App_Store Clustering

Data description

Column Name	Description
id	App ID
track_name	App Name
size_bytes	Size in bytes
currency	Currency Type
price	Price ammount
ratingcounttot	User Rating counts (for all versions)
ratingcountver	User Rating counts (for current version)
user_rating	Average User Rating value (for all versions)
userratingver	Average User Rating value (for current version)
ver	Latest version code
cont_rating	Content Rating
prime_genre	Primary Genre
sup_devices.num	Number of supporting devices
ipadSc_urls.num	Number of screenshots showed for display

Column Name	Description
lang.num	Number of supported languages
vpp_lic	Vpp Device Based Licensing Enabled

Important Libraries

In [204]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
from warnings import filterwarnings
filterwarnings("ignore")
!pip3 install ppscore
import ppscore as pps
#Import Library RobustScaler
from sklearn.preprocessing import RobustScaler
#Cluster Model
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
```

```
Requirement already satisfied: ppscore in c:\users\shaimaa\anaconda3\lib\site-p
ackages (1.2.0)
Requirement already satisfied: pandas<2.0.0,>=1.0.0 in c:\users\shaimaa\anacond
a3\lib\site-packages (from ppscore) (1.2.4)
Requirement already satisfied: scikit-learn<1.0.0,>=0.20.2 in c:\users\shaimaa
\anaconda3\lib\site-packages (from ppscore) (0.24.1)
Requirement already satisfied: pytz>=2017.3 in c:\users\shaimaa\anaconda3\lib\s
ite-packages (from pandas<2.0.0,>=1.0.0->ppscore) (2021.1)
Requirement already satisfied: numpy>=1.16.5 in c:\users\shaimaa\anaconda3\lib
\site-packages (from pandas<2.0.0,>=1.0.0->ppscore) (1.20.1)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\shaimaa\anaco
nda3\lib\site-packages (from pandas<2.0.0,>=1.0.0->ppscore) (2.8.1)
Requirement already satisfied: six>=1.5 in c:\users\shaimaa\anaconda3\lib\site-
packages (from python-dateutil>=2.7.3->pandas<2.0.0,>=1.0.0->ppscore) (1.15.0)
Requirement already satisfied: scipy>=0.19.1 in c:\users\shaimaa\anaconda3\lib
\site-packages (from scikit-learn<1.0.0,>=0.20.2->ppscore) (1.6.2)
Requirement already satisfied: joblib>=0.11 in c:\users\shaimaa\anaconda3\lib\s
ite-packages (from scikit-learn<1.0.0,>=0.20.2->ppscore) (1.0.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\shaimaa\anacond
a3\lib\site-packages (from scikit-learn<1.0.0,>=0.20.2->ppscore) (2.1.0)
```

Read Data

In [205]:

```
#load_data
data = pd.read_csv('C:\\Users\\Shaimaa\\Desktop\\AppleStore.csv' ,sep =',' , enco
data.head()
```

Out[205]:

	Unnamed: 0	id	track_name	size_bytes	currency	price	rating_count_tot	rating_count_ve
0	1	281656475	PAC-MAN Premium	100788224	USD	3.99	21292	2
1	2	281796108	Evernote - stay organized	158578688	USD	0.00	161065	2
2	3	281940292	WeatherBug - Local Weather, Radar, Maps, Alerts	100524032	USD	0.00	188583	282
3	4	282614216	eBay: Best App to Buy, Sell, Save! Online Shop...	128512000	USD	0.00	262241	64
4	5	282935706	Bible	92774400	USD	0.00	985920	532

In [206]:

```
#drop column (Unnamed) as semiler ID column
data.drop(['Unnamed: 0'], axis=1 ,inplace=True)
#show data after drop
data.head(2)
```

Out[206]:

	id	track_name	size_bytes	currency	price	rating_count_tot	rating_count_ver	user_rat
0	281656475	PAC-MAN Premium	100788224	USD	3.99	21292	26	
1	281796108	Evernote - stay organized	158578688	USD	0.00	161065	26	

In [207]:
```
#data about data
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7197 entries, 0 to 7196
Data columns (total 16 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   id                7197 non-null   int64  
 1   track_name        7197 non-null   object 
 2   size_bytes        7197 non-null   int64  
 3   currency          7197 non-null   object 
 4   price             7197 non-null   float64
 5   rating_count_tot  7197 non-null   int64  
 6   rating_count_ver  7197 non-null   int64  
 7   user_rating       7197 non-null   float64
 8   user_rating_ver   7197 non-null   float64
 9   ver               7197 non-null   object 
 10  cont_rating       7197 non-null   object 
 11  prime_genre       7197 non-null   object 
 12  sup_devices.num   7197 non-null   int64  
 13  ipadSc_urls.num   7197 non-null   int64  
 14  lang.num          7197 non-null   int64  
 15  vpp_lic           7197 non-null   int64  
dtypes: float64(3), int64(8), object(5)
memory usage: 899.8+ KB
```

data has float64(for 3 columns) , int64(8 columns), object(5 columns)

will Apply encoding for object columns

In [208]:
```
#show shape of data 7197 Row and 16 columns
data.shape
```

Out[208]:
```
(7197, 16)
```

In [209]:
```
data.isnull().sum().sum()
#not found null data
```

Out[209]:
```
0
```

In [210]:
```
data.currency.value_counts()
#All of Apps has same currency paid
```

Out[210]:
```
USD    7197
Name: currency, dtype: int64
```

In [211]:
```
data.nunique()
#target maybe vpp_lic
```

Out[211]:
```
id                  7197
track_name          7195
size_bytes          7107
currency               1
price                 36
rating_count_tot    3185
rating_count_ver    1138
user_rating           10
user_rating_ver       10
ver                 1590
cont_rating            4
prime_genre           23
sup_devices.num       20
ipadSc_urls.num        6
lang.num              57
vpp_lic                2
dtype: int64
```

Exploratory Data Analaysis

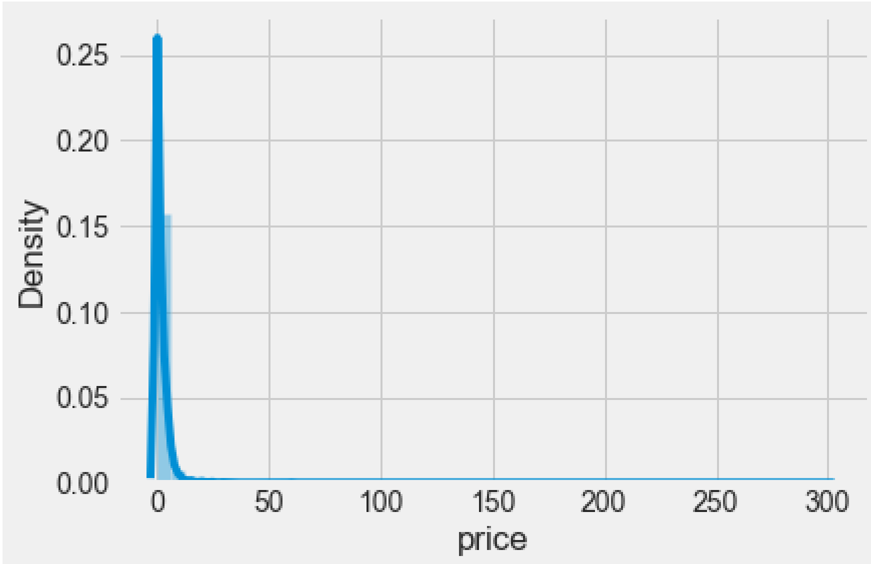
How do you visualize price distribution of paid apps ?

In [212]:
```
data.price.value_counts()
#4056 free apps
#another apps is paid
```

Out[212]:
```
0.00      4056
0.99       728
2.99       683
1.99       621
4.99       394
3.99       277
6.99       166
9.99        81
5.99        52
7.99        33
14.99       21
19.99       13
8.99         9
24.99        8
13.99        6
11.99        6
29.99        6
12.99        5
15.99        4
59.99        3
17.99        3
39.99        2
27.99        2
49.99        2
22.99        2
23.99        2
20.99        2
16.99        2
21.99        1
47.99        1
99.99        1
249.99       1
18.99        1
34.99        1
299.99       1
74.99        1
Name: price, dtype: int64
```

In [213]:
```
sns.distplot(data.price)
```

Out[213]: <AxesSubplot:xlabel='price', ylabel='Density'>



In [214]:
```
free_apps = data[(data.price==0.00)]

paid_apps  = data[(data.price>0)]
```

In [215]:
```
free_apps.head(2)
```

Out[215]:

	id	track_name	size_bytes	currency	price	rating_count_tot	rating_count_ver	user_rat
1	281796108	Evernote - stay organized	158578688	USD	0.0	161065	26	
2	281940292	WeatherBug - Local Weather, Radar, Maps, Alerts	100524032	USD	0.0	188583	2822	

In [216]:
```
paid_apps.head(2)
```

Out[216]:

	id	track_name	size_bytes	currency	price	rating_count_tot	rating_count_ver	user_
0	281656475	PAC-MAN Premium	100788224	USD	3.99	21292	26	
5	283619399	Shanghai Mahjong	10485713	USD	0.99	8253	5516	

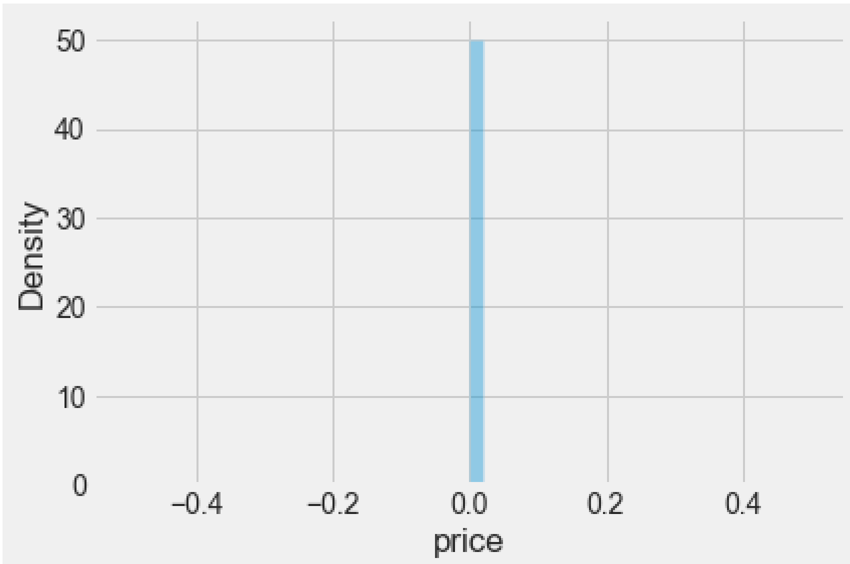
In [217]:
```
paid_apps.price.value_counts()
```

Out[217]:
```
0.99      728
2.99      683
1.99      621
4.99      394
3.99      277
6.99      166
9.99       81
5.99       52
7.99       33
14.99      21
19.99      13
8.99        9
24.99       8
29.99       6
11.99       6
13.99       6
12.99       5
15.99       4
17.99       3
59.99       3
16.99       2
39.99       2
23.99       2
27.99       2
20.99       2
22.99       2
49.99       2
74.99       1
47.99       1
18.99       1
249.99      1
34.99       1
99.99       1
21.99       1
299.99      1
Name: price, dtype: int64
```

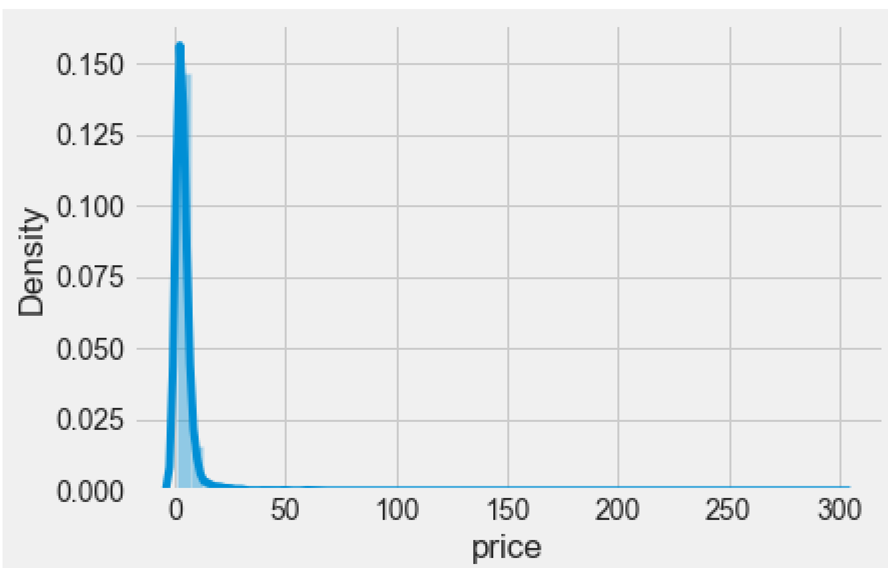
The number of apps decreases with increasing his price

In [218]:
```
free_apps.price.value_counts()
```
Out[218]:
```
0.0    4056
Name: price, dtype: int64
```

In [219]:
```
sns.distplot(free_apps['price'])
```
Out[219]:
```
<AxesSubplot:xlabel='price', ylabel='Density'>
```

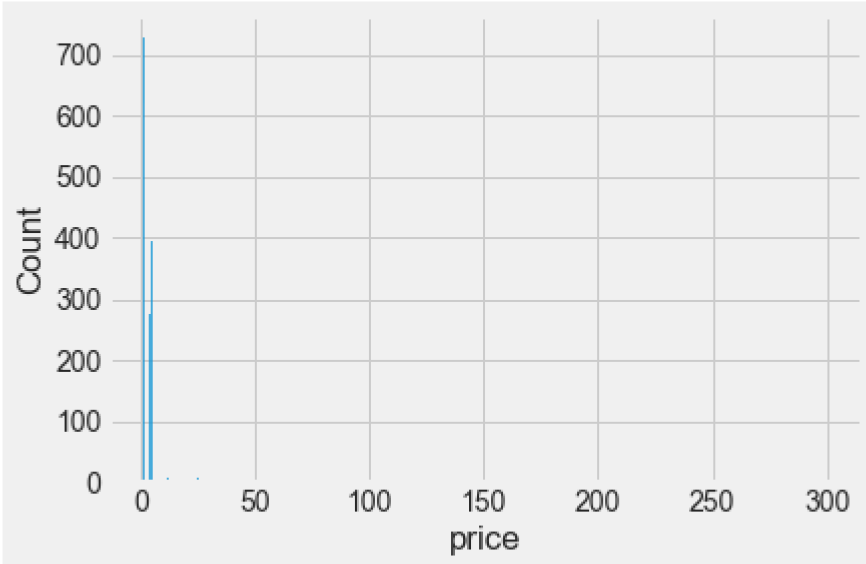


In [220]:
```
sns.distplot(paid_apps['price'])
```
Out[220]:
```
<AxesSubplot:xlabel='price', ylabel='Density'>
```

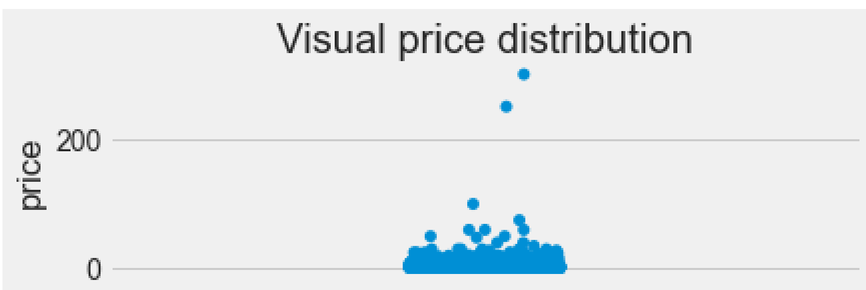


In [221]:
```
sns.histplot(paid_apps['price'])
```

Out[221]: <AxesSubplot:xlabel='price', ylabel='Count'>



In [222]:
```
plt.style.use('fivethirtyeight')
plt.figure(figsize=(6,4))

