# Predicting User Ratings Of Mobile Games

Author: Christine Li

## Background

The mobile phone games industry has become increasingly popular over the years with the introduction of smart phones. While 'classic' gamers still own powerful computers or consoles for their gaming needs, smart phones and their gaming ability have become powerful in their own right. As technology has advanced, each of these platforms have turned into major sources of profits for companies. Whether it be on public transport, waiting for family and friends at meet-up locations or just taking the time at the end of the day to unwind, these platforms offer unlimited opportunies for people to escape into games.

One of the primary factors is accessibility of the smart phone. With the rapid evolution of the smartphone, mobile gaming's popularity is propelled by the smartphone's widespread availability, convenience and portability. This allows gamers to play almost virtually anywhere. Mobile games are also easy to download and can be played on the go without investing into any special equipment making it a valuable source of entertainment.

User reviews are just one factor among many that affect gaming sales. Game developers can leverage positive reviews to increase excitement and attract players. Game ratings can play a pivotal role in purchasing the game and gamers may have varying thresholds for ratings which affect their buying.

## Business Problem - Predicting User Ratings of Mobile Strategy Games

Objective: To create a predictive multiple regression model to predict the user rating of a mobile strategy game.

Using this tool, game developers can create a mobile strategy game that capitalises on features that lead to increased user ratings. Mobile game ratings are a crucial metric for developers and companies to gauge the success of their games and overall revenue generation.

## Data

The mobile gaming industry is incredibly large, so we will be focussing on a subset of mobile games classified as mobile strategy games.

This dataset, '17K Mobile Strategy Games' was downloaded from Kaggle. This can be accessed with the URL https://www.kaggle.com/datasets/tristan581/17k-apple-app-store-strategy-games?resource=download (https://www.kaggle.com/datasets/tristan581/17k-apple-app-store-strategy-games?resource=download) The data was collected by the owner of the dataset by mostly using the iTunes API, App Store sitemap, along with some web scraping on 3rd of August 2019.

Column Names and Descriptions:

- *URL* - The URL for the mobile game
- *ID* - The assigned ID for the game
- *Name* - The name of the game
- *Subtitle* - The secondary text under the name.
- *Icon URL* - The URL for the game icon
- *Average User Rating* - Rounded to nearest 0.5, requires at least 5 ratings.
- *User Rating Count* - Number of ratings internationally, null means it is below 5
- *Price* - Price in USD
- *In-app Purchases* - Prices of available in-app purchases
- *Description* - Mobile Game App description
- *Developer* - App developer
- *Age Rating* - Either 4+, 9+, 12+ or 17+
- *Languages* - Language codes using ISO Language Codes
- *Size* - Size of the app in bytes
- *Primary Genre* - The main genre of the app
- *Genres* - Genres of the app
- *Original Release Date* - When it was released
- *Current Version Release Date* - When it was last updated

In [1]:
```
#importing the necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

from sklearn import linear_model
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler, MinMaxScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.model_selection import train_test_split, cross_val_score
import statsmodels.api as sm
import scipy.stats as stats
```

In [2]:
```
games = pd.read_csv('/Users/christineli/Desktop/Capstone Project/appstore_games.csv')
games.head()
```

Out[2]:

| | URL | ID | Name | Subtitle | Icon URL | Average User Rating | User Rating Count | Price | In-a Purchas |
|---|---|---|---|---|---|---|---|---|---|
| 0 | https://apps.apple.com/us/app/sudoku/id284921427 | 284921427 | Sudoku | NaN | https://is2-ssl.mzstatic.com/image/thumb/Purpl... | 4.0 | 3553.0 | 2.99 | Na |
| 1 | https://apps.apple.com/us/app/reversi/id284926400 | 284926400 | Reversi | NaN | https://is4-ssl.mzstatic.com/image/thumb/Purpl... | 3.5 | 284.0 | 1.99 | Na |
| 2 | https://apps.apple.com/us/app/morocco/id284946595 | 284946595 | Morocco | NaN | https://is5-ssl.mzstatic.com/image/thumb/Purpl... | 3.0 | 8376.0 | 0.00 | Na |
| 3 | https://apps.apple.com/us/app/sudoku-free/id28... | 285755462 | Sudoku (Free) | NaN | https://is3-ssl.mzstatic.com/image/thumb/Purpl... | 3.5 | 190394.0 | 0.00 | Na |
| 4 | https://apps.apple.com/us/app/senet-deluxe/id2... | 285831220 | Senet Deluxe | NaN | https://is1-ssl.mzstatic.com/image/thumb/Purpl... | 3.5 | 28.0 | 2.99 | Na |

In [3]:
```
games.describe()
```

Out[3]:

| | ID | Average User Rating | User Rating Count | Price | Size |
|---|---|---|---|---|---|
| count | 1.700700e+04 | 7561.000000 | 7.561000e+03 | 16983.000000 | 1.700600e+04 |
| mean | 1.059614e+09 | 4.060905 | 3.306531e+03 | 0.813419 | 1.157064e+08 |
| std | 2.999676e+08 | 0.751428 | 4.232256e+04 | 7.835732 | 2.036477e+08 |
| min | 2.849214e+08 | 1.000000 | 5.000000e+00 | 0.000000 | 5.132800e+04 |
| 25% | 8.996543e+08 | 3.500000 | 1.200000e+01 | 0.000000 | 2.295014e+07 |
| 50% | 1.112286e+09 | 4.500000 | 4.600000e+01 | 0.000000 | 5.676895e+07 |
| 75% | 1.286983e+09 | 4.500000 | 3.090000e+02 | 0.000000 | 1.330271e+08 |
| max | 1.475077e+09 | 5.000000 | 3.032734e+06 | 179.990000 | 4.005591e+09 |

In [4]:
```
games.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17007 entries, 0 to 17006
Data columns (total 18 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   URL                           17007 non-null  object 
 1   ID                            17007 non-null  int64  
 2   Name                          17007 non-null  object 
 3   Subtitle                      5261 non-null   object 
 4   Icon URL                      17007 non-null  object 
 5   Average User Rating           7561 non-null   float64
 6   User Rating Count             7561 non-null   float64
 7   Price                         16983 non-null  float64
 8   In-app Purchases              7683 non-null   object 
 9   Description                   17007 non-null  object 
 10  Developer                     17007 non-null  object 
 11  Age Rating                    17007 non-null  object 
 12  Languages                     16947 non-null  object 
 13  Size                          17006 non-null  float64
 14  Primary Genre                 17007 non-null  object 
 15  Genres                        17007 non-null  object 
 16  Original Release Date         17007 non-null  object 
 17  Current Version Release Date  17007 non-null  object 
dtypes: float64(4), int64(1), object(13)
memory usage: 2.3+ MB
```

We can see that this is a large dataset with more than 17000 entries and 18 columns.

At first glance, we can see that majority of this data is an object type which first needs to be transformed before it is able to be used in a multiple regression model.

We can also see that there are some features of this dataset which won't be relevent for this project which will be dealt with in our Scrub section below.

## Scrub:

Data preparation

- Deal with missing values
- Deal with outliers
- Fix data values, data types
- Clean data

Our target variable is the average **user rating** and the other features in our dataset will be the predictors.

In [5]:
```
#Convert object types columns 'In-app Purchases', 'Age Rating', into numeric data types
games["In-app Purchases"] = pd.to_numeric(games["In-app Purchases"], errors = 'coerce')

#Remove the + from 4+, 9+, 7+, 12+ etc in Age Rating Column to convert into float
games["Age Rating"] = [float(str(i).replace("+", "")) for i in games["Age Rating"]]
games['Age Rating'].value_counts()
```

Out[5]:
```
4.0     11806
9.0      2481
12.0     2055
17.0      665
Name: Age Rating, dtype: int64
```

In [6]:
```
#Confirming that 'In-app Purchases' and 'Age Rating' are now numeric types.
games.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17007 entries, 0 to 17006
Data columns (total 18 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   URL                           17007 non-null  object 
 1   ID                            17007 non-null  int64  
 2   Name                          17007 non-null  object 
 3   Subtitle                      5261 non-null   object 
 4   Icon URL                      17007 non-null  object 
 5   Average User Rating           7561 non-null   float64
 6   User Rating Count             7561 non-null   float64
 7   Price                         16983 non-null  float64
 8   In-app Purchases              2498 non-null   float64
 9   Description                   17007 non-null  object 
 10  Developer                     17007 non-null  object 
 11  Age Rating                    17007 non-null  float64
 12  Languages                     16947 non-null  object 
 13  Size                          17006 non-null  float64
 14  Primary Genre                 17007 non-null  object 
 15  Genres                        17007 non-null  object 
 16  Original Release Date         17007 non-null  object 
 17  Current Version Release Date  17007 non-null  object 
dtypes: float64(6), int64(1), object(11)
memory usage: 2.3+ MB
```

