Google Play Store Data

Complete Exploratory Data Analysis

By Komal Baloch

Email: komalbalochhasni@gmail.com

About Dataset

`Description`: This Data Set was downloaded from Kaggle, from the following link

`Content`:Each app (row) has values for catergory, rating, size, and more.

- Import Libraries

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
#this is for jupyter notebook to show the plot in the notebook itself instead of opening a new window for the
%matplotlib inline
```

- Read Csv File

In [2]:
```
df = pd.read_csv('gps.csv')
```

- Viewing the first five Rows of the data

In [3]:
```
df.head(5)
```

Out [3]:

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres	Last Updated	Current Ver
0	Photo Editor & Candy Camera & Grid & ScrapBook	ART_AND_DESIGN	4.1	159	19M	10,000+	Free	0	Everyone	Art & Design	January 7, 2018	1.0.0
1	Coloring book moana	ART_AND_DESIGN	3.9	967	14M	500,000+	Free	0	Everyone	Art & Design;Pretend Play	January 15, 2018	2.0.0
2	U Launcher Lite – FREE Live Cool Themes, Hide ...	ART_AND_DESIGN	4.7	87510	8.7M	5,000,000+	Free	0	Everyone	Art & Design	August 1, 2018	1.2.4
3	Sketch - Draw & Paint	ART_AND_DESIGN	4.5	215644	25M	50,000,000+	Free	0	Teen	Art & Design	June 8, 2018	Varies with device
4	Pixel Draw - Number Art Coloring Book	ART_AND_DESIGN	4.3	967	2.8M	100,000+	Free	0	Everyone	Art & Design;Creativity	June 20, 2018	1.1

- Some the output of notebook does not present the complete output, therefore we can increase the limit of columns view and row view by using these commands:

In [4]:
```
pd.set_option('display.max_columns', None) # this is to display all the columns in the dataframe
pd.set_option('display.max_rows', None) # this is to display all the rows in the dataframe
```

In [5]:
```
# hide all warnings runtime
import warnings
warnings.filterwarnings('ignore')
```

- let's see the exact column names which can be easily copied later on from Google Playstore Dataset

In [6]:
```
df.columns
```

Out [6]:
```
Index(['App', 'Category', 'Rating', 'Reviews', 'Size', 'Installs', 'Type',
       'Price', 'Content Rating', 'Genres', 'Last Updated', 'Current Ver',
       'Android Ver'],
      dtype='object')
```

- Let's have a look on the shape of the dataset

In [7]:
```
df.shape
```

Out [7]:
```
(10841, 13)
```

- Not enough, let's have a look on the columns and their data types using detailed info function

In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10841 entries, 0 to 10840
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   App             10841 non-null  object 
 1   Category        10841 non-null  object 
 2   Rating          9367 non-null   float64
 3   Reviews         10841 non-null  object 
 4   Size            10841 non-null  object 
 5   Installs        10841 non-null  object 
 6   Type            10840 non-null  object 
 7   Price           10841 non-null  object 
 8   Content Rating  10840 non-null  object 
 9   Genres          10841 non-null  object 
 10  Last Updated    10841 non-null  object 
 11  Current Ver     10833 non-null  object 
 12  Android Ver     10838 non-null  object 
dtypes: float64(1), object(12)
memory usage: 1.1+ MB
```

Observations

1. There are 10841 rows and 13 columns in the dataset
2. The columns are of different data types
3. The columns in the datasets are: 'App', 'Category', 'Rating', 'Reviews', 'Size', 'Installs', 'Type', 'Price', 'Content Rating', 'Genres', 'Last Updated', 'Current Ver', 'Android Ver'
4. There are some missing values in the dataset which we will read in details and deal later on in the notebook.
5. There are some columns which are of object data type but they should be of numeric data type, we will convert them later on in the notebook. 'Size', 'Installs', 'Price'

In [9]:
```
df.describe()
```

Out [9]:

	Rating
count	9367.000000
mean	4.193338
std	0.537431
min	1.000000
25%	4.000000
50%	4.300000
75%	4.500000
max	19.000000

- Observations: We have only 1 column as numeric data type, rest all are object data type (according to python), but we can see that 'Size', 'Installs', 'Price','Reviews' also numeric, we must convert them to numeric data type in data wrangling process.

- Let's clean the Size column first

In [10]:
```
# check for null values
df['Size'].isnull().sum()
```

Out [10]:
```
0
```

- no null values, we are good to go.

In [11]:
```
# check unique values
df['Size'].unique()
```

Out [11]:
```
array(['19M', '14M', '8.7M', '25M', '2.8M', '5.6M', '29M', '33M', '3.1M',
       '28M', '12M', '20M', '21M', '37M', '2.7M', '5.5M', '17M', '39M',
       '31M', '4.2M', '7.0M', '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', '2.5M', '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', '3.7M', '22M', '7.4M', '6.4M', '3.2M', '8.2M', '9.9M',
       '4.9M', '9.5M', '5.0M', '5.9M', '13M', '73M', '6.8M', '3.5M',
       '4.0M', '2.3M', '7.2M', '2.1M', '42M', '7.3M', '9.1M', '55M',
       '23k', '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', '118k', '44M', '695k', '1.6M',
       '6.2M', '18k', '53M', '1.4M', '3.0M', '5.8M', '3.8M', '9.6M',
       '45M', '63M', '49M', '77M', '4.4M', '4.8M', '70M', '6.9M', '9.3M',
       '10.0M', '8.1M', '36M', '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', '34M', '93M', '65M', '79M',
       '100M', '58M', '50M', '68M', '64M', '67M', '60M', '94M', '232k',
       '99M', '624k', '95M', '8.5k', '41k', '292k', '11k', '80M', '1.7M',
       '74M', '62M', '69M', '75M', '98M', '85M', '82M', '96M', '87M',
       '71M', '86M', '91M', '81M', '92M', '83M', '88M', '704k', '862k',
       '899k', '378k', '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', '902k', '818k',
       '81k', '939k', '169k', '45k', '475k', '965k', '90M', '545k', '61k',
       '283k', '655k', '714k', '93k', '872k', '121k', '322k', '1.0M',
       '976k', '172k', '238k', '549k', '206k', '954k', '444k', '717k',
       '210k', '609k', '308k', '705k', '306k', '904k', '473k', '175k',
       '350k', '383k', '454k', '421k', '70k', '812k', '442k', '842k',
       '417k', '412k', '459k', '478k', '335k', '782k', '721k', '430k',
       '429k', '192k', '200k', '460k', '728k', '496k', '816k', '414k',
       '506k', '887k', '613k', '243k', '569k', '778k', '683k', '592k',
       '319k', '186k', '840k', '647k', '191k', '373k', '437k', '598k',
       '716k', '585k', '982k', '222k', '219k', '55k', '948k', '323k',
       '691k', '511k', '951k', '963k', '25k', '554k', '351k', '27k',
       '82k', '208k', '913k', '514k', '551k', '29k', '103k', '898k',
       '743k', '116k', '153k', '209k', '353k', '499k', '173k', '597k',
       '809k', '122k', '411k', '400k', '801k', '787k', '237k', '50k',
       '643k', '986k', '97k', '516k', '837k', '780k', '961k', '269k',
       '20k', '498k', '600k', '749k', '642k', '881k', '72k', '656k',
       '601k', '221k', '228k', '108k', '940k', '176k', '33k', '663k',
       '34k', '942k', '259k', '164k', '458k', '245k', '629k', '28k',
       '288k', '775k', '785k', '636k', '916k', '994k', '309k', '485k',
       '914k', '903k', '608k', '500k', '54k', '562k', '847k', '957k',
       '688k', '811k', '270k', '48k', '329k', '523k', '921k', '874k',
       '981k', '784k', '280k', '24k', '518k', '754k', '892k', '154k',
       '860k', '364k', '387k', '626k', '161k', '879k', '39k', '970k',
       '170k', '141k', '160k', '144k', '143k', '190k', '376k', '193k',
       '246k', '73k', '658k', '992k', '253k', '420k', '404k', '1,000+',
       '470k', '226k', '240k', '89k', '234k', '257k', '861k', '467k',
       '157k', '44k', '676k', '67k', '552k', '885k', '1020k', '582k',
       '619k'], dtype=object)
```

- There are several uniques values in the Size column, we have to first make the unit into one common unit from M and K to bytes, and then remove the M and K from the values and convert them into numeric data type.