plt.subplot(2,1,2)
plt.title('Visual price distribution')
sns.stripplot(data=paid_apps,y='price',jitter= True,orient = 'h' ,size=6)
plt.show()
```



from this graph The number of apps that have a price greater than 50 is few compared to before 50 USD

In [223]:
```
Top_Apps=paid_apps[paid_apps.price>50][['track_name','price','prime_genre','user_
Top_Apps
#7 Top apps with price, prime_genre and user rating
```

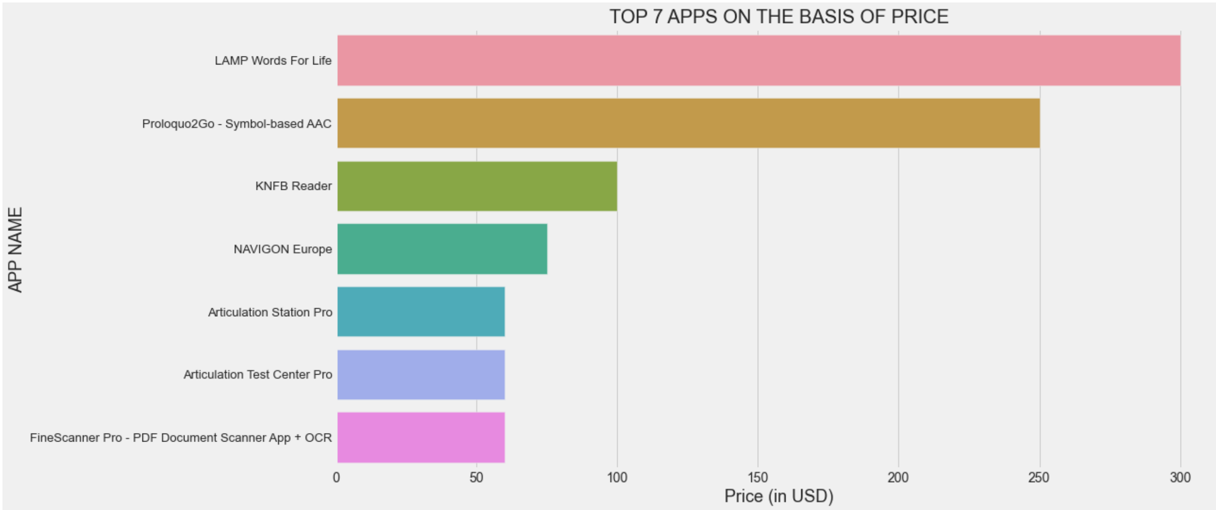
Out[223]:

	track_name	price	prime_genre	user_rating
115	Proloquo2Go - Symbol-based AAC	249.99	Education	4.0
162	NAVIGON Europe	74.99	Navigation	3.5
1136	Articulation Station Pro	59.99	Education	4.5
1479	LAMP Words For Life	299.99	Education	4.0
2181	Articulation Test Center Pro	59.99	Education	4.5
2568	KNFB Reader	99.99	Productivity	4.5
3238	FineScanner Pro - PDF Document Scanner App + OCR	59.99	Business	4.0

Top 7 apps on the basis of price

In [224]:
```
#Function for visualizaiton
def visualizer(x, y, plot_type, title, xlabel, ylabel, rotation=False, rotation_v
    plt.figure(figsize=figsize)

    if plot_type == "bar":
        sns.barplot(x=x, y=y)
    elif plot_type == "count":
        sns.countplot(x)

    plt.title(title, fontsize=20)
    plt.xlabel(xlabel, fontsize=18)
    plt.ylabel(ylabel, fontsize=18)
    plt.yticks(fontsize=13)
    if rotation == True:
        plt.xticks(fontsize=13,rotation=rotation_value)
    plt.show()
```

In [225]:
```
Top_Apps = Top_Apps.sort_values('price', ascending=False)

visualizer(Top_Apps.price,Top_Apps.track_name, "bar", "TOP 7 APPS ON THE BASIS OF
#names of track in y axis to be readable
```



In [226]:
```
paid_apps.head(2)
```

Out[226]:

	id	track_name	size_bytes	currency	price	rating_count_tot	rating_count_ver	user_rat
0	281656475	PAC-MAN Premium	100788224	USD	3.99	21292	26	
5	283619399	Shanghai Mahjong	10485713	USD	0.99	8253	5516	

In [227]:
```
#sum of all paid apps
sum_paid = paid_apps.price.value_counts().sum()
sum_paid
```

Out[227]: 3141

In [228]:
```
#sum of all free apps
sum_free = free_apps.price.value_counts().sum()
sum_free
```

Out[228]: 4056

How does the price distribution get affected by category ?

In [229]:
```
data.prime_genre.value_counts()
```

Out[229]:
```
Games                3862
Entertainment         535
Education             453
Photo & Video         349
Utilities             248
Health & Fitness      180
Productivity          178
Social Networking     167
Lifestyle             144
Music                 138
Shopping              122
Sports                114
Book                  112
Finance               104
Travel                 81
News                   75
Weather                72
Reference              64
Food & Drink           63
Business               57
Navigation             46
Medical                23
Catalogs               10
Name: prime_genre, dtype: int64
```

Top app category is Games Games # is 3862 and Entertainment # is 535

In [230]:
```
data.head()
```

Out[230]:

	id	track_name	size_bytes	currency	price	rating_count_tot	rating_count_ver	user_rat
0	281656475	PAC-MAN Premium	100788224	USD	3.99	21292	26	
1	281796108	Evernote - stay organized	158578688	USD	0.00	161065	26	
2	281940292	WeatherBug - Local Weather, Radar, Maps, Alerts	100524032	USD	0.00	188583	2822	
3	282614216	eBay: Best App to Buy, Sell, Save! Online Shop...	128512000	USD	0.00	262241	649	
4	282935706	Bible	92774400	USD	0.00	985920	5320	

In [231]:
```
new_data_cate = data.groupby([data.prime_genre])[['id']].count().reset_index().so
new_data_cate.columns = ['prime_genre','# of Apps']
new_data_cate.head()
#Categories and number of apps in each category
```

Out[231]:

	prime_genre	# of Apps
7	Games	3862
4	Entertainment	535
3	Education	453
14	Photo & Video	349
21	Utilities	248

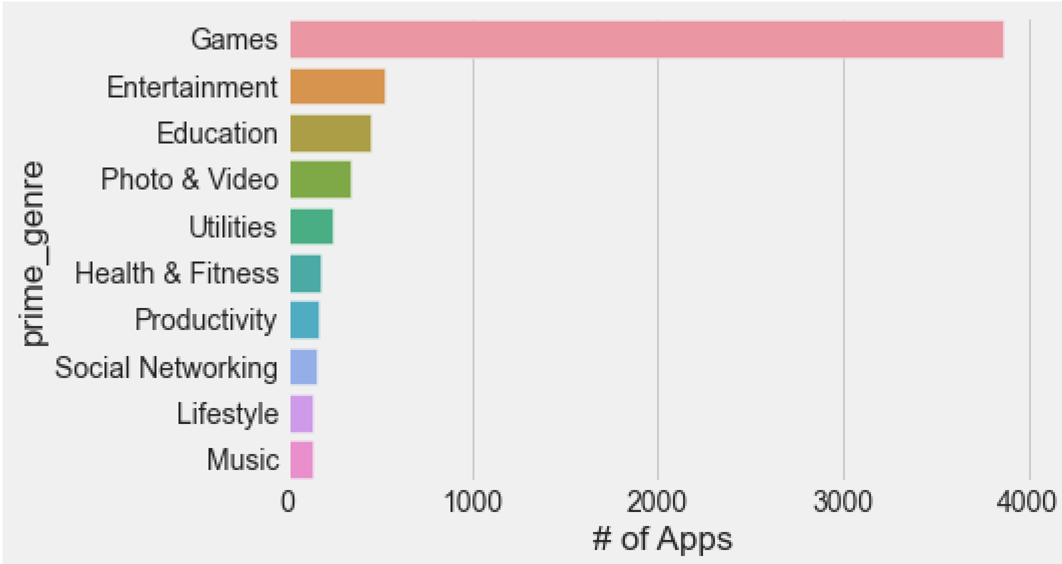
In [232]:
```
#Top_Categories accorrding number of apps
new_data_cate.head(10)
```

Out[232]:

	prime_genre	# of Apps
7	Games	3862
4	Entertainment	535
3	Education	453
14	Photo & Video	349
21	Utilities	248
8	Health & Fitness	180
15	Productivity	178
18	Social Networking	167
9	Lifestyle	144
11	Music	138

In [233]:
```
sns.barplot(y = 'prime_genre',x = '# of Apps', data=new_data_cate.head(10))
```

Out[233]: <AxesSubplot:xlabel='# of Apps', ylabel='prime_genre'>



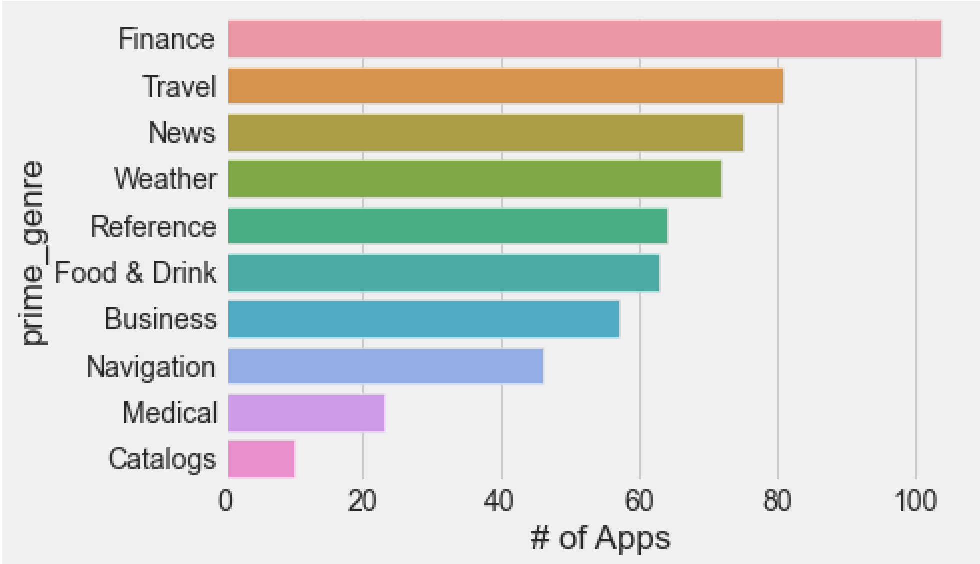
In [234]:
```
#Lower Categories according number of apps Categories unpopular
new_data_cate.tail(10)
```

Out[234]:

	prime_genre	# of Apps
5	Finance	104
20	Travel	81
13	News	75
22	Weather	72
16	Reference	64
6	Food & Drink	63
1	Business	57
12	Navigation	46
10	Medical	23
2	Catalogs	10

In [235]:
```
sns.barplot(x= '# of Apps' , y = 'prime_genre' , data = new_data_cate.tail(10))
```

Out[235]:
```
<AxesSubplot:xlabel='# of Apps', ylabel='prime_genre'>
```



In [236]:
```
plt.figure(figsize=(10,5))
plt.scatter(y=paid_apps.prime_genre ,x=paid_apps.price,c='DarkBlue')
plt.title('Price & Category')
plt.xlabel('Price')
plt.ylabel('Category')
plt.show()
```

- Top Price in important Category (Business , Navigation , Education , Productivity)
- in another side price for all of apps less than 50 USD
- Education Apps has a higher price
- Shopping Apps has a lower price

What about paid apps Vs Free apps ?

In [237]:
```
free_apps.head(2)
```

Out[237]:

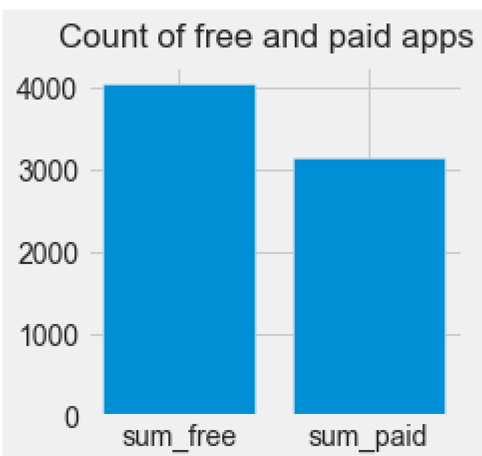
	id	track_name	size_bytes	currency	price	rating_count_tot	rating_count_ver	user_rat
1	281796108	Evernote - stay organized	158578688	USD	0.0	161065	26	
2	281940292	WeatherBug - Local Weather, Radar, Maps, Alerts	100524032	USD	0.0	188583	2822	

In [238]:
```
paid_apps.head(2)
```

Out[238]:

	id	track_name	size_bytes	currency	price	rating_count_tot	rating_count_ver	user_rat
0	281656475	PAC-MAN Premium	100788224	USD	3.99	21292	26	
5	283619399	Shanghai Mahjong	10485713	USD	0.99	8253	5516	

In [239]:
```
names = ['sum_free', 'sum_paid']
values = [sum_free, sum_paid]
plt.figure(figsize=(3, 3))
plt.suptitle('Count of free and paid apps')
plt.bar(names, values)
plt.show()
```



count of paid apps less than count of free apps

In [240]:
```
print('number of Catigories in free apps is' ,  len(free_apps.prime_genre.value_c
print('number of Catigories in paid apps is' ,  len(paid_apps.prime_genre.value_c
#all categories has free & paid apps
```