In [7]:
```
#Extract year from 'Original Release Date' and 'Current Version Release Date'
games['Original Release Date'] = pd.to_datetime(games['Original Release Date'])
games['Current Version Release Date'] = pd.to_datetime(games['Current Version Release Date'])

games['Original Year'] = games['Original Release Date'].dt.year
games['Current Version Year'] = games['Current Version Release Date'].dt.year
```

In [8]:
```
#Drop unnecessary columns and repeated Original Release Date and Current Version Release Date columns
games_clean = games.drop(columns = ['Name','URL', 'ID', 'Subtitle', 'Icon URL', 'Description', 'Original Release

games_clean.head()
```

Out[8]:

| | Average User Rating | User Rating Count | Price | In-app Purchases | Developer | Age Rating | Languages | Size | Primary Genre | Genres | Original Year | Current Version Year |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 4.0 | 3553.0 | 2.99 | NaN | Mighty Mighty Good Games | 4.0 | DA, NL, EN, FI, FR, DE, IT, JA, KO, NB, PL, PT... | 15853568.0 | Games | Games, Strategy, Puzzle | 2008 | 2017 |
| **1** | 3.5 | 284.0 | 1.99 | NaN | Kiss The Machine | 4.0 | EN | 12328960.0 | Games | Games, Strategy, Board | 2008 | 2018 |
| **2** | 3.0 | 8376.0 | 0.00 | NaN | Bayou Games | 4.0 | EN | 674816.0 | Games | Games, Board, Strategy | 2008 | 2017 |
| **3** | 3.5 | 190394.0 | 0.00 | NaN | Mighty Mighty Good Games | 4.0 | DA, NL, EN, FI, FR, DE, IT, JA, KO, NB, PL, PT... | 21552128.0 | Games | Games, Strategy, Puzzle | 2008 | 2017 |
| **4** | 3.5 | 28.0 | 2.99 | NaN | RoGame Software | 4.0 | DA, NL, EN, FR, DE, EL, IT, JA, KO, NO, PT, RU... | 34689024.0 | Games | Games, Strategy, Board, Education | 2008 | 2018 |

In [9]:
```
#Check for missing values
games_clean.isna().sum()

#There are missing values in the Average User Rating, User Rating Count, In-app Purchases,
#Languages and Size columns
```

Out[9]:
```
Average User Rating      9446
User Rating Count        9446
Price                      24
In-app Purchases        14509
Developer                   0
Age Rating                  0
Languages                  60
Size                        1
Primary Genre               0
Genres                      0
Original Year               0
Current Version Year        0
dtype: int64
```

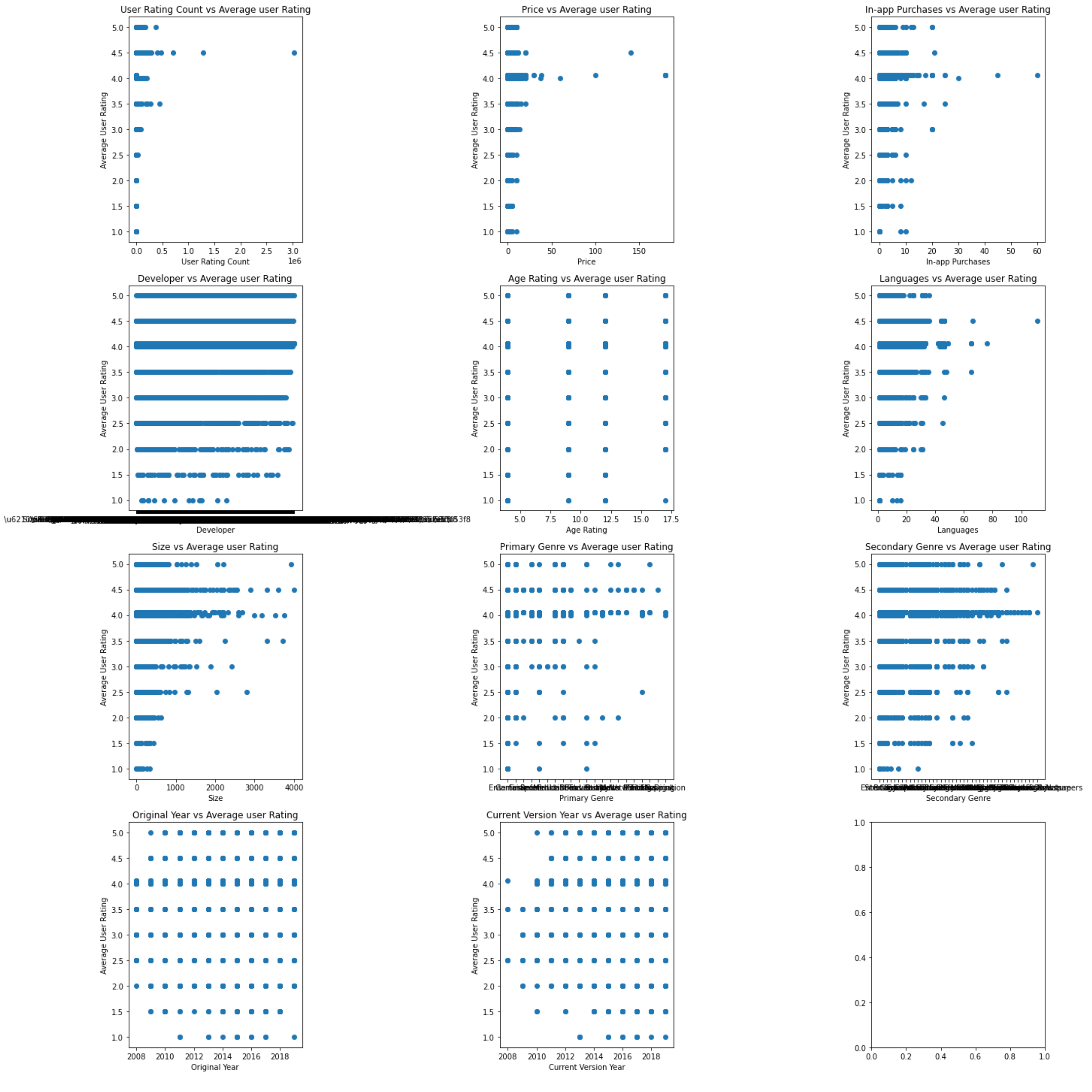
In [10]:
```
#In-app purchases: For no in-app purchases, rather leaving it empty replace it with 0.
games_clean['In-app Purchases'].fillna(0, inplace = True)

#Languages: Replace it with english, 'EN', rather than leaving it missing as English
# is one of the most global and spoken languages
games_clean['Languages'].fillna('EN', inplace = True)

#Price: Replace missing values with mean value since only small number missing
games_clean['Price'].fillna(np.mean(games_clean['Price']), inplace = True)

#Convert Size into Megabytes and replace missing value with mean value
games_clean['Size'] = round(games['Size']/1000000, 2)
games_clean['Size'].fillna(np.mean(games_clean['Size']), inplace=True)

#Average User Rating and User Rating Count: Replace with mean value as they are important columns
games_clean['Average User Rating'].fillna(np.mean(games_clean['Average User Rating']), inplace = True)
games_clean['User Rating Count'].fillna(np.mean(games_clean['User Rating Count']), inplace = True)

#Double check all missing values are dealt with
games_clean.isna().sum()
```

Out[10]:
```
Average User Rating     0
User Rating Count       0
Price                   0
In-app Purchases        0
Developer               0
Age Rating              0
Languages               0
Size                    0
Primary Genre           0
Genres                  0
Original Year           0
Current Version Year    0
dtype: int64
```

In [11]:
```
#Fix Languages
games_clean['Languages'] = games['Languages'].apply(lambda x: len(str(x).split(',')))
```

In [12]:
```
#Fix genres

Primary_Genre = []
Secondary_Genre = []
for x in games['Genres']:
    Primary_Genre.append(x.split(',')[0])
    Secondary_Genre.append(x.split(',')[1])

games_clean['Primary Genre'] = Primary_Genre
games_clean['Genres'] = Secondary_Genre

games_clean.rename(columns = {'Genres':'Secondary Genre'}, inplace = True)
```

In [13]:
```
#Identify outliers
#Visualise relationship between features and the response variable using scatter plots

features = games_clean.drop(columns = ['Average User Rating'])

columns_to_plot = features

fig, axes = plt.subplots(nrows=4, ncols=3, figsize=(20, 20))
axes = axes.flatten()

for index, column in enumerate(columns_to_plot):
    ax = axes[index]
    ax.scatter(columns_to_plot[column], games_clean['Average User Rating'])
    ax.set_title(f'{column} vs Average user Rating')
    ax.set_xlabel(column)
    ax.set_ylabel('Average User Rating')

# Adjust layout to prevent overlapping
plt.tight_layout()
plt.show()
```




We can also see that there are some outliers in these scatterplots.
Outliers: User Rating Count, Price, In-App Purchases, Languages