In [12]:
```python
# find the values in size column which has 'M' in it
df['Size'].loc[df['Size'].str.contains('M')].value_counts().sum()
```

Out [12]:
```
8829
```

In [13]:
```python
# find the values in size column which has 'k' in it
df['Size'].loc[df['Size'].str.contains('k')].value_counts().sum()
```

Out [13]:
```
316
```

In [14]:
```python
# find the values in size column which has 'Varies with device' in it
df['Size'].loc[df['Size'].str.contains('Varies with device')].value_counts().sum()
```

Out [14]:
```
1695
```

In [15]:
```python
# Total Values in Size column
df['Size'].value_counts().sum()
```

Out [15]:
```
10841
```

- We have 8830 values in M units
- We have 316 values in k units
- We have 1695 value in Varies with device
- Let's convert the M and K units into bytes and then remove the M and K from the values and convert them into numeric data type.

In [16]:
```python
# convert the size column to numeric by multiplying the values with 1024 if it has 'k' in it and 1024*1024 if
# this function will convert the size column to numeric
def convert_size(size):
    # add function details here
    '''
    This function will convert the size column to numeric by multiplying the values with 1024 if it has 'k' i
    '''

    if isinstance(size, str):
        if 'k' in size:
            return float(size.replace('k', '')) * 1024
        elif 'M' in size:
            return float(size.replace('M', '')) * 1024 * 1024
        elif 'Varies with device' in size:
            return np.nan
    return size

df['Size'] = df['Size'].apply(convert_size)
```

In [17]:
```
# rename the column name 'Size' to 'Size_in_bytes'
df.rename(columns={'Size': 'Size_in_bytes'}, inplace=True)
```

In [18]:
```
df['Size_in_bytes'].unique()
```

Out [18]:
```
array([19922944.0, 14680064.0, 9122611.2, 26214400.0, 2936012.8,
       5872025.6, 30408704.0, 34603008.0, 3250585.6, 29360128.0,
       12582912.0, 20971520.0, 22020096.0, 38797312.0, 2831155.2,
       5767168.0, 17825792.0, 40894464.0, 32505856.0, 4404019.2,
       7340032.0, 24117248.0, 6291456.0, 6396313.6, 4823449.6, 9646899.2,
       5452595.2, 11534336.0, 25165824.0, nan, 9856614.4, 15728640.0,
       10485760.0, 1258291.2, 27262976.0, 8388608.0, 8283750.4,
       58720256.0, 59768832.0, 36700160.0, 56623104.0, 205824.0,
       3774873.6, 5976883.2, 9017753.6, 2516582.4, 28311552.0, 2621440.0,
       16777216.0, 3565158.4, 9332326.4, 4089446.4, 3040870.4, 39845888.0,
       33554432.0, 5662310.4, 18874368.0, 1153433.6, 2306867.2, 4718592.0,
       10276044.8, 54525952.0, 9437184.0, 7025459.2, 31457280.0,
       2726297.6, 7444889.6, 3879731.2, 23068672.0, 7759462.4, 6710886.4,
       3355443.2, 8598323.2, 10380902.4, 5138022.4, 9961472.0, 5242880.0,
       6186598.4, 13631488.0, 76546048.0, 7130316.8, 3670016.0, 4194304.0,
       2411724.8, 7549747.2, 2202009.6, 44040192.0, 7654604.8, 9542041.6,
       57671680.0, 23552.0, 6815744.0, 1572864.0, 7864320.0, 53477376.0,
       42991616.0, 50331648.0, 8912896.0, 48234496.0, 8703180.8,
       4508876.8, 4928307.2, 3460300.8, 41943040.0, 8178892.8, 9227468.8,
       6920601.6, 5347737.6, 63963136.0, 69206016.0, 80896.0, 8808038.4,
       120832.0, 46137344.0, 711680.0, 1677721.6, 6501171.2, 18432.0,
       55574528.0, 1468006.4, 3145728.0, 6081740.8, 3984588.8, 10066329.6,
       47185920.0, 66060288.0, 51380224.0, 80740352.0, 4613734.4,
       5033164.8, 73400320.0, 7235174.4, 9751756.8, 8493465.6, 37748736.0,
       88080384.0, 101711872.0, 2097152.0, 1992294.4, 1887436.8,
       5557452.8, 49283072.0, 569344.0, 538624.0, 79691776.0, 7969177.6,
       61865984.0, 10171187.2, 81788928.0, 75497472.0, 45088768.0,
       8074035.2, 6606028.8, 342016.0, 35651584.0, 97517568.0, 68157440.0,
       82837504.0, 104857600.0, 60817408.0, 52428800.0, 71303168.0,
       67108864.0, 70254592.0, 62914560.0, 98566144.0, 237568.0,
       103809024.0, 638976.0, 99614720.0, 8704.0, 41984.0, 299008.0,
       11264.0, 83886080.0, 1782579.2, 77594624.0, 65011712.0, 72351744.0,
       78643200.0, 102760448.0, 89128960.0, 85983232.0, 100663296.0,
       91226112.0, 74448896.0, 90177536.0, 95420416.0, 84934656.0,
       96468992.0, 87031808.0, 92274688.0, 720896.0, 882688.0, 920576.0,
       387072.0, 272384.0, 384000.0, 1363148.8, 998400.0, 1003520.0,
       4299161.6, 93323264.0, 712704.0, 557056.0, 537600.0, 942080.0,
       797696.0, 873472.0, 737280.0, 730112.0, 790528.0, 325632.0,
       59392.0, 246784.0, 200704.0, 877568.0, 52224.0, 975872.0, 885760.0,
       257024.0, 952320.0, 552960.0, 320512.0, 763904.0, 207872.0,
       26624.0, 321536.0, 244736.0, 379904.0, 225280.0, 747520.0,
       774144.0, 93184.0, 300032.0, 17408.0, 75776.0, 14336.0, 324608.0,
       79872.0, 946176.0, 923648.0, 837632.0, 82944.0, 961536.0, 173056.0,
       46080.0, 486400.0, 988160.0, 94371840.0, 558080.0, 62464.0,
       289792.0, 670720.0, 731136.0, 95232.0, 892928.0, 123904.0,
       329728.0, 1048576.0, 999424.0, 176128.0, 243712.0, 562176.0,
       210944.0, 976896.0, 454656.0, 734208.0, 215040.0, 623616.0,
       315392.0, 721920.0, 313344.0, 925696.0, 484352.0, 179200.0,
       358400.0, 392192.0, 464896.0, 431104.0, 71680.0, 831488.0,
       452608.0, 862208.0, 427008.0, 421888.0, 470016.0, 489472.0,
       343040.0, 800768.0, 738304.0, 440320.0, 439296.0, 196608.0,
       204800.0, 471040.0, 745472.0, 507904.0, 835584.0, 423936.0,
       518144.0, 908288.0, 627712.0, 248832.0, 582656.0, 796672.0,
       699392.0, 606208.0, 326656.0, 190464.0, 860160.0, 662528.0,
       195584.0, 381952.0, 447488.0, 612352.0, 733184.0, 599040.0,
       1005568.0, 227328.0, 224256.0, 56320.0, 970752.0, 330752.0,
       707584.0, 523264.0, 973824.0, 986112.0, 25600.0, 567296.0,
       359424.0, 27648.0, 83968.0, 212992.0, 934912.0, 526336.0, 564224.0,
       29696.0, 105472.0, 919552.0, 760832.0, 118784.0, 156672.0,
       214016.0, 361472.0, 510976.0, 177152.0, 611328.0, 828416.0,
       124928.0, 420864.0, 409600.0, 820224.0, 805888.0, 242688.0,
       51200.0, 658432.0, 1009664.0, 99328.0, 528384.0, 857088.0,
       798720.0, 984064.0, 275456.0, 20480.0, 509952.0, 614400.0,
       766976.0, 657408.0, 902144.0, 73728.0, 671744.0, 615424.0,
       226304.0, 233472.0, 110592.0, 962560.0, 180224.0, 33792.0,
       678912.0, 34816.0, 964608.0, 265216.0, 167936.0, 468992.0,
       250880.0, 644096.0, 28672.0, 294912.0, 793600.0, 803840.0,
       651264.0, 937984.0, 1017856.0, 316416.0, 496640.0, 935936.0,
       924672.0, 622592.0, 512000.0, 55296.0, 575488.0, 867328.0,
       979968.0, 704512.0, 830464.0, 276480.0, 49152.0, 336896.0,
       535552.0, 943104.0, 894976.0, 1004544.0, 802816.0, 286720.0,
       24576.0, 530432.0, 772096.0, 913408.0, 157696.0, 880640.0,
       372736.0, 396288.0, 641024.0, 164864.0, 900096.0, 39936.0,
       993280.0, 174080.0, 144384.0, 163840.0, 147456.0, 146432.0,
       194560.0, 385024.0, 197632.0, 251904.0, 74752.0, 673792.0,
       1015808.0, 259072.0, 430080.0, 413696.0, '1,000+', 481280.0,
       231424.0, 245760.0, 91136.0, 239616.0, 263168.0, 881664.0,
       478208.0, 160768.0, 45056.0, 692224.0, 68608.0, 565248.0, 906240.0,
       1044480.0, 595968.0, 633856.0], dtype=object)
```