```
number of Catigories in free apps is 23
number of Catigories in paid apps is 23
```

In [241]:
```
free_apps.head()
```

Out[241]:

	id	track_name	size_bytes	currency	price	rating_count_tot	rating_count_ver	user_rat
1	281796108	Evernote - stay organized	158578688	USD	0.0	161065	26	
2	281940292	WeatherBug - Local Weather, Radar, Maps, Alerts	100524032	USD	0.0	188583	2822	
3	282614216	eBay: Best App to Buy, Sell, Save! Online Shop...	128512000	USD	0.0	262241	649	
4	282935706	Bible	92774400	USD	0.0	985920	5320	
6	283646709	PayPal - Send and request money safely	227795968	USD	0.0	119487	879	

In [242]:
```
free = free_apps.prime_genre.value_counts().sort_index().to_frame()
paid = paid_apps.prime_genre.value_counts().sort_index().to_frame()
total = data.prime_genre.value_counts().sort_index().to_frame()
free.columns=['free']
paid.columns=['paid']
total.columns=['total']
fig  =free.join(paid).join(total)
fig['%paid'] = fig.paid*100 /fig.total
fig['%free'] = fig.free*100/ fig.total
fig
```

Out[242]:

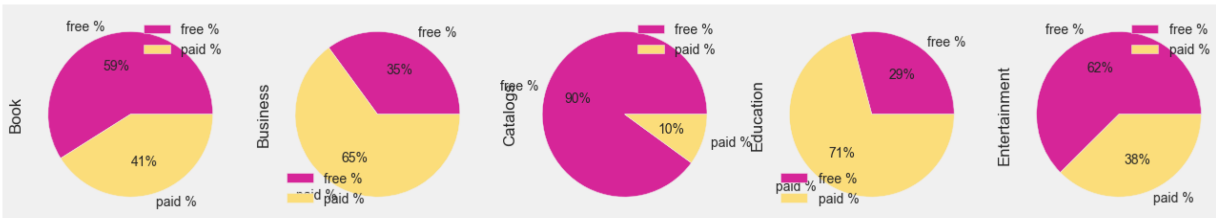
	free	paid	total	%paid	%free
Book	66	46	112	41.071429	58.928571
Business	20	37	57	64.912281	35.087719
Catalogs	9	1	10	10.000000	90.000000
Education	132	321	453	70.860927	29.139073
Entertainment	334	201	535	37.570093	62.429907
Finance	84	20	104	19.230769	80.769231
Food & Drink	43	20	63	31.746032	68.253968
Games	2257	1605	3862	41.558778	58.441222
Health & Fitness	76	104	180	57.777778	42.222222
Lifestyle	94	50	144	34.722222	65.277778
Medical	8	15	23	65.217391	34.782609
Music	67	71	138	51.449275	48.550725
Navigation	20	26	46	56.521739	43.478261
News	58	17	75	22.666667	77.333333
Photo & Video	167	182	349	52.148997	47.851003
Productivity	62	116	178	65.168539	34.831461
Reference	20	44	64	68.750000	31.250000
Shopping	121	1	122	0.819672	99.180328
Social Networking	143	24	167	14.371257	85.628743
Sports	79	35	114	30.701754	69.298246
Travel	56	25	81	30.864198	69.135802
Utilities	109	139	248	56.048387	43.951613
Weather	31	41	72	56.944444	43.055556

In general # of Free apps greater than # of paid apps but in (Business , Education ,Health ,Fitness,Medical,Music,Navigation ,Photo & Video,Productivity,Reference,Utilities,Weather) #of paid apps greater than # of free apps

In [243]:
```
# for pie chart
pies = fig[['%free','%paid']].head()
pies.columns=['free %','paid %']
```

In [244]:
```
plt.figure(figsize=(15,10))
pies.T.plot.pie(subplots=True,figsize=(20,4),colors=['#D62598','#FBDD7A'],autopct
plt.show()
```

```
<Figure size 1080x720 with 0 Axes>
```



Cataloge has Higher # of Free-Apps

Education has Higher # of Paid-Apps

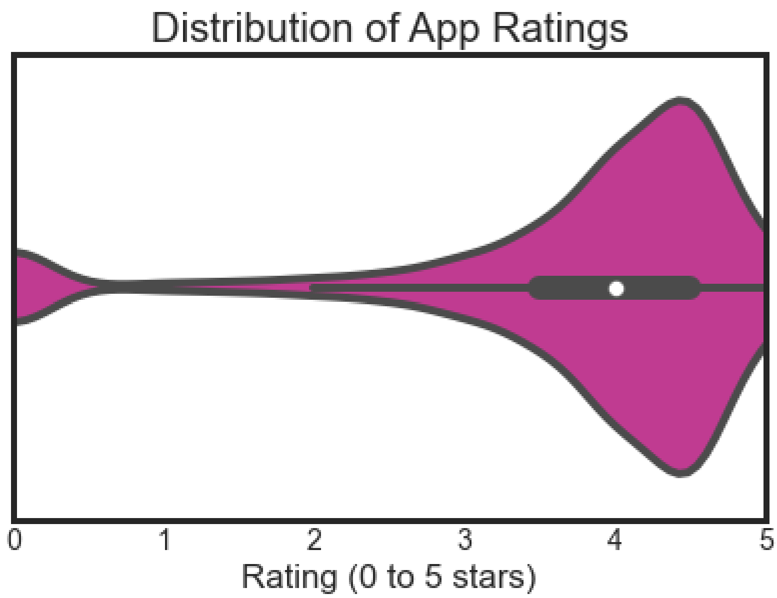
Are paid apps good enough ?

In [245]:
```
data[data['rating_count_tot']==data['rating_count_tot'].max()]
#Most rated & highest total rating for all version app:
```

Out[245]:

	id	track_name	size_bytes	currency	price	rating_count_tot	rating_count_ver	user_ra
16	284882215	Facebook	389879808	USD	0.0	2974676	212	

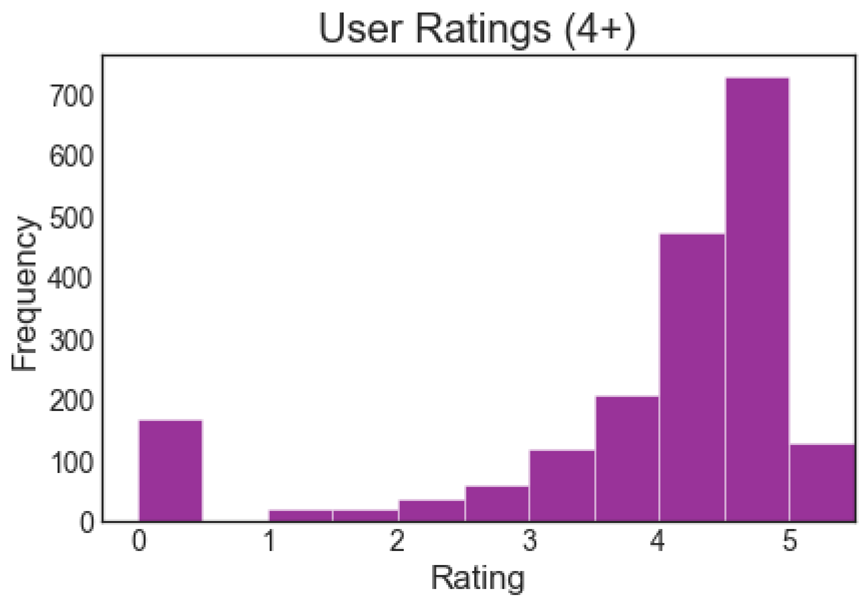
In [246]:
```
sns.set_style('white')
sns.violinplot(x=paid_apps['user_rating'],color='#D62598')
plt.xlim(0,5)
plt.xlabel('Rating (0 to 5 stars)')
_ = plt.title('Distribution of App Ratings')
```



In [247]:
```
paid_apps.cont_rating.value_counts()
```

Out[247]:
```
4+     1967
9+      549
12+     450
17+     175
Name: cont_rating, dtype: int64
```

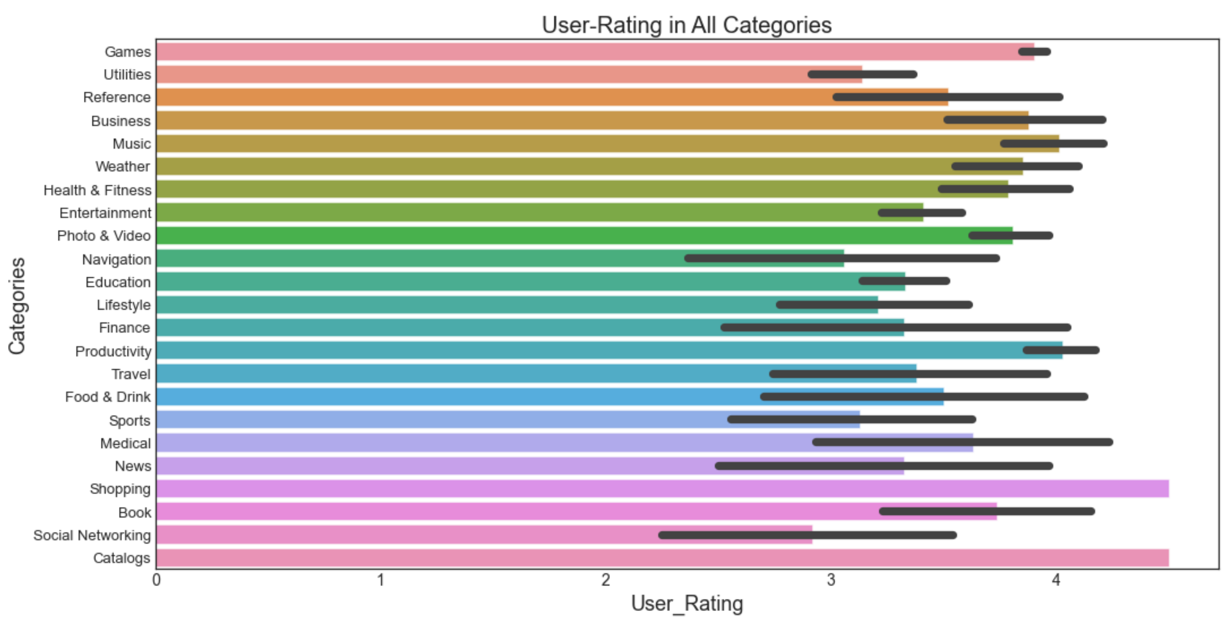
In [248]:
```
bins = (0,0.5,1,1.5,2,2.5,3,3.5,4,4.5,5,5.5)
plt.style.use('seaborn-white')
plt.hist(paid_apps[paid_apps['cont_rating']=='4+']['user_rating'],alpha=.8,bins=b
plt.xticks((0,1,2,3,4,5))
plt.title('User Ratings (4+)')
plt.xlabel('Rating')
plt.ylabel('Frequency')
_ = plt.xlim(right=5.5)
```



Rating of paid Apps between 3.5 - 5

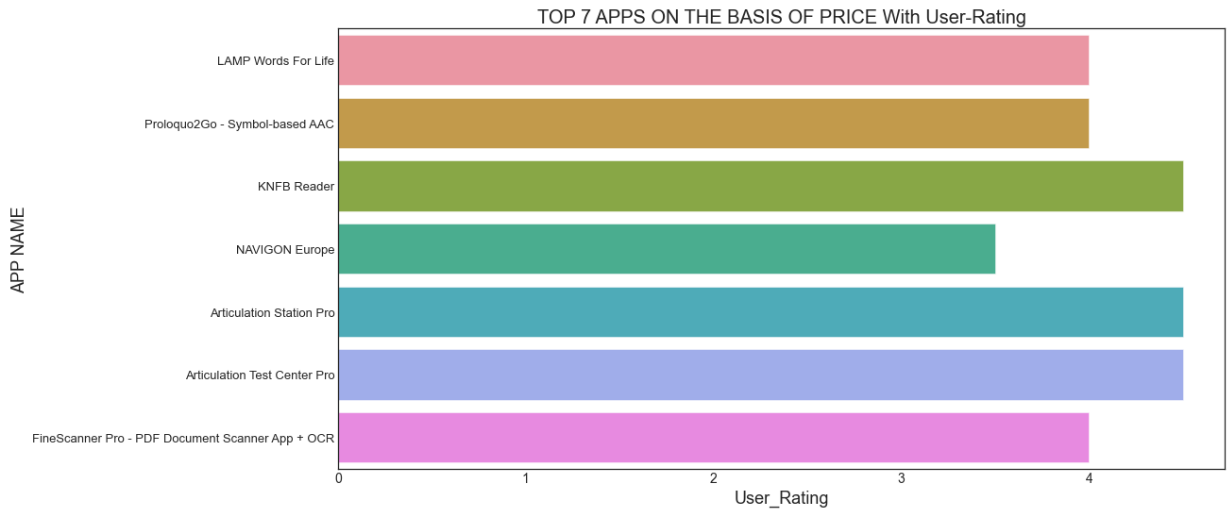
maybe say is good paid apps

In [249]:
```
visualizer(paid_apps['user_rating'],paid_apps.prime_genre, "bar", "User-Rating ir
```



higher Rating in Shopping & Catalogs

In [250]:
```
Top_Apps = Top_Apps.sort_values('price', ascending=False)