In [14]:
```
#Deal with the outliers in 'User Rating Count'
z_user_count = np.abs(stats.zscore(games_clean['User Rating Count']))

threshold = 3
outliers_user_count = games_clean[z_user_count>threshold]
print('Outliers:', len(outliers_user_count)) #There are 53 outliers in User Rating Count

games_clean1 = games_clean.drop(outliers_user_count.index, axis =0)
games_clean1.info()
```

```
Outliers: 53
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16954 entries, 0 to 17006
Data columns (total 12 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   Average User Rating  16954 non-null  float64
 1   User Rating Count    16954 non-null  float64
 2   Price                16954 non-null  float64
 3   In-app Purchases     16954 non-null  float64
 4   Developer            16954 non-null  object 
 5   Age Rating           16954 non-null  float64
 6   Languages            16954 non-null  int64  
 7   Size                 16954 non-null  float64
 8   Primary Genre        16954 non-null  object 
 9   Secondary Genre      16954 non-null  object 
 10  Original Year        16954 non-null  int64  
 11  Current Version Year 16954 non-null  int64  
dtypes: float64(6), int64(3), object(3)
memory usage: 1.7+ MB
```

In [15]:
```
#Deal with outliers in Price

z_price = np.abs(stats.zscore(games_clean1['Price']))
#threshold is 3 as indicated above
outliers_price = games_clean1[z_price>threshold]
print('Outliers:', len(outliers_price)) #There are 37 outliers in User Rating Count

games_clean2 = games_clean1.drop(outliers_price.index, axis =0)
games_clean2.info()
```

```
Outliers: 37
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16917 entries, 0 to 17006
Data columns (total 12 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   Average User Rating  16917 non-null  float64
 1   User Rating Count    16917 non-null  float64
 2   Price                16917 non-null  float64
 3   In-app Purchases     16917 non-null  float64
 4   Developer            16917 non-null  object 
 5   Age Rating           16917 non-null  float64
 6   Languages            16917 non-null  int64  
 7   Size                 16917 non-null  float64
 8   Primary Genre        16917 non-null  object 
 9   Secondary Genre      16917 non-null  object 
 10  Original Year        16917 non-null  int64  
 11  Current Version Year 16917 non-null  int64  
dtypes: float64(6), int64(3), object(3)
memory usage: 1.7+ MB
```

In [16]:
```
#Deal with outliers in In-App Purchases

z_in_app_purchases = np.abs(stats.zscore(games_clean2['In-app Purchases']))
#threshold is 3 as indicated above
outliers_in_app_purchases = games_clean2[z_in_app_purchases>threshold]
print('Outliers:', len(outliers_in_app_purchases)) #There are 282 outliers in User Rating Count

games_clean3 = games_clean2.drop(outliers_in_app_purchases.index, axis=0)
games_clean3.info()
```

```
Outliers: 282
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16635 entries, 0 to 17006
Data columns (total 12 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   Average User Rating  16635 non-null  float64
 1   User Rating Count    16635 non-null  float64
 2   Price                16635 non-null  float64
 3   In-app Purchases     16635 non-null  float64
 4   Developer            16635 non-null  object 
 5   Age Rating           16635 non-null  float64
 6   Languages            16635 non-null  int64  
 7   Size                 16635 non-null  float64
 8   Primary Genre        16635 non-null  object 
 9   Secondary Genre      16635 non-null  object 
 10  Original Year        16635 non-null  int64  
 11  Current Version Year 16635 non-null  int64  
dtypes: float64(6), int64(3), object(3)
memory usage: 1.6+ MB
```

In [17]:
```
#Assign final removed outlier dataframe back to clean dataframe
games_clean = games_clean3
```

In [18]:
```
#Checking to make sure data is now cleaned.
#Other than Developer, Primary and Secondary Genre, the rest of the data should be numeric
games_clean.info()
```

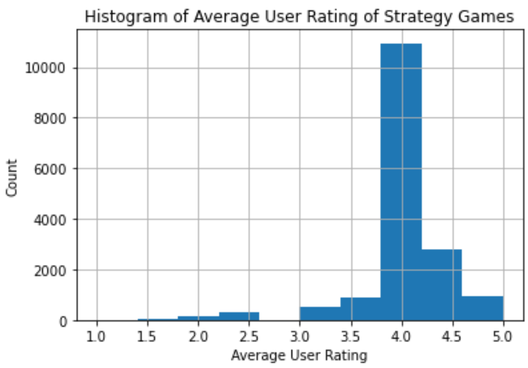
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16635 entries, 0 to 17006
Data columns (total 12 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   Average User Rating  16635 non-null  float64
 1   User Rating Count    16635 non-null  float64
 2   Price                16635 non-null  float64
 3   In-app Purchases     16635 non-null  float64
 4   Developer            16635 non-null  object 
 5   Age Rating           16635 non-null  float64
 6   Languages            16635 non-null  int64  
 7   Size                 16635 non-null  float64
 8   Primary Genre        16635 non-null  object 
 9   Secondary Genre      16635 non-null  object 
 10  Original Year        16635 non-null  int64  
 11  Current Version Year 16635 non-null  int64  
dtypes: float64(6), int64(3), object(3)
memory usage: 1.6+ MB
```

## Exploratory Data Analysis

#### Average user rating

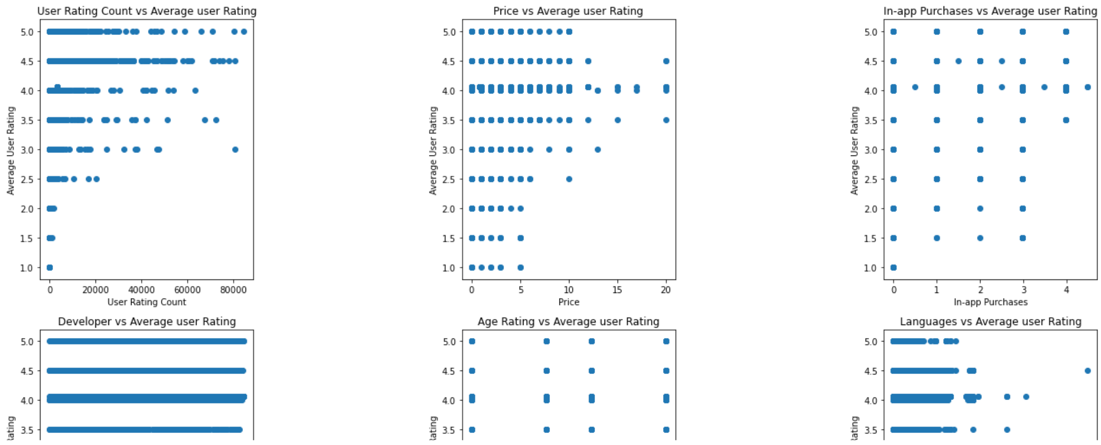
In [19]:
```
games_clean['Average User Rating'].hist()
plt.title ('Histogram of Average User Rating of Strategy Games')
plt.xlabel('Average User Rating')
plt.ylabel('Count')
```

Out[19]: Text(0, 0.5, 'Count')



The average user rating of strategy games are rated 4.0.

In [20]:
```
#Visualise relationship between features and the response variable using scatter plots

features = games_clean.drop(columns = ['Average User Rating'])

columns_to_plot = features

fig, axes = plt.subplots(nrows=4, ncols=3, figsize=(20, 20))
axes = axes.flatten()

for index, column in enumerate(columns_to_plot):
    ax = axes[index]
    ax.scatter(columns_to_plot[column], games_clean['Average User Rating'])
    ax.set_title(f'{column} vs Average user Rating')
    ax.set_xlabel(column)
    ax.set_ylabel('Average User Rating')