In [19]:
```
def string_to_int(value):
    try:
        return int(value)
    except ValueError:
        # Handle non-integer values here (e.g., return a default value or NaN)
        return 0
```

```
# Apply the custom function to the column
df['Size_in_bytes'] = df['Size_in_bytes'].apply(string_to_int)
```

In [20]:
```
df['Size_in_bytes'].dtypes
```

Out [20]:
```
dtype('int64')
```

- Now we have converted every value into bytes and removed the M and K from the values and converted them into numeric data type. 'Varies with device' was a string value, therefore we intentionally converted them into null values, which we can fill later on according to our needs.
- Let's have a look on the Installs column

In [21]:
```
# check the unique values in size column
df['Installs'].unique()
```

Out [21]:
```
array(['10,000+', '500,000+', '5,000,000+', '50,000,000+', '100,000+',
       '50,000+', '1,000,000+', '10,000,000+', '5,000+', '100,000,000+',
       '1,000,000,000+', '1,000+', '500,000,000+', '50+', '100+', '500+',
       '10+', '1+', '5+', '0+', '0', 'Free'], dtype=object)
```

In [22]:
```
# let's have a values counts
df['Installs'].value_counts()
```

Out [22]:
```
Installs
1,000,000+        1579
10,000,000+       1252
100,000+          1169
10,000+           1054
1,000+             907
5,000,000+         752
100+               719
500,000+           539
50,000+            479
5,000+             477
100,000,000+       409
10+                386
500+               330
50,000,000+        289
50+                205
5+                  82
500,000,000+        72
1+                  67
1,000,000,000+      58
0+                  14
0                    1
Free                 1
Name: count, dtype: int64
```

In [23]:
```
# find how many values has '+' in it
df['Installs'].loc[df['Installs'].str.contains('\+')].value_counts().sum()
```

Out [23]:
```
10839
```

In [24]:
```
# Total values in Installs column
df['Installs'].value_counts().sum()
```

Out [24]:
```
10841
```

- The only problem I see here is the + sign in the values, let's remove them and convert the column into numeric data type.
- The total values in the Installs column are 10841 and there are no null values in the column.
- However, one value 0 has no plus sign
- Let's remove the plus sign + and , from the values and convert them into numeric data type

In [25]:
```
# remove the plus sign from install column and convert it to numeric
df['Installs'] = df['Installs'].apply(lambda x: x.replace('+', '') if '+' in str(x) else x)
# also remove the commas from the install column
df['Installs'] = df['Installs'].apply(lambda x: x.replace(',', '') if ',' in str(x) else x)
```

In [26]:
```

def string_to_int(value):
    try:
        return int(value)
    except ValueError:
        # Handle non-integer values here (e.g., return a default value or NaN)
        return 0

# Apply the custom function to the column
df['Installs'] = df['Installs'].apply(string_to_int)
```

In [27]:
```
df['Installs'].unique()
```

Out [27]:
```
array([     10000,      500000,     5000000,    50000000,      100000,
            50000,     1000000,    10000000,        5000,   100000000,
         1000000000,        1000,  500000000,          50,         100,
                500,          10,          1,           5,           0],
        dtype=int64)
```

In [28]:
```
df.head() # check the head of the dataframe
```

Out [28]:

	App	Category	Rating	Reviews	Size_in_bytes	Installs	Type	Price	Content Rating	Genres	Last Updated	C
0	Photo Editor & Candy Camera & Grid & ScrapBook	ART_AND_DESIGN	4.1	159	19922944	10000	Free	0	Everyone	Art & Design	January 7, 2018	1.
1	Coloring book moana	ART_AND_DESIGN	3.9	967	14680064	500000	Free	0	Everyone	Art & Design;Pretend Play	January 15, 2018	2.
2	U Launcher Lite – FREE Live Cool Themes, Hide ...	ART_AND_DESIGN	4.7	87510	9122611	5000000	Free	0	Everyone	Art & Design	August 1, 2018	1.
3	Sketch - Draw & Paint	ART_AND_DESIGN	4.5	215644	26214400	50000000	Free	0	Teen	Art & Design	June 8, 2018	V. w d
4	Pixel Draw - Number Art Coloring Book	ART_AND_DESIGN	4.3	967	2936012	100000	Free	0	Everyone	Art & Design;Creativity	June 20, 2018	1.