visualizer(Top_Apps.user_rating,Top_Apps.track_name, "bar", "TOP 7 APPS ON THE BA
#names of track in y axis to be readable
```



apps has high price also has high user-rating acording for that paid apps is good enough

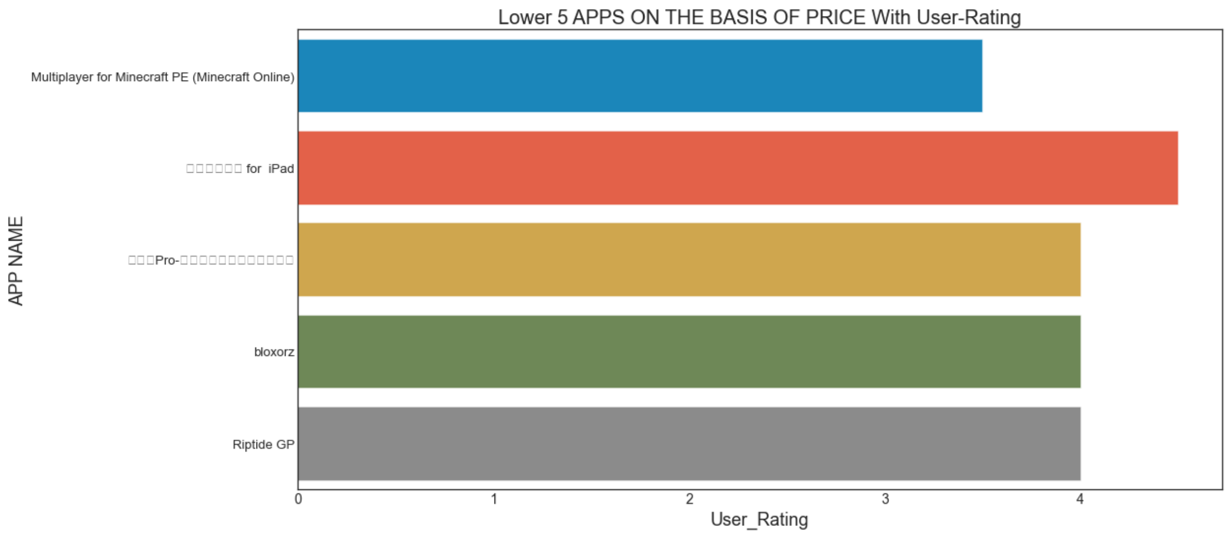
In [251]:
```
Lower_Apps=paid_apps[paid_apps.price<=50][['track_name','price','prime_genre','us
Lower_Apps.head()
```

Out[251]:

	track_name	price	prime_genre	user_rating
0	PAC-MAN Premium	3.99	Games	4.0
5	Shanghai Mahjong	0.99	Games	4.0
8	PCalc - The Best Calculator	9.99	Utilities	4.5
9	Ms. PAC-MAN	3.99	Games	4.0
10	Solitaire by MobilityWare	4.99	Games	4.5

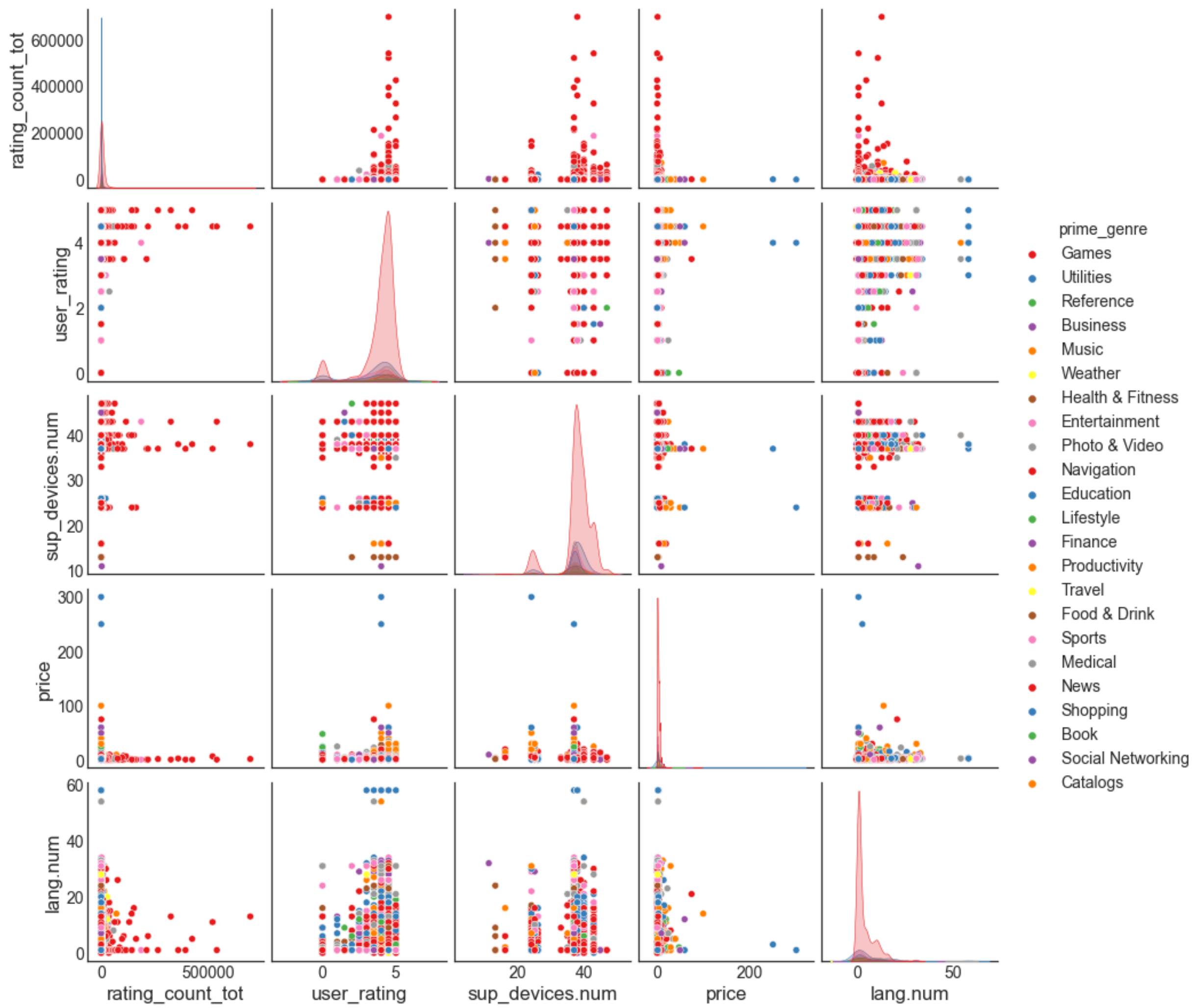
lower_apps in price also has high user-rating so paid apps is good

In [252]:
```
Lower_Apps = Lower_Apps.sort_values('price', ascending=True)
lower = Lower_Apps.head()
visualizer(lower.user_rating,lower.track_name, "bar", "Lower 5 APPS ON THE BASIS
```



In [253]:
```
numCol = paid_apps[['rating_count_tot', 'user_rating', 'sup_devices.num', 'price'
sns.pairplot(data = numCol, hue='prime_genre',palette='Set1')
```

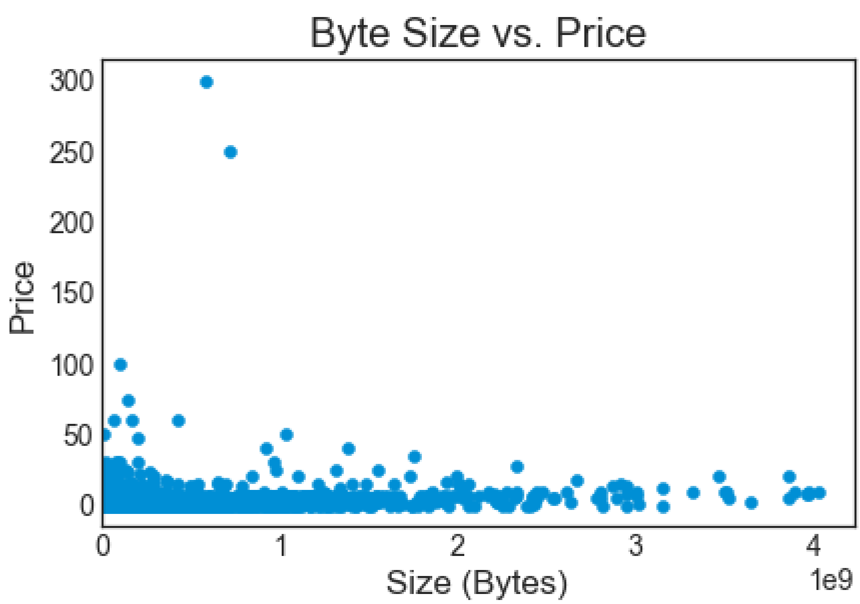
Out[253]: <seaborn.axisgrid.PairGrid at 0x2272768d3a0>



As the size of the app increases do they get pricier ?

In [254]:
```
plt.style.use('seaborn-white')
plt.scatter(data['size_bytes'],data['price'])
plt.title('Byte Size vs. Price')
plt.xlabel('Size (Bytes)')
plt.ylabel('Price')
plt.xlim(0)
```

Out[254]: (0.0, 4227238656.0)



size of App not corelated with price

we show that if size is big ,price is low

the value of an app to the user isn't necessarily related to its size.

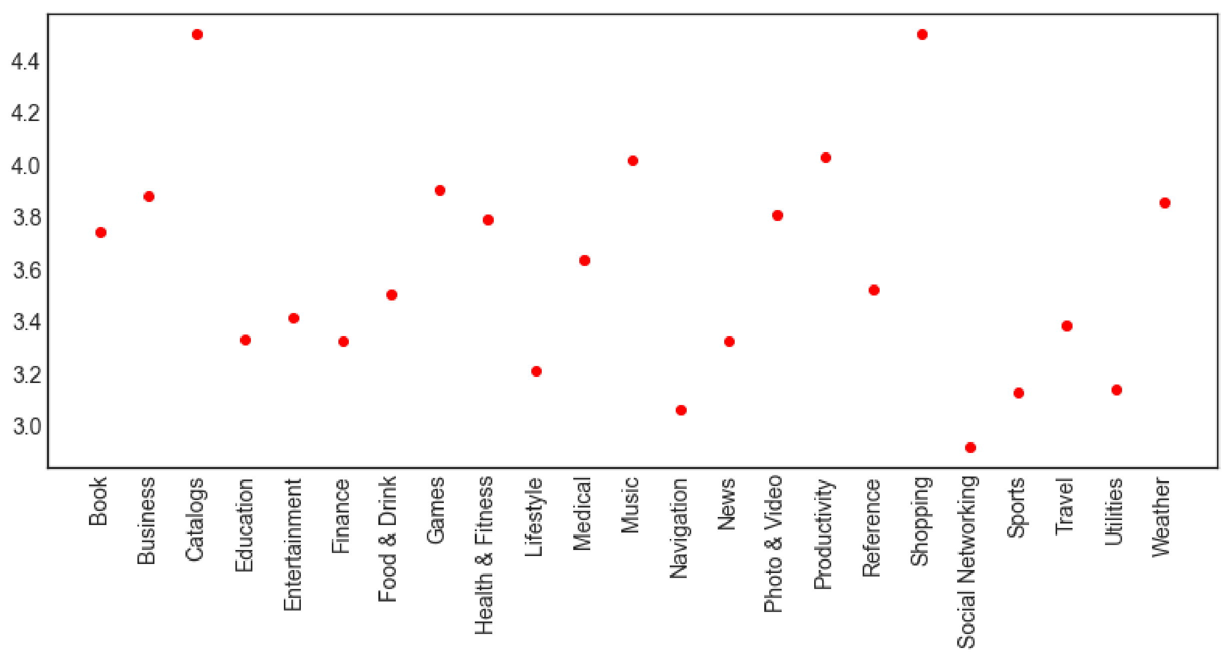
How are the apps distributed category wise ? Can we split by paid category ?

In [255]:
```
grp = paid_apps.groupby('prime_genre')
x = grp['user_rating'].agg(np.mean)
y = grp['price'].agg(np.sum)
z = grp['user_rating_ver'].agg(np.mean)
print(x)
print(y)
print(z)
```

```
prime_genre
Book                 3.739130
Business             3.878378
Catalogs             4.500000
Education            3.331776
Entertainment        3.410448
Finance              3.325000
Food & Drink         3.500000
Games                3.904984
Health & Fitness     3.788462
Lifestyle            3.210000
Medical              3.633333
Music                4.014085
Navigation           3.057692
News                 3.323529
Photo & Video        3.807692
Productivity         4.030172
Reference            3.522727
Shopping             4.500000
Social Networking    2.916667
Sports               3.128571
Travel               3.380000
Utilities            3.140288
Weather              3.853659
Name: user_rating, dtype: float64
prime_genre
Book                  200.54
Business              291.63
Catalogs                7.99
Education            1824.79
Entertainment         475.99
Finance                43.80
Food & Drink           97.80
Games                5533.95
Health & Fitness      344.96
Lifestyle             127.50
Medical               201.85
Music                 667.29
Navigation            189.74
News                   38.83
Photo & Video         514.18
Productivity          770.84
Reference             309.56
Shopping                1.99
Social Networking      56.76
Sports                108.65
Travel                 90.75
Utilities             408.61
Weather               115.59
```

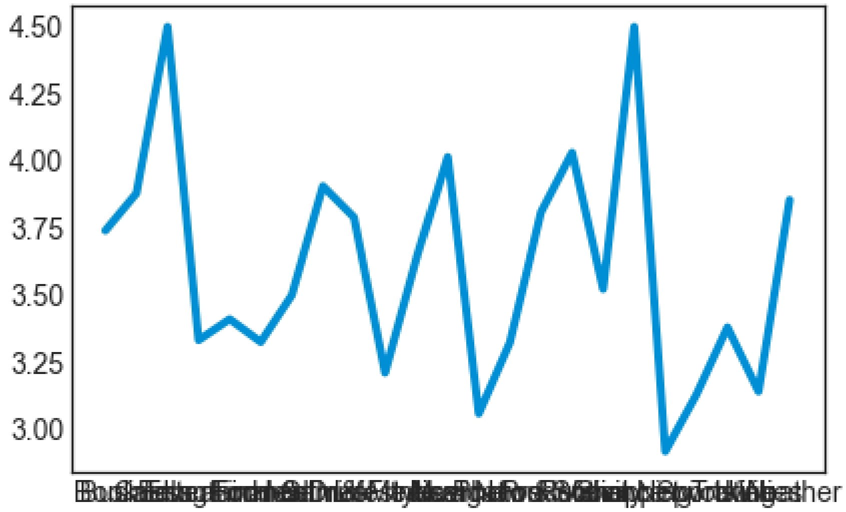
```
Name: price, dtype: float64
prime_genre
Book                 3.163043
Business             3.729730
Catalogs             5.000000
Education            2.992212
Entertainment        3.129353
Finance              2.000000
Food & Drink         2.575000
Games                3.777882
Health & Fitness     3.485577
Lifestyle            2.960000
Medical              3.366667
Music                3.683099
Navigation           2.500000
News                 2.647059
Photo & Video        3.681319
Productivity         3.689655
Reference            2.920455
Shopping             5.000000
Social Networking    2.729167
Sports               2.885714
Travel               3.640000
Utilities            2.899281
Weather              3.597561
Name: user_rating_ver, dtype: float64
```

In [256]:
```
#again need to expand
plt.figure(figsize=(12,5))
plt.plot(x, 'ro')
plt.xticks(rotation=90)
plt.show()
```



In [257]:
```
# lets plot
plt.plot(x)
```

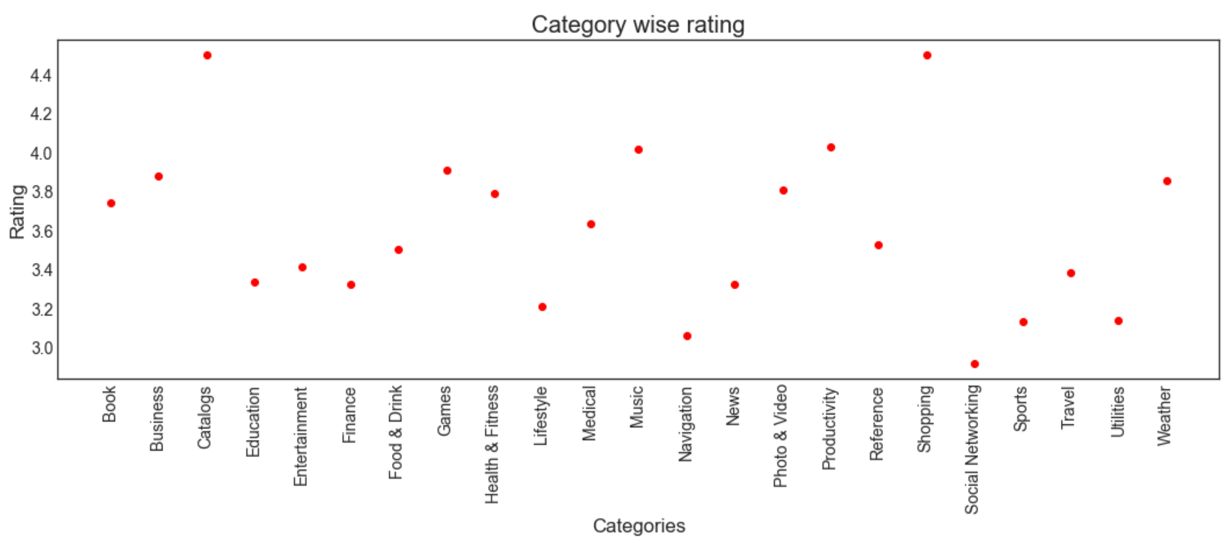
Out[257]: `[<matplotlib.lines.Line2D at 0x22727a087f0>]`



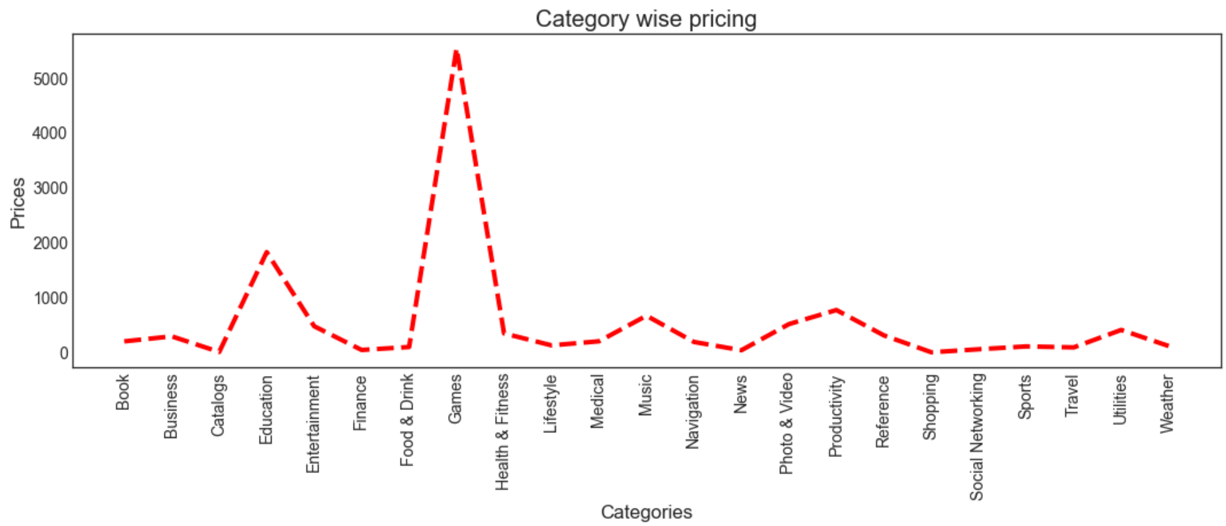
High Rating with catalogs , shopping

Low rating in social networking

In [258]:
```
# for x
plt.figure(figsize=(16,5))
plt.plot(x, 'ro')
plt.xticks(rotation=90)
plt.title('Category wise rating')
plt.xlabel('Categories')
plt.ylabel('Rating')
plt.show()
```



In [259]:
```
# for Y
plt.figure(figsize=(16,5))
plt.plot(y, 'r--')
plt.xticks(rotation=90)
plt.title('Category wise pricing')
plt.xlabel('Categories')
plt.ylabel('Prices')
plt.show()
```

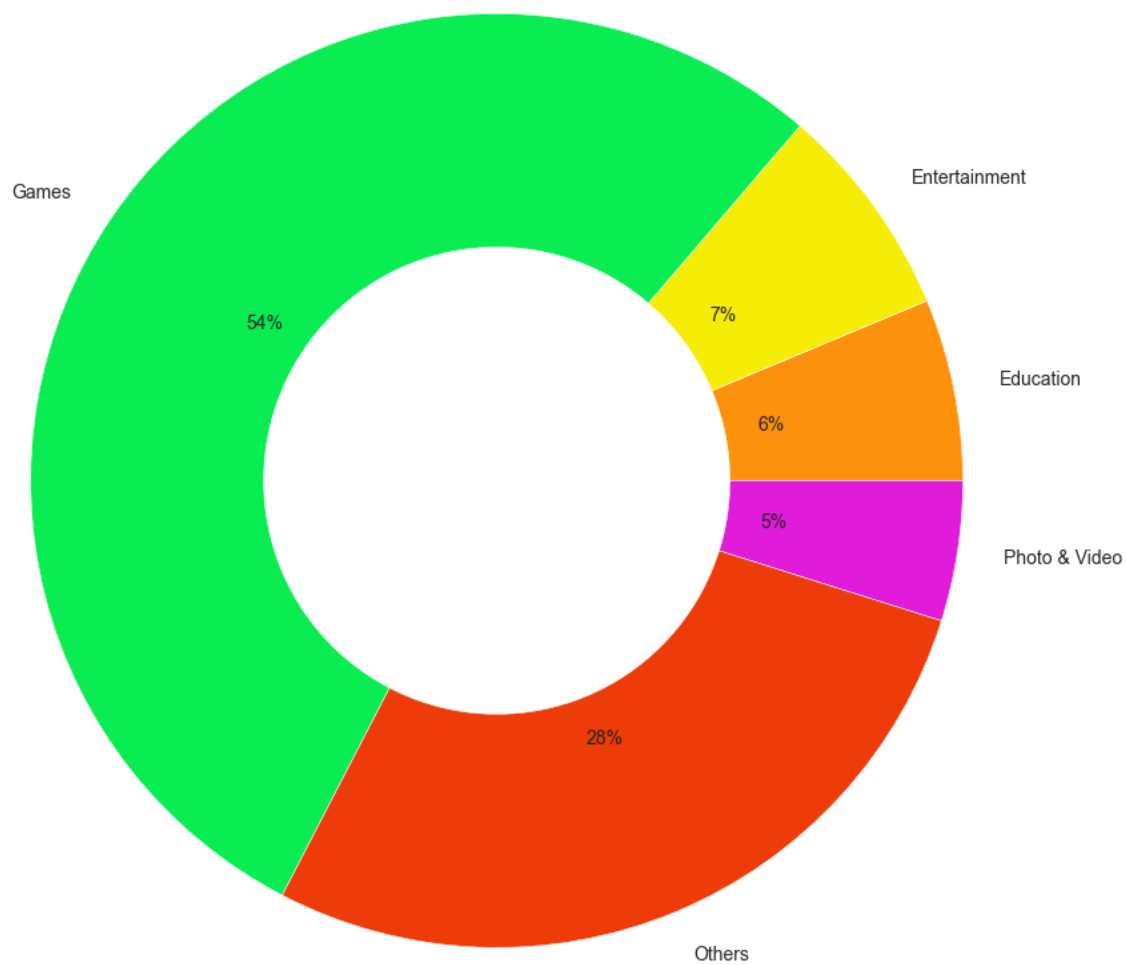


In [260]:
```
# reducing the number of categories to 5 categories