# Adjust layout to prevent overlapping
plt.tight_layout()
plt.show()
```

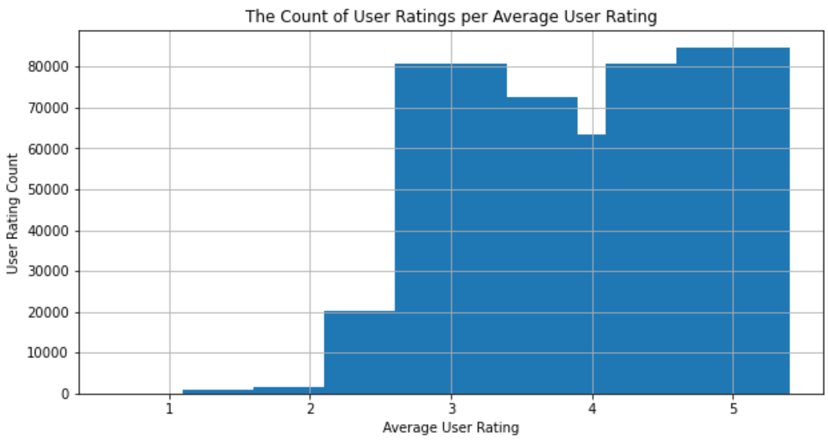



Since we have removed the outliers above, we can see the relationship between the feature variables and the target variable clearer than before.

Overall, there does not appear to be any strong linear relationships between the feature variables and the user ratings.

**User Rating Count**

In [21]:
```
plt.figure(figsize = (10,5))
plt.bar(games_clean['Average User Rating'], games_clean['User Rating Count'])
plt.xlabel('Average User Rating')
plt.ylabel('User Rating Count')
plt.title('The Count of User Ratings per Average User Rating')
plt.grid()
```

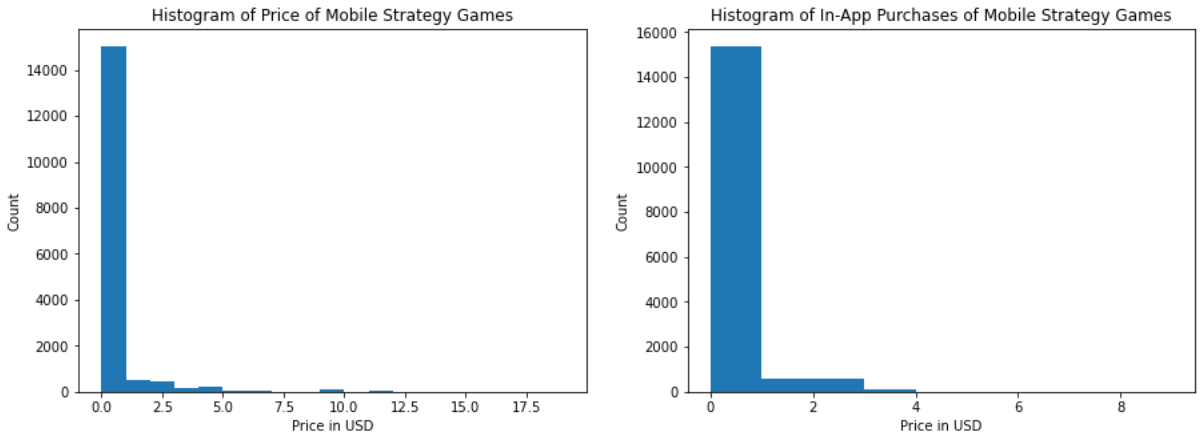



This looks very similar to our average user rating graph, that the majority of the data is centered around the 4.0. We can see that the higher the number of User Ratings, the higher the rating.

**Price and In-App Purchases**

In [22]:
```
plt.figure(figsize = (15, 5))
plt.subplot(1, 2, 1)
plt.hist(games_clean['Price'], range(0, 20))
plt.title('Histogram of Price of Mobile Strategy Games')
plt.xlabel('Price in USD')
plt.ylabel('Count')
#There is an outlier of an app price with $179.99 therefore limited the range
#for visualisation purposes only

plt.subplot(1, 2, 2)
plt.hist(games_clean['In-app Purchases'], range(0, 10))
plt.title('Histogram of In-App Purchases of Mobile Strategy Games')
plt.xlabel('Price in USD')
plt.ylabel('Count')
```

Out[22]:
```
Text(0, 0.5, 'Count')
```

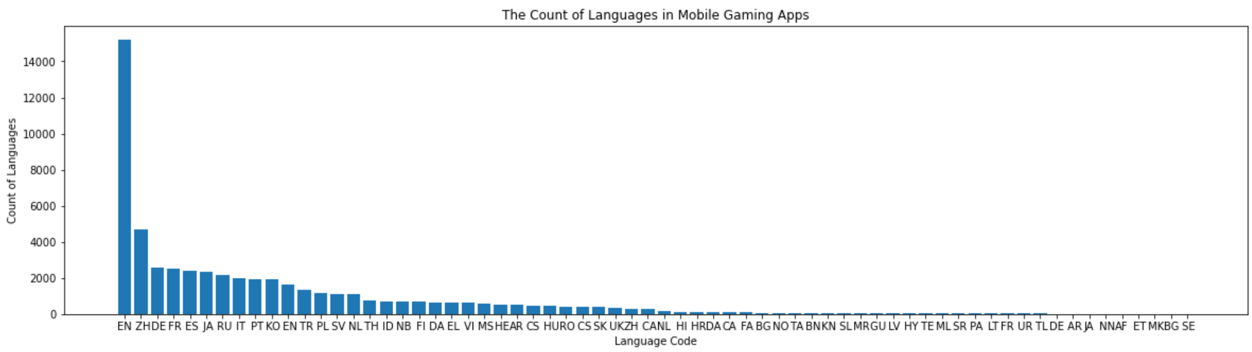



These two graphs look very similar as well. Majority of these apps are free-to-play or cost less than $5 as an upfront fee to download and begin playing. These apps also have low-cost in-app purchases.

**Languages**

In [23]:
```
games_languages = games['Languages'].str.split(',', expand = True).stack()
lang_count = games_languages.value_counts().to_dict()
len(lang_count.keys()) #139

#For the purpose of the visualisation, as there are 139 unique values in total
#remove any dictionary value that is 10 or less so it can all fit on the graph
lang_vis = {key: value for key, value in lang_count.items() if value > 10}

plt.figure(figsize = (20,5))
plt.bar(lang_vis.keys(), lang_vis.values())
plt.xlabel('Language Code')
plt.ylabel('Count of Languages')
plt.title('The Count of Languages in Mobile Gaming Apps')
```

Out[23]:
```
Text(0.5, 1.0, 'The Count of Languages in Mobile Gaming Apps')
```




Our most popular language for these mobile app games is **English** (EN), followed by Chinese (ZH) and then German (DE).

I have attached a link to the list of ISO language codes for further clarification of the codes - https://en.wikipedia.org/wiki/List_of_ISO_639_language_codes (https://en.wikipedia.org/wiki/List_of_ISO_639_language_codes)

Another important feature to look at are the genres of these mobile games.

**Genres**

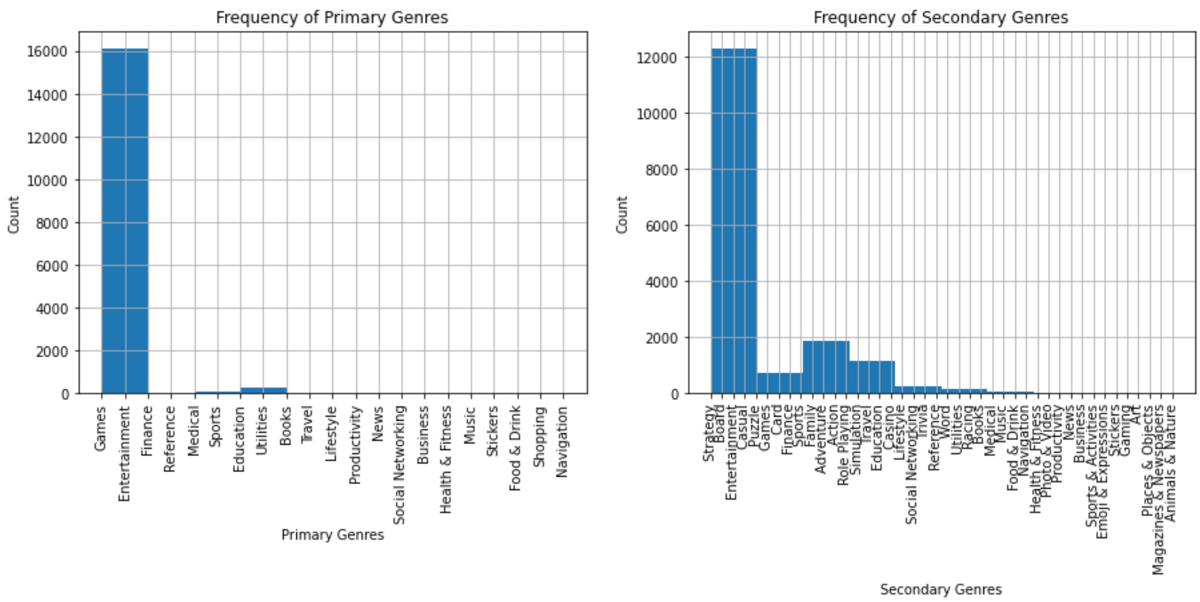
In [24]:
```
games['Genres'].value_counts()
```

Out[24]:
```
Games, Strategy, Puzzle                                      778
Games, Puzzle, Strategy                                      694
Games, Strategy                                              588
Games, Strategy, Action                                      483
Games, Simulation, Strategy                                  465
                                                            ...
Health & Fitness, Casual, Games, Strategy                      1
Stickers, Places & Objects, Strategy, Games, Adventure, Gaming      1
Finance, Strategy, Simulation, Games                           1
Social Networking, Adventure, Strategy, Games                  1
Games, Travel, Board, Strategy                                 1
Name: Genres, Length: 1004, dtype: int64
```

In [25]:
```
#Some games have multiple genres, so want to split the genre column into
#its unique genres.
games_genres = games['Genres'].str.split(',', expand = True).stack()

genre_count = games_genres.value_counts().to_dict()
len(genre_count.keys()) #68

plt.figure(figsize = (20,5))
plt.bar(genre_count.keys(), genre_count.values())
plt.xticks(rotation = 90)
```