- Let's have a look on the Price column

In [29]:
```
# check the unique values in the 'Price' column
df['Price'].unique()
```

Out [29]:
```
array(['0', '$4.99', '$3.99', '$6.99', '$1.49', '$2.99', '$7.99', '$5.99',
       '$3.49', '$1.99', '$9.99', '$7.49', '$0.99', '$9.00', '$5.49',
       '$10.00', '$24.99', '$11.99', '$79.99', '$16.99', '$14.99',
       '$1.00', '$29.99', '$12.99', '$2.49', '$10.99', '$1.50', '$19.99',
       '$15.99', '$33.99', '$74.99', '$39.99', '$3.95', '$4.49', '$1.70',
       '$8.99', '$2.00', '$3.88', '$25.99', '$399.99', '$17.99',
       '$400.00', '$3.02', '$1.76', '$4.84', '$4.77', '$1.61', '$2.50',
       '$1.59', '$6.49', '$1.29', '$5.00', '$13.99', '$299.99', '$379.99',
       '$37.99', '$18.99', '$389.99', '$19.90', '$8.49', '$1.75',
       '$14.00', '$4.85', '$46.99', '$109.99', '$154.99', '$3.08',
       '$2.59', '$4.80', '$1.96', '$19.40', '$3.90', '$4.59', '$15.46',
       '$3.04', '$4.29', '$2.60', '$3.28', '$4.60', '$28.99', '$2.95',
       '$2.90', '$1.97', '$200.00', '$89.99', '$2.56', '$30.99', '$3.61',
       '$394.99', '$1.26', 'Everyone', '$1.20', '$1.04'], dtype=object)
```

In [30]:
```
df['Price'].isnull().sum()
```

Out [30]:
```
0
```

- No null values

In [31]:
```
df['Price'].value_counts() # check the value counts of the 'Price' column
```

Out [31]:
```
Price
0            10040
$0.99          148
$2.99          129
$1.99           73
$4.99           72
$3.99           63
$1.49           46
$5.99           30
$2.49           26
$9.99           21
$6.99           13
$399.99         12
$14.99          11
$4.49            9
$29.99           7
$24.99           7
$3.49            7
$7.99            7
$5.49            6
$19.99           6
$11.99           5
$6.49            5
$12.99           5
$8.99            5
$10.00           3
$16.99           3
$1.00            3
$2.00            3
$13.99           2
$8.49            2
$17.99           2
```

```
$1.70          2
$3.95          2
$79.99         2
$7.49          2
$9.00          2
$10.99         2
$39.99         2
$33.99         2
$19.40         1
$3.90          1
$1.96          1
$4.60          1
$15.46         1
$3.04          1
$4.29          1
$2.60          1
$3.28          1
$4.80          1
$4.59          1
$3.08          1
$28.99         1
$2.95          1
$2.90          1
$1.97          1
$200.00        1
$89.99         1
$2.56          1
$30.99         1
$1.20          1
Everyone       1
$3.61          1
$394.99        1
$1.26          1
$2.59          1
$1.61          1
$154.99        1
$1.59          1
$1.50          1
$15.99         1
$74.99         1
$3.88          1
$25.99         1
$400.00        1
$3.02          1
$1.76          1
$4.84          1
$4.77          1
$2.50          1
$1.29          1
$109.99        1
$5.00          1
$299.99        1
$379.99        1
$37.99         1
$18.99         1
$389.99        1
$19.90         1
$1.75          1
$14.00         1
$4.85          1
$46.99         1
$1.04          1
Name: count, dtype: int64
```

- We need to confirm if the values in the Price column are only with $ sign or not

In [32]:
```python
# count the values having $ in the 'Price' column
df['Price'].loc[df['Price'].str.contains('\$')].value_counts().sum()
```

Out [32]:
```
800
```

In [33]:
```python
# This code counts the number of values in the 'Price' column which contains 0 but does not contain $ sign
df['Price'].loc[(df['Price'].str.contains('0')) & (~df['Price'].str.contains('\$'))].value_counts().sum()
```

Out [33]:
```
10040
```

- Now we can confirm that the only currency used is $ in the Price column or 0 value, as 800+10040=10840 Total values.
- The only problem is $ sign let's remove it and convert the column into numeric data type.

In [34]:
```python

df['Price'] = df['Price'].apply(lambda x: x.replace('', '') if '$' in str(x) else x)
# convert the price column to numeric (float because this is the price)
def string_to_float(value):
    try:
        return float(value)
    except ValueError:
        # Handle non-float values here (e.g., return a default value or NaN)
        return 0

# Apply the custom function to the column
df['Price'] = df['Price'].apply(string_to_float)
```

In [35]:
```python
df['Price'].dtype # this will show the data type of the column
```

Out [35]:
```
dtype('float64')
```

In [36]:
```
# using f string to print the min, max and average price of the apps
print(f"Min price is: {df['Price'].min()} $")
print(f"Max price is: {df['Price'].max()} $")
print(f"Average price is: {df['Price'].mean()} $")
```

```
Min price is: 0.0 $
Max price is: 0.0 $
Average price is: 0.0 $
```

In [37]:
```
df.describe()
```

Out [37]:

	Rating	Size_in_bytes	Installs	Price
count	9367.000000	1.084100e+04	1.084100e+04	10841.0
mean	4.193338	1.903177e+07	1.546291e+07	0.0
std	0.537431	2.324747e+07	8.502557e+07	0.0
min	1.000000	0.000000e+00	0.000000e+00	0.0
25%	4.000000	2.726297e+06	1.000000e+03	0.0
50%	4.300000	9.646899e+06	1.000000e+05	0.0
75%	4.500000	2.726298e+07	5.000000e+06	0.0
max	19.000000	1.048576e+08	1.000000e+09	0.0

- Descriptive Statistics

In [38]:
```
df.describe()
```

Out [38]:

	Rating	Size_in_bytes	Installs	Price
count	9367.000000	1.084100e+04	1.084100e+04	10841.0
mean	4.193338	1.903177e+07	1.546291e+07	0.0
std	0.537431	2.324747e+07	8.502557e+07	0.0
min	1.000000	0.000000e+00	0.000000e+00	0.0
25%	4.000000	2.726297e+06	1.000000e+03	0.0
50%	4.300000	9.646899e+06	1.000000e+05	0.0
75%	4.500000	2.726298e+07	5.000000e+06	0.0
max	19.000000	1.048576e+08	1.000000e+09	0.0

- Observations:
- Now, we have only 5 columns as numeric data type.
- We can observe their descriptive statistics. and make tons of observations as per our hypotheses.
- We can see that the Rating column has a minimum value of 1 and a maximum value of 19, which is the range of rating, and the mean is 4.19 which is a good rating. On an average people give this rating.
- Similarly, we can observe the other columns as well.
- -Therefore, the most important thing is to classify as app based on the correlation matrix and then observe the descriptive statistics of the app category and number of installs, reviews, ratings, etc.

- But even before that we have to think about the missing values in the dataset.

- Dealing with the missing values is one of the most important part of the data wrangling process, we must deal with the missing values in order to get the correct insights from the data.

In [39]:
```
df.isnull().sum() # this will show the number of null values in each column
```

Out [39]:
```
App                  0
Category             0
Rating            1474
Reviews              0
Size_in_bytes        0
Installs             0
Type                 1
Price                0
Content Rating       1
Genres               0
Last Updated         0
Current Ver          8
Android Ver          3
dtype: int64
```

In [40]:
```
df.isnull().sum().sort_values(ascending=False) # this will show the number of null values in each column in c
```

Out [40]:
```
Rating            1474
Current Ver          8
Android Ver          3
Type                 1
Content Rating       1
App                  0
Category             0
```

```
Reviews               0
Size_in_bytes         0
Installs              0
Price                 0
Genres                0
Last Updated          0
dtype: int64
```

In [41]:
```
df.isnull().sum().sum() # this will show the total number of null values in the dataframe
```

Out [41]:
```
1487
```

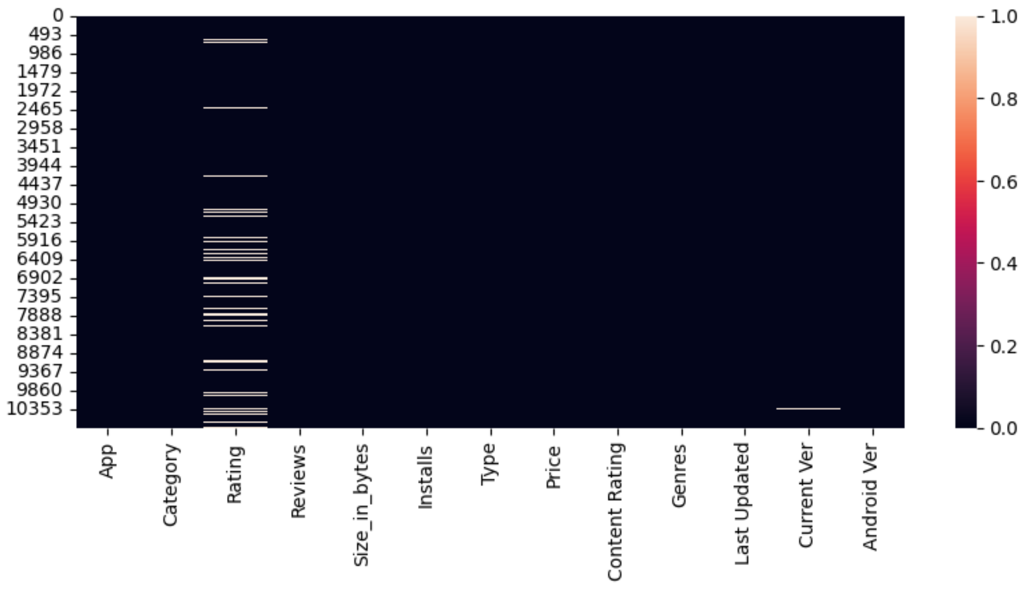
In [42]:
```
(df.isnull().sum() / len(df) * 100).sort_values(ascending=False) # this will show the percentage of null valu
```

Out [42]:
```
Rating            13.596532
Current Ver        0.073794
Android Ver        0.027673
Type               0.009224
Content Rating     0.009224
App                0.000000
Category           0.000000
Reviews            0.000000
Size_in_bytes      0.000000
Installs           0.000000
Price              0.000000
Genres             0.000000
Last Updated       0.000000
dtype: float64
```

- Let's plot the missing values in the dataset

In [43]:
```
# make a figure size
plt.figure(figsize=(10, 4))
#plot the null values in each column
sns.heatmap(df.isnull())
```

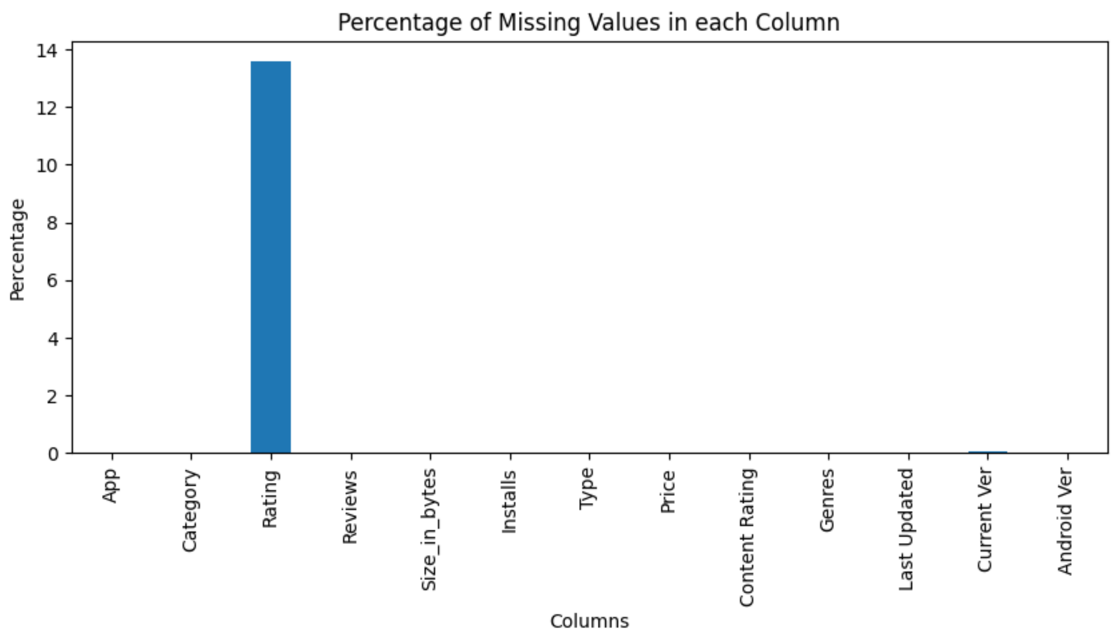
Out [43]:
```
<Axes: >
```



- There is another way, let's plot the missing values by percentage

In [44]:
```
# make figure size
plt.figure(figsize=(10, 4))
# plot the null values by their percentage in each column
missing_percentage = df.isnull().sum()/len(df)*100
missing_percentage.plot(kind='bar')
# add the labels
plt.xlabel('Columns')
plt.ylabel('Percentage')
plt.title('Percentage of Missing Values in each Column')
```

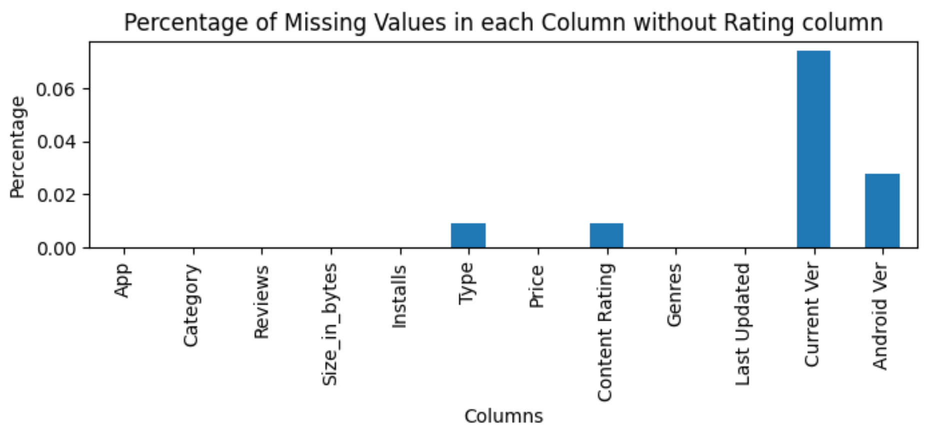
Out [44]:
```
Text(0.5, 1.0, 'Percentage of Missing Values in each Column')
```



- We have missing percentage columns other than rating having less than one percent of missing values, we will plot them as follows

In [45]:
```
plt.figure(figsize=(8, 2)) # make figure size
missing_percentage[missing_percentage < 1].plot(kind='bar') # plot the null values by their percentage in eac
plt.xlabel('Columns') # add the x-axis labels
plt.ylabel('Percentage') # add the labels for y-axis
plt.title('Percentage of Missing Values in each Column without Rating column')  # add the title for the plot
```

Out [45]:
```
Text(0.5, 1.0, 'Percentage of Missing Values in each Column without Rating column')
```



In [46]:
```
df.isnull().sum().sort_values(ascending=False) # this will show the number of null values in each column in c
```

Out [46]:
```
Rating            1474
Current Ver          8
Android Ver          3
Type                 1
Content Rating       1
App                  0
Category             0
Reviews              0
Size_in_bytes        0
Installs             0
Price                0
Genres               0
Last Updated         0
dtype: int64
```

In [47]:
```
(df.isnull().sum() / len(df) * 100).sort_values(ascending=False) # this will show the percentage of null valu
```

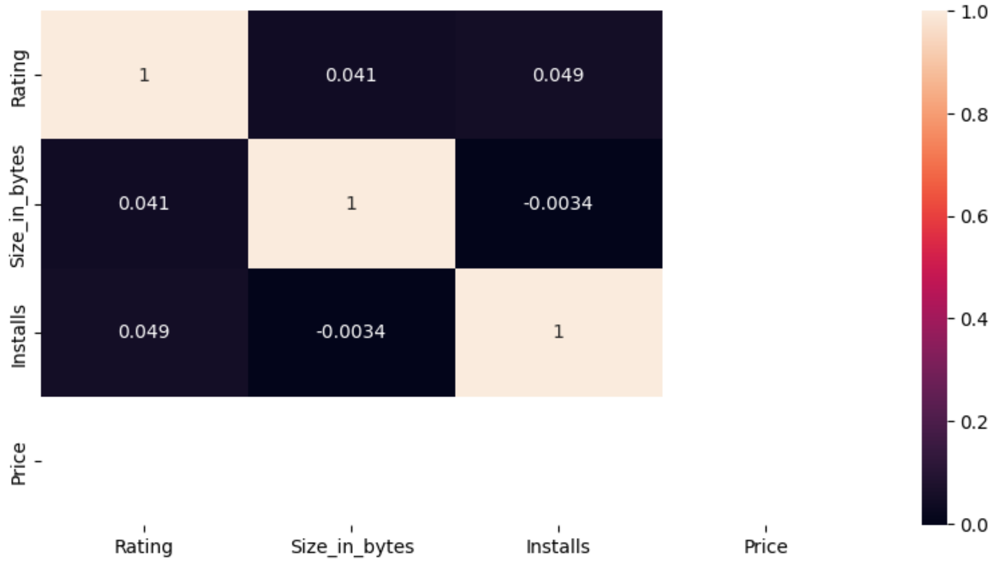
Out [47]:
```
Rating            13.596532
Current Ver        0.073794
Android Ver        0.027673
Type               0.009224
Content Rating     0.009224
App                0.000000
Category           0.000000
Reviews            0.000000
Size_in_bytes      0.000000
Installs           0.000000
Price              0.000000
Genres             0.000000
Last Updated       0.000000
dtype: float64
```

- Observaions:
- We have 1474 missing values in the 'Rating' column, which is 13.6% of the total values in the column.

- Dealing with the missing values:
- We can not impute the Rating column as is is directly linked with the installation column. To test this Hypothesis we need to plot the Rating column with the Installs and Size columns and statistically test it using pearson correlation test.

In [48]:
```
# Make a correlation matrix of numeric columns
plt.figure(figsize=(10,5)) # make figure size
numeric_cols = ['Rating', 'Size_in_bytes', 'Installs', 'Price'] # make a list of numeric columns
sns.heatmap(df[numeric_cols].corr(), annot=True) # plot the correlation matrix
```

Out [48]:
```
<Axes: >
```



In [49]:
```
# we can also calculate the correlation matrix using pandas
df[numeric_cols].corr() # this will show the correlation matrix
```

Out [49]:

	Rating	Size_in_bytes	Installs	Price
Rating	1.000000	0.041476	0.048652	NaN
Size_in_bytes	0.041476	1.000000	-0.003440	NaN
Installs	0.048652	-0.003440	1.000000	NaN
Price	NaN	NaN	NaN	NaN

- Before going ahead, let's remove the rows with missing values in the Current Ver, Android Ver, Category, Type and Genres columns, as they are very less in number and will not affect our analysis.

In [52]:
```
# length before removing null values
print(f"Length of the dataframe before removing null values: {len(df)}")
```

```
Length of the dataframe before removing null values: 10841
```

In [53]:
```
# remove the rows having null values in the 'Current Ver', 'Android Ver', 'Category', 'Type' and 'Genres' col
df.dropna(subset=['Current Ver', 'Android Ver', 'Category', 'Type', 'Genres'], inplace=True)
```

In [54]:
```
# length after removing null values
print(f"Length of the dataframe after removing null values: {len(df)}")
```

```
Length of the dataframe after removing null values: 10829
```

- We have removed 12 rows having null values in the Current Ver, Android Ver, Category, Type and Genres columns.

In [55]:
```
# let's check the null values again
df.isnull().sum().sort_values(ascending=False)
```

Out [55]:
```
Rating            1469
App                  0
Category             0
Reviews              0
Size_in_bytes        0
Installs             0
Type                 0
Price                0
Content Rating       0
Genres               0
```

```
Last Updated        0
Current Ver         0
Android Ver         0
dtype: int64
```

- Observations: Only Rating is left with missing values.
- We know that we have to be carefull while deadling with Rating column, as it is directly linked with the Installs column. In Size columns we already know about Varies with device values, which we have converted into null values, we do not need to impute at the moment, as every app has different size and nobody can predict that as nearly as possible.

In [56]:
```
df.columns
```

Out [56]:
```
Index(['App', 'Category', 'Rating', 'Reviews', 'Size_in_bytes', 'Installs',
       'Type', 'Price', 'Content Rating', 'Genres', 'Last Updated',
       'Current Ver', 'Android Ver'],
      dtype='object')
```

In [58]:
```
# use groupby function to find the trend of Rating in each Installs_category
df.groupby('Installs')['Rating'].describe()
```

Out [58]:

	count	mean	std	min	25%	50%	75%	max
Installs								
0	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	3.0	5.000000	0.000000	5.0	5.00	5.0	5.000	5.0
5	9.0	4.611111	1.166667	1.5	5.00	5.0	5.000	5.0
10	69.0	4.624638	0.821119	1.0	4.80	5.0	5.000	5.0
50	56.0	4.419643	1.046799	1.0	4.35	5.0	5.000	5.0
100	309.0	4.363430	0.847641	1.0	4.00	4.7	5.000	5.0
500	201.0	4.176617	0.801873	1.0	3.90	4.4	4.700	5.0
1000	712.0	4.066292	0.784476	1.0	3.70	4.3	4.600	5.0
5000	431.0	4.026450	0.636302	1.4	3.70	4.2	4.500	5.0
10000	1009.0	4.039247	0.590572	1.7	3.80	4.2	4.500	5.0
50000	466.0	4.051288	0.551604	1.6	3.80	4.2	4.400	4.9
100000	1150.0	4.110261	0.484969	1.6	3.90	4.2	4.500	4.9
500000	537.0	4.168156	0.446599	1.8	3.90	4.3	4.500	4.9
1000000	1576.0	4.220939	0.348702	2.2	4.00	4.3	4.500	4.9
5000000	752.0	4.243218	0.317626	2.0	4.10	4.3	4.500	4.9
10000000	1252.0	4.313419	0.277183	3.0	4.20	4.4	4.500	4.9
50000000	289.0	4.351211	0.223312	3.1	4.20	4.4	4.500	4.8
100000000	409.0	4.411491	0.163719	3.5	4.30	4.4	4.500	4.8
500000000	72.0	4.350000	0.151053	4.0	4.30	4.3	4.425	4.7
1000000000	58.0	4.258621	0.212780	3.7	4.10	4.3	4.400	4.5

In [60]:
```
df['Rating'].isnull().sum()
```

Out [60]:
```
14
```

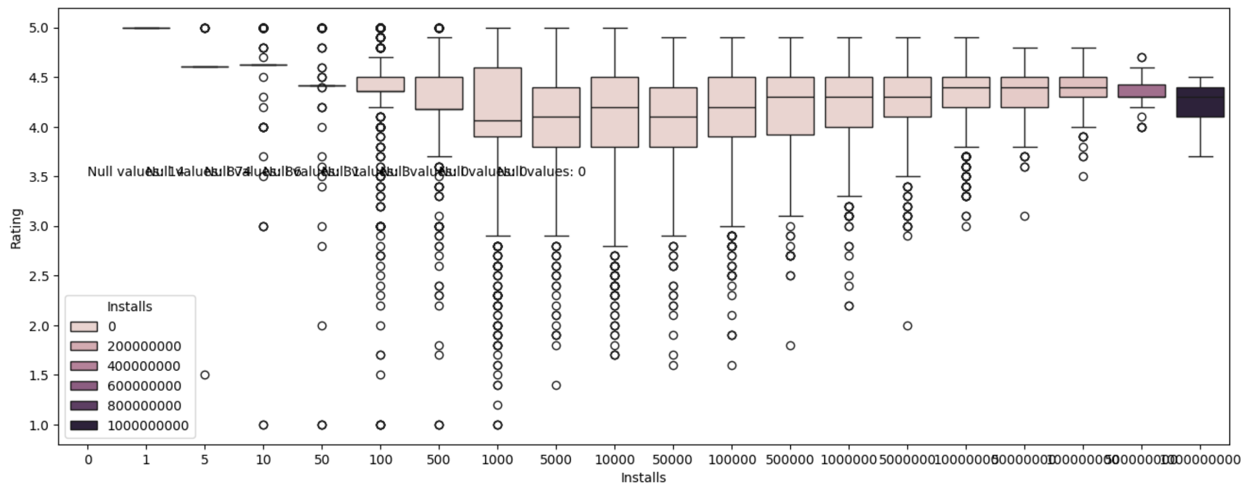
In [62]:
```
# in which Install_category the Rating has NaN values
df['Installs'].loc[df['Rating'].isnull()].value_counts()
```

Out [62]:
```
Installs
0    14
Name: count, dtype: int64
```

- Let's plot this and have a look

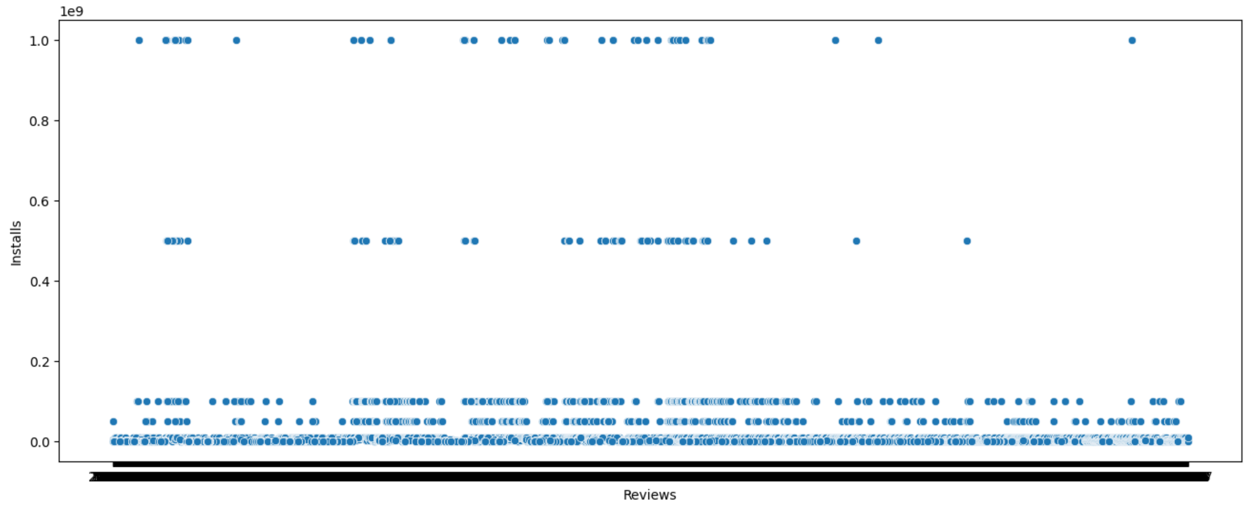
In [64]:
```
# plot the boxplot of Rating in each Installs_category
plt.figure(figsize=(16, 6)) # make figure size
sns.boxplot(x='Installs', y='Rating', hue='Installs', data=df) # plot the boxplot
# add the text of number of null values in each category
plt.text(0, 3.5, 'Null values: 14')
plt.text(1, 3.5, 'Null values: 874')
plt.text(2, 3.5, 'Null values: 86')
plt.text(3, 3.5, 'Null values: 31')
plt.text(4, 3.5, 'Null values: 3')
plt.text(5, 3.5, 'Null values: 0')
plt.text(6, 3.5, 'Null values: 0')
plt.text(7, 3.5, 'Null values: 0')
```

Out [64]:
```
Text(7, 3.5, 'Null values: 0')
```



```
In [67]: # plot reviews and installs in a scatter plot
         plt.figure(figsize=(16, 6)) # make figure size
         sns.scatterplot(x='Reviews', y='Installs', data=df) # plot the scatter plot
```

```
Out [67]: <Axes: xlabel='Reviews', ylabel='Installs'>
```



- replace the respective missing values of Rating, with respect to the average give in each 'Installs'!

```
In [83]: average_rating = df.groupby('Installs')['Rating'].median()
```

```
In [84]: df['Rating']  = df.apply(lambda row:average_rating[row['Installs']]if pd.isna(row['Rating']) else row['Rating
```

- Duplicates Removing duplicates is one of the most important part of the data wrangling process, we must remove the duplicates in order to get the correct insights from the data. If you do not remove duplicates from a dataset, it can lead to incorrect insights and analysis. Duplicates can skew statistical measures such as mean, median, and standard deviation, and can also lead to over-representation of certain data points. It is important to remove duplicates to ensure the accuracy and reliability of your data analysis.

```
In [89]: df.columns
```

```
Out [89]: Index(['App', 'Category', 'Rating', 'Reviews', 'Size_in_bytes', 'Installs',
                 'Type', 'Price', 'Content Rating', 'Genres', 'Last Updated',
                 'Current Ver', 'Android Ver'],
                dtype='object')
```

```
In [88]: # find duplicate if any
         df.duplicated().sum()
```

```
Out [88]: 483
```

This shows us total duplicates, but we can also check based on the app name, as we know that every app has a unique name.

```
In [90]: # find duplicate if any in the 'App' column
         df['App'].duplicated().sum()
```

```
Out [90]: 1181
```

- Oops! we have 1181 dupicate app names

- Can we find a column which can help us to remove the duplicates
- let's check for number of duplicates in each column usi# ng a for loop and printing the output

In [91]:
```python
for col in df.columns:
    print(f"Number of duplicates in {col} column are: {df[col].duplicated().sum()}")
```

```
Number of duplicates in App column are: 1181
Number of duplicates in Category column are: 10796
Number of duplicates in Rating column are: 10777
Number of duplicates in Reviews column are: 4830
Number of duplicates in Size_in_bytes column are: 10373
Number of duplicates in Installs column are: 10809
Number of duplicates in Type column are: 10827
Number of duplicates in Price column are: 10828
Number of duplicates in Content Rating column are: 10823
Number of duplicates in Genres column are: 10710
Number of duplicates in Last Updated column are: 9453
Number of duplicates in Current Ver column are: 7998
Number of duplicates in Android Ver column are: 10796
```

In [95]:
```python
# print the number of duplicates in df
print(f"Number of duplicates in df are: {df.duplicated().sum()}")
```

```
Number of duplicates in df are: 483
```

In [96]:
```python
# find exact duplicates and print them
# df[df['App'].duplicated(keep=False)].sort_values(by='App')
```

In [97]:
```python
# print the number of rows and columns after removing duplicates
print(f"Number of rows after removing duplicates: {df.shape[0]}")
```

```
Number of rows after removing duplicates: 10829
```

- Now we have removed 483 duplicates from the dataset. and have 10346 rows left.

- Insights from Data
- Which category has the highest number of apps?

In [98]:
```python
df['Category'].value_counts().head(10) # this will show the top 10 categories with highest number of apps
```

Out [98]:
```
Category
FAMILY             1968
GAME               1144
TOOLS               841
MEDICAL             463
BUSINESS            460
PRODUCTIVITY        424
PERSONALIZATION     390
COMMUNICATION       387
SPORTS              384
LIFESTYLE           382
Name: count, dtype: int64
```

- Which category has the highest number of installs?

In [99]:
```python
df.groupby('Category')['Installs'].sum().sort_values(ascending=False).head(10)
```

Out [99]:
```
Category
GAME                  35086024415
COMMUNICATION         32647276251
PRODUCTIVITY          14176091369
SOCIAL                14069867902
TOOLS                 11452271905
FAMILY                10258203405
PHOTOGRAPHY           10088247655
NEWS_AND_MAGAZINES     7496317760
TRAVEL_AND_LOCAL       6868887146
VIDEO_PLAYERS          6222002720
Name: Installs, dtype: int64
```

- Which category has the highest number of reviews?

In [100]:
```python
# Category with highest number of Reviews
df.groupby('Category')['Reviews'].sum().sort_values(ascending=False).head(10)
```

Out [100]:
```
Category
PARENTING             8617941763614343114133936378967107559497632484...
SOCIAL                7815830666577313860625949173295532622492170147...
MAPS_AND_NAVIGATION   7232629156815348110480050459432695644349284201...
EDUCATION             6289924181893254485375314299776977032346407510...
COMMUNICATION         5664284769119316125257964299514290354604324341...
ENTERTAINMENT         5456208116562894829677147008910939985095241225...
SPORTS                5211381802283662828824597951338259119951733342...
FAMILY                4706944214544499101477412753339832026757611161...
GAME                  4447388277222642242667725425814889736920352341...
HEALTH_AND_FITNESS    4281561577380983113927233722012540059220098117...
Name: Reviews, dtype: object
```

- Which category has the highest rating?

```
In [101]: # Category with highest average Rating
          df.groupby('Category')['Rating'].mean().sort_values(ascending=False).head(10)
```

```
Out [101]: Category
           EVENTS                 4.414972
           EDUCATION              4.387245
           ART_AND_DESIGN         4.367249
           BOOKS_AND_REFERENCE    4.360748
           PERSONALIZATION        4.352251
           HEALTH_AND_FITNESS     4.299333
           GAME                   4.290438
           SOCIAL                 4.266821
           SHOPPING               4.264005
           SPORTS                 4.261570
           Name: Rating, dtype: float64
```

- Make Questions and Inspect the data find the answers

```
In [104]: df.columns
```

```
Out [104]: Index(['App', 'Category', 'Rating', 'Reviews', 'Size_in_bytes', 'Installs',
                  'Type', 'Price', 'Content Rating', 'Genres', 'Last Updated',
                  'Current Ver', 'Android Ver'],
                 dtype='object')
```

```
In [107]: # Which rating has highest reviews
          q = df.groupby('Rating')['Reviews'].sum().sort_values(ascending=False).head(3)
          q
```

```
Out [107]: Rating
           4.3    967194216845542388464021135741383488572601181 7...
           3.9    967136129397129535614217393209022771433233177 8...
           1.9                     95395395410571638250167612179141
           Name: Reviews, dtype: object
```

```
In [111]: # Which category has maximum size
          df.groupby('Category')['Size_in_bytes'].max()
```

```
Out [111]: Category
           ART_AND_DESIGN          40894464
           AUTO_AND_VEHICLES      101711872
           BEAUTY                  59768832
           BOOKS_AND_REFERENCE     91226112
           BUSINESS               102760448
           COMICS                  41943040
           COMMUNICATION           69206016
           DATING                  80740352
           EDUCATION              101711872
           ENTERTAINMENT           81788928
           EVENTS                  63963136
           FAMILY                 104857600
           FINANCE                104857600
           FOOD_AND_DRINK          79691776
           GAME                   104857600
           HEALTH_AND_FITNESS     104857600
           HOUSE_AND_HOME          80740352
           LIBRARIES_AND_DEMO     103809024
           LIFESTYLE              104857600
           MAPS_AND_NAVIGATION     81788928
           MEDICAL                104857600
           NEWS_AND_MAGAZINES      65011712
           PARENTING              102760448
           PERSONALIZATION         95420416
           PHOTOGRAPHY            100663296
           PRODUCTIVITY            79691776
           SHOPPING               102760448
           SOCIAL                 100663296
           SPORTS                 104857600
           TOOLS                  103809024
           TRAVEL_AND_LOCAL        94371840
           VIDEO_PLAYERS           95420416
           WEATHER                 56623104
           Name: Size_in_bytes, dtype: int64
```

```
In [112]: # Which content rating has highest reviews
          df.groupby('Content Rating')['Reviews'].sum().sort_values(ascending=False).head(3)
```

```
Out [112]: Content Rating
           Mature 17+        774733091615177883410101446120881151881461538 1...
           Everyone 10+      769935741823544783411573999521818931818933238 1...
           Adults only 18+                                       24005500177326
           Name: Reviews, dtype: object
```

```
In [114]: # which app is most installs
          df.groupby('App')['Installs'].sum().sort_values(ascending=False).head(3)
```

```
Out [114]: App
           Subway Surfers    6000000000
           Instagram         4000000000
           Google Photos     4000000000
           Name: Installs, dtype: int64
```

In [115]:
```
# Current version updated last time
df.groupby('Current Ver')['Last Updated'].sum().sort_values(ascending=False).head(3)
```

Out [115]:
```
Current Ver
4.2         September 8, 2016August 4, 2018October 16, 201...
2.2.0.23                                   September 8, 2015
5.26                                       September 7, 2017
Name: Last Updated, dtype: object
```

In [116]:
```
df.columns
```

Out [116]:
```
Index(['App', 'Category', 'Rating', 'Reviews', 'Size_in_bytes', 'Installs',
       'Type', 'Price', 'Content Rating', 'Genres', 'Last Updated',
       'Current Ver', 'Android Ver'],
      dtype='object')
```

In [118]:
```
# which tpye of content most reviews
df.groupby('Type')['Reviews'].sum()
```

Out [118]:
```
Type
Free    1599678751021564496716717836815137911211388087...
Paid    1144210295114421029518247154557004215461121310...
Name: Reviews, dtype: object
```

In [120]:
```
# which apps have occupy  more size
df.groupby('App')['Size_in_bytes'].sum().sort_values(ascending=False).head(3)
```

Out [120]:
```
App
ROBLOX                 632291328
Candy Crush Saga       543162368
Angry Birds Classic    508559360
Name: Size_in_bytes, dtype: int64
```

In [122]:
```
# which app has which version
df.groupby('App')['Current Ver'].sum().sort_values(ascending=False).head(3)
```

Out [122]:
```
App
MHD F-Series                                             version 0.994
Daily K-Talk                                                v8[1.0.10]
Calculator - free calculator, multi calculator app    v8.0.1.8.0629.1
Name: Current Ver, dtype: object
```