s = data.prime_genre.value_counts().index[:4]
def categ(x):
    if x in s:
        return x
    else :
        return "Others"

data['broad_genre']= data.prime_genre.apply(lambda x : categ(x))
```

In [261]:
```
data['broad_genre'].value_counts()
```

Out[261]:
```
Games            3862
Others           1998
Entertainment     535
Education         453
Photo & Video     349
Name: broad_genre, dtype: int64
```

In [262]:
```
BlueOrangeWapang = ['#fc910d','#f5ed05','#09ed52','#ed3b09','#e01bda']
plt.figure(figsize=(15,15))
label_names=data.broad_genre.value_counts().sort_index().index
size = data.broad_genre.value_counts().sort_index().tolist()

my_circle=plt.Circle( (0,0), 0.5, color='white')
plt.pie(size, labels=label_names, colors=BlueOrangeWapang ,autopct = '%1.0f%%',)
p=plt.gcf()
p.gca().add_artist(my_circle)
plt.show()
```



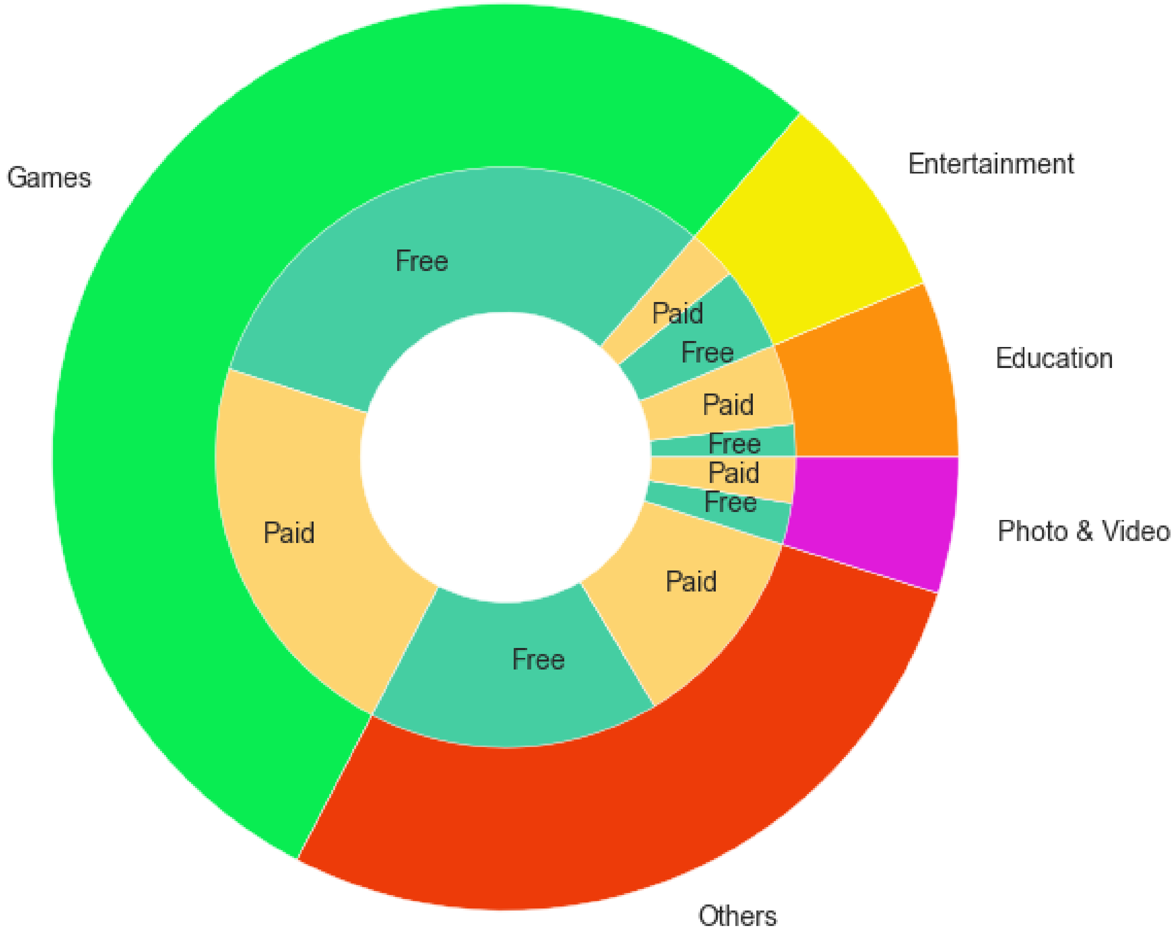
In [263]:
```
free = data[data.price==0].broad_genre.value_counts().sort_index().to_frame()
paid = data[data.price>0].broad_genre.value_counts().sort_index().to_frame()
total = data.broad_genre.value_counts().sort_index().to_frame()
free.columns=['free']
paid.columns=['paid']
total.columns=['total']
five_ca  =free.join(paid).join(total)
five_ca['Paid_per'] = five_ca.paid*100 /five_ca.total
five_ca['Free_per'] = five_ca.free*100/ five_ca.total
five_ca
```

Out[263]:

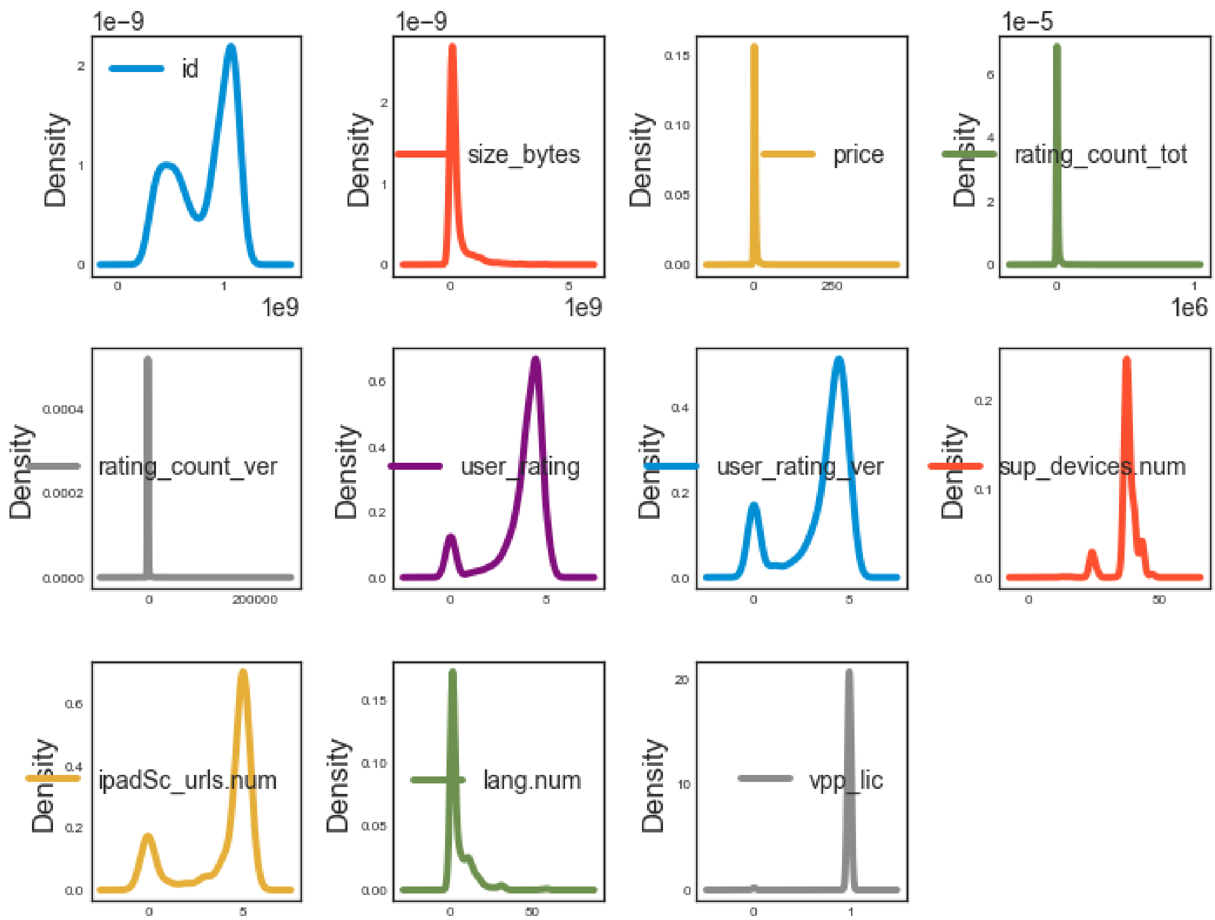
	free	paid	total	Paid_per	Free_per
Education	132	321	453	70.860927	29.139073
Entertainment	334	201	535	37.570093	62.429907
Games	2257	1605	3862	41.558778	58.441222
Others	1166	832	1998	41.641642	58.358358
Photo & Video	167	182	349	52.148997	47.851003

In [264]:
```
plt.figure(figsize=(15,15))
f=pd.DataFrame(index=np.arange(0,10,2),data=five_ca['free'].values,columns=['num'
p=pd.DataFrame(index=np.arange(1,11,2),data=five_ca['paid'].values,columns=['num'
final = pd.concat([f,p],names=['labels']).sort_index()
final.num.tolist()

plt.figure(figsize=(25,25))
group_names=data.broad_genre.value_counts().sort_index().index
group_size=data.broad_genre.value_counts().sort_index().tolist()
h = ['Free', 'Paid']
subgroup_names= 5*h
sub= ['#45cea2','#fdd470']
subcolors= 5*sub
subgroup_size=final.num.tolist()


