN
df = df[~pd.isna(df['Rating'])]
#On écrase le dataframe
```

In [50]:

```
#Afficher les 10 premiers
df['Rating'][0:10]
```

Out[50]:

```
0    4.1
1    3.9
2    4.7
3    4.5
4    4.3
5    4.4
6    3.8
7    4.1
8    4.4
9    4.7
Name: Rating, dtype: float64
```

Etape 2 : Anayse univariable

Dans Anaconda Prompt :

- pip install plotly
- pip install cufflinks
- pip install warnings
- pip install seaborn
- pip install matplotlib

In [51]:

```
#Librairies
plt.style.use('ggplot')

import seaborn as sns # for making plots with seaborn
color = sns.color_palette()
sns.set(rc={'figure.figsize':(25,15)})

import plotly
#connected=True means it will download the latest version of plotly javascript library.
plotly.offline.init_notebook_mode(connected=True)
import plotly.graph_objs as go

import plotly.figure_factory as ff
import cufflinks as cf
```

1) Analyse univariable de 'Category'

In [52]:

```
#Afficher les modalités
np.unique(df['Category'])
```

Out[52]:

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

In [53]:

```
#Compter le nombre d'applications par catégories
number_of_apps_in_category = df['Category'].value_counts().sort_values(ascending=True)
number_of_apps_in_category
```

Out[53]:

```
BEAUTY                   42
EVENTS                   45
PARENTING                50
COMICS                   54
ART_AND_DESIGN           61
HOUSE_AND_HOME           62
LIBRARIES_AND_DEMO       64
WEATHER                  72
AUTO_AND_VEHICLES        73
FOOD_AND_DRINK           94
ENTERTAINMENT           102
EDUCATION               118
MAPS_AND_NAVIGATION     118
DATING                  134
VIDEO_PLAYERS           148
BOOKS_AND_REFERENCE     169
SHOPPING                180
TRAVEL_AND_LOCAL        187
SOCIAL                  203
NEWS_AND_MAGAZINES      204
HEALTH_AND_FITNESS      244
COMMUNICATION           256
SPORTS                  260
PHOTOGRAPHY             263
BUSINESS                263
MEDICAL                 290
PERSONALIZATION         298
PRODUCTIVITY            301
LIFESTYLE               301
FINANCE                 302
TOOLS                   718
GAME                    912
FAMILY                 1608
Name: Category, dtype: int64
```

In [54]:

```
#Le dernier élément du tableau
number_of_apps_in_category.index[-1]
```

Out[54]:

```
'FAMILY'
```

In [55]:

```
#Le dernier élément du tableau
number_of_apps_in_category[-1]/len(df)*100
```

Out[55]:

```
19.619326500732065
```

In [56]:

```
#L'avant dernier élément du tableau
number_of_apps_in_category.index[-2]
```

Out[56]:

```
'GAME'
```

In [57]:

```
#L'avant dernier élément du tableau
number_of_apps_in_category[-2]/len(df)*100
```

Out[57]:

```
11.127379209370424
```

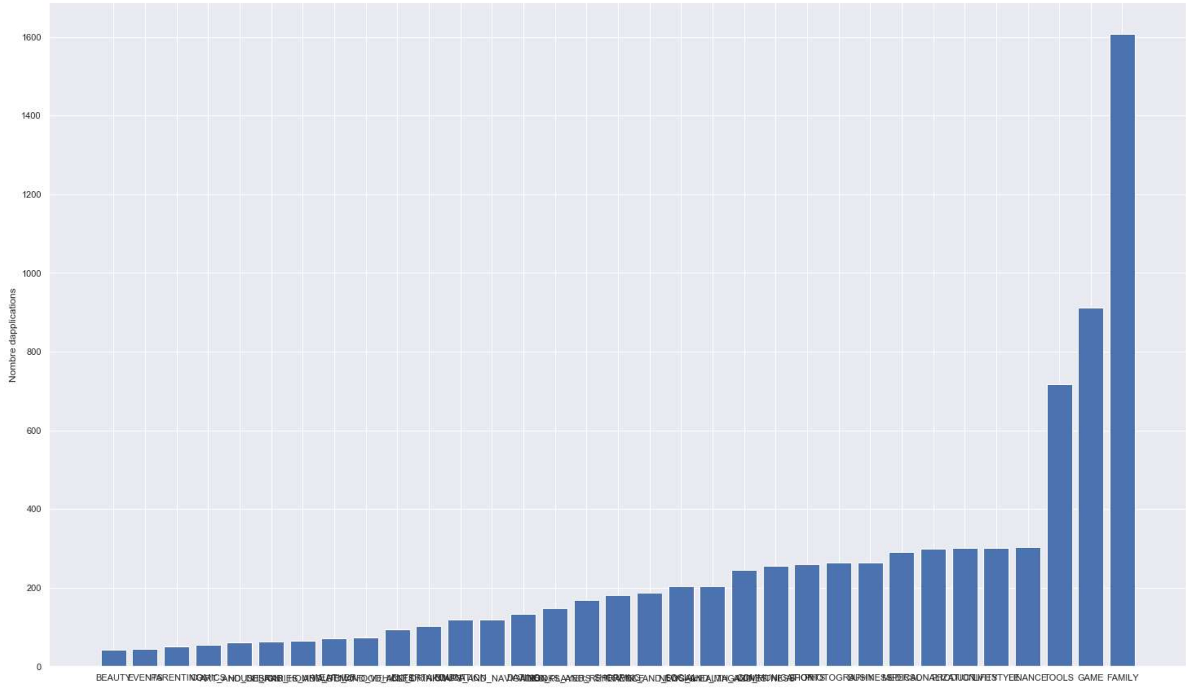
Les catégories proposant le plus d'applications sur le marché sont "Family" et "Games" avec respectivement 19,61% et 11,12%

In [58]:

```
#Figure de distribution des applications par catégories
plt.bar(number_of_apps_in_category.index, number_of_apps_in_category)
plt.ylabel('Nombre dapplications')

#Pour Python 3.6
#plt.bar(range(len(number_of_apps_in_category.index)), number_of_apps_in_category)
#plt.xticks(range(len(number_of_apps_in_category.index)), number_of_apps_in_category.index)
```

Out[58]:

```
Text(0, 0.5, 'Nombre dapplications')
```

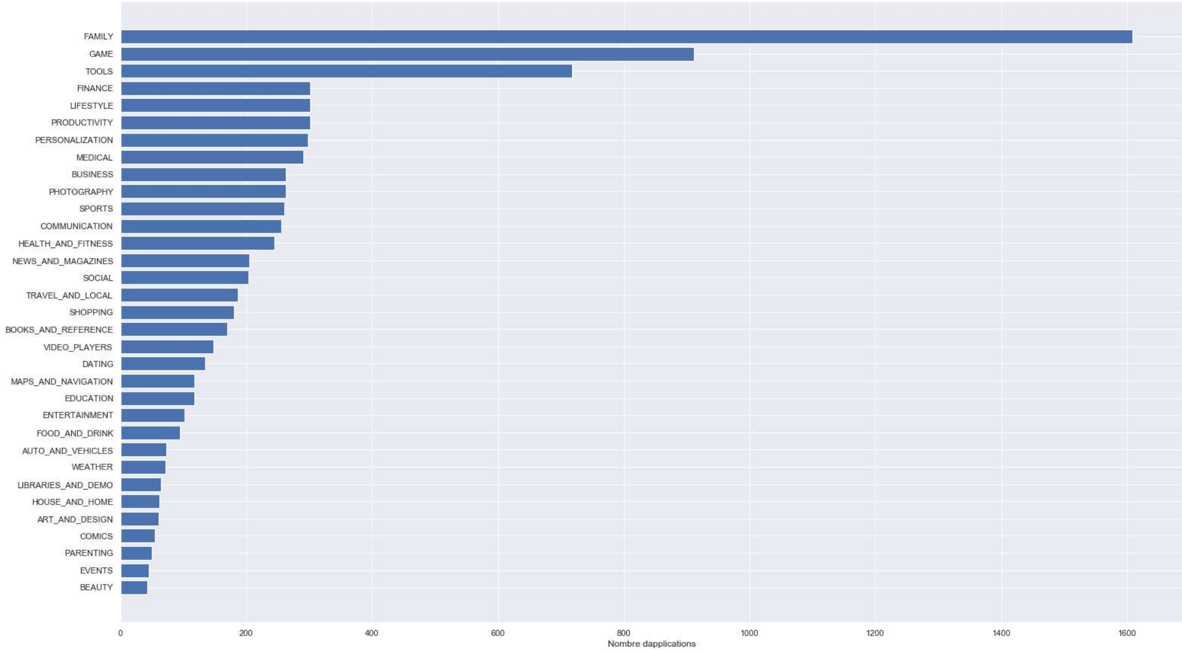


In [59]:

```
#Figure de distribution des applications par catégories
plt.barh(number_of_apps_in_category.index, number_of_apps_in_category)
plt.xlabel('Nombre dapplications')

#Pour Python 3.6
#plt.barh(range(len(number_of_apps_in_category.index)), number_of_apps_in_category)
#plt.yticks(range(len(number_of_apps_in_category.index)), number_of_apps_in_category.index)
```

Out[59]:

```
Text(0.5, 0, 'Nombre dapplications')
```



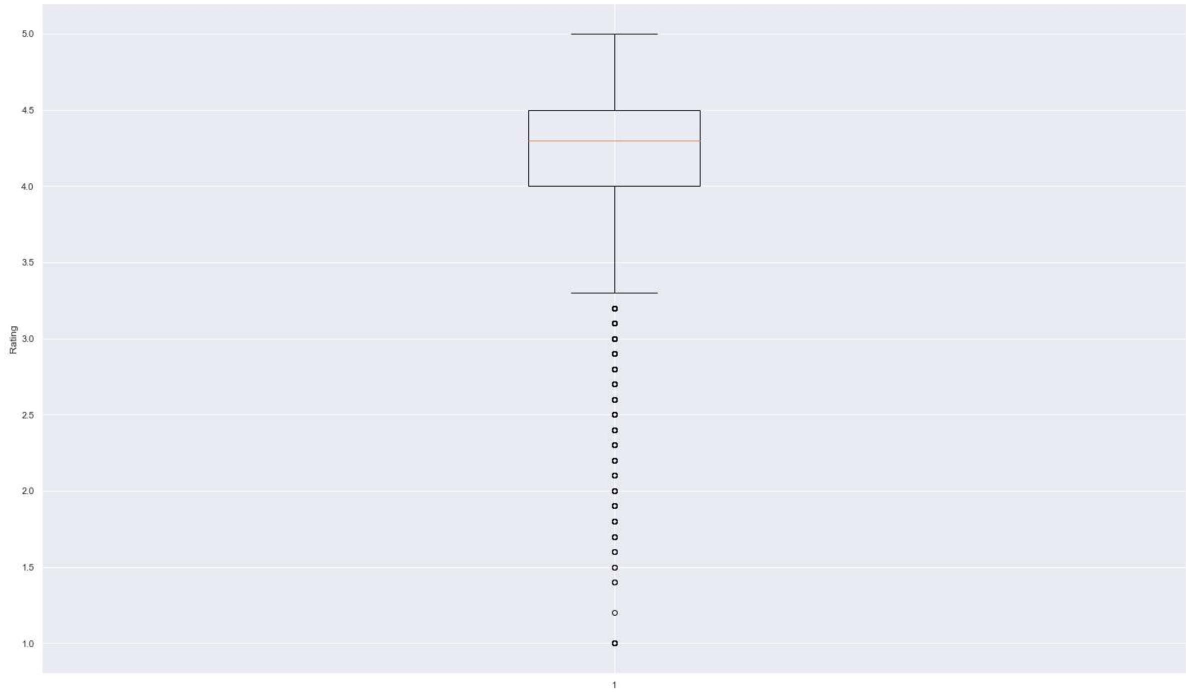
2) Analyse univariable de 'Rating'

In [60]:

```
#Boîte à moustaches
bx = plt.boxplot(df['Rating'])
#bx = plt.boxplot(df['Rating'], 0, 'gD') #Pour afficher les outliers sur la boîte à moustac

plt.ylabel('Rating')
```

Out[60]:

Text(0, 0.5, 'Rating')



In [61]:

```
#Moyenne
np.mean(df['Rating'])
```

Out[61]:

4.173243045387998

In [62]:

```
#Médianne
np.median(df['Rating'])
```

Out[62]:

4.3

In [63]:

```
#Ecart-type
np.std(df['Rating'])
```

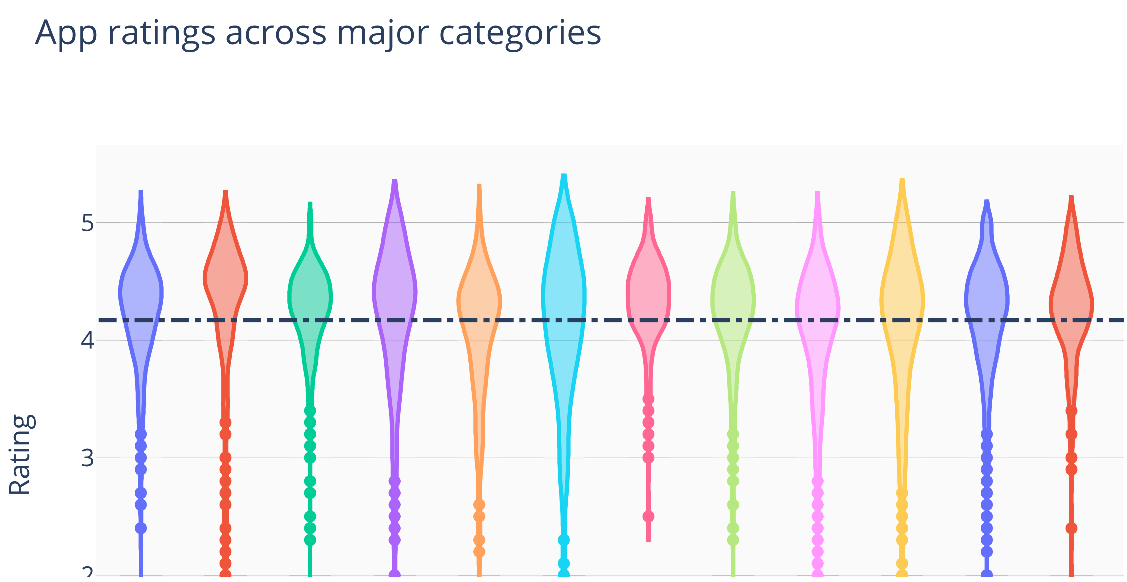
Out[63]:

0.536591992399412

3) Analyse univariable de 'Category'

In [64]:

```
# Affichage en boîte à moustache

groups = df.groupby('Category').filter(lambda x: len(x) >= 170).reset_index()
#print(type(groups.item.['BUSINESS']))
#print(len(groups.loc[df.Category == 'DATING']))

#df_sorted = df.groupby('Category').agg({'Rating':'median'}).reset_index().sort_values(by='
#print(df_sorted)

layout = {'title' : 'App ratings across major categories',
        'xaxis': {'tickangle':-40},
        'yaxis': {'title': 'Rating'},
          'plot_bgcolor': 'rgb(250,250,250)',
          'shapes': [{
              'type' :'line',
              'x0': -.5,
              'y0': np.nanmean(list(groups.Rating)),
              'x1': 19,
              'y1': np.nanmean(list(groups.Rating)),
              'line': { 'dash': 'dashdot'}
          }]
          }

data = [{
    'y': df.loc[df.Category==category]['Rating'],
    'type':'violin',
    'name' : category,
    'showlegend':False,
    #'marker': {'color': 'Set2'},
    } for i,category in enumerate(list(set(groups.Category)))]

plotly.offline.iplot({'data': data, 'layout': layout})
#plotly.offline.iplot({'data': data, 'layout': layout}, validate=False) #Pour forcer l'affi
```



Analyse

- La distribution évaluations tendents à être similaires et ont plutôt de très bonnes notes.
- Certaines catégories ont une queue de dispertion à droite plus importante que d'autres catégories (**Dating** par rapport à **Personalization** par exemple).
- Les catégories **Health and Fitness** et **Books and Reference** sont les plus qualitative avec **50% de ceux-ci ayant une évaluation supérieur à 4.5 étoiles**, ce qui est particulièrement bon.

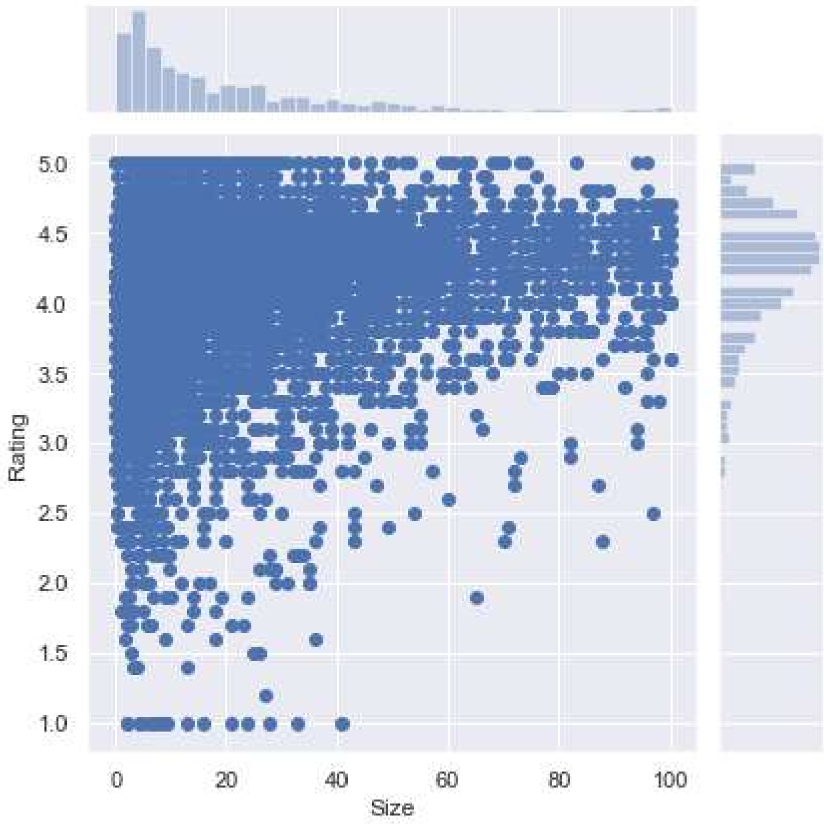
Etape 3 : Analyse bi-variables

1) Analyse bi-variables de 'Rating' et 'Size'

Est-ce que la taille impacte l'évaluation ?

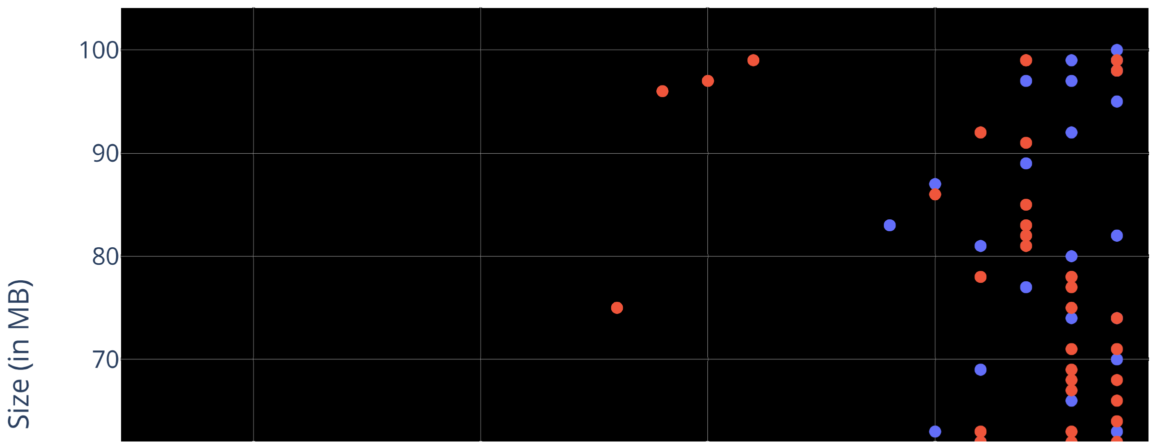
In [65]:

```
# sns.set_style('ticks')
# fig, ax = plt.subplots()
# fig.set_size_inches(8, 8)
sns.set_style("darkgrid")
ax = sns.jointplot(df['Size'], df['Rating'])
#ax.set_title('Rating Vs Size')
```



In [66]:

```
subset_df = df[df.Size > 40]
groups_temp = subset_df.groupby('Category').filter(lambda x: len(x) >20)

# for category in enumerate(list(set(groups_temp.Category))):
#     print (category)

data = [{
    'x': groups_temp.loc[subset_df.Category==category[1]]['Rating'],
    'type':'scatter',
    'y' : subset_df['Size'],
    'name' : str(category[1]),
    'mode' : 'markers',
    'showlegend': True,
    #'marker': {'color':c[i]}
    #'text' : df['rating'],
    } for category in enumerate(['GAME', 'FAMILY'])]


layout = {'title':"",
          'xaxis': {'title' : 'Rating'},
          'yaxis' : {'title' : 'Size (in MB)'},
         'plot_bgcolor': 'rgb(0,0,0)'}

plotly.offline.iplot({'data': data, 'layout': layout})

# heavy_categories = [ 'ENTERTAINMENT', 'MEDICAL', 'DATING']

# data = [{
#     'x': groups.loc[df.Category==category]['Rating'],
#     'type':'scatter',
#     'y' : df['Size'],
#     'name' : category,
#     'mode' : 'markers',
#     'showlegend': True,
#     #'text' : df['rating'],
#     } for category in heavy_categories]
```

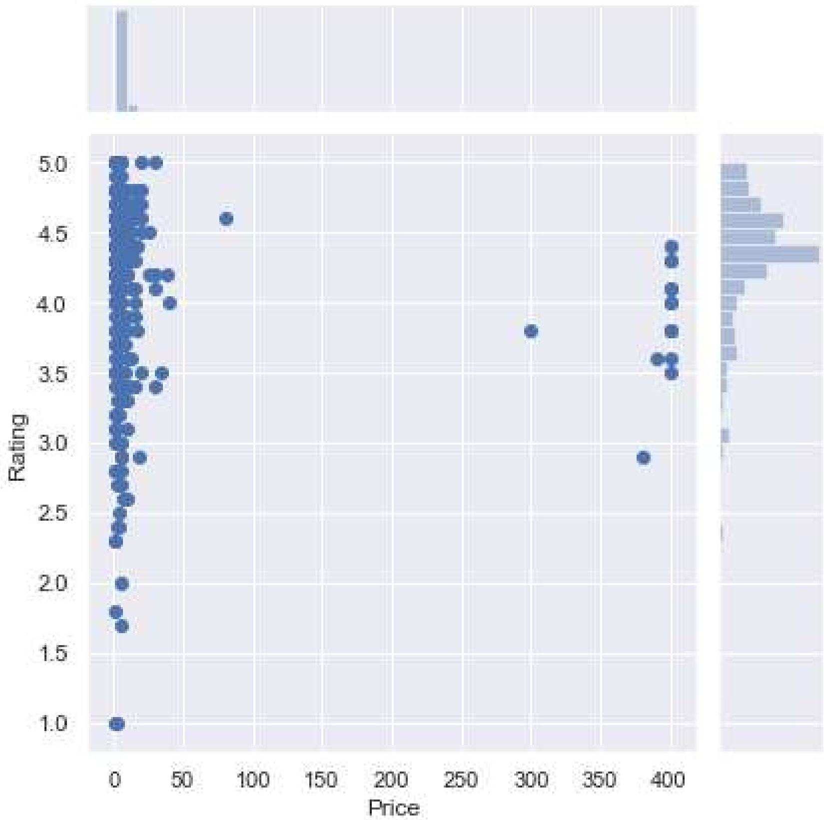


2) Analyse bi-variables de 'Rating' et 'Price'

Analyse de la stratégie de Pricing

In [67]:

```
paid_apps = df[df.Price > 0]
p = sns.jointplot( "Price", "Rating", paid_apps)
```



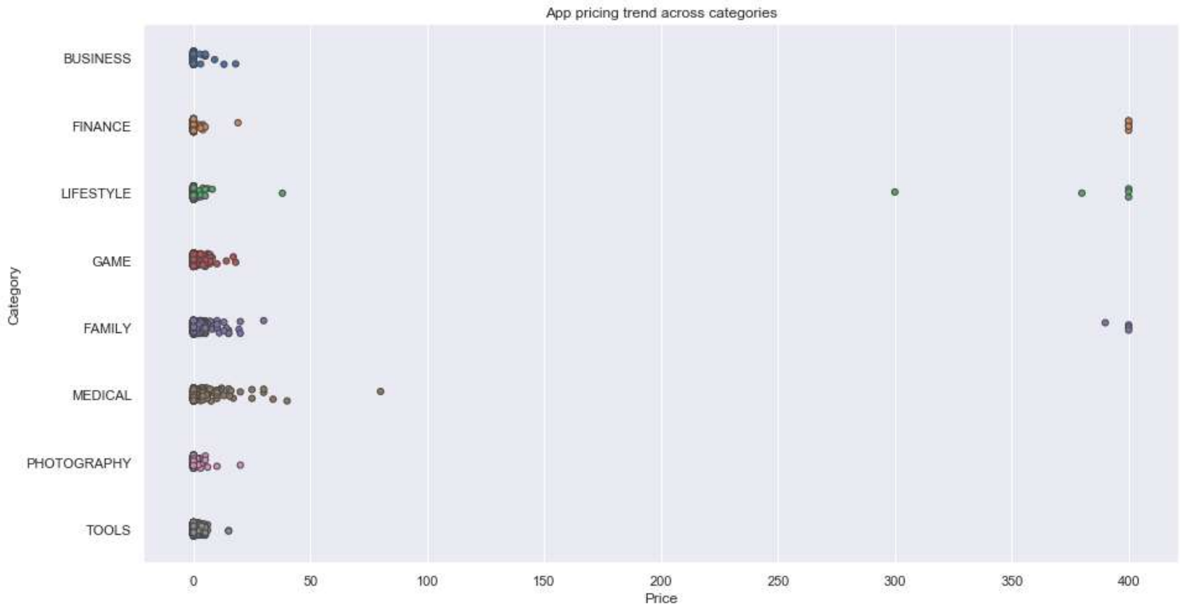
La plupart des applications les mieux évaluées se positionnent sur un prix variant entre **~1$ to ~30$**.

Il y a quelques applications au dessus de **20$**

3) Analyse bi-variables de 'Category' et 'Price'

In [68]:

```
#Analayse du prix discrétisé par catégorie
subset_df = df[df.Category.isin(['GAME', 'FAMILY', 'PHOTOGRAPHY', 'MEDICAL', 'TOOLS', 'FINA
                                 'LIFESTYLE','BUSINESS'])]
sns.set_style('darkgrid')
fig, ax = plt.subplots()
fig.set_size_inches(15, 8)
p = sns.stripplot(x="Price", y="Category", data=subset_df, jitter=True, linewidth=1)
title = ax.set_title('App pricing trend across categories')
```



In [69]:

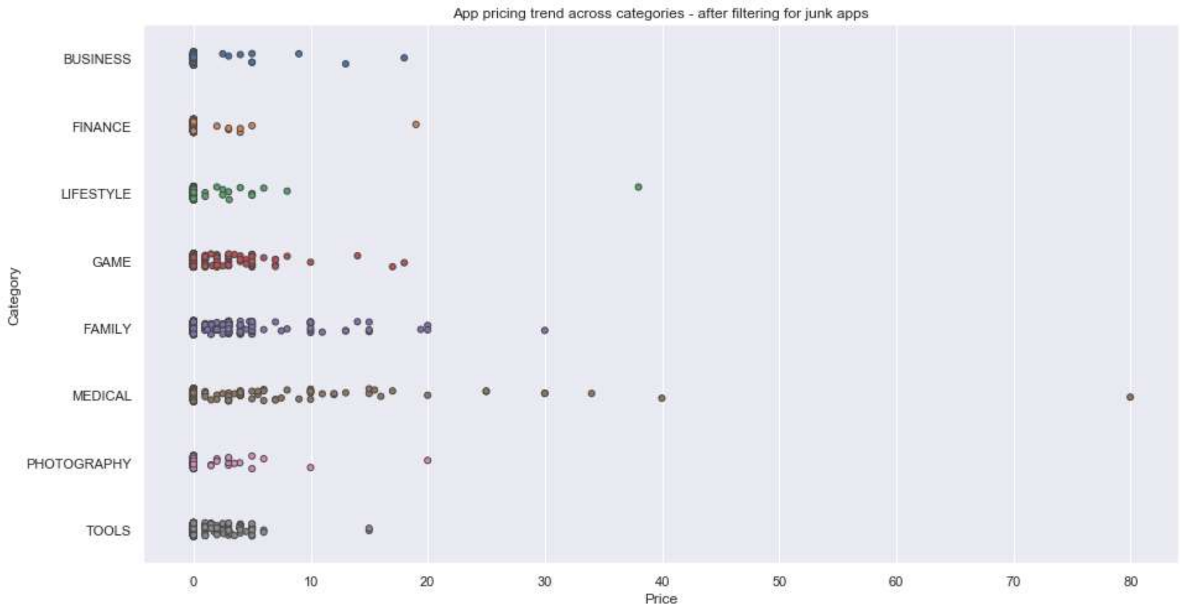
```
#print('Junk apps priced above 350$')
df[['Category', 'App']][df.Price > 200]
```

Out[69]:

	Category	App
4197	FAMILY	most expensive app (H)
4362	LIFESTYLE	💎 I'm rich
4367	LIFESTYLE	I'm Rich - Trump Edition
5351	LIFESTYLE	I am rich
5354	FAMILY	I am Rich Plus
5355	LIFESTYLE	I am rich VIP
5356	FINANCE	I Am Rich Premium
5357	LIFESTYLE	I am extremely Rich
5358	FINANCE	I am Rich!
5359	FINANCE	I am rich(premium)
5362	FAMILY	I Am Rich Pro
5364	FINANCE	I am rich (Most expensive app)
5366	FAMILY	I Am Rich
5369	FINANCE	I am Rich
5373	FINANCE	I AM RICH PRO PLUS

In [70]:

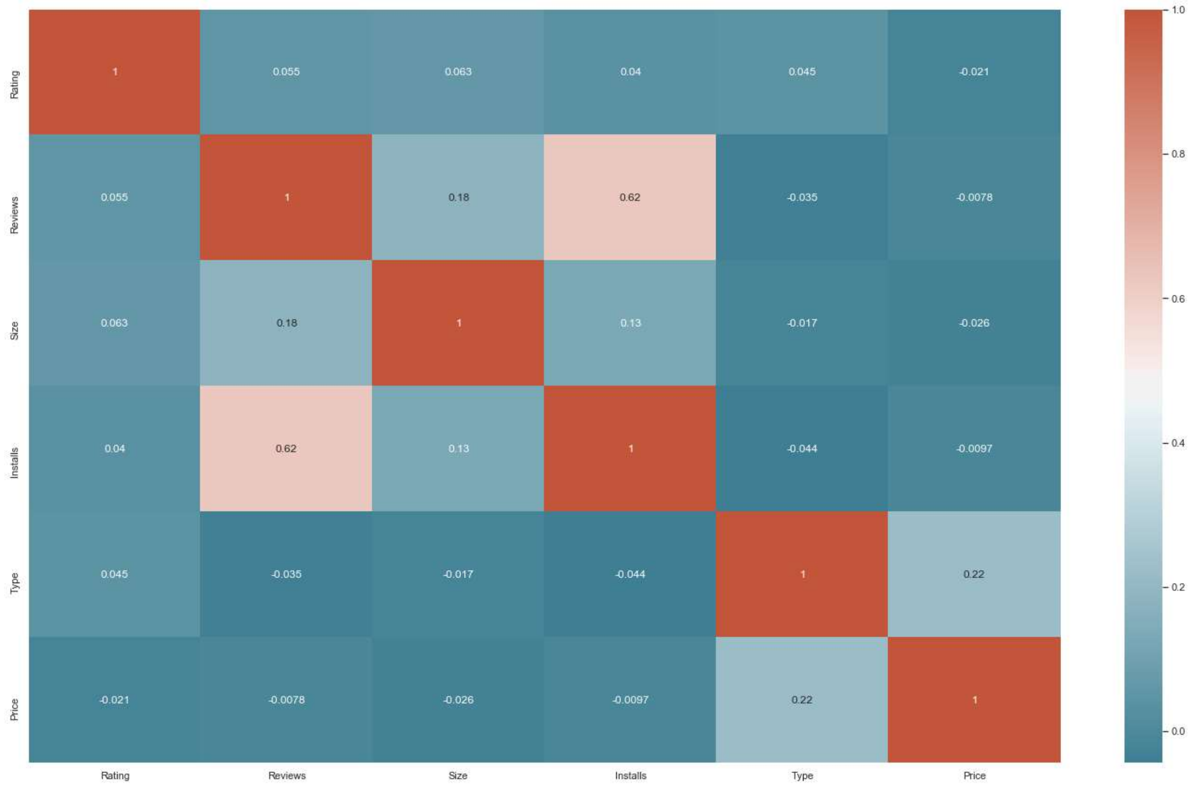
```
fig, ax = plt.subplots()
fig.set_size_inches(15, 8)
subset_df_price = subset_df[subset_df.Price<100]
p = sns.stripplot(x="Price", y="Category", data=subset_df_price, jitter=True, linewidth=1)
title = ax.set_title('App pricing trend across categories - after filtering for junk apps')
```



4) Analyse des coorélations par paire de variables quantitatives

In [71]:

```
#df['Installs'].corr(df['Reviews'])#df['Insta
#print(np.corrcoef(l, rating))

corrmat = df.corr()
#f, ax = plt.subplots()
p = sns.heatmap(corrmat, annot=True, cmap=sns.diverging_palette(220, 20, as_cmap=True))
```



Une corrélation modérée existe entre le nombre de reviews et le nombre de téléchargements.