```
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, '')])
```

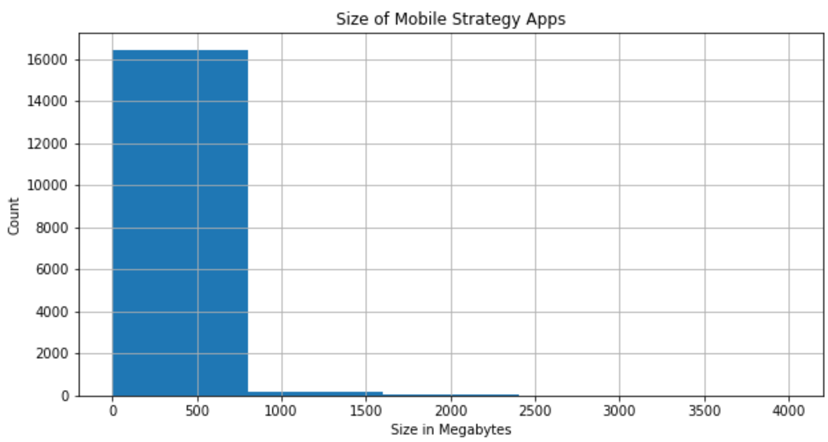
In [26]:
```
plt.figure(figsize = (15, 5))
plt.subplot(1, 2, 1)
plt.hist(games_clean['Primary Genre'], align = 'mid')
plt.title('Frequency of Primary Genres')
plt.xlabel('Primary Genres')
plt.ylabel('Count')
plt.xticks(rotation = 90)
plt.grid()

plt.subplot(1, 2, 2)
plt.hist(games_clean['Secondary Genre'], align = 'mid')
plt.title('Frequency of Secondary Genres')
plt.xlabel('Secondary Genres')
plt.ylabel('Count')
plt.xticks(rotation = 90)
plt.grid()
```




Games are our most frequent genre with Strategy games as the most popular form of game.

**Size**

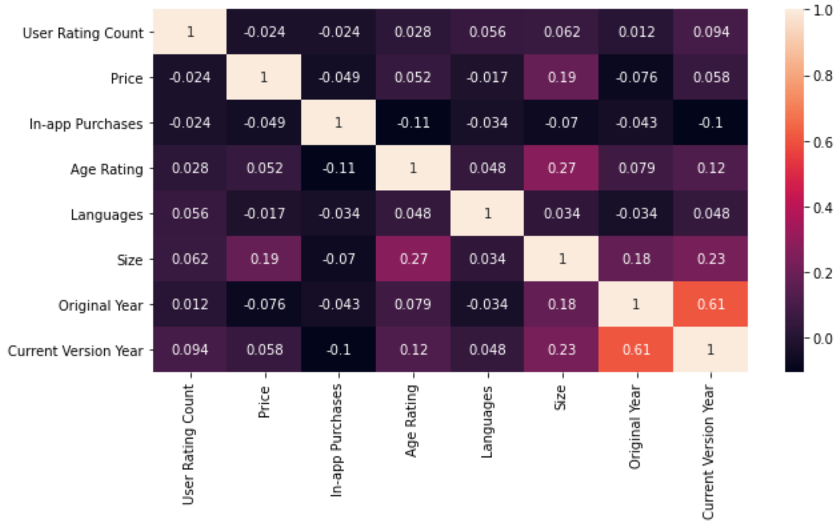
In [27]:
```
plt.figure(figsize = (10, 5))
plt.hist(games_clean['Size'], bins = 5)
plt.title('Size of Mobile Strategy Apps')
plt.xlabel('Size in Megabytes')
plt.ylabel('Count')
plt.grid()
```




Majority of these mobile game apps are within 7500MB in size.

In [28]:
```
#To understand the correlation structure, removing the target variable Average User Rating
#to see how the predictors relate to each other

games_corr = features
```

In [29]:
```
fig, ax = plt.subplots(figsize = (10,5))
sns.heatmap(games_corr.corr(), annot = True)
plt.show()
```




Looking at this dataset, there are **no** variables that are highly correlated with each other suggesting that we will most likely not have issues with multicollinearity.

## Preparing the Model

- Encode the data
- Split the data
- Scale the data

*Encode the Data*

Current variables in our dataset are:
Categorical: Developer, Age Rating, Languages, Primary Genre, Secondary Genre
Continuous: Average User Rating, User Rating Count, Price, In-app Purchases, Size, Original Year, Current Version Year

In [30]:
```
categorical = pd.DataFrame(games_clean.select_dtypes(include=['object']))
# This leads to Developer, Primary Genre, Secondary Genre
```

In [31]:
```
games_ohe = pd.get_dummies(categorical, drop_first = True)
games_prepped = games_clean.drop(columns = ['Average User Rating','Developer', 'Primary Genre', 'Secondary Genre
games_ohe_prepped = pd.concat([games_ohe.reset_index(drop=True), games_prepped.reset_index(drop=True)], axis=1)
```

In [32]:
```
games_ohe_prepped
```

Out[32]:

| | Developer_"ByteRockers' Games GmbH & Co. KG" | Developer_"Daniel O'Sullivan" | Developer_"Don't Blink Studios" | Developer_"Ellie's Games, LLC" | Developer_"Galen O'Shea" | Developer_"Igor's Software Labs LLC" | Developer_"It's All A Game LLC" | Developer_ o'donc |
|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **2** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **3** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **4** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | |
| **16630** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **16631** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **16632** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **16633** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **16634** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

16635 rows × 8663 columns

*Split the data*

In [33]:
```
#Split data into train/test split to prevent data leakage
X = games_ohe_prepped
y = games_clean['Average User Rating']
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

print("Length of the training and test set:", len(X_train), len(X_test), len(y_train), len(y_test))
```

```
Length of the training and test set: 12476 4159 12476 4159
```

*Scale the data*

In [34]:
```
scaler = StandardScaler()
X_train_normalised = scaler.fit_transform(X_train)
X_test_normalised = scaler.transform(X_test)
```

## Build the First Model

In [35]:
```
#Run the model

X_train_int = sm.add_constant(X_train_normalised)
model = sm.OLS(y_train, X_train_int).fit()
model.summary()
```

Out[35]:

OLS Regression Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | Average User Rating | **R-squared:** | 0.677 |
| **Model:** | OLS | **Adj. R-squared:** | 0.256 |
| **Method:** | Least Squares | **F-statistic:** | 1.606 |
| **Date:** | Mon, 26 Feb 2024 | **Prob (F-statistic):** | 7.52e-75 |
| **Time:** | 19:10:40 | **Log-Likelihood:** | -2081.4 |
| **No. Observations:** | 12476 | **AIC:** | 1.830e+04 |
| **Df Residuals:** | 5407 | **BIC:** | 7.083e+04 |
| **Df Model:** | 7068 | | |
| **Covariance Type:** | nonrobust | | |

| | **coef** | **std err** | **t** | **P>\|t\|** | **[0.025** | **0.975]** |
|---|---|---|---|---|---|---|
| const | 4.0619 | 0.004 | 1044.684 | 0.000 | 4.054 | 4.069 |

R^2: 0.677 - 67% of the variance in the target variable User Ratings can be explained by the predictor features. Around 67% of the data fits on the regression model.
Adjusted R-squared value: 0.256 - This is a modified R^2 that has been adjusted by the number of predictors in the model. There are some variables here that do not appear to contribute to the model.

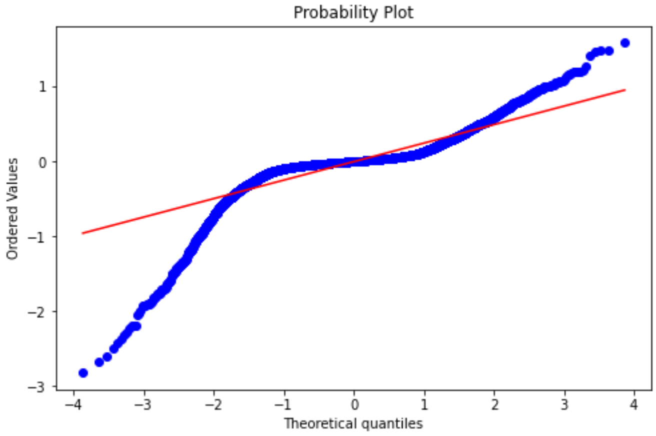
The adjusted R2 increases when a new variable improves the model more than would be expected by chance. It decreases when a predictor improves the model less than expected. A very low R2 value generally indicates underfitting, which means adding additional relevant features or using a more complex model might help.

p_value (lists as Prob F-statistic), since this is less than 0 we can reject the null hypothesis.

#### Check the Linearity Assumptions

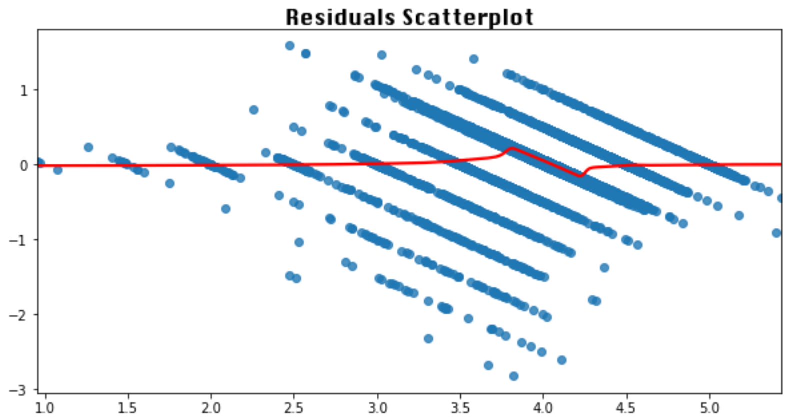
In [36]:
```
residuals = model.resid
```

In [37]:
```
import scipy as sp