# First Ring (outside)
fig, ax = plt.subplots()
ax.axis('equal')
mypie, _ = ax.pie(group_size, radius=2.5, labels=group_names, colors=BlueOrangeWa
plt.setp( mypie, width=1.2, edgecolor='white')

# Second Ring (Inside)
mypie2, _ = ax.pie(subgroup_size, radius=1.6, labels=subgroup_names, labeldistanc
plt.setp( mypie2, width=0.8, edgecolor='white')
plt.margins(0,0)

# show it
plt.show()
```

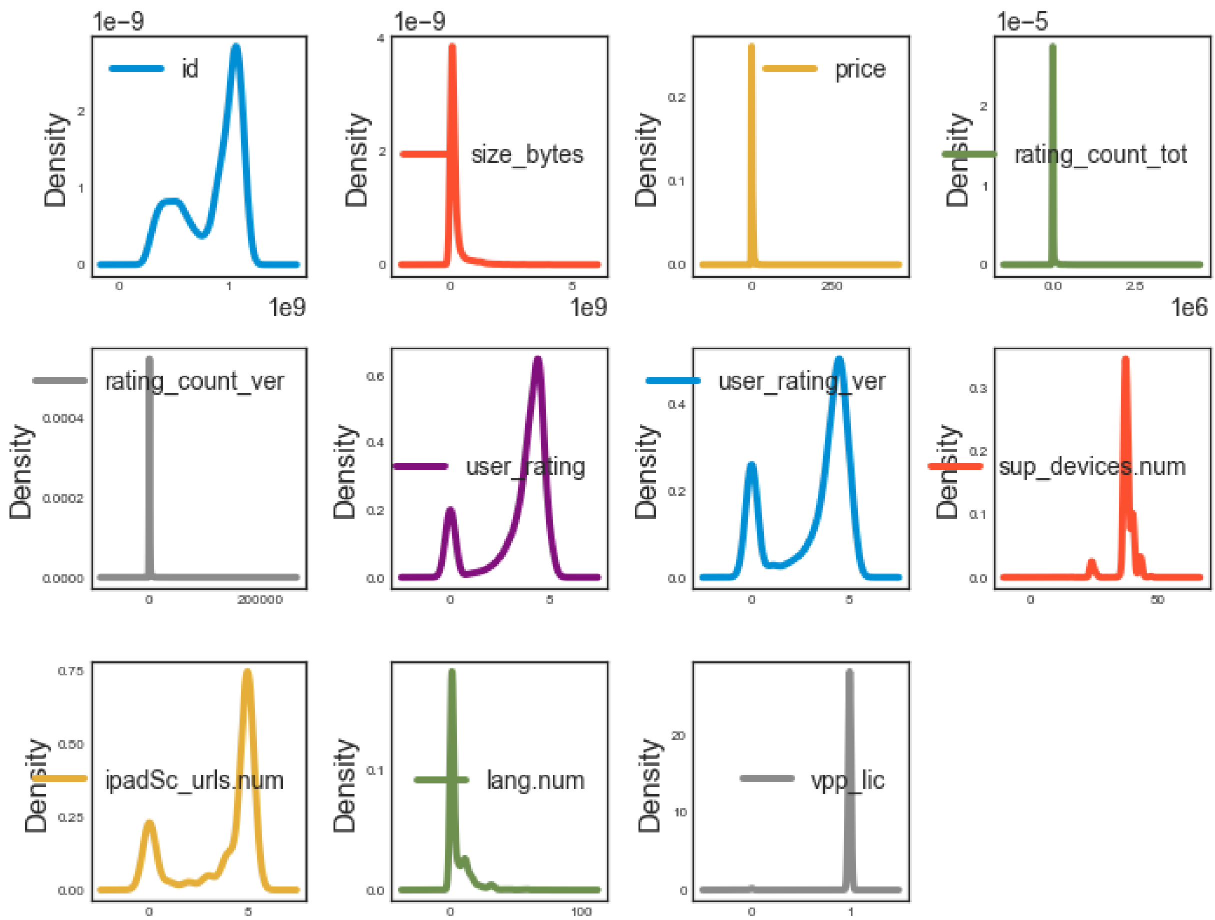
```
<Figure size 1080x1080 with 0 Axes>

<Figure size 1800x1800 with 0 Axes>
```


Games
Entertainment
Education
Photo & Video
Others
Free
Paid
Free
Paid
Free
Paid
Free
Paid
Free
Paid

In [265]:
```
paid_apps.plot(kind='density' , subplots=True , layout=(4,4) , sharex=False ,
               fontsize=8 , figsize=(10,10))
plt.tight_layout()
```



In [266]:
```
data.plot(kind='density' , subplots=True , layout=(4,4) , sharex=False ,
          fontsize=8 , figsize=(10,10))
plt.tight_layout()
```



In [267]:
```
data.rating_count_tot.value_counts()
```

Out[267]:
```
0        929
1        120
7         48
9         47
5         46
        ...
3116       1
3120       1
1073       1
3124       1
10239      1
Name: rating_count_tot, Length: 3185, dtype: int64
```

In [268]:
```
free_apps.vpp_lic.value_counts()
```

Out[268]:
```
1    4035
0      21
Name: vpp_lic, dtype: int64
```

In [269]:
```
paid_apps.vpp_lic.value_counts()
```

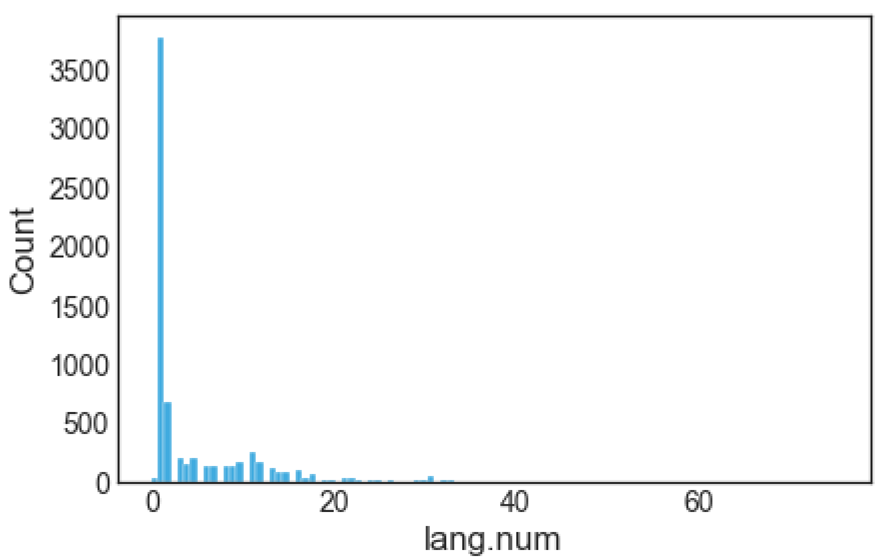
Out[269]:
```
1    3112
0      29
Name: vpp_lic, dtype: int64
```

In [270]:
```
data.vpp_lic.value_counts()
```

Out[270]:
```
1    7147
0      50
Name: vpp_lic, dtype: int64
```

In [271]:
```
sns.histplot(data['lang.num'])
```

Out[271]:
```
<AxesSubplot:xlabel='lang.num', ylabel='Count'>
```



Feature Engineering

In [272]:
```
from sklearn.preprocessing import LabelEncoder
USD_LABEL =  LabelEncoder()
data['currency']= USD_LABEL.fit_transform(data['currency'])
```

In [273]:
```
data.drop(['broad_genre'] ,
          #['currency'],
          axis = 1, inplace = True)
```

In [274]:
```
data.drop(['currency'],
          axis = 1, inplace = True)
```

In [275]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7197 entries, 0 to 7196
Data columns (total 15 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   id                7197 non-null   int64  
 1   track_name        7197 non-null   object 
 2   size_bytes        7197 non-null   int64  
 3   price             7197 non-null   float64
 4   rating_count_tot  7197 non-null   int64  
 5   rating_count_ver  7197 non-null   int64  
 6   user_rating       7197 non-null   float64
 7   user_rating_ver   7197 non-null   float64
 8   ver               7197 non-null   object 
 9   cont_rating       7197 non-null   object 
 10  prime_genre       7197 non-null   object 
 11  sup_devices.num   7197 non-null   int64  
 12  ipadSc_urls.num   7197 non-null   int64  
 13  lang.num          7197 non-null   int64  
 14  vpp_lic           7197 non-null   int64  
dtypes: float64(3), int64(8), object(4)
memory usage: 843.5+ KB
```

In [276]:
```
#encoding object columns int
track_name_LABEL =  LabelEncoder()
data['track_name']= track_name_LABEL.fit_transform(data['track_name'])
```

In [277]:
```
ver_LABEL =  LabelEncoder()
data['ver']= ver_LABEL.fit_transform(data['ver'])

prime_genre_LABEL =  LabelEncoder()
data['prime_genre']= prime_genre_LABEL.fit_transform(data['prime_genre'])


cont_rating_LABEL =  LabelEncoder()
data['cont_rating']= cont_rating_LABEL.fit_transform(data['cont_rating'])
```

In [278]:
```
data.head()
```

Out[278]:

	id	track_name	size_bytes	price	rating_count_tot	rating_count_ver	user_rating	user_
0	281656475	3676	100788224	3.99	21292	26	4.0	
1	281796108	1664	158578688	0.00	161065	26	4.0	
2	281940292	5870	100524032	0.00	188583	2822	3.5	
3	282614216	6132	128512000	0.00	262241	649	4.0	
4	282935706	527	92774400	0.00	985920	5320	4.5	

In [279]:
```
data.drop(['id'] , axis =1 , inplace = True)
```

In [280]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7197 entries, 0 to 7196
Data columns (total 14 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   track_name        7197 non-null   int32  
 1   size_bytes        7197 non-null   int64  
 2   price             7197 non-null   float64
 3   rating_count_tot  7197 non-null   int64  
 4   rating_count_ver  7197 non-null   int64  
 5   user_rating       7197 non-null   float64
 6   user_rating_ver   7197 non-null   float64
 7   ver               7197 non-null   int32  
 8   cont_rating       7197 non-null   int32  
 9   prime_genre       7197 non-null   int32  
 10  sup_devices.num   7197 non-null   int64  
 11  ipadSc_urls.num   7197 non-null   int64  
 12  lang.num          7197 non-null   int64  
 13  vpp_lic           7197 non-null   int64  
dtypes: float64(3), int32(4), int64(7)
memory usage: 674.8 KB
```

In [281]:
```
#Data about Data
data.describe().style.background_gradient(cmap='Purples')
```

Out[281]:

	track_name	size_bytes	price	rating_count_tot	rating_count_ver	user_ratin
count	7197.000000	7197.000000	7197.000000	7197.000000	7197.000000	7197.00000
mean	3597.221203	199134453.825066	1.726218	12892.907184	460.373906	3.52695
std	2077.028362	359206913.538703	5.833006	75739.408675	3920.455183	1.51794
min	0.000000	589824.000000	0.000000	0.000000	0.000000	0.00000
25%	1799.000000	46922752.000000	0.000000	28.000000	1.000000	3.50000
50%	3597.000000	97153024.000000	0.000000	300.000000	23.000000	4.00000
75%	5396.000000	181924864.000000	1.990000	2793.000000	140.000000	4.50000
max	7194.000000	4025969664.000000	299.990000	2974676.000000	177050.000000	5.00000

we found variation in data as standard deviation is different

some of columns has big outliers

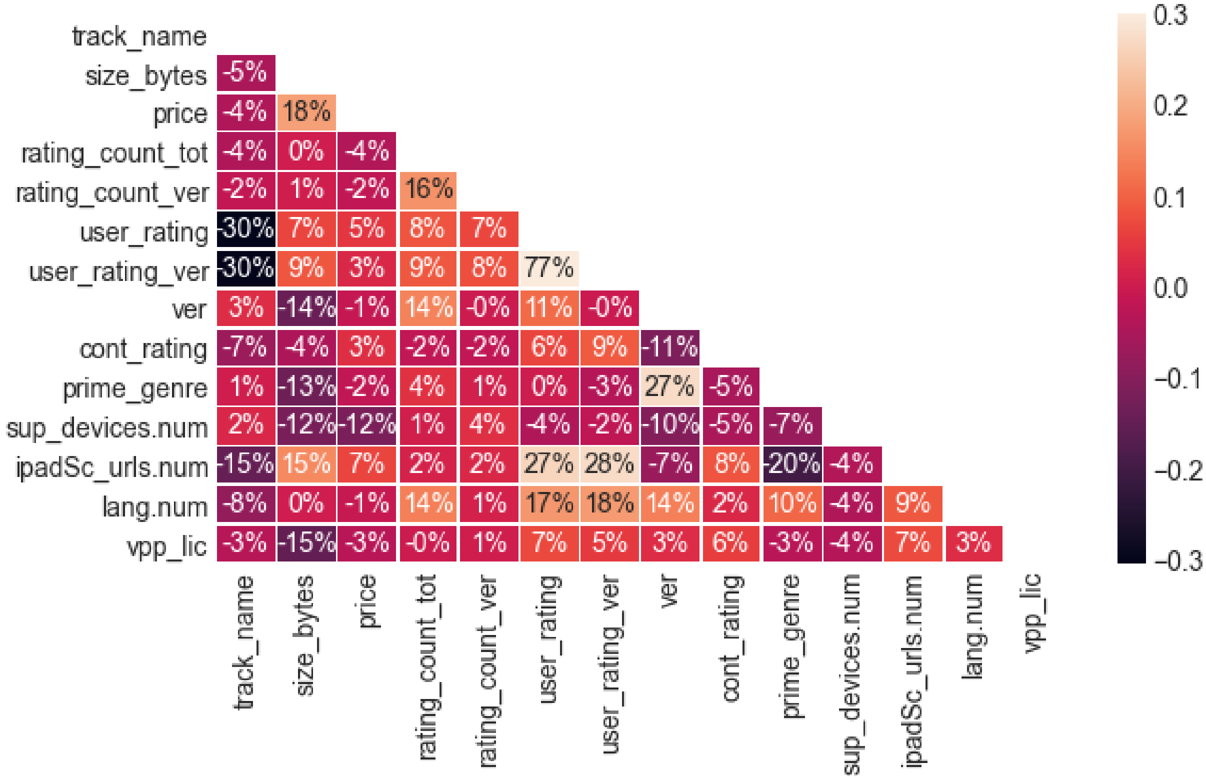
In [282]:
```
D_corr = data.corr()
D_corr.style.background_gradient()
```

Out[282]:

	track_name	size_bytes	price	rating_count_tot	rating_count_ver	user_rating
track_name	1.000000	-0.049030	-0.039913	-0.043531	-0.015036	-0.303899
size_bytes	-0.049030	1.000000	0.182392	0.004486	0.006337	0.066256
price	-0.039913	0.182392	1.000000	-0.039044	-0.018012	0.046601
rating_count_tot	-0.043531	0.004486	-0.039044	1.000000	0.163645	0.083310
rating_count_ver	-0.015036	0.006337	-0.018012	0.163645	1.000000	0.068754
user_rating	-0.303899	0.066256	0.046601	0.083310	0.068754	1.000000
user_rating_ver	-0.300058	0.086075	0.025173	0.088744	0.077840	0.774140
ver	0.031308	-0.139159	-0.010842	0.142502	-0.000678	0.113104
cont_rating	-0.068895	-0.044634	0.033551	-0.016398	-0.016948	0.064212
prime_genre	0.006130	-0.134438	-0.017413	0.039188	0.011090	0.000975
sup_devices.num	0.021808	-0.118347	-0.115361	0.008832	0.037951	-0.042451
ipadSc_urls.num	-0.145207	0.152697	0.066100	0.015734	0.024333	0.265671
lang.num	-0.081477	0.004614	-0.006713	0.137675	0.013287	0.170976
vpp_lic	-0.030828	-0.150418	-0.029942	-0.000982	0.006460	0.069816

Corellation between columns is very low

In [283]:
```
mask = np.zeros_like(data.corr())
mask[np.triu_indices_from(mask)] = True
with sns.axes_style("ticks"):
    f, ax = plt.subplots(figsize=(9, 5))
    ax = sns.heatmap(data.corr(), mask=mask, vmax=.3,annot=True,fmt=".0%",linewic
```



Correlation between user_rating_var , user_rating is high 77%

ipadSc_urls.num , user_rating is 27%

ipadSc_urls.num , user_rating_var is 28%

PPS(Predictive Power Score)

Non Linear Data

In [284]:
```
#Calculating ppscore
import ppscore
c=pps.matrix(data)
c
```

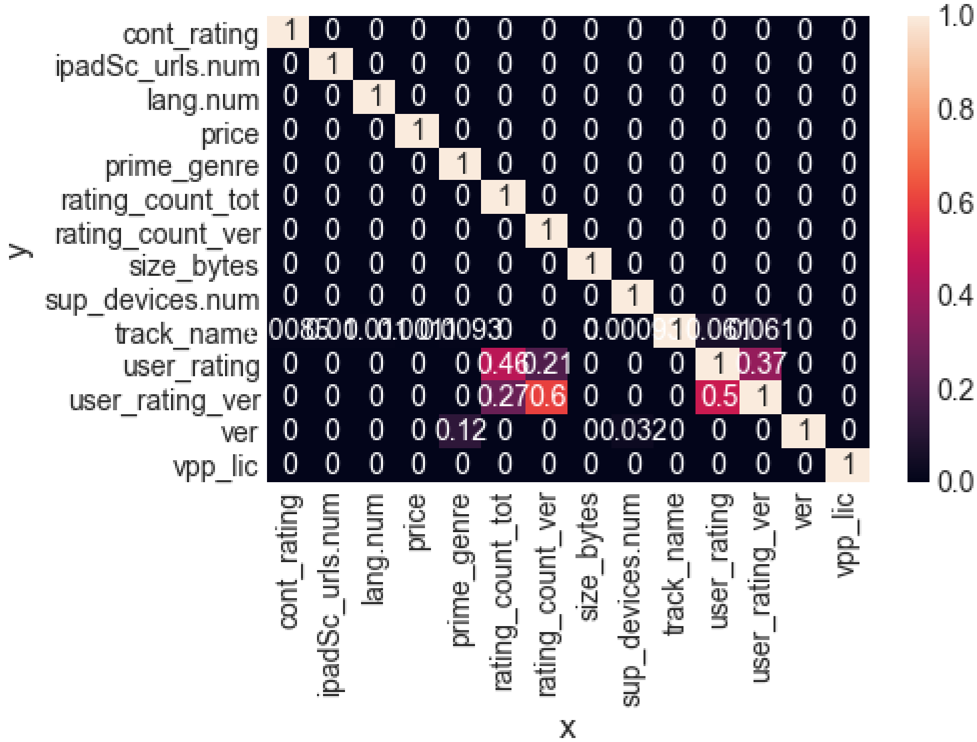
Out[284]:

	x	y	ppscore	case	is_valid_score	metric	baseline_score	r
0	track_name	track_name	1.0	predict_itself	True	None	0.000000e+00	1
1	track_name	size_bytes	0.0	regression	True	mean absolute error	1.507944e+08	2
2	track_name	price	0.0	regression	True	mean absolute error	1.771204e+00	2
3	track_name	rating_count_tot	0.0	regression	True	mean absolute error	1.216113e+04	2
4	track_name	rating_count_ver	0.0	regression	True	mean absolute error	4.637924e+02	7
...	...	...	...	...	...	...	...	
191	vpp_lic	prime_genre	0.0	regression	True	mean absolute error	2.912000e+00	3
192	vpp_lic	sup_devices.num	0.0	regression	True	mean absolute error	1.827400e+00	1
193	vpp_lic	ipadSc_urls.num	0.0	regression	True	mean absolute error	1.280600e+00	1
194	vpp_lic	lang.num	0.0	regression	True	mean absolute error	4.398600e+00	5
195	vpp_lic	vpp_lic	1.0	predict_itself	True	None	0.000000e+00	1

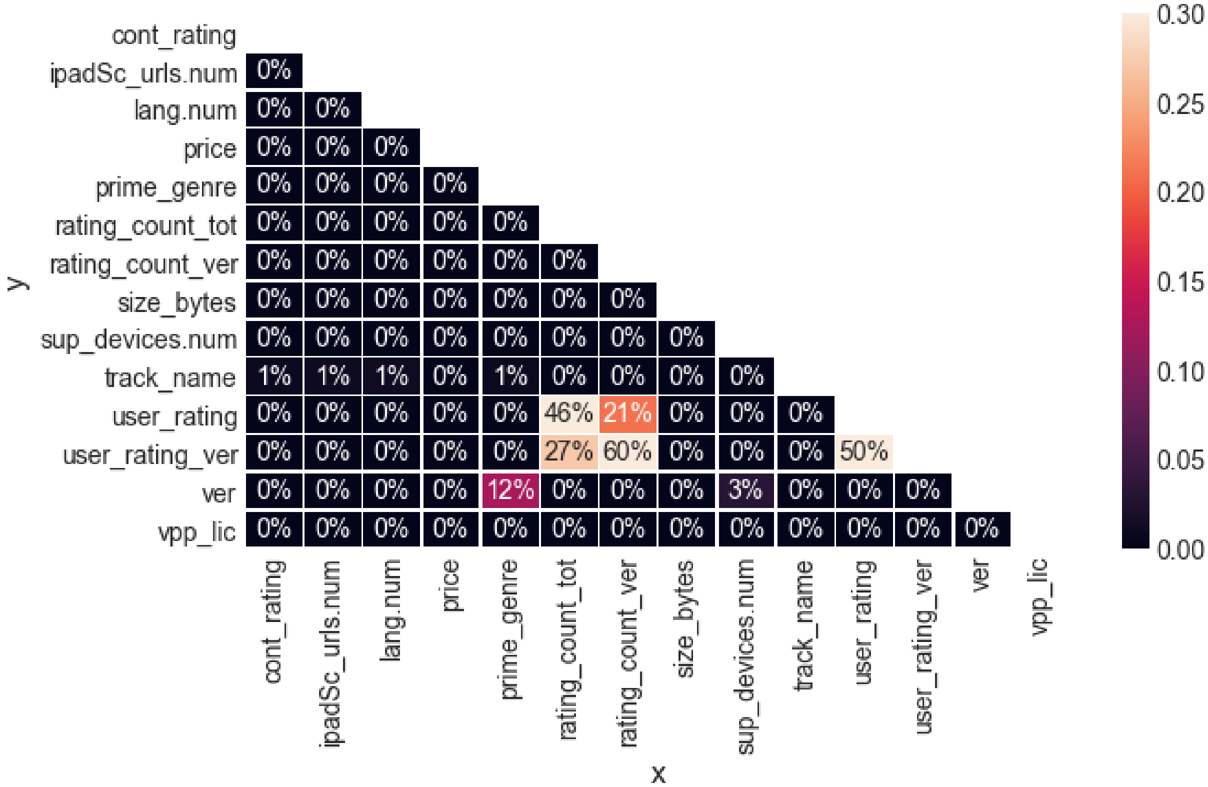
196 rows × 9 columns

In [285]:
```
figsize=(20,20)
a = pps.matrix(data).pivot(columns='x', index='y', values='ppscore')
sns.heatmap(a, annot=True)
```

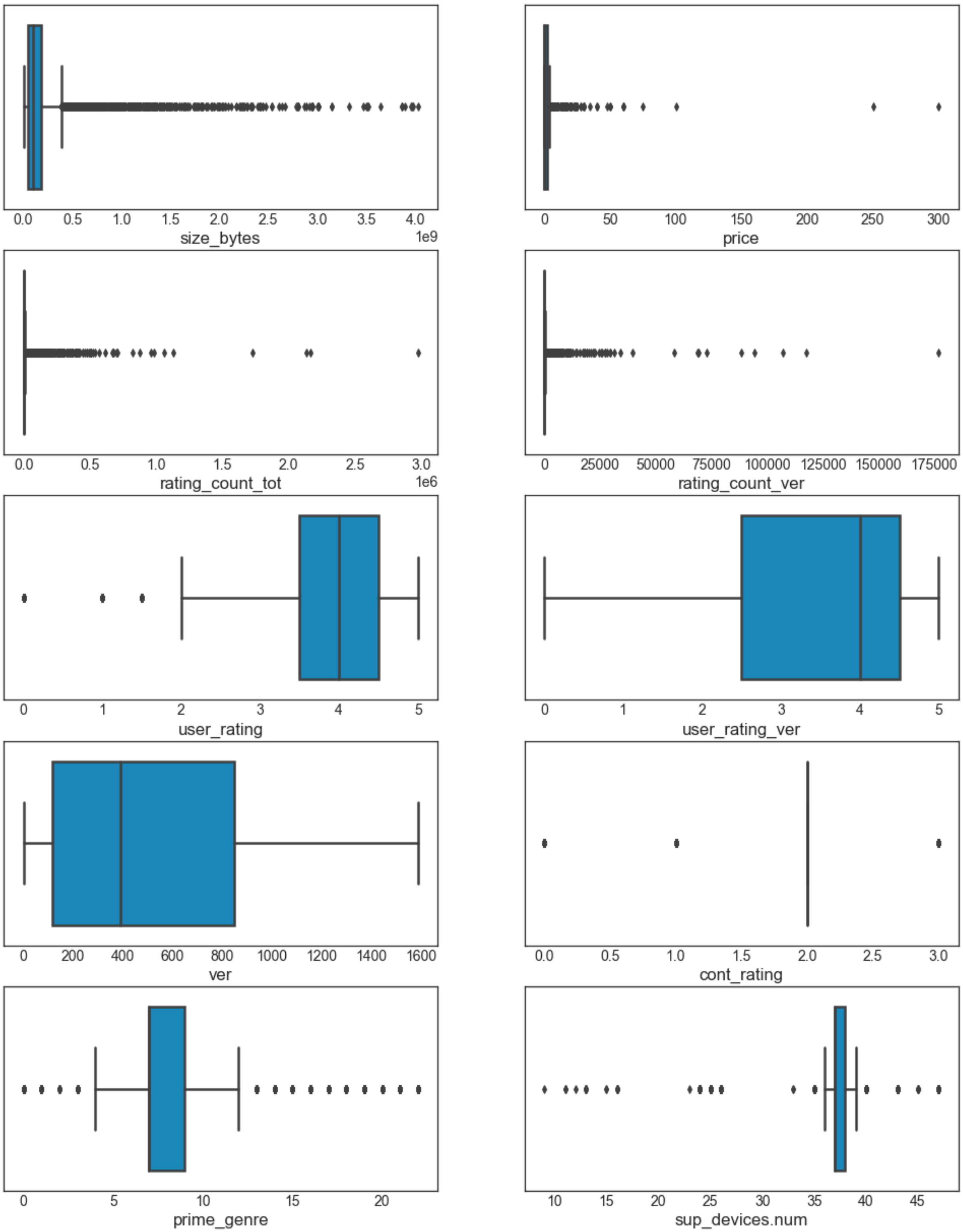
Out[285]: <AxesSubplot:xlabel='x', ylabel='y'>



In [286]:
```
mask = np.zeros_like(a)
mask[np.triu_indices_from(mask)] = True
with sns.axes_style("ticks"):
    f, ax = plt.subplots(figsize=(9, 5))
    ax = sns.heatmap(a, mask=mask, vmax=.3,annot=True,fmt=".0%",linewidth=0.5,squ
```



In [287]:
```
#Show outliers with boxplot
plt.figure(figsize = (15,20))
col_names = [ 'size_bytes', 'price', 'rating_count_tot',
       'rating_count_ver', 'user_rating', 'user_rating_ver', 'ver',
       'cont_rating', 'prime_genre', 'sup_devices.num']
for i in range(10):
    plt.subplot(5,2,i+1)#3 number of row #2 number of columns
    sns.boxplot(x=data[col_names[i]], linewidth=2.5)
plt.show()
```



In [288]:
```
data.shape
```

Out[288]:
```
(7197, 14)
```

In [289]:
```
 outliers_list = []
# For each feature find the data points with extreme high or low values
for feature in data.keys():

    # Calculate Q1 (25th percentile of the data) for the given feature
    Q1 = np.percentile(data[feature], 25)

    # Calculate Q3 (75th percentile of the data) for the given feature
    Q3 = np.percentile(data[feature], 75)

    # Use the interquartile range to calculate an outlier step (1.5 times the int
    step = (Q3 - Q1) * 1.5

    # Display the outliers
    print("Data points considered outliers for the feature '{}':".format(feature)
    outliers = list(data[~((data[feature] >= Q1 - step) & (data[feature] <= Q3 +

    display(data[~((data[feature] >= Q1 - step) & (data[feature] <= Q3 + step))])
    print('-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
    outliers_list.extend(outliers)