fig, ax = plt.subplots(figsize = (8, 5))
sp.stats.probplot(residuals, plot = ax, fit = True)
plt.show()

#Most of the data points fall somewhere on the line so meets the normality assumption
```




In [38]:
```
plt.figure(figsize=(10,5))
sns.regplot(x=model.predict(), y=model.resid, lowess=True, line_kws={'color': 'red'})
plt.title('Residuals Scatterplot', fontsize=16, y=.99, fontname='Silom')

#Residuals are uniform across in a linear line
```

Out[38]:
```
Text(0.5, 0.99, 'Residuals Scatterplot')
```



### Build the Second Model

In [39]:
```
#Build 2nd model, attempt to reduce size of condition number
#Pick new categoricals: Age Rating, Primary Genre, Secondary Genre
#Dropping Developer as it an extremely large increase in new columns with dummy implementation
#therefore it may be affecting how the model runs

games2_clean = games_clean.drop(columns = ['Average User Rating', 'Developer'])

categoricals2 = ['Age Rating','Primary Genre', 'Secondary Genre']

#Implement dummies
games_ohe2 = pd.get_dummies(games2_clean[categoricals2], drop_first = True)
games_prepped2 = games2_clean.drop(columns = ['Age Rating','Primary Genre', 'Secondary Genre'])
games_ohe_prepped2 = pd.concat([games_ohe2, games_prepped2], axis = 1)

games_ohe_prepped2
```

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 3553.000000 | 2.99 | 0.00 | 17 | 15.85 | 2008 | 2017 |
| 0 | 1 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 284.000000 | 1.99 | 0.00 | 1 | 12.33 | 2008 | 2018 |
| 0 | 1 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 8376.000000 | 0.00 | 0.00 | 1 | 0.67 | 2008 | 2017 |
| 0 | 1 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 28.000000 | 2.99 | 0.00 | 15 | 34.69 | 2008 | 2018 |
| 0 | 1 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 47.000000 | 0.00 | 1.99 | 1 | 48.67 | 2008 | 2019 |
| .. | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 0 | 1 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 3306.531279 | 0.00 | 0.00 | 1 | 64.80 | 2019 | 2019 |
| 0 | 1 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 3306.531279 | 0.00 | 0.00 | 1 | 110.34 | 2019 | 2019 |
| 0 | 1 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 3306.531279 | 0.00 | 0.00 | 1 | 23.21 | 2019 | 2019 |
| 0 | 1 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 3306.531279 | 0.00 | 0.00 | 1 | 196.75 | 2019 | 2019 |
| 0 | 1 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 3306.531279 | 0.00 | 0.00 | 2 | 22.95 | 2019 | 2019 |

In [40]:
```
X2 = games_ohe_prepped2
y2 = games_clean['Average User Rating']
X2_train, X2_test, y2_train, y2_test = train_test_split(X2, y2, random_state=42)
```

In [41]:
```
X2_train_normalised = scaler.fit_transform(X2_train)
X2_test_normalised = scaler.transform(X2_test)
```

In [42]:
```
X2_train_int = sm.add_constant(X2_train_normalised)
model2 = sm.OLS(y2_train, X2_train_int).fit()
model2.summary()
```

Out[42]:

OLS Regression Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | Average User Rating | **R-squared:** | 0.053 |
| **Model:** | OLS | **Adj. R-squared:** | 0.048 |
| **Method:** | Least Squares | **F-statistic:** | 10.30 |
| **Date:** | Mon, 26 Feb 2024 | **Prob (F-statistic):** | 2.13e-102 |
| **Time:** | 19:11:03 | **Log-Likelihood:** | -8795.3 |
| **No. Observations:** | 12476 | **AIC:** | 1.773e+04 |
| **Df Residuals:** | 12407 | **BIC:** | 1.824e+04 |
| **Df Model:** | 68 | | |
| **Covariance Type:** | nonrobust | | |

| | **coef** | **std err** | **t** | **P>\|t\|** | **[0.025** | **0.975]** |
|---|---|---|---|---|---|---|
| const | [illegible] | [illegible] | [illegible] | 0.000 | [illegible] | [illegible] |

R^2: 0.053 - 5% of the variance in the target variable User Ratings can be explained by the predictor features. Around 5% of the data fits on the regression model.
Adjusted R-squared value: 0.048 - This is a modified R^2 that has been adjusted by the number of predictors in the model. The R^2 and adjusted R^2 are quite similar in values which means that this model has not been too penalised by the addition of multiple features.

p_value (lists as Prob F-statistic), since this is less than 0 we can reject the null hypothesis.

### Build the 3rd model

In [43]:
```
#Build 3rd model, attempt to reduce size of condition number and increase R2

games3_clean = games_clean.drop(columns = ['Average User Rating'])

categoricals3 = ['Age Rating','Developer', 'Languages', 'Primary Genre', 'Secondary Genre']

#Implement dummies
games_ohe3 = pd.get_dummies(games3_clean[categoricals3], drop_first = True)
games_prepped3 = games3_clean.drop(columns = ['Age Rating','Developer', 'Languages', 'Primary Genre', 'Secondary
games_ohe_prepped3 = pd.concat([games_ohe3, games_prepped3], axis = 1)

games_ohe_prepped3
```

Out[43]:

| | Age Rating | Languages | Developer_"ByteRockers' Games GmbH & Co. KG" | Developer_"Daniel O'Sullivan" | Developer_"Don't Blink Studios" | Developer_"Ellie's Games, LLC" | Developer_"Galen O'Shea" | Developer_"Igor's Software Labs LLC" | Develo Al |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 4.0 | 17 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **1** | 4.0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **2** | 4.0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **4** | 4.0 | 15 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **5** | 4.0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | |
| **17002** | 4.0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **17003** | 4.0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **17004** | 4.0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **17005** | 4.0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **17006** | 4.0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | |

16635 rows × 8663 columns

In [44]:
```
X3 = games_ohe_prepped3
y3 = games_clean['Average User Rating']
X3_train, X3_test, y3_train, y3_test = train_test_split(X3, y3, random_state=42)
```

In [45]:
```
X3_train_normalised = scaler.fit_transform(X3_train)
X3_test_normalised = scaler.transform(X3_test)
```

In [46]:
```
X3_train_int = sm.add_constant(X3_train_normalised)
model3 = sm.OLS(y3_train, X3_train_int).fit()
```

In [47]:
```
model3.summary()
```

| | | | | | | |
|---|---|---|---|---|---|---|
| **x45** | -0.0026 | 0.006 | -0.472 | 0.637 | -0.014 | 0.008 |
| **x46** | -1.085e+12 | 5.73e+11 | -1.895 | 0.058 | -2.21e+12 | 3.74e+10 |
| **x47** | -0.0010 | 0.006 | -0.180 | 0.857 | -0.012 | 0.010 |
| **x48** | -0.0065 | 0.006 | -1.166 | 0.244 | -0.017 | 0.004 |
| **x49** | -0.0068 | 0.008 | -0.863 | 0.388 | -0.022 | 0.009 |
| **x50** | -0.0003 | 0.006 | -0.045 | 0.964 | -0.011 | 0.011 |
| **x51** | -0.0016 | 0.006 | -0.285 | 0.775 | -0.013 | 0.009 |
| **x52** | 0.0014 | 0.006 | 0.249 | 0.803 | -0.010 | 0.012 |
| **x53** | 0.0009 | 0.006 | 0.155 | 0.877 | -0.010 | 0.012 |
| **x54** | -0.0019 | 0.006 | -0.341 | 0.733 | -0.013 | 0.009 |
| **x55** | 0.0087 | 0.008 | 1.121 | 0.263 | -0.007 | 0.024 |
| **x56** | 0.0015 | 0.007 | 0.218 | 0.828 | -0.012 | 0.015 |
| **x57** | -0.0009 | 0.006 | -0.161 | 0.872 | -0.012 | 0.010 |

R^2: 0.677 - 67% of the variance in the target variable User Ratings can be explained by the predictor features. Around 67% of the data fits on the regression model.
Adjusted R-squared value: 0.255 - This is a modified R^2 that has been adjusted by the number of predictors in the model. There are some variables here that do not appear to contribute to the model.

p_value (lists as Prob F-statistic), since this is less than 0 we can reject the null hypothesis.

This model has very similar performance to our first model which our categorical variables were Developer, Primary Genre and Secondary Genre.

**Build the Fourth model**

In [48]:
```
#Build 4th model, attempt to reduce size of condition number and increase R2

games4_clean = games_clean.drop(columns = ['Average User Rating', 'Developer', 'Secondary Genre'])

categoricals4 = ['Age Rating', 'Languages', 'Primary Genre', 'Original Year', 'Current Version Year']

#Implement dummies
games_ohe4 = pd.get_dummies(games4_clean[categoricals4], drop_first = True)
games_prepped4 = games4_clean.drop(columns = ['Age Rating', 'Languages', 'Primary Genre', 'Original Year', 'Curr
games_ohe_prepped4 = pd.concat([games_ohe4, games_prepped4], axis = 1)

games_ohe_prepped4
```

Out[48]:

| | Age Rating | Languages | Original Year | Current Version Year | Primary Genre_Business | Primary Genre_Education | Primary Genre_Entertainment | Primary Genre_Finance | Primary Genre_Food & Drink | Primary Genre_Games | ... | Ge |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 4.0 | 17 | 2008 | 2017 | 0 | 0 | 0 | 0 | 0 | 1 | ... | |
| **1** | 4.0 | 1 | 2008 | 2018 | 0 | 0 | 0 | 0 | 0 | 1 | ... | |
| **2** | 4.0 | 1 | 2008 | 2017 | 0 | 0 | 0 | 0 | 0 | 1 | ... | |
| **4** | 4.0 | 15 | 2008 | 2018 | 0 | 0 | 0 | 0 | 0 | 1 | ... | |
| **5** | 4.0 | 1 | 2008 | 2019 | 0 | 0 | 0 | 0 | 0 | 1 | ... | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **17002** | 4.0 | 1 | 2019 | 2019 | 0 | 0 | 0 | 0 | 0 | 1 | ... | |
| **17003** | 4.0 | 1 | 2019 | 2019 | 0 | 0 | 0 | 0 | 0 | 1 | ... | |
| **17004** | 4.0 | 1 | 2019 | 2019 | 0 | 0 | 0 | 0 | 0 | 1 | ... | |
| **17005** | 4.0 | 1 | 2019 | 2019 | 0 | 0 | 0 | 0 | 0 | 1 | ... | |
| **17006** | 4.0 | 2 | 2019 | 2019 | 0 | 0 | 0 | 0 | 0 | 1 | ... | |

16635 rows × 28 columns

In [49]:
```
X4 = games_ohe_prepped4
y4 = games_clean['Average User Rating']
X4_train, X4_test, y4_train, y4_test = train_test_split(X4, y4, random_state=42)
```

In [50]:
```
X4_train_normalised = scaler.fit_transform(X4_train)
X4_test_normalised = scaler.transform(X4_test)
```

In [51]:
```
X4_train_int = sm.add_constant(X4_train_normalised)
model4 = sm.OLS(y4_train, X4_train_int).fit()
```

In [52]:
```
model4.summary()
```

Out[52]:

OLS Regression Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | Average User Rating | **R-squared:** | 0.048 |
| **Model:** | OLS | **Adj. R-squared:** | 0.046 |
| **Method:** | Least Squares | **F-statistic:** | 23.43 |
| **Date:** | Mon, 26 Feb 2024 | **Prob (F-statistic):** | 6.14e-113 |
| **Time:** | 19:21:04 | **Log-Likelihood:** | -8828.8 |
| **No. Observations:** | 12476 | **AIC:** | 1.771e+04 |
| **Df Residuals:** | 12448 | **BIC:** | 1.792e+04 |
| **Df Model:** | 27 | | |
| **Covariance Type:** | nonrobust | | |

| | coef | std err | t | P>\|t\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **const** | 4.0619 | 0.004 | 922.954 | 0.000 | 4.053 | 4.070 |
| **x1** | -0.0074 | 0.005 | -1.601 | 0.109 | -0.016 | 0.002 |
| **x2** | 0.0021 | 0.004 | 0.483 | 0.629 | -0.007 | 0.011 |
| **x3** | 0.0515 | 0.006 | 9.056 | 0.000 | 0.040 | 0.063 |
| **x4** | 0.0571 | 0.006 | 9.914 | 0.000 | 0.046 | 0.068 |
| **x5** | -0.0029 | 0.006 | -0.492 | 0.623 | -0.015 | 0.009 |
| **x6** | -0.0160 | 0.024 | -0.665 | 0.506 | -0.063 | 0.031 |
| **x7** | -0.0209 | 0.022 | -0.946 | 0.344 | -0.064 | 0.022 |
| **x8** | -0.0077 | 0.008 | -1.010 | 0.312 | -0.023 | 0.007 |
| **x9** | 0.0021 | 0.005 | 0.395 | 0.693 | -0.008 | 0.013 |
| **x10** | -0.0216 | 0.040 | -0.534 | 0.593 | -0.101 | 0.058 |
| **x11** | -0.0009 | 0.005 | -0.169 | 0.865 | -0.011 | 0.010 |
| **x12** | -0.0177 | 0.006 | -2.727 | 0.006 | -0.030 | -0.005 |
| **x13** | -0.0073 | 0.005 | -1.532 | 0.126 | -0.017 | 0.002 |
| **x14** | 0.0044 | 0.005 | 0.818 | 0.413 | -0.006 | 0.015 |
| **x15** | -0.0029 | 0.005 | -0.572 | 0.567 | -0.013 | 0.007 |
| **x16** | 0.0009 | 0.006 | 0.155 | 0.877 | -0.011 | 0.013 |
| **x17** | -0.0023 | 0.007 | -0.325 | 0.745 | -0.016 | 0.011 |
| **x18** | -0.0202 | 0.009 | -2.158 | 0.031 | -0.038 | -0.002 |
| **x19** | 0.0022 | 0.005 | 0.470 | 0.638 | -0.007 | 0.012 |
| **x20** | -0.0072 | 0.006 | -1.263 | 0.207 | -0.018 | 0.004 |
| **x21** | -0.0063 | 0.013 | -0.501 | 0.616 | -0.031 | 0.018 |
| **x22** | -0.0095 | 0.009 | -1.021 | 0.307 | -0.028 | 0.009 |
| **x23** | 4.638e-18 | 1.94e-18 | 2.386 | 0.017 | 8.27e-19 | 8.45e-18 |
| **x24** | -0.0098 | 0.014 | -0.682 | 0.496 | -0.038 | 0.018 |
| **x25** | 0.0385 | 0.004 | 8.655 | 0.000 | 0.030 | 0.047 |
| **x26** | 0.0026 | 0.005 | 0.564 | 0.573 | -0.006 | 0.011 |
| **x27** | -0.0015 | 0.004 | -0.333 | 0.739 | -0.010 | 0.007 |
| **x28** | 0.0011 | 0.005 | 0.224 | 0.823 | -0.008 | 0.010 |

| | | | |
|---|---|---|---|
| **Omnibus:** | 4296.714 | **Durbin-Watson:** | 2.005 |
| **Prob(Omnibus):** | 0.000 | **Jarque-Bera (JB):** | 24363.933 |
| **Skew:** | -1.547 | **Prob(JB):** | 0.00 |
| **Kurtosis:** | 9.107 | **Cond. No.** | 1.37e+16 |

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 1.31e-28. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

Model 1 and 3 are very similar in performance, Model 2 and 4 are very similar performance so I will evaluate between Model 1 and Model 2 to see which performs better.

## Model Evaluation

*First model evaluation*

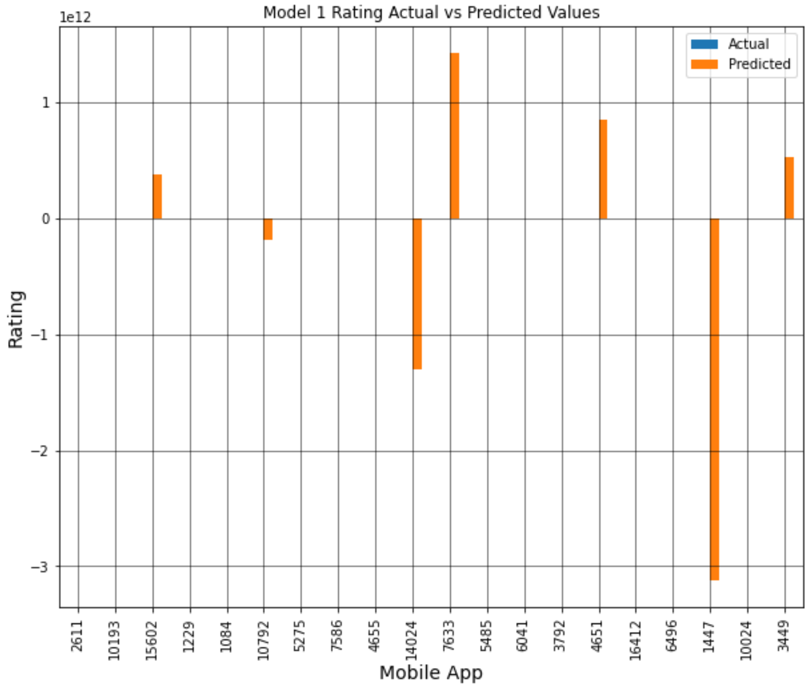
In [53]:
```
model1 = LinearRegression()
model1.fit(X_train_normalised, y_train)
```

Out[53]: LinearRegression()

In [54]:
```
y_pred_test = model1.predict(X_test_normalised)
y_pred_train = model1.predict(X_train_normalised)
accuracy = model1.score(X_test_normalised, y_test)
print('Accuracy:', accuracy)

#Actual vs Predicted Rating values
df1 = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred_test})
df2 = df1.head(20)

#Visualisation
df2.plot(kind='bar',figsize=(10,8))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='black')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='white')
plt.xlabel('Mobile App',color='black',fontsize=14)
plt.ylabel('Rating',color='black',fontsize = 14)
plt.title('Model 1 Rating Actual vs Predicted Values')
```

```
Accuracy: -7.429308859187968e+25
```

Out[54]: Text(0.5, 1.0, 'Model 1 Rating Actual vs Predicted Values')




In [55]:
```
print ('Train Mean Squared Error: '+ str(metrics.mean_squared_error(y_train,y_pred_train)))
print ('Test Mean Squared Error: '+ str(metrics.mean_squared_error(y_test,y_pred_test)))
```

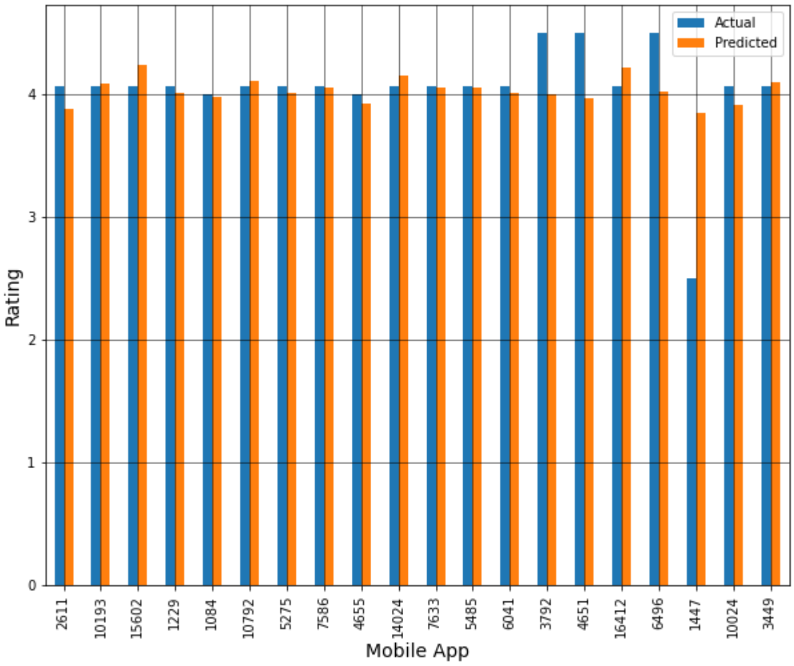
```
Train Mean Squared Error: 0.1458837018829821
Test Mean Squared Error: 1.7520400213040188e+25
```

*Second model evaluation*

In [56]:
```
model2 = LinearRegression()
model2.fit(X2_train_normalised, y2_train)

y2_pred_test = model2.predict(X2_test_normalised)
y2_pred_train = model2.predict(X2_train_normalised)
accuracy2 = model2.score(X2_test_normalised, y2_test)
print('Accuracy:', accuracy2)

#Actual vs Predicted Values
df3 = pd.DataFrame({'Actual': y2_test, 'Predicted': y2_pred_test})
df4 = df3.head(20)

#Visualisation
df4.plot(kind='bar',figsize=(10,8))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='black')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='white')
plt.xlabel('Mobile App',color='black',fontsize=14)
plt.ylabel('Rating',color='black',fontsize = 14)
```

```
Accuracy: 0.05765435489999371
```

Out[56]:
```
Text(0, 0.5, 'Rating')
```




In [57]:
```
print ('Train Mean Squared Error: '+ str(metrics.mean_squared_error(y_train, y2_pred_train)))
print ('Test Mean Squared Error: '+ str(metrics.mean_squared_error(y_test, y2_pred_test)))
```

```
Train Mean Squared Error: 0.2398071778731519
Test Mean Squared Error: 0.22223161203951125
```

Our Model 1 has a better R2 value however it has a high MSE which suggests high overfitting. Not only that, there is a large difference between the train and test MSE which also demonstrates overfitting. This model would not perform well for new data.

Our Model 2 while has lower R2 value it has a much better MSE which means we are not overfitting and will perform similarly when exposed to new data. The difference between train and test MSE are very similar which also indicates that this model is not overfitting.

**Pick Model 2 as our Final Model**

In [58]:
```
column_names = []
for column in games_ohe_prepped2:
    column_names.append(column)

coeff_df = pd.DataFrame(model2.coef_,[column_names], columns = ['Coefficients'])
sorted_df = coeff_df.sort_values(by = 'Coefficients', ascending = False)

#These coefficients tell describe the nature of the dependence of User Ratings on these coefficients
#If the coefficient is positive/negative, then the User Rating increases/decreases as the value of Rating increa:

sorted_df.head(20)
```

Out[58]:

| | Coefficients |
|---|---|
| **Current Version Year** | 0.059908 |
| **Original Year** | 0.049366 |
| **User Rating Count** | 0.038149 |
| **Secondary Genre_ Puzzle** | 0.009813 |
| **Secondary Genre_ Magazines & Newspapers** | 0.007247 |
| **Secondary Genre_ Travel** | 0.007054 |
| **Price** | 0.004515 |
| **Secondary Genre_ Casual** | 0.004197 |
| **Primary Genre_Music** | 0.003977 |
| **Secondary Genre_ Navigation** | 0.003711 |
| **Secondary Genre_ Medical** | 0.003610 |
| **Secondary Genre_ Health & Fitness** | 0.003529 |
| **Secondary Genre_ Photo & Video** | 0.003233 |
| **Secondary Genre_ Music** | 0.002900 |
| **Secondary Genre_ News** | 0.002450 |
| **Languages** | 0.002197 |
| **Primary Genre_Food & Drink** | 0.002141 |
| **Primary Genre_Shopping** | 0.001807 |
| **Secondary Genre_ Finance** | 0.001668 |
| **Size** | 0.001502 |

## Conclusions

Mobile game ratings are important to developers and companies for several reasons:

- Ratings provide direct feedback from plays about their experiences with the game.
- Games with positive ratings are more likely to be downloaded and played by users, increasing the game's active user base. Engaged users are more likely to spend time and money on in-app purchases which can contribute to the game's revenue.
- Games with higher ratings are more likely to be more visible in App Stores and improved visibility can lead to higher download numbers.
- Games with higher ratings are more likely to stand out from competitors. These ratings also contribute to the overall reputation of the developer.
- Higher-rated games are generally more successful in monetising their user base. Satisfied players are more likely to make in-app purchases leading to higher revenue generation.

In order to have a prediction for a mobile app's rating for strategy games, our top 5 most important features include the **Current Version Year, Original Year, the User Rating Count and various Secondary Genre options**.
This proposed prediction model for User Ratings will provide companies with practical recommendations and the probability of successful development of mobile games.
Knowing the priority features helps the developer understand the user's needs and trends which helps them then develop a successful application based on these needs and the potential to predict their app rating based on an assumption of Price, Age Rating, Genres and other features.

**Limitations** There was a lot of missing values that had to be accounted for. I chose to keep as much missing values as possible however depending on the business requirement, this could change how the data was used.

Data had to be transformed in order to be suitable for machine learning. This means that that this same transformation needs to be applied to both training and testing dataset otherwise this could lead to inaccuracy of the model

In [ ]:
```

```