#print("List of Outliers -> \n :{}".format(outliers_list))
```

Data points considered outliers for the feature 'track_name':

track_name	size_bytes	price	rating_count_tot	rating_count_ver	user_rating	user_rating_ver

-=-
=
Data points considered outliers for the feature 'size_bytes':

	track_name	size_bytes	price	rating_count_tot	rating_count_ver	user_rating	user_ratir
16	1743	389879808	0.00	2974676	212	3.5	
103	4440	431771648	2.99	35074	403	4.5	
115	4054	723764224	249.99	773	10	4.0	
152	5327	430128128	6.99	54408	65	3.5	
281	2232	878883840	4.99	15142	73	4.0	

In [290]:
```
#duplicate_outliers_list = list(set([x for x in outliers_list if outliers_list.co
#duplicate_outliers_list.sort()
#print("\nList of Common Outliers -> {}".format(duplicate_outliers_list))
# Remove the outliers
#outliers  = duplicate_outliers_list
#new_data = data.drop(data.index[outliers]).reset_index(drop = True)
```

In [291]:
```
#new_data.head()
```

In [292]:
```
#new_data.shape
```

Standardizing - RobustScaler

In [293]:
```
#Before clustering, we transform features from original version to standardize ve
#Creat Object from RobustScaler
s = RobustScaler()
#fit_transform for dataset
data_robustscaler = s.fit_transform(data)
```

In [294]:
```
data.columns
```

Out[294]:
```
Index(['track_name', 'size_bytes', 'price', 'rating_count_tot',
       'rating_count_ver', 'user_rating', 'user_rating_ver', 'ver',
       'cont_rating', 'prime_genre', 'sup_devices.num', 'ipadSc_urls.num',
       'lang.num', 'vpp_lic'],
      dtype='object')
```

In [295]:
```
df_robust = pd.DataFrame(data_robustscaler , columns=['track_name', 'size_bytes',
       'rating_count_ver', 'user_rating', 'user_rating_ver', 'ver',
       'cont_rating', 'prime_genre', 'sup_devices.num', 'ipadSc_urls.num',
       'lang.num', 'vpp_lic'])
df_robust.head()
```

Out[295]:

	track_name	size_bytes	price	rating_count_tot	rating_count_ver	user_rating	user_rating_ve
0	0.021963	0.026927	2.005025	7.592043	0.021583	0.0	0.2
1	-0.537392	0.454998	0.000000	58.142857	0.021583	0.0	-0.2
2	0.631915	0.024970	0.000000	68.095118	20.136691	-0.5	0.2
3	0.704754	0.232285	0.000000	94.734539	4.503597	0.0	0.2
4	-0.853489	-0.032434	0.000000	356.462929	38.107914	0.5	0.5

Clustering Model

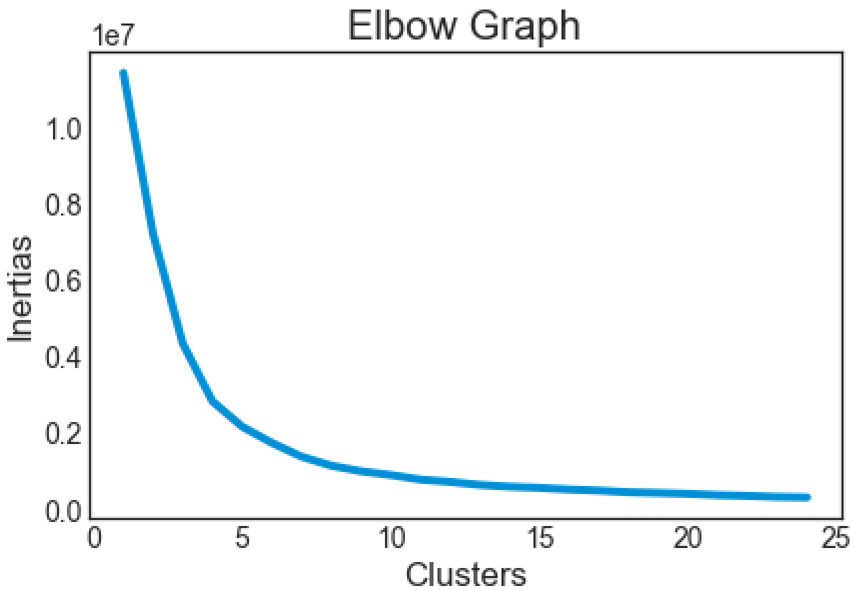
In [296]:
```
ilist = [] #list of inertias #sum of distance between data point and center of cl
n=25 #number of clusters
for i in range (1,n):
    KMeanModel = KMeans(n_clusters=i , init='k-means++' , random_state=33 , algor
    KMeanModel.fit(df_robust)#Fitting Model
    ilist.append(KMeanModel.inertia_)

ilist
```

Out[296]:
```
[11429215.571860189,
 7211367.732265211,
 4350808.28678941,
 2837358.064677662,
 2174276.895143953,
 1748572.0318372592,
 1387628.102377955,
 1152193.8861806358,
 1004561.1560823298,
 906868.0514840924,
 788015.5258390473,
 725511.5157764495,
 646151.2028919919,
 599251.6179133644,
 571054.0553285419,
 528035.342347889,
 498806.78537939937,
 455298.4605952475,
 436829.02094646613,
 413957.3789655431,
 379320.59971496544,
 360431.5459413131,
 332858.50885922246,
 320955.69714040525]
```

when number of cluster is big ,inertia is low

Elbow Graph for Inertias & # of clusters

In [297]:
```
plt.plot(range(1,n) , ilist)
plt.title('Elbow Graph')
plt.xlabel('Clusters')
plt.ylabel('Inertias')
plt.show()
#elbo graph to show less inertias as accuracy will be increasing
```



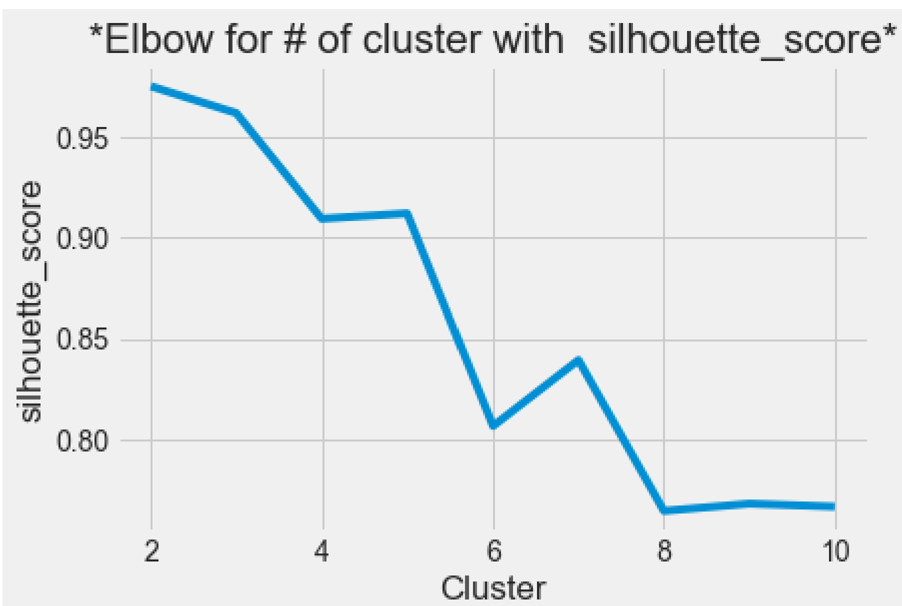
In [298]:
```
#calculate silhouette_score
score = []
for n in range(2,11):
    KMean = KMeans(n_clusters=n , init='k-means++' , random_state=33 , algorithm=
    KMean.fit(df_robust)
    result = KMean.labels_
    print(n ,"     " , silhouette_score(df_robust , result))
    score.append(silhouette_score(df_robust , result))
```

```
2      0.9752360014334556
3      0.9619417777367756
4      0.9094899476542044
5      0.912181616881109
6      0.8068755368146527
7      0.8396241892059226
8      0.7647708561974061
9      0.7683509898886909
10      0.7668877539481522
```

**Cluster 2 or 3 *

Elbow for # of cluster with silhouette_score

In [299]:
```
plt.style.use("fivethirtyeight")
plt.plot(range(2,11) , score)
plt.title('*Elbow for # of cluster with  silhouette_score*')
plt.xlabel('Cluster')
plt.ylabel('silhouette_score')
plt.show()
```



3 Cluster

In [300]:
```
KMeanModel = KMeans(n_clusters= 7, init='k-means++' , random_state=33 , algorithm
#algorithm is auto , full or elkan
#Fitting Model
KMeanModel.fit(df_robust)
y_predict=KMeanModel.predict(df_robust)
centers = KMeanModel.cluster_centers_
labels  = KMeanModel.labels_
inertial= KMeanModel.inertia_
iteration=KMeanModel.n_iter_
```

In [301]:
```
silhouette_Score = silhouette_score(df_robust , labels)
print('Silutescore Score for KMean :: ',silhouette_Score)
```

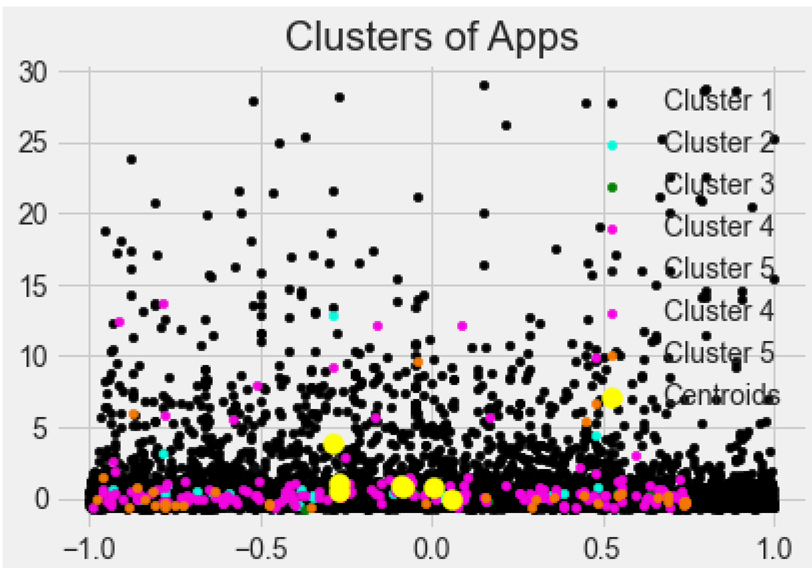
```
Silutescore Score for KMean ::  0.8396241892059226
```

In [302]:
```
print('\n Centers of 3 clusters :: \n' , centers)
print('\n Labels is :: \n',labels)
print('\n Y_Predictions :: \n' , y_predict)
print('\n Inertial is :: ',inertial)
print('\n Iteration is :: ',iteration)
```

```
 Centers of 3 clusters :: 
 [[ 3.63350681e-03  7.51607033e-01  8.92174357e-01  1.44515669e+00
   1.37572656e+00 -4.99568718e-01 -3.90921507e-01  1.70131433e-01
  -2.65526164e-01  7.74942496e-01  3.46032202e-01 -6.52458309e-01
   6.09586790e-01 -6.90051754e-03]
 [-2.67493443e-01  1.13559362e+00  1.96198383e-01  2.40553141e+02
   1.85442602e+01  3.04347826e-01  1.63043478e-01  8.74865992e-01
  -3.04347826e-01  1.54347826e+00  1.13043478e+00 -3.91304348e-01
   2.27329193e+00 -2.60208521e-18]
 [ 6.01195441e-02 -4.64170951e-02  6.21859296e-02  8.16063743e+01
   6.08585432e+02  3.12500000e-01  1.56250000e-01  4.26369863e-02
  -3.75000000e-01  1.43750000e+00  7.50000000e-01 -7.50000000e-01
   6.25000000e-01 -1.73472348e-18]
 [-2.88851821e-01  3.90330772e+00  4.97487437e-01  1.17968174e+02
   1.27357554e+03  1.00000000e+00  5.00000000e-01 -5.61643836e-02
  -2.00000000e+00  0.00000000e+00  6.00000000e+00  0.00000000e+00
   1.71428571e+00  0.00000000e+00]
 [-2.71545733e-01  5.51140415e-01  0.00000000e+00  8.12873870e+02
   1.04856115e+01  2.50000000e-01  0.00000000e+00  8.61986301e-01
  -1.11022302e-16  2.25000000e+00  1.00000000e+00 -1.12500000e+00
   2.60714286e+00  0.00000000e+00]
 [-8.71363391e-02  8.88801388e-01  9.41472066e-02  7.02542453e+01
   8.83977994e+00  2.50000000e-01  1.08823529e-01  6.95777599e-01
  -4.23529412e-01  1.19411765e+00  5.05882353e-01 -4.79411765e-01
   1.37394958e+00 -1.17647059e-02]
 [-8.49279161e-02  7.29381205e-01  4.43790380e-01  3.50635805e+01
   1.41796095e+02  4.71428571e-01  2.64285714e-01  1.79804305e-01
  -4.00000000e-01  1.12857143e+00  1.97142857e+00 -3.71428571e-01
   4.69387755e-01 -1.73472348e-18]]

 Labels is :: 
 [0 5 5 ... 0 0 0]

 Y_Predictions :: 
 [0 5 5 ... 0 0 0]

 Inertial is ::  1387628.102377955

 Iteration is ::  8
```

Visualising the Clusters

In [303]:
```
#convert data fram to np.array to avoid error
df_robust = np.array(df_robust) #that all
# Visualising the clusters
plt.scatter(df_robust[y_predict == 0, 0], df_robust[y_predict == 0, 1], s = 20, c
plt.scatter(df_robust[y_predict == 1, 0], df_robust[y_predict == 1, 1], s = 20, c
plt.scatter(df_robust[y_predict == 2, 0], df_robust[y_predict == 2, 1], s = 20, c
plt.scatter(df_robust[y_predict == 3, 0], df_robust[y_predict == 3, 1], s = 20, c
plt.scatter(df_robust[y_predict == 4, 0], df_robust[y_predict == 4, 1], s = 20, c

plt.scatter(df_robust[y_predict == 5, 0], df_robust[y_predict == 5, 1], s = 20, c
plt.scatter(df_robust[y_predict == 6, 0], df_robust[y_predict == 6, 1], s = 20, c


plt.scatter(KMeanModel.cluster_centers_[:, 0], KMeanModel.cluster_centers_[:, 1],
plt.title('Clusters of Apps')
plt.legend()

plt.show()
```



Finally!...

after clustering we have 3 clusters from Apps according all features

future work use this cluster to apply classification model

In [304]:
```
df_robust
```

Out[304]:
```
array([[ 2.19627467e-02,  2.69269861e-02,  2.00502513e+00, ...,
         0.00000000e+00,  1.28571429e+00,  0.00000000e+00],
       [-5.37392271e-01,  4.54997800e-01,  0.00000000e+00, ...,
         0.00000000e+00,  3.14285714e+00,  0.00000000e+00],
       [ 6.31915485e-01,  2.49700390e-02,  0.00000000e+00, ...,
         0.00000000e+00,  2.85714286e-01,  0.00000000e+00],
       ...,
       [-8.12621629e-01,  1.04954565e-01,  1.00000000e+00, ...,
        -2.00000000e+00,  0.00000000e+00,  0.00000000e+00],
       [ 5.82429803e-01,  6.14390388e-04,  0.00000000e+00, ...,
        -2.50000000e+00,  1.42857143e-01,  0.00000000e+00],
       [-5.40450375e-01, -4.63295863e-02,  0.00000000e+00, ...,
        -2.50000000e+00,  1.42857143e-01,  0.00000000e+00]])
```

In [305]:
```
df_robust = pd.DataFrame(df_robust ,columns=['track_name', 'size_bytes', 'price',
       'rating_count_ver', 'user_rating', 'user_rating_ver', 'ver',
       'cont_rating', 'prime_genre', 'sup_devices.num', 'ipadSc_urls.num',
       'lang.num', 'vpp_lic'])
df_robust.head()
```

Out[305]:

	track_name	size_bytes	price	rating_count_tot	rating_count_ver	user_rating	user_rating_ve
0	0.021963	0.026927	2.005025	7.592043	0.021583	0.0	0.2
1	-0.537392	0.454998	0.000000	58.142857	0.021583	0.0	-0.2
2	0.631915	0.024970	0.000000	68.095118	20.136691	-0.5	0.2
3	0.704754	0.232285	0.000000	94.734539	4.503597	0.0	0.2
4	-0.853489	-0.032434	0.000000	356.462929	38.107914	0.5	0.5

In [306]:
```
pre = pd.DataFrame(y_predict ,columns=['cluster'])
pre.head()
```

Out[306]:

	cluster
0	0
1	5
2	5
3	5
4	1

In [307]:
```
# Now we can use supervised learning
#df_robust['Cluster'] =pre['cluster']
#df_robust.head()
```

In [ ]: