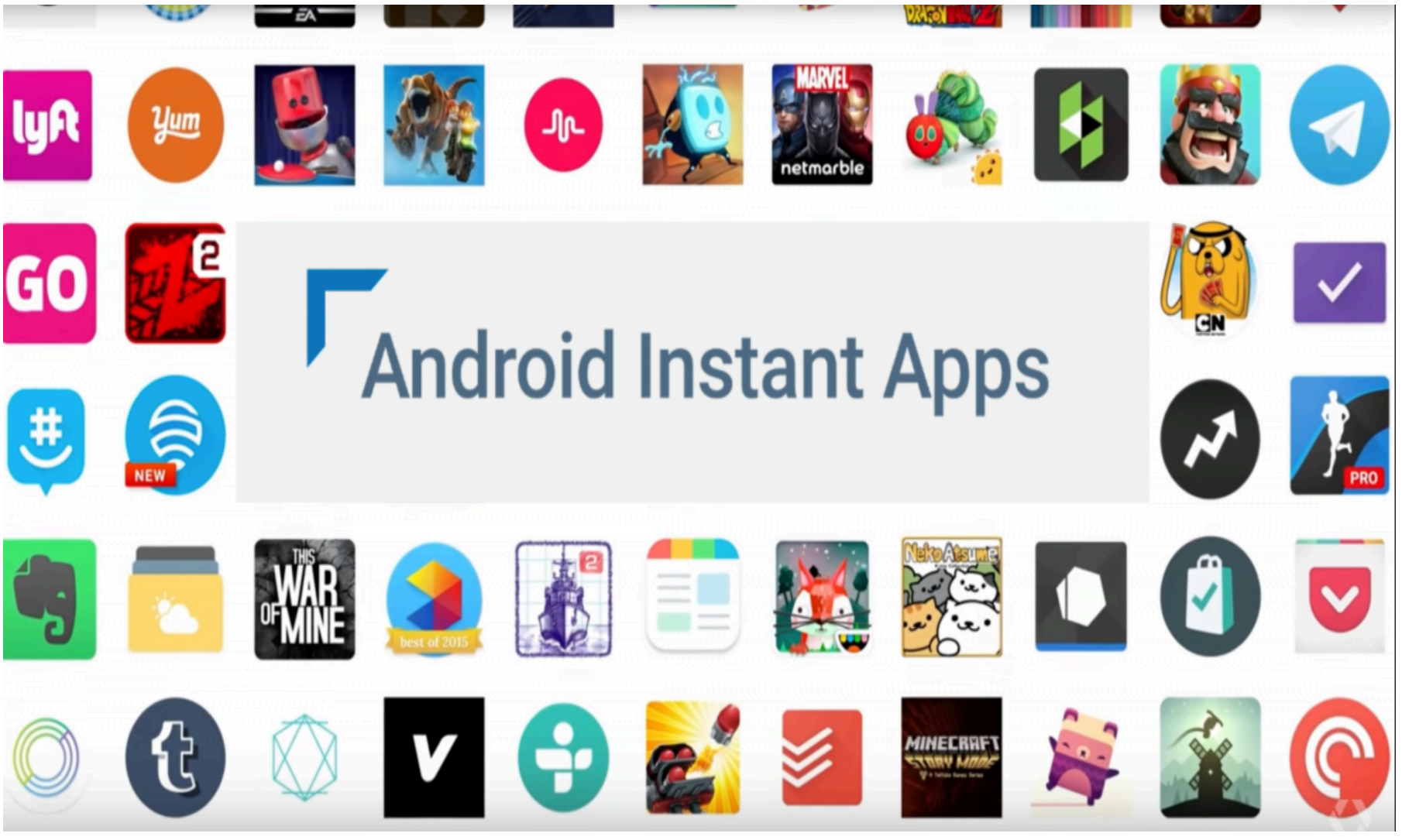
Play Store Applications Analysis

Success Prediction of Google_Play_Store_Apps



UPDATE - Fix These : (Agenda)

Issues found with the model/process used in this Jupyter Notebook:

- A K-fold Cross Validation set should have been used to find tune the model before using the test set.
- Other models could have been used during the cross validation process to determine the best binary classifier model to use.
- ROC AUC or precision/recall or jaccard Index should have been created to determine the best decision threshold for the model or measure the accuracy

Dataset Attributes:

- App : name
- Category : Category in which the app lies.
- Rating : Average rating of an app given by users
- Reviews : Users reviews
- Size : app_size
- Installs: No. of times app has been installs on particlur user id
- Type : (Free & Paid apps)
- Price : Price of an app
- Content Rating : This app is suitable for which users(for e.g 'teen' ,'everyone' ,etc.)
- Genres : Genre of app
- Last Updated : Last Updated
- Current Ver : Current version pf app
- Android Ver : Andriod version of app

Algorithms USED to Create Our Model:

- Dicision_tree_classifier (CART)
- Random_Forest_Classifier (RFC)
- Support_Vector_Machine(SVM with 'rbf' kernal trick)
- KNN (K-nearest neighbour) (Later we found best algorithm for our Data_Set)

Work_Flow:

- Data_Collection and preprocessing step
- EDA step
- Modelling step
- Results and analysis

Required Libraries :

In [530]:

```
import pandas as pd
import numpy as np
import seaborn as sns #plotting
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
import plotly.plotly as py
import plotly.graph_objs as go
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
%matplotlib inline
init_notebook_mode(connected=True)
```

Look at The Data_Set :

In [531]:

```
df_reviews = pd.read_csv("googleplaystore_user_reviews.csv")
df_reviews.head()
```

Out[531]:

	App	Translated_Review	Sentiment	Sentiment_Polarity	Sentiment_Subjectivity
0	10 Best Foods for You	I like eat delicious food. That's I'm cooking ...	Positive	1.00	0.533333
1	10 Best Foods for You	This help eating healthy exercise regular basis	Positive	0.25	0.288462
2	10 Best Foods for You	NaN	NaN	NaN	NaN
3	10 Best Foods for You	Works great especially going grocery store	Positive	0.40	0.875000
4	10 Best Foods for You	Best idea us	Positive	1.00	0.300000

In [532]:

```
df_apps = pd.read_csv("googleplaystore.csv")
df_apps.head()
```

Out[532]:

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating
0	Photo Editor & Candy Camera & Grid & ScrapBook	ART_AND_DESIGN	4.1	159	19M	10,000+	Free	0	Everyone
1	Coloring book moana	ART_AND_DESIGN	3.9	967	14M	500,000+	Free	0	Everyone
2	U Launcher Lite – FREE Live Cool Themes, Hide ...	ART_AND_DESIGN	4.7	87510	8.7M	5,000,000+	Free	0	Everyone
3	Sketch - Draw & Paint	ART_AND_DESIGN	4.5	215644	25M	50,000,000+	Free	0	Teen
4	Pixel Draw - Number Art Coloring Book	ART_AND_DESIGN	4.3	967	2.8M	100,000+	Free	0	Everyone

Data_preprocessing_Steps/EDA :

In [533]:

```
categories = list(df_apps["Category"].unique())
```

In [534]:

```
print("There are {0:.0f} categories! (Excluding/Removing Category 1.9)".format(le
```

```
There are 33 categories! (Excluding/Removing Category 1.9)
```

In [535]:

```
print(categories)
```

```
['ART_AND_DESIGN', 'AUTO_AND_VEHICLES', 'BEAUTY', 'BOOKS_AND_REFEREN
CE', 'BUSINESS', 'COMICS', 'COMMUNICATION', 'DATING', 'EDUCATION', '
ENTERTAINMENT', 'EVENTS', 'FINANCE', 'FOOD_AND_DRINK', 'HEALTH_AND_F
ITNESS', 'HOUSE_AND_HOME', 'LIBRARIES_AND_DEMO', 'LIFESTYLE', 'GAME'
, 'FAMILY', 'MEDICAL', 'SOCIAL', 'SHOPPING', 'PHOTOGRAPHY', 'SPORTS'
, 'TRAVEL_AND_LOCAL', 'TOOLS', 'PERSONALIZATION', 'PRODUCTIVITY', 'P
ARENTING', 'WEATHER', 'VIDEO_PLAYERS', 'NEWS_AND_MAGAZINES', 'MAPS_A
ND_NAVIGATION', '1.9']
```

In [536]:

```
#Remove Category 1.9
categories.remove('1.9')
```

In [537]:

```
a = df_apps.loc[df_apps["Category"] == "1.9"]
```

In [538]:

```
print(a.head())
```

```
                                                App Category  Rating Revi
ews   \
10472  Life Made WI-Fi Touchscreen Photo Frame      1.9     19.0     3
.0M

         Size Installs Type     Price Content Rating             Gen
res  \
10472  1,000+     Free    0  Everyone            NaN  February 11, 2
018

      Last Updated Current Ver Android Ver
10472       1.0.19  4.0 and up         NaN
```

In [539]:

```
a.head()
```

Out[539]:

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	G
10472	Life Made WI-Fi Touchscreen Photo Frame	1.9	19.0	3.0M	1,000+	Free	0	Everyone	NaN	Fe 11

In [540]:

```
print("This mislabeled app category affects {} app at index {}.".format(len(a),in
```

```
This mislabeled app category affects 1 app at index 10472.
```

In [541]:

```
df_apps['Rating'].isnull().sum()
```

Out[541]:

```
1474
```

In [542]:

```
df_apps = df_apps.drop(df_apps[df_apps['Rating'].isnull()].index, axis=0)
```

In [543]:

```
df_apps.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9367 entries, 0 to 10840
Data columns (total 13 columns):
App                9367 non-null object
Category           9367 non-null object
Rating             9367 non-null float64
Reviews            9367 non-null object
Size               9367 non-null object
Installs           9367 non-null object
Type               9367 non-null object
Price              9367 non-null object
Content Rating     9366 non-null object
Genres             9367 non-null object
Last Updated       9367 non-null object
Current Ver        9363 non-null object
Android Ver        9364 non-null object
dtypes: float64(1), object(12)
memory usage: 1.0+ MB
```

In [544]:

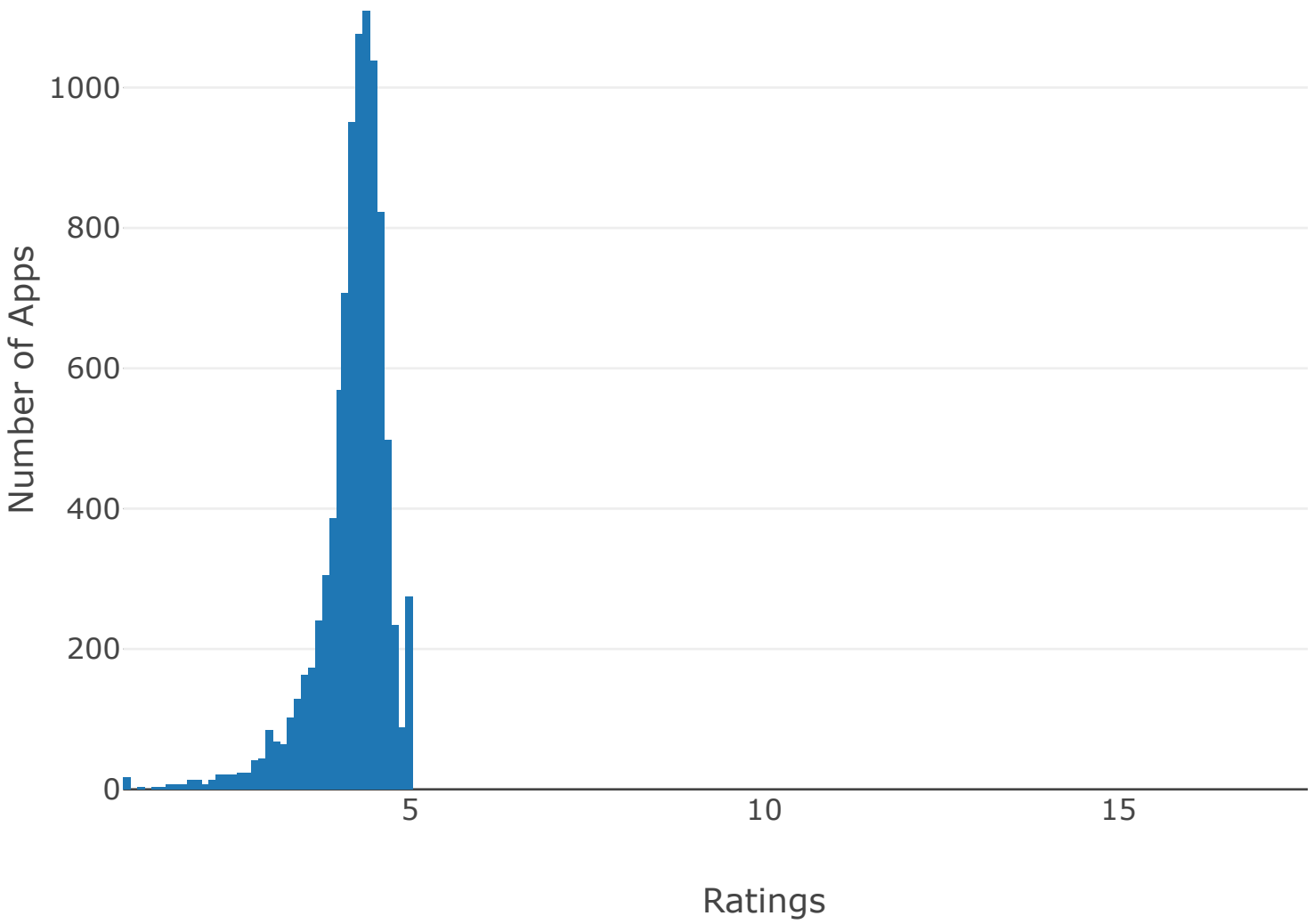
```
df_apps["Rating"].describe()
```

Out[544]:

```
count    9367.000000
mean        4.193338
std         0.537431
min         1.000000
25%         4.000000
50%         4.300000
75%         4.500000
max        19.000000
Name: Rating, dtype: float64
```

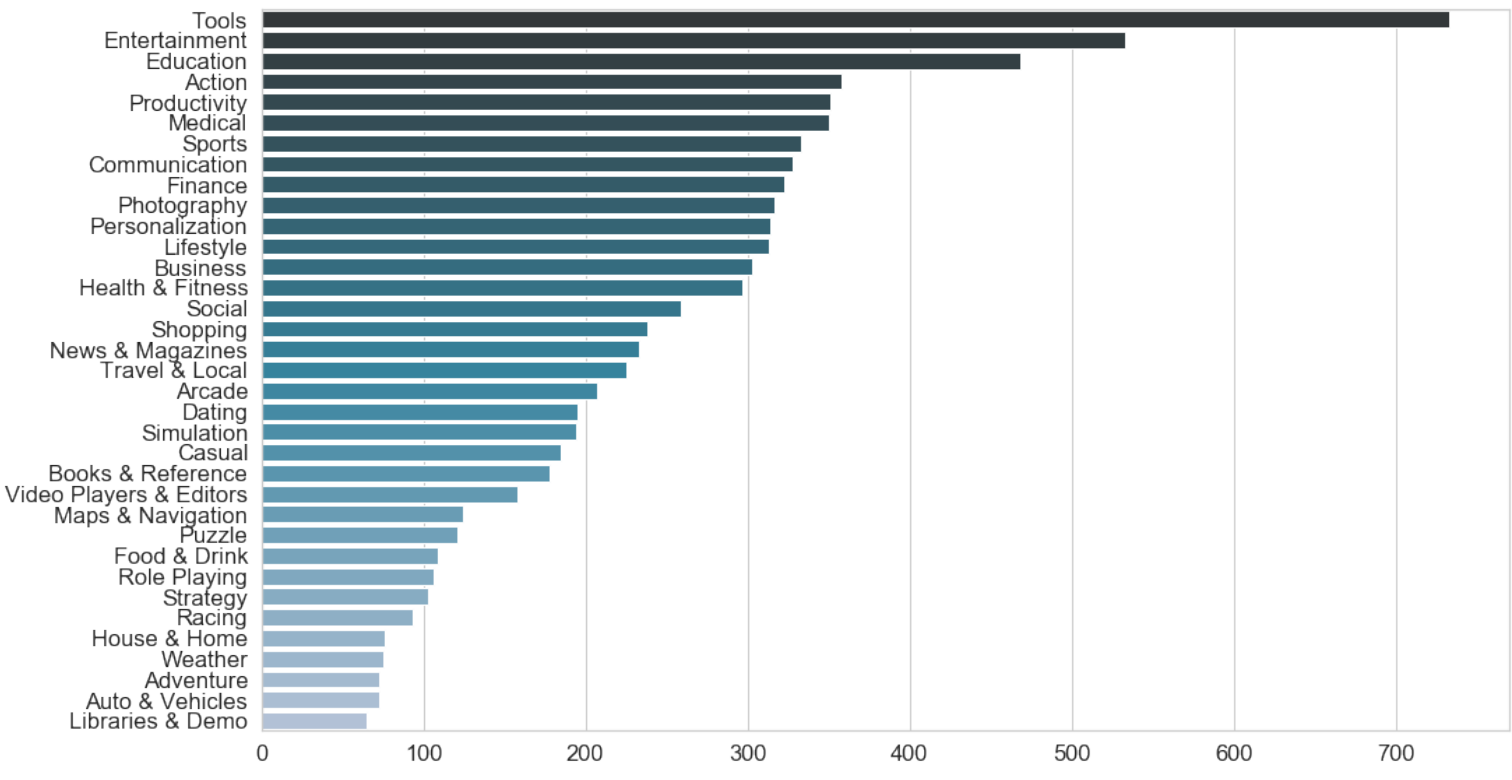
In [545]:

```
layout = go.Layout(
    xaxis=dict(title='Ratings'),yaxis=dict(title='Number of Apps'))
data = [go.Histogram(x=df_apps["Rating"])]
fig = go.Figure(data=data, layout=layout)
iplot(fig, filename='basic histogram')
```



In [546]:

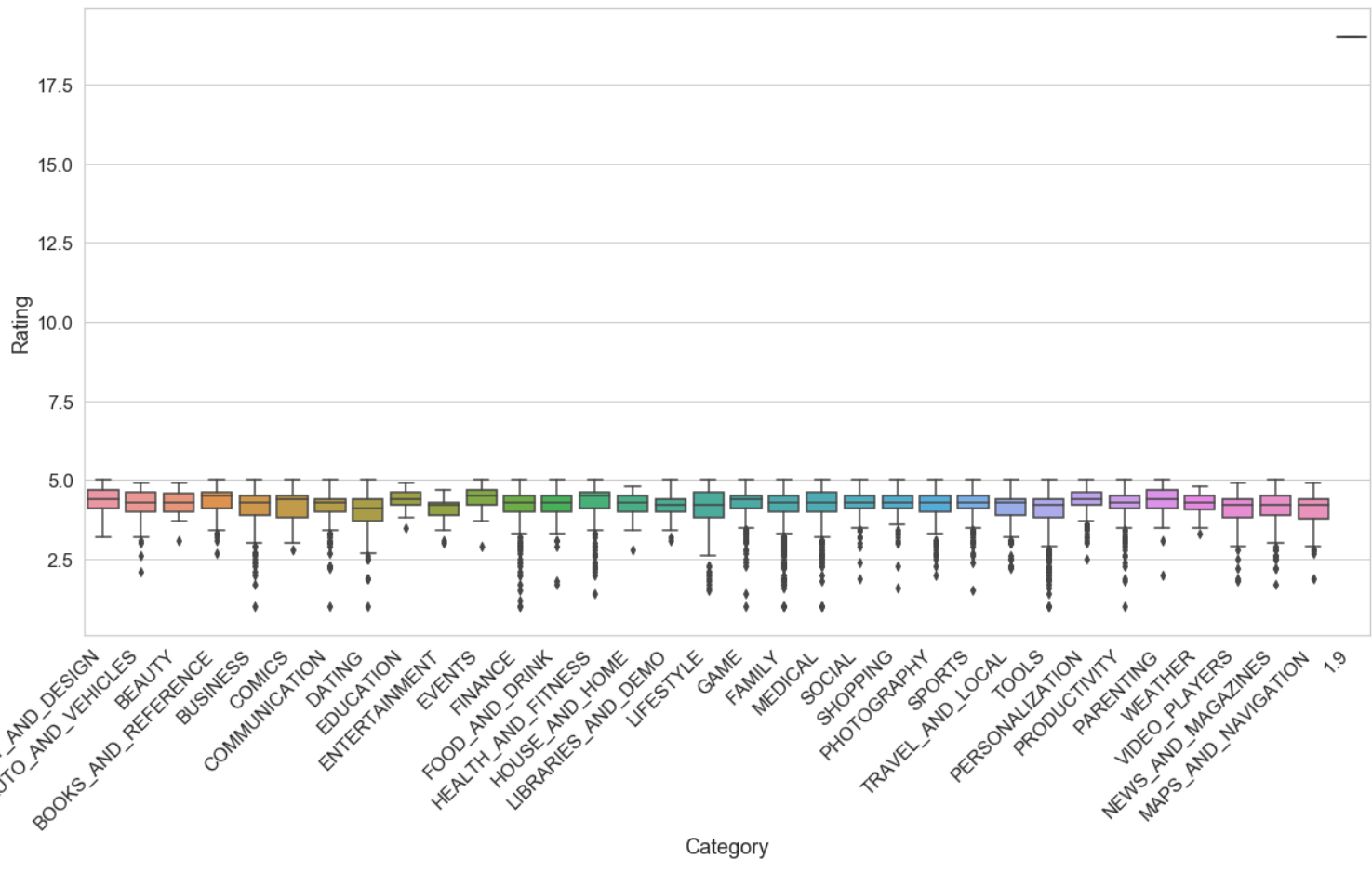
```
#Show top 35 app genres
plt.figure(figsize=(16, 9.5))
genres = df_apps["Genres"].value_counts()[:35]
ax = sns.barplot(x=genres.values, y=genres.index, palette="PuBuGn_d")
```



Which categories have the best overall rating? Also, which category had the most installs? Let's find out!

In [547]:

```
sns.set(rc={'figure.figsize':(20,10)}, font_scale=1.5, style='whitegrid')
ax = sns.boxplot(x="Category",y="Rating",data=df_apps)
labels = ax.set_xticklabels(ax.get_xticklabels(), rotation=45,ha='right')
```



All of the categories have close rating averages. In order to further define which categories are the highest rated, we will only look at the data for each category that has more than or equal to 4.0 in rating.

In [548]:

```
#Cut away rows which have < 4.0 ratings
highRating = df_apps.copy()
highRating = highRating.loc[highRating["Rating"] >= 4.0]
highRateNum = highRating.groupby('Category')['Rating'].nunique()
highRateNum
```

Out[548]:

```
Category
1.9                    1
ART_AND_DESIGN        10
AUTO_AND_VEHICLES      8
BEAUTY                10
BOOKS_AND_REFERENCE   11
BUSINESS              11
COMICS                10
COMMUNICATION         10
DATING                11
EDUCATION             10
ENTERTAINMENT          8
EVENTS                11
FAMILY                11
FINANCE               11
FOOD_AND_DRINK        10
GAME                  11
HEALTH_AND_FITNESS    11
HOUSE_AND_HOME         9
LIBRARIES_AND_DEMO     9
LIFESTYLE             11
MAPS_AND_NAVIGATION   10
MEDICAL               11
NEWS_AND_MAGAZINES    11
PARENTING             11
PERSONALIZATION       11
PHOTOGRAPHY           11
PRODUCTIVITY          11
SHOPPING              10
SOCIAL                11
SPORTS                11
TOOLS                 11
TRAVEL_AND_LOCAL      10
VIDEO_PLAYERS         10
WEATHER                9
Name: Rating, dtype: int64
```

There are many categories of apps that are equal in terms of being the highest rated. This is great, however, the interest should lie within the app categories which have the lowest number of high ratings. These poorly rated apps deserve more attention because if a new sleek new app in that category were to be put on the app store, then the developers could satisfy the demand for innovation in this area. In this case, The Categories of Importance are "AUTO_AND_VEHICLES"and "ENTERTAINMENT."

Now to analyze the apps which would produce the most ad revenue

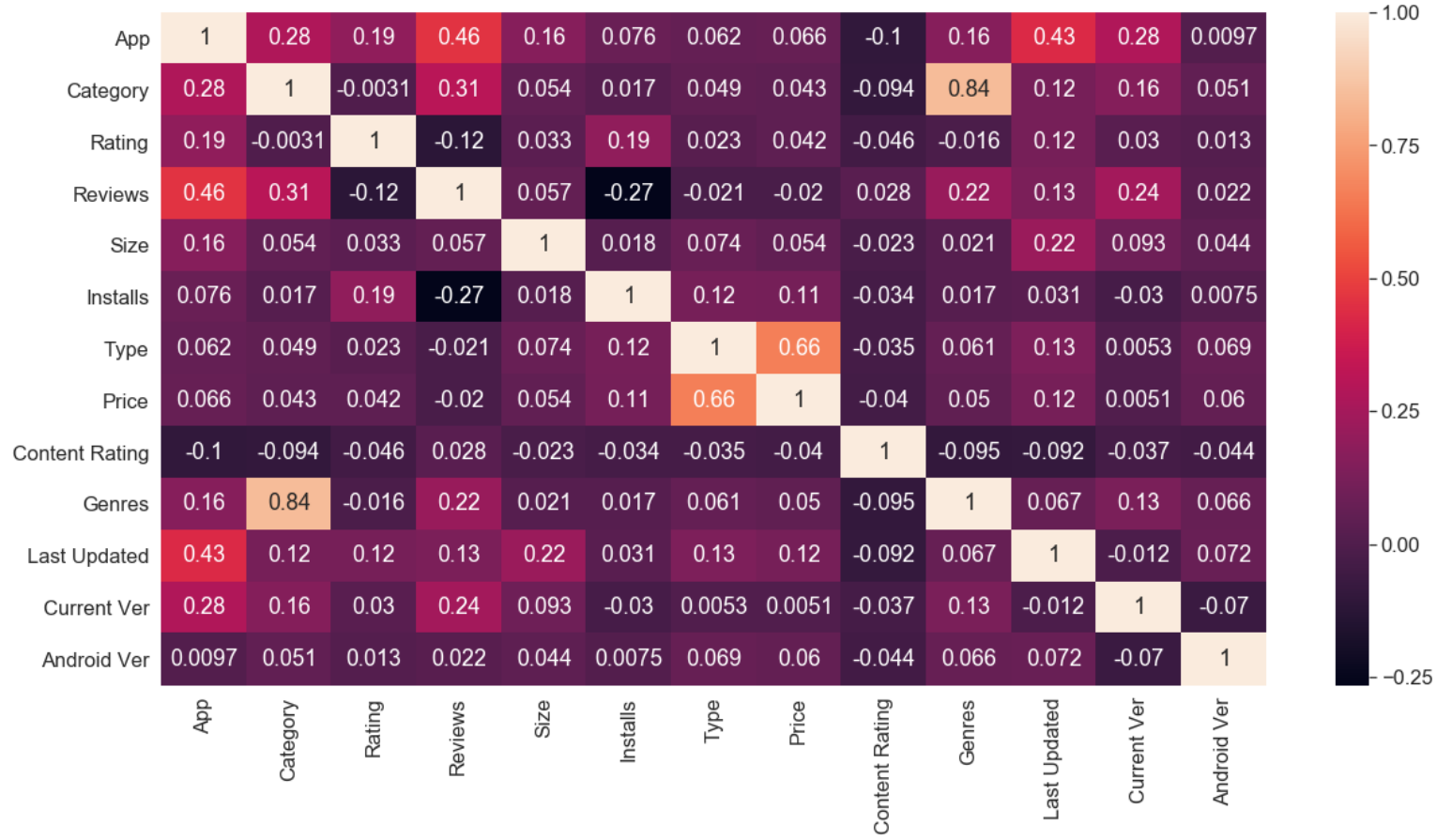
One parameter that would affect ad revenue the most is the number of installs an app has. More installs means more people are opening the app and viewing the embedded ads, hence, there is more money being made. A free application may lead to more installs, however, other parameters may alter how many installs an app will have. Let's see if there is a correlation between installs and other parameters!

In [549]:

```
df_apps.dtypes
df_apps["Type"] = (df_apps["Type"] == "Paid").astype(int)
corr = df_apps.apply(lambda x: x.factorize()[0]).corr()
sns.heatmap(corr, xticklabels=corr.columns, yticklabels=corr.columns,annot=True)
```

Out[549]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a2ee29940>
```



Above, we can see that Installs and Reviews has the strongest inverse correlation. This is reasonable because more reviews are conducted on apps that are the most popular. Since Installs was not correlated to Type, this disproves our intuition that free apps lead to more installs. Since the Installs parameter is independent and not correlated to any other parameters, we must only use Installs to show the popularity of an app. Apps with larger amounts of installs would generate the most revenue. Let's take a look at the Top 40 Apps that businesses should consider signing advertising deals with!

In [550]:

```
#Extract App, Installs, & Content Rating from df_apps
popApps = df_apps.copy()
popApps = popApps.drop_duplicates()
popApps.drop(popApps[popApps['Installs'] == 'Free'].index, inplace = True)
```

In [551]:

```
#Remove characters preventing values from being floats and integers
popApps["Installs"] = popApps["Installs"].str.replace("+","")
popApps["Installs"] = popApps["Installs"].str.replace(",","")
popApps["Installs"] = popApps["Installs"].astype("int64")
popApps["Price"] = popApps["Price"].str.replace("$","")
popApps["Price"] = popApps["Price"].astype("float64")
popApps["Size"] = popApps["Size"].str.replace("Varies with device","0")
popApps["Size"] = (popApps["Size"].replace(r'[kM]+$', '', regex=True).astype(floa
popApps["Size"].str.extract(r'[\d\.]+([kM]+)', expand=False).fillna(1).replace(['
```

In [552]:

```
popApps = popApps.sort_values(by="Installs",ascending=False)
popApps.reset_index(inplace=True)
popApps.drop(["index"],axis=1,inplace=True)
popApps.loc[:40,['App','Installs','Content Rating']]
```

Out[552]:

	App	Installs	Content Rating
0	Messenger – Text and Video Chat for Free	1000000000	Everyone
1	Google Drive	1000000000	Everyone
2	Instagram	1000000000	Teen
3	Google	1000000000	Everyone
4	Instagram	1000000000	Teen
5	Google+	1000000000	Teen
6	Subway Surfers	1000000000	Everyone 10+
7	Maps - Navigate & Explore	1000000000	Everyone
8	Google	1000000000	Everyone
9	Hangouts	1000000000	Everyone
10	Google+	1000000000	Teen
11	Google Drive	1000000000	Everyone
12	Google Play Movies & TV	1000000000	Teen
13	Google Photos	1000000000	Everyone
14	Google Street View	1000000000	Everyone
15	Subway Surfers	1000000000	Everyone 10+
16	Maps - Navigate & Explore	1000000000	Everyone

17	Subway Surfers	1000000000	Everyone 10+
18	Google Drive	1000000000	Everyone
19	Instagram	1000000000	Teen
20	Google Chrome: Fast & Secure	1000000000	Everyone
21	Subway Surfers	1000000000	Everyone 10+
22	YouTube	1000000000	Teen
23	Google Play Books	1000000000	Teen
24	Google Photos	1000000000	Everyone
25	WhatsApp Messenger	1000000000	Everyone
26	Google Photos	1000000000	Everyone
27	Facebook	1000000000	Teen
28	Google Play Games	1000000000	Teen
29	YouTube	1000000000	Teen
30	Google Photos	1000000000	Everyone
31	Facebook	1000000000	Teen
32	Google Street View	1000000000	Everyone
33	Google News	1000000000	Teen
34	Subway Surfers	1000000000	Everyone 10+
35	Messenger – Text and Video Chat for Free	1000000000	Everyone
36	Gmail	1000000000	Everyone
37	Hangouts	1000000000	Everyone
38	Gmail	1000000000	Everyone
39	Hangouts	1000000000	Everyone
40	WhatsApp Messenger	1000000000	Everyone

In order to predict if an app will be successful, we must first determine what shows success. In this case a popular app has a high install value. The way in which we will go about preprocessing the data is by binarizing the Installs column. Anything above 100,000 will be considered equal to 1, and everything below that threshold will be equal to 0. This data split is not symmetric and will cause the model to be biased when predicting popularity of an app. We will pop off the enough values of each group to make a 50-50 training set, and the rest will be used for our test set. Also, we will encode the object labels of desired features.

Label encoding to change categorical variable :

In [553]:

```
popAppsCopy = popApps.copy()
label_encoder = preprocessing.LabelEncoder()
```

In [554]:

```
# Encode labels in column 'Category'.
popAppsCopy['Category']= label_encoder.fit_transform(popAppsCopy['Category'])
popAppsCopy['Content Rating']= label_encoder.fit_transform(popAppsCopy['Content R
popAppsCopy['Genres']= label_encoder.fit_transform(popAppsCopy['Genres'])
popAppsCopy.dtypes
```

Out[554]:

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

```
Since the important data is already preprocessed into floats and integers, we
can drop the object features and build an 80/20 training/test split.
```

Creating Train_and_Test sets :

In [555]:

```
popAppsCopy.head()
```

Out[555]:

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres
0	Messenger – Text and Video Chat for Free	6	4.0	56646578	0.0	1000000000	0	0.0	1	34
1	Google Drive	25	4.4	2731171	0.0	1000000000	0	0.0	1	80
2	Instagram	27	4.5	66577313	0.0	1000000000	0	0.0	4	98
3	Google	29	4.4	8033493	0.0	1000000000	0	0.0	1	105
4	Instagram	27	4.5	66509917	0.0	1000000000	0	0.0	4	98

In [556]:

```
popAppsCopy = popAppsCopy.drop(["App","Last Updated","Current Ver","Android Ver"]
```

In [557]:

```
popAppsCopy.head()
```

Out[557]:

	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres
0	6	4.0	56646578	0.0	1000000000	0	0.0	1	34
1	25	4.4	2731171	0.0	1000000000	0	0.0	1	80
2	27	4.5	66577313	0.0	1000000000	0	0.0	4	98
3	29	4.4	8033493	0.0	1000000000	0	0.0	1	105
4	27	4.5	66509917	0.0	1000000000	0	0.0	4	98

In [558]:

```
print("There are {} total rows.".format(popAppsCopy.shape[0]))
countPop = popAppsCopy[popAppsCopy["Installs"] > 100000].count()
```

```
There are 8892 total rows.
```

In [559]:

```
print("{} Apps are Popular!".format(countPop[0]))
print("{} Apps are Unpopular!\n".format((popAppsCopy.shape[0]-countPop)[0]))
```

```
4568 Apps are Popular!
4324 Apps are Unpopular!
```

In [560]:

```
print("For an 80-20 training/test split, we need about {} apps for testing\n".for
```

```
For an 80-20 training/test split, we need about 1778.4 apps for test
ing
```

In [561]:

```
popAppsCopy["Installs"] = (popAppsCopy["Installs"] > 100000)*1 #Installs Binarize
```

In [562]:

```
popAppsCopy.head()
```

Out[562]:

	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres
0	6	4.0	56646578	0.0	1	0	0.0	1	34
1	25	4.4	2731171	0.0	1	0	0.0	1	80
2	27	4.5	66577313	0.0	1	0	0.0	4	98
3	29	4.4	8033493	0.0	1	0	0.0	1	105
4	27	4.5	66509917	0.0	1	0	0.0	4	98

In [563]:

```
print("Cut {} apps off Popular df for a total of 3558 Popular training apps.".for
print("Cut {} apps off Unpopular df for a total of 3558 Unpopular training apps.\
```

```
Cut 1010 apps off Popular df for a total of 3558 Popular training ap
ps.
Cut 766 apps off Unpopular df for a total of 3558 Unpopular training
apps.
```

In [564]:

```
testPop1 = popAppsCopy[popAppsCopy["Installs"] == 1].sample(1010,random_state=0)
```

In [565]:

```
popAppsCopy = popAppsCopy.drop(testPop1.index)
```

In [566]:

```
print("Values were not dropped from training dataframe.",testPop1.index[0] in pop
```

```
Values were not dropped from training dataframe. False
```

In [567]:

```
testPop0 = popAppsCopy[popAppsCopy["Installs"] == 0].sample(766,random_state=0)
```

In [568]:

```
popAppsCopy = popAppsCopy.drop(testPop0.index)
```

In [569]:

```
print("Values were not dropped from training dataframe.",testPop0.index[0] in pop
```

```
Values were not dropped from training dataframe. False
```

In [570]:

```
testDf = testPop1.append(testPop0)
trainDf = popAppsCopy
```

In [571]:

```
#Shuffle rows in test & training data set
testDf = testDf.sample(frac=1,random_state=0).reset_index(drop=True)
trainDf = trainDf.sample(frac=1,random_state=0).reset_index(drop=True)
```

In [572]:

```
#Form training and test data split
y_train = trainDf.pop("Installs")
X_train = trainDf.copy()
y_test = testDf.pop("Installs")
X_test = testDf.copy()
```

In [573]:

```
X_train = X_train.drop(['Reviews', 'Rating'], axis=1) #REMOVE ROW TO INCLUDE REVI
X_test = X_test.drop(['Reviews', 'Rating'], axis=1)   #REMOVE ROW TO INCLUDE REVI
```

In [574]:

```
print("{} Apps are used for Training.".format(y_train.count()))
print("{} Apps are used for Testing.".format(y_test.count()))
X_test.head(3)
```

```
7116 Apps are used for Training.
1776 Apps are used for Testing.
```

Out[574]:

	Category	Size	Type	Price	Content Rating	Genres
0	11	60000000.0	0	0.0	1	101
1	14	31000000.0	0	0.0	4	0
2	11	48000000.0	0	0.0	1	94

Fit Data in model :

The Model 1 - Modelling with Decision Tree Classifier (DTC)

In [575]:

```
popularity_classifier = DecisionTreeClassifier(max_leaf_nodes=29, random_state=0)
popularity_classifier.fit(X_train, y_train)
```

Out[575]:

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_dept
h=None,
            max_features=None, max_leaf_nodes=29,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_stat
e=0,
            splitter='best')
```

Predict on Test_Set

In [576]:

```
predictions = popularity_classifier.predict(X_test)
print("Predicted: ",predictions[:30])
print("Actual:    ",np.array(y_test[:30]))
```

```
Predicted:  [1 1 1 0 0 0 0 1 1 1 1 0 0 0 1 0 1 1 0 1 1 0 0 1 1 0 0 1
1 0]
Actual:     [1 1 0 1 0 0 0 1 1 0 1 1 1 0 1 1 1 0 0 1 1 0 0 1 0 0 1 1
1 1]
```

Measure Accuracy of Classifier

In [577]:

```
predict =popularity_classifier.predict(X_test)
print(classification_report(y_test,predict))
```

```
              precision    recall  f1-score   support

           0       0.65      0.76      0.70       766
           1       0.79      0.69      0.74      1010

   micro avg       0.72      0.72      0.72      1776
   macro avg       0.72      0.73      0.72      1776
weighted avg       0.73      0.72      0.72      1776
```

In [578]:

```
print("Train Accuracy:",popularity_classifier.score(X_train,y_train))
print("Test Accuracy:",popularity_classifier.score(X_test,y_test))
```

```
Train Accuracy: 0.7349634626194491
Test Accuracy: 0.722972972972973
```

The Model 2 - Modelling with Random Forest Classifier (R_F)

In [579]:

```
from sklearn.ensemble import  RandomForestClassifier

classifier = RandomForestClassifier(n_estimators=50, random_state=0)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
```

Now the time to Evaluate our Model (Accuracy)

In [580]:

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_sco

print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
print(accuracy_score(y_test, y_pred))
```

```
[[541 225]
 [271 739]]
              precision    recall  f1-score   support

           0       0.67      0.71      0.69       766
           1       0.77      0.73      0.75      1010

   micro avg       0.72      0.72      0.72      1776
   macro avg       0.72      0.72      0.72      1776
weighted avg       0.72      0.72      0.72      1776

0.7207207207207207
```

The Model 3 : Modelling with KNN (K - Nearest Neighbour Classification)

In [581]:

```
from sklearn.neighbors import KNeighborsClassifier
```

Training: Lets start the algorithm with k=4 for now;

In [582]:

```
k = 4
#Train Model and Predict
neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
neigh
```

Out[582]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkows
ki',
           metric_params=None, n_jobs=None, n_neighbors=4, p=2,
           weights='uniform')
```

In [583]:

```
yhat = neigh.predict(X_test)
yhat[0:5]
```

Out[583]:

```
array([1, 0, 0, 0, 0])
```

In [584]:

```
from sklearn import metrics
print("Train set Accuracy: ", metrics.accuracy_score(y_train, neigh.predict(X_tra
print("Test set Accuracy: ", metrics.accuracy_score(y_test, yhat))
```

```
Train set Accuracy:  0.7700955593029792
Test set Accuracy:  0.642454954954955
```

but this time with k=6? let see what'll happens

In [585]:

```
k = 6
#Train Model and Predict
neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
yhat = neigh.predict(X_test)
from sklearn import metrics
print("Train set Accuracy: ", metrics.accuracy_score(y_train, neigh.predict(X_tra
print("Test set Accuracy: ", metrics.accuracy_score(y_test, yhat))
```

```
Train set Accuracy:  0.7483136593591906
Test set Accuracy:  0.6413288288288288
```

* Let's check with k-folds Cross Validation to find which k is optimum for our data_set

In [586]:

```
from sklearn.model_selection import cross_val_score
```

In [587]:

```
# 10-fold cross-validation with K=5 for KNN (the n_neighbors parameter)
knn = KNeighborsClassifier(n_neighbors=5)
scores = cross_val_score(knn, X, y, cv=10, scoring='accuracy')
print(scores)
```

```
[0.74 0.68 0.63 0.66 0.66 0.66 0.6  0.6  0.65 0.58]
```

In [588]:

```
#use average accuracy as an estimate of out-of-sample accuracy
print(scores.mean())
```

```
0.6448670675739832
```

In [589]:

```
# search for an optimal value of K for KNN
k_range = list(range(1, 31))
k_scores = []
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X, y, cv=10, scoring='accuracy')
    k_scores.append(scores.mean())
print(k_scores)
```

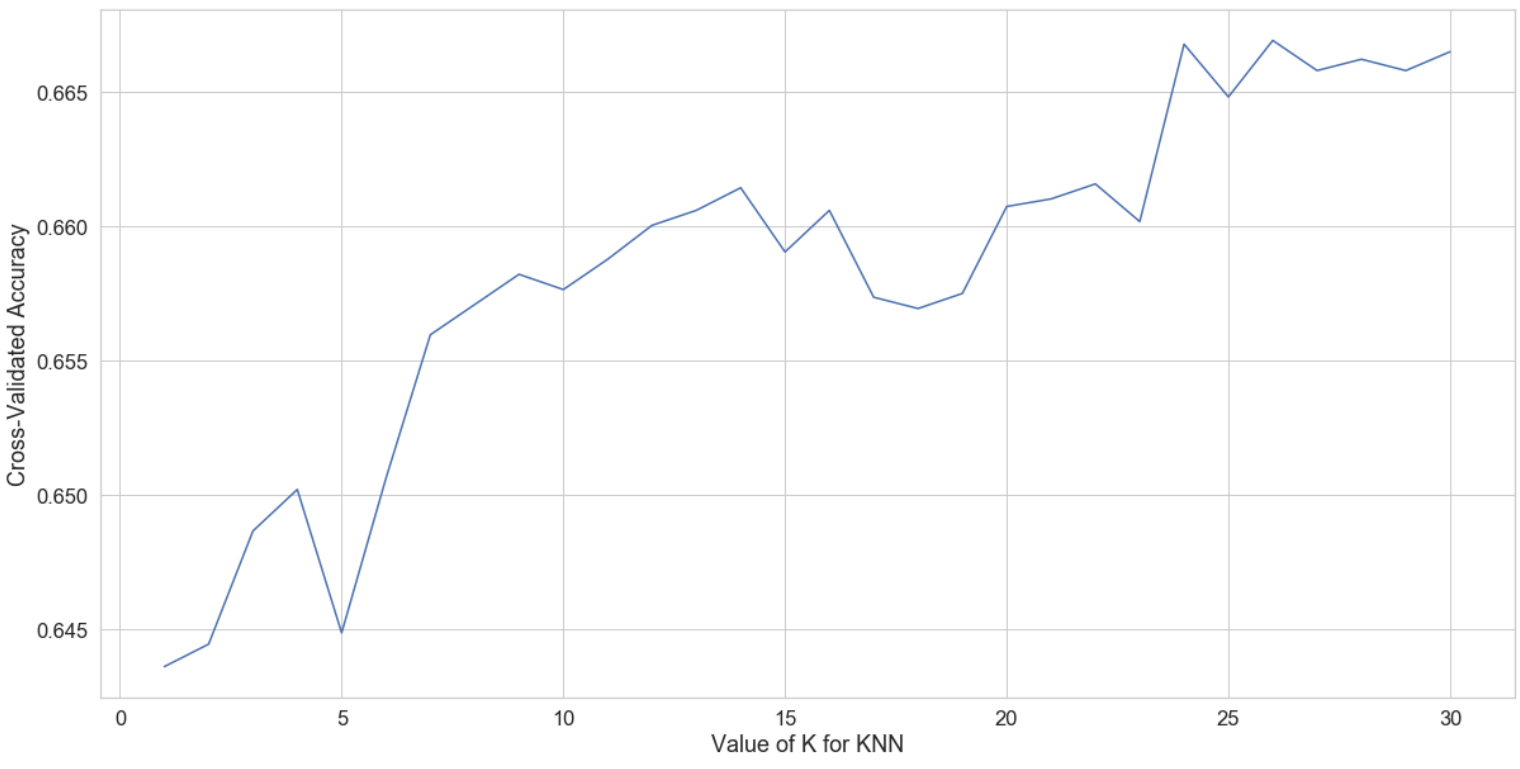
```
[0.6436089571134673, 0.6444457192593764, 0.6486599936698845, 0.65020
4541857889, 0.6448670675739832, 0.6506227251147334, 0.65596296882418
09, 0.6570881468586801, 0.6582141161576199, 0.6576463839215065, 0.65
87751226459883, 0.6600375850609272, 0.6605958221237538, 0.6614369362
240862, 0.6590473176135465, 0.6605938439626523, 0.6573607374584587,
0.6569405760405129, 0.6575035606899826, 0.6607398322519387, 0.661021
1267605633, 0.6615797594556101, 0.6601780344991296, 0.66678232315239
75, 0.6648184048108877, 0.6669227725905998, 0.6658003639816428, 0.66
622092103180 89, 0.6657991770849818, 0.666501424275993]
```

In [590]:

```
# plot the value of K for KNN (x-axis) versus the cross-validated accuracy (y-axi
plt.plot(k_range, k_scores)
plt.xlabel('Value of K for KNN')
plt.ylabel('Cross-Validated Accuracy')
```

Out[590]:

```
Text(0, 0.5, 'Cross-Validated Accuracy')
```



Model 4 : Modelling with SVM (Support_Vector_Machine)

In [591]:

```
from sklearn import svm
clf = svm.SVC(kernel='rbf')
clf.fit(X_train, y_train)
```

```
/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:196: Futu
reWarning:

The default value of gamma will change from 'auto' to 'scale' in ver
sion 0.22 to account better for unscaled features. Set gamma explici
tly to 'auto' or 'scale' to avoid this warning.
```

Out[591]:

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
```

In [592]:

```
yhat = clf.predict(X_test)
yhat [0:5]
```

Out[592]:

```
array([0, 0, 1, 0, 0])
```

evaluation

In [593]:

```
from sklearn.metrics import classification_report, confusion_matrix
import itertools
```

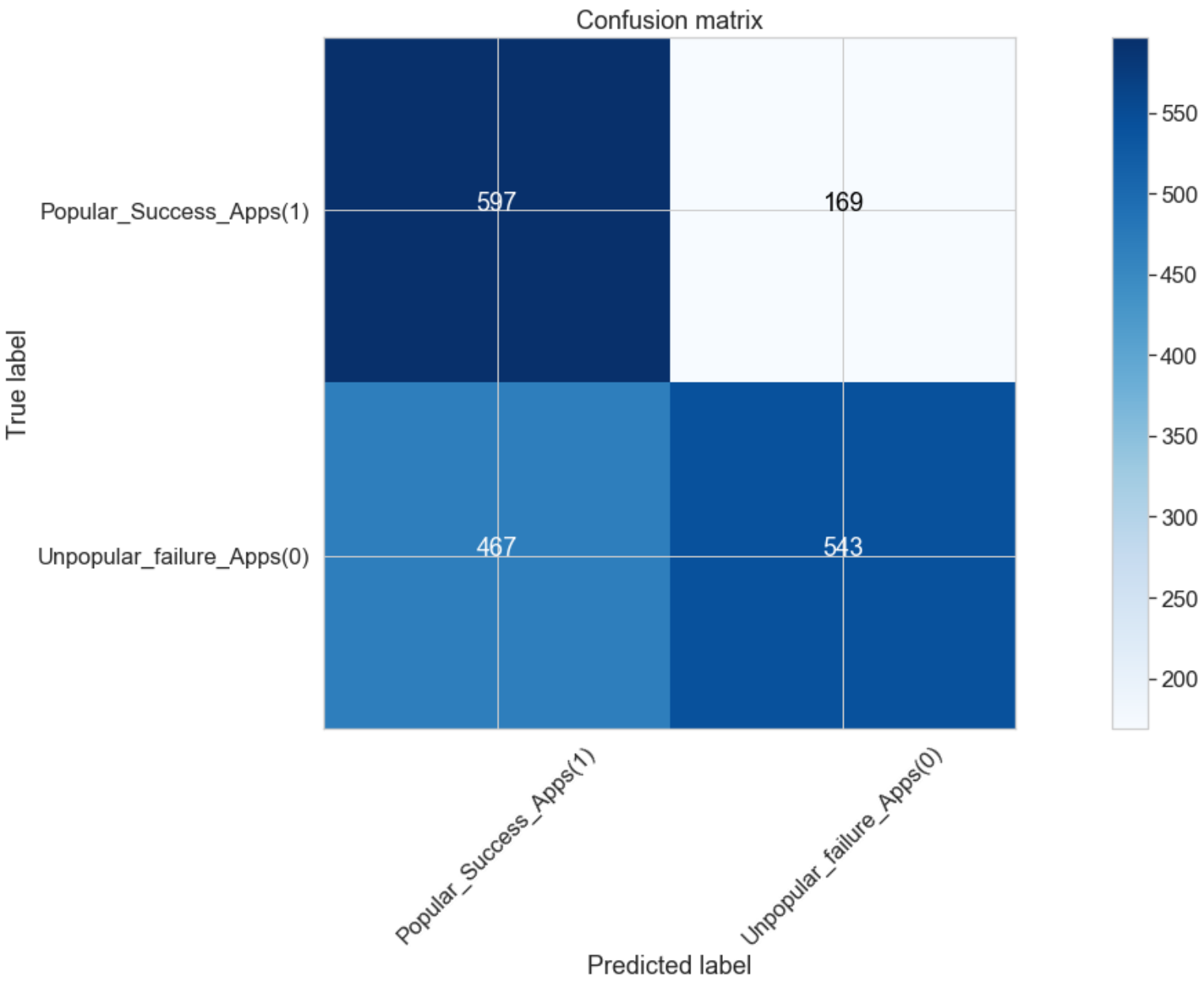
In [594]:

```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

In [595]:

```
# Compute confusion matrix
cnf_matrix = confusion_matrix(y_test, yhat, labels=[0,1])
np.set_printoptions(precision=2)

print (classification_report(y_test, yhat))

# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=['Popular_Success_Apps(1)','Unpopular_f
```

```
              precision    recall  f1-score   support

           0       0.56      0.78      0.65       766
           1       0.76      0.54      0.63      1010

   micro avg       0.64      0.64      0.64      1776
   macro avg       0.66      0.66      0.64      1776
weighted avg       0.68      0.64      0.64      1776

Confusion matrix, without normalization
[[597 169]
 [467 543]]
```



In [596]:

```
from sklearn.metrics import jaccard_similarity_score
jaccard_similarity_score(y_test, yhat)
```

Out[596]:

```
0.6418918918918919
```

```
In all of these classifier models we found, that our model Accuracy is best
suited with KNN
```

Find out what caused higher popularity

If different apps with the same app sizes are compared, we can see that the Category and the Genres columns are the only parameters that differ when determining popularity. Shown below, the 1 in the "Popular?" column may be an outlier, so as a whole, given all columns below, we can predict with ~72% accuracy the success of an app.

In [597]:

```
X_testCopy = X_test.copy()
X_testCopy["Popular?"] = y_test
X_testCopy[X_test["Size"] == 3600000].head(10)
```

Out[597]:

	Category	Size	Type	Price	Content Rating	Genres	Popular?
112	11	3600000.0	0	0.00	1	50	0
616	12	3600000.0	0	0.00	1	58	0
1297	19	3600000.0	0	0.00	1	68	1
1310	31	3600000.0	0	0.00	4	110	0
1352	23	3600000.0	1	0.99	1	78	0

When running the kernel, the Accuracy of this Decision Tree Classifier will be about 95% (IF INCLUDING REVIEWS & RATINGS). When not including the rating and reviews features, the Classifier has around 72% Accuracy. This shows that **given the Size, Type, Price, Content Rating, and Genre of an app, we can predict within 72% certainty if an app will have more than 100,000 installs and be a hit on the Google Play Store.**

Conclusion

For Innovation - Developers should focus in on apps with a category of Auto and Vehicles and Entertainment, as there are not many highly rated apps in these categories.

For Revenue - Marketers should advertise on the top 40 most installed apps list above, in order to reach the maximum viewing of their advertisements.

For Popularity - Everyone building apps should consider that the Category and Genre of an app may strongly dictate if an app will be popular or not. However, the Size, Type, Price, Content Rating, and Genre features should all be used to most accurately determine if an app will gain maximum installs.

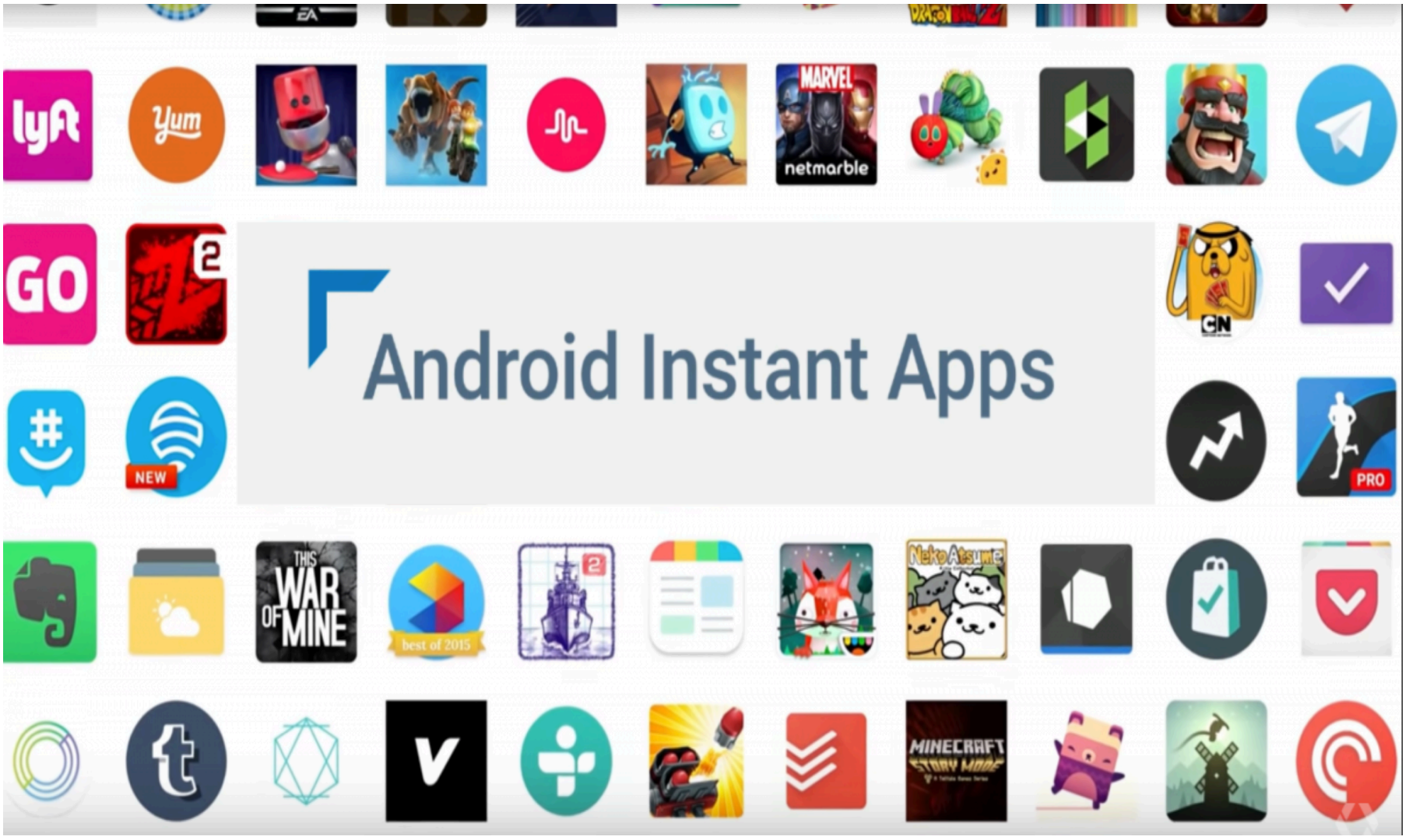
Model Limitations Better performance for predicting app success would come by using alternative ML models. Random Forests, Logistic Regression, and K Nearest-Neighbors would be great models to test against Decision Trees. More than likely, we could achieve up to 80% accuracy using one of these models rather than the current ML model.

Thank you everyone !

In [ ]:

Play Store Applications Analysis

Success Prediction of Google_Play_Store_Apps



UPDATE - Fix These : (Agenda)

- A K-fold Cross Validation set should have been used to find tune the model before using the test set.
- Other models could have been used during the cross validation process to determine the best binary classifier model to use.
- ROC AUC or precision/recall or jaccard Index should have been created to determine the best decision threshold for the model or measure the accuracy

Dataset Attributes:

- App : name
- Category : Category in which the app lies.
- Rating : Average rating of an app given by users
- Reviews : Users reviews
- Size : app_size
- Installs: No. of times app has been installs on particlur user id
- Type : (Free & Paid apps)
- Price : Price of an app
- Content Rating : This app is suitable for which users(for e.g 'teen' ,'everyone' ,etc.)
- Genres : Genre of app
- Last Updated : Last Updated
- Current Ver : Current version pf app
- Android Ver : Andriod version of app

Algorithms USED to Create Our Model:

- Dicision_tree_classifier (CART)
- Random_Forest_Classifier (RFC)
- Support_Vector_Machine(SVM with 'rbf' kernal trick)
- KNN (K-nearest neighbour) (Later we found best algorithm for our Data_Set)

Work_Flow:

- Data_Collection and preprocessing step
- EDA step
- Modelling step
- Results and analysis

Required Libraries :

In [530]:

Look at The Data_Set :

In [531]:

Out[531]:

	App	Translated_Review	Sentiment	Sentiment_Polarity	Sentiment_Subjectivity
0	10 Best Foods for You	I like eat delicious food. That's I'm cooking ...	Positive	1.00	0.533333
1	10 Best Foods for You	This help eating healthy exercise regular basis	Positive	0.25	0.288462
2	10 Best Foods for You	NaN	NaN	NaN	NaN
3	10 Best Foods for You	Works great especially going grocery store	Positive	0.40	0.875000
4	10 Best Foods for You	Best idea us	Positive	1.00	0.300000

In [532]:

Out[532]:

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating
0	Photo Editor & Candy Camera & Grid & ScrapBook	ART_AND_DESIGN	4.1	159	19M	10,000+	Free	0	Everyone
1	Coloring book moana	ART_AND_DESIGN	3.9	967	14M	500,000+	Free	0	Everyone
2	U Launcher Lite – FREE Live Cool Themes, Hide ...	ART_AND_DESIGN	4.7	87510	8.7M	5,000,000+	Free	0	Everyone
3	Sketch - Draw & Paint	ART_AND_DESIGN	4.5	215644	25M	50,000,000+	Free	0	Teen
4	Pixel Draw - Number Art Coloring Book	ART_AND_DESIGN	4.3	967	2.8M	100,000+	Free	0	Everyone

Data_preprocessing_Steps/EDA :

In [533]:

In [534]:

```
There are 33 categories! (Excluding/Removing Category 1.9)
```

In [535]:

```
['ART_AND_DESIGN', 'AUTO_AND_VEHICLES', 'BEAUTY', 'BOOKS_AND_REFEREN
CE', 'BUSINESS', 'COMICS', 'COMMUNICATION', 'DATING', 'EDUCATION', '
ENTERTAINMENT', 'EVENTS', 'FINANCE', 'FOOD_AND_DRINK', 'HEALTH_AND_F
ITNESS', 'HOUSE_AND_HOME', 'LIBRARIES_AND_DEMO', 'LIFESTYLE', 'GAME'
, 'FAMILY', 'MEDICAL', 'SOCIAL', 'SHOPPING', 'PHOTOGRAPHY', 'SPORTS'
, 'TRAVEL_AND_LOCAL', 'TOOLS', 'PERSONALIZATION', 'PRODUCTIVITY', 'P
ARENTING', 'WEATHER', 'VIDEO_PLAYERS', 'NEWS_AND_MAGAZINES', 'MAPS_A
ND_NAVIGATION', '1.9']
```

In [536]:

In [537]:

In [538]:

```
                                             App Category  Rating Revi
ews  \
10472  Life Made WI-Fi Touchscreen Photo Frame      1.9    19.0    3
.0M

         Size Installs Type     Price Content Rating             Gen
res  \
10472  1,000+     Free    0  Everyone            NaN  February 11, 2
018

      Last Updated Current Ver Android Ver
10472       1.0.19  4.0 and up         NaN
```

In [539]:

Out[539]:

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	G
10472	Life Made WI-Fi Touchscreen Photo Frame	1.9	19.0	3.0M	1,000+	Free	0	Everyone	NaN	Fe 11

In [540]:

```
This mislabeled app category affects 1 app at index 10472.
```

In [541]:

Out[541]:

```
1474
```

In [542]:

In [543]:

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9367 entries, 0 to 10840
Data columns (total 13 columns):
App               9367 non-null object
Category          9367 non-null object
Rating            9367 non-null float64
Reviews           9367 non-null object
Size              9367 non-null object
Installs          9367 non-null object
Type              9367 non-null object
Price             9367 non-null object
Content Rating    9366 non-null object
Genres            9367 non-null object
Last Updated      9367 non-null object
Current Ver       9363 non-null object
Android Ver       9364 non-null object
dtypes: float64(1), object(12)
memory usage: 1.0+ MB
```

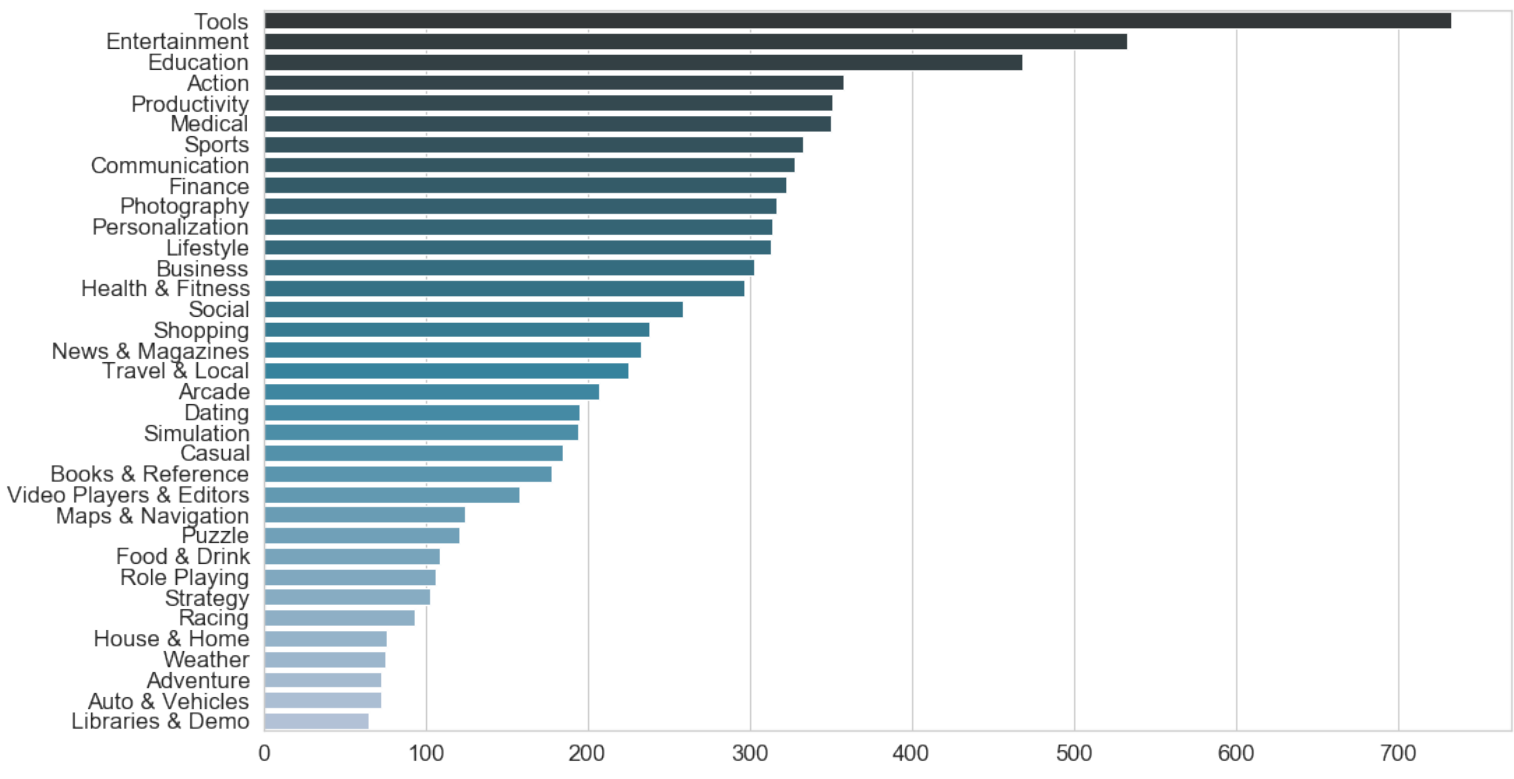
In [544]:

Out[544]:

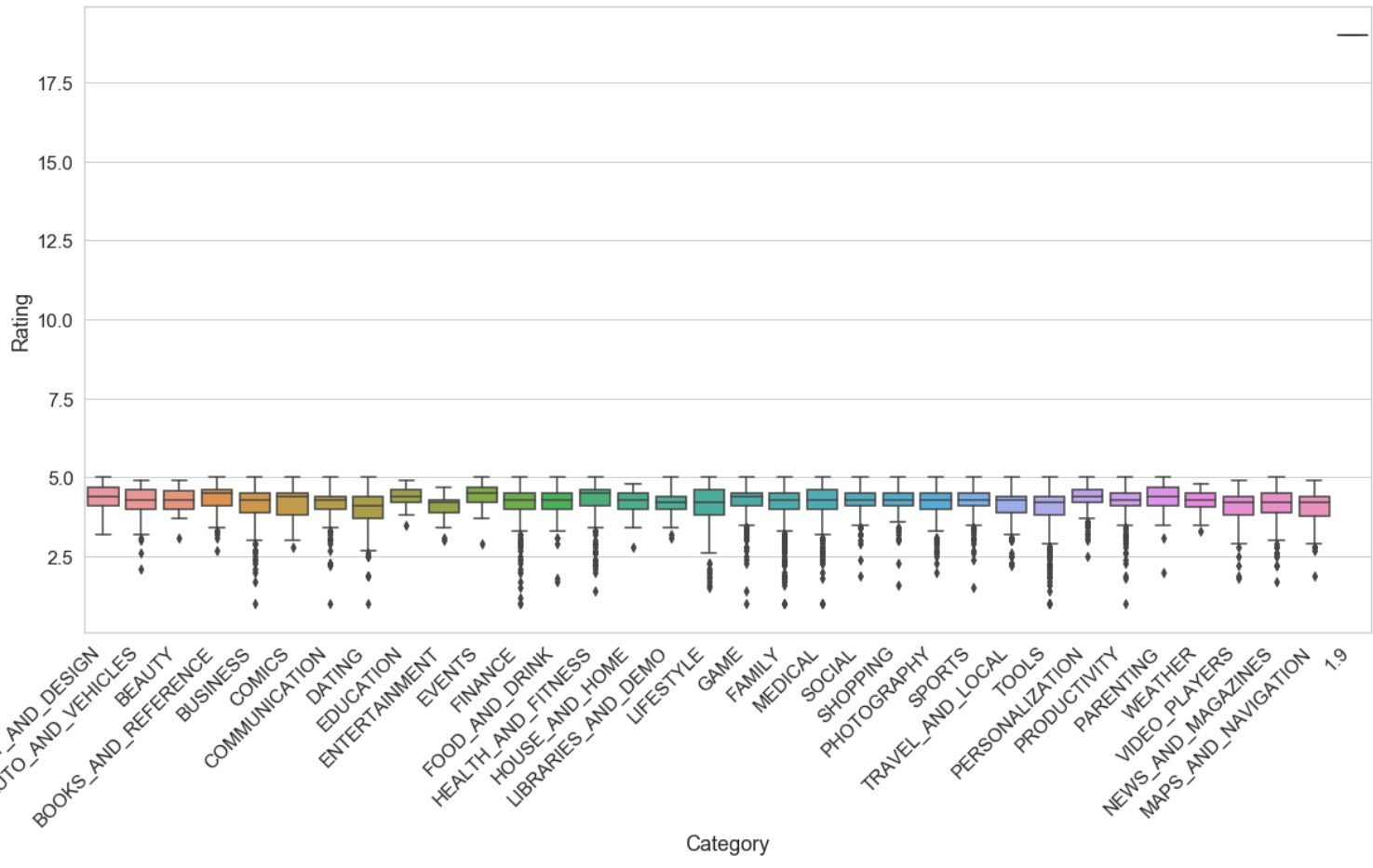
```
count    9367.000000
mean        4.193338
std         0.537431
min         1.000000
25%         4.000000
50%         4.300000
75%         4.500000
max        19.000000
Name: Rating, dtype: float64
```

In [545]:

In [546]:


Tools
Entertainment
Education
Action
Productivity
Medical
Sports
Communication
Finance
Photography
Personalization
Lifestyle
Business
Health & Fitness
Social
Shopping
News & Magazines
Travel & Local
Arcade
Dating
Simulation
Casual
Books & Reference
Video Players & Editors
Maps & Navigation
Puzzle
Food & Drink
Role Playing
Strategy
Racing
House & Home
Weather
Adventure
Auto & Vehicles
Libraries & Demo
0
100
200
300
400
500
600
700

In [547]:



All of the categories have close rating averages. In order to further define which categories are the highest rated, we will only look at the data for each category that has more than or equal to 4.0 in rating.

In [548]:

Out[548]:

```
Category
1.9                    1
ART_AND_DESIGN        10
AUTO_AND_VEHICLES      8
BEAUTY                10
BOOKS_AND_REFERENCE   11
BUSINESS              11
COMICS                10
COMMUNICATION         10
DATING                11
EDUCATION             10
ENTERTAINMENT          8
EVENTS                11
FAMILY                11
FINANCE               11
FOOD_AND_DRINK        10
GAME                  11
HEALTH AND FITNESS    11
```

There are many categories of apps that are equal in terms of being the highest rated. This is great, however, the interest should lie within the app categories which have the lowest number of high ratings. These poorly rated apps deserve more attention because if a new sleek new app in that category were to be put on the app store, then the developers could satisfy the demand for innovation in this area. In this case, The Categories of Importance are "AUTO_AND_VEHICLES"and "ENTERTAINMENT."

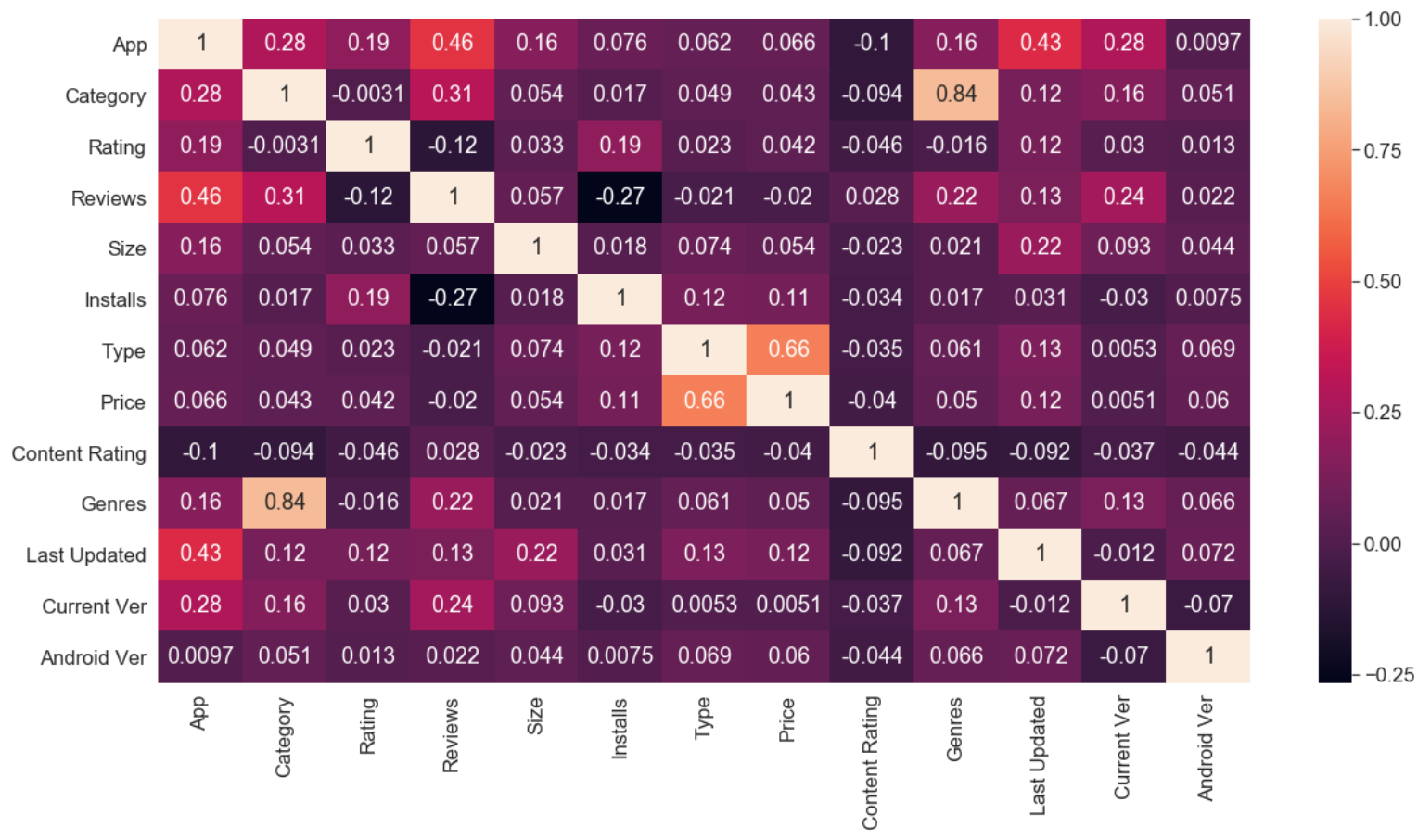
Now to analyze the apps which would produce the most ad revenue

One parameter that would affect ad revenue the most is the number of installs an app has. More installs means more people are opening the app and viewing the embedded ads, hence, there is more money being made. A free application may lead to more installs, however, other parameters may alter how many installs an app will have. Let's see if there is a correlation between installs and other parameters!

In [549]:

Out[549]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a2ee29940>
```



Above, we can see that Installs and Reviews has the strongest inverse correlation. This is reasonable because more reviews are conducted on apps that are the most popular. Since Installs was not correlated to Type, this disproves our intuition that free apps lead to more installs. Since the Installs parameter is independent and not correlated to any other parameters, we must only use Installs to show the popularity of an app. Apps with larger amounts of installs would generate the most revenue. Let's take a look at the Top 40 Apps that businesses should consider signing advertising deals with!

In [550]:

In [551]:

In [552]:

Out[552]:

	App	Installs	Content Rating
0	Messenger – Text and Video Chat for Free	1000000000	Everyone
1	Google Drive	1000000000	Everyone
2	Instagram	1000000000	Teen
3	Google	1000000000	Everyone
4	Instagram	1000000000	Teen
5	Google+	1000000000	Teen
6	Subway Surfers	1000000000	Everyone 10+
7	Maps - Navigate & Explore	1000000000	Everyone
8	Google	1000000000	Everyone
9	Hangouts	1000000000	Everyone
10	Google+	1000000000	Teen
11	Google Drive	1000000000	Everyone
12	Google Play Movies & TV	1000000000	Teen
13	Google Photos	1000000000	Everyone
14	Google Street View	1000000000	Everyone
15	Subway Surfers	1000000000	Everyone 10+
16	Maps - Navigate & Explore	1000000000	Everyone
17	Subway Surfers	1000000000	Everyone 10+
18	Google Drive	1000000000	Everyone
19	Instagram	1000000000	Teen
20	Google Chrome: Fast & Secure	1000000000	Everyone
21	Subway Surfers	1000000000	Everyone 10+
22	YouTube	1000000000	Teen
23	Google Play Books	1000000000	Teen
24	Google Photos	1000000000	Everyone
25	WhatsApp Messenger	1000000000	Everyone
26	Google Photos	1000000000	Everyone
27	Facebook	1000000000	Teen
28	Google Play Games	1000000000	Teen

29	YouTube	1000000000	Teen
30	Google Photos	1000000000	Everyone
31	Facebook	1000000000	Teen
32	Google Street View	1000000000	Everyone
33	Google News	1000000000	Teen
34	Subway Surfers	1000000000	Everyone 10+
35	Messenger – Text and Video Chat for Free	1000000000	Everyone
36	Gmail	1000000000	Everyone
37	Hangouts	1000000000	Everyone
38	Gmail	1000000000	Everyone
39	Hangouts	1000000000	Everyone
40	WhatsApp Messenger	1000000000	Everyone

In order to predict if an app will be successful, we must first determine what shows success. In this case a popular app has a high install value. The way in which we will go about preprocessing the data is by binarizing the Installs column. Anything above 100,000 will be considered equal to 1, and everything below that threshold will be equal to 0. This data split is not symmetric and will cause the model to be biased when predicting popularity of an app. We will pop off the enough values of each group to make a 50-50 training set, and the rest will be used for our test set. Also, we will encode the object labels of desired features.

Label encoding to change categorical variable

In [553]:

In [554]:

Out[554]:

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

Creating Train_and_Test sets :

In [555]:

Out[555]:

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres
0	Messenger – Text and Video Chat for Free	6	4.0	56646578	0.0	1000000000	0	0.0	1	34
1	Google Drive	25	4.4	2731171	0.0	1000000000	0	0.0	1	80
2	Instagram	27	4.5	66577313	0.0	1000000000	0	0.0	4	98
3	Google	29	4.4	8033493	0.0	1000000000	0	0.0	1	105
4	Instagram	27	4.5	66509917	0.0	1000000000	0	0.0	4	98

In [556]:

In [557]:

Out[557]:

	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres
0	6	4.0	56646578	0.0	1000000000	0	0.0	1	34
1	25	4.4	2731171	0.0	1000000000	0	0.0	1	80
2	27	4.5	66577313	0.0	1000000000	0	0.0	4	98
3	29	4.4	8033493	0.0	1000000000	0	0.0	1	105
4	27	4.5	66509917	0.0	1000000000	0	0.0	4	98

In [558]:

```
There are 8892 total rows.
```

In [559]:

```
4568 Apps are Popular!
4324 Apps are Unpopular!
```

In [560]:

```
For an 80-20 training/test split, we need about 1778.4 apps for test
ing
```

In [561]:

In [562]:

Out[562]:

	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres
0	6	4.0	56646578	0.0	1	0	0.0	1	34
1	25	4.4	2731171	0.0	1	0	0.0	1	80
2	27	4.5	66577313	0.0	1	0	0.0	4	98
3	29	4.4	8033493	0.0	1	0	0.0	1	105
4	27	4.5	66509917	0.0	1	0	0.0	4	98

In [563]:

```
Cut 1010 apps off Popular df for a total of 3558 Popular training ap
ps.
Cut 766 apps off Unpopular df for a total of 3558 Unpopular training
apps.
```

In [564]:

In [565]:

In [566]:

```
Values were not dropped from training dataframe. False
```

In [567]:

In [568]:

In [569]:

```
Values were not dropped from training dataframe. False
```

In [570]:

In [571]:

In [572]:

In [573]:

In [574]:

```
7116 Apps are used for Training.
1776 Apps are used for Testing.
```

Out[574]:

	Category	Size	Type	Price	Content Rating	Genres
0	11	60000000.0	0	0.0	1	101
1	14	31000000.0	0	0.0	4	0
2	11	48000000.0	0	0.0	1	94

Fit Data in model :

The Model 1 - Modelling with Decision Tree Classifier (DTC)

In [575]:

Out[575]:

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_dept
h=None,
            max_features=None, max_leaf_nodes=29,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_stat
e=0,
            splitter='best')
```

Predict on Test_Set

In [576]:

```
Predicted:  [1 1 1 0 0 0 0 1 1 1 1 0 0 0 1 0 1 1 0 1 1 0 0 1 1 0 0 1
1 0]
Actual:     [1 1 0 1 0 0 0 1 1 0 1 1 1 0 1 1 1 0 0 1 1 0 0 1 0 0 1 1
1 1]
```

Measure Accuracy of Classifier

In [577]:

```
              precision    recall  f1-score   support

           0       0.65      0.76      0.70       766
           1       0.79      0.69      0.74      1010

   micro avg       0.72      0.72      0.72      1776
   macro avg       0.72      0.73      0.72      1776
weighted avg       0.73      0.72      0.72      1776
```

In [578]:

```
Train Accuracy: 0.7349634626194491
Test Accuracy: 0.722972972972973
```

The Model 2 - Modelling with Random Forest Classifier (R_F)

In [579]:

Now the time to Evaluate our Model (Accuracy)

In [580]:

```
[[541 225]
 [271 739]]
              precision    recall  f1-score   support

           0       0.67      0.71      0.69       766
           1       0.77      0.73      0.75      1010

   micro avg       0.72      0.72      0.72      1776
   macro avg       0.72      0.72      0.72      1776
weighted avg       0.72      0.72      0.72      1776

0.7207207207207207
```

The Model 3 : Modelling with KNN (K - Nearest Neighbour Classification)

In [581]:

Training: Lets start the algorithm with k=4 for now;

In [582]:

Out[582]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkows
ki',
           metric_params=None, n_jobs=None, n_neighbors=4, p=2,
           weights='uniform')
```

In [583]:

Out[583]:

```
array([1, 0, 0, 0, 0])
```

In [584]:

```
Train set Accuracy:  0.7700955593029792
Test set Accuracy:  0.642454954954955
```

In [585]:

```
Train set Accuracy:  0.7483136593591906
Test set Accuracy:  0.6413288288288288
```

* Let's check with k-folds Cross Validation to find which k is optimum for our data_set

In [586]:

In [587]:

```
[0.74 0.68 0.63 0.66 0.66 0.66 0.6  0.6  0.65 0.58]
```

In [588]:

```
0.6448670675739832
```

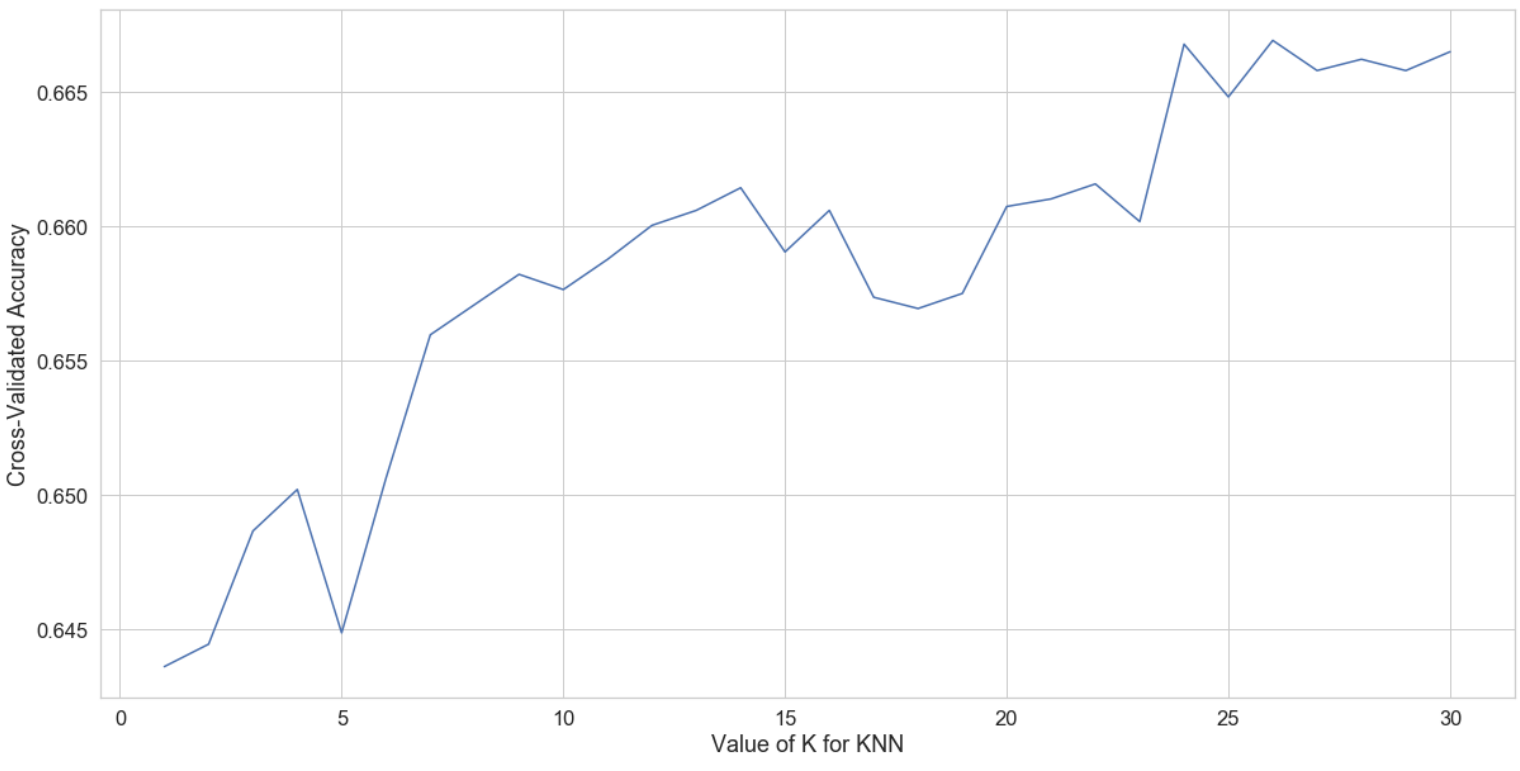
In [589]:

```
[0.6436089571134673, 0.6444457192593764, 0.6486599936698845, 0.65020
4541857889, 0.6448670675739832, 0.6506227251147334, 0.65596296882418
09, 0.6570881468586801, 0.6582141161576199, 0.6576463839215065, 0.65
87751226459883, 0.6600375850609272, 0.6605958221237538, 0.6614369362
240862, 0.6590473176135465, 0.6605938439626523, 0.6573607374584587,
0.6569405760405129, 0.6575035606899826, 0.6607398322519387, 0.661021
1267605633, 0.6615797594556101, 0.6601780344991296, 0.66678232315239
75, 0.6648184048108877, 0.6669227725905998, 0.6658003639816428, 0.66
622209210318089, 0.6657991770849818, 0.666501424275993]
```

In [590]:

Out[590]:

```
Text(0, 0.5, 'Cross-Validated Accuracy')
```



Model 4 : Modelling with SVM (Support_Vector_Machine)

In [591]:

```
/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:196: Futu
reWarning:

The default value of gamma will change from 'auto' to 'scale' in ver
sion 0.22 to account better for unscaled features. Set gamma explici
tly to 'auto' or 'scale' to avoid this warning.
```

Out[591]:

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
```

In [592]:

Out[592]:

```
array([0, 0, 1, 0, 0])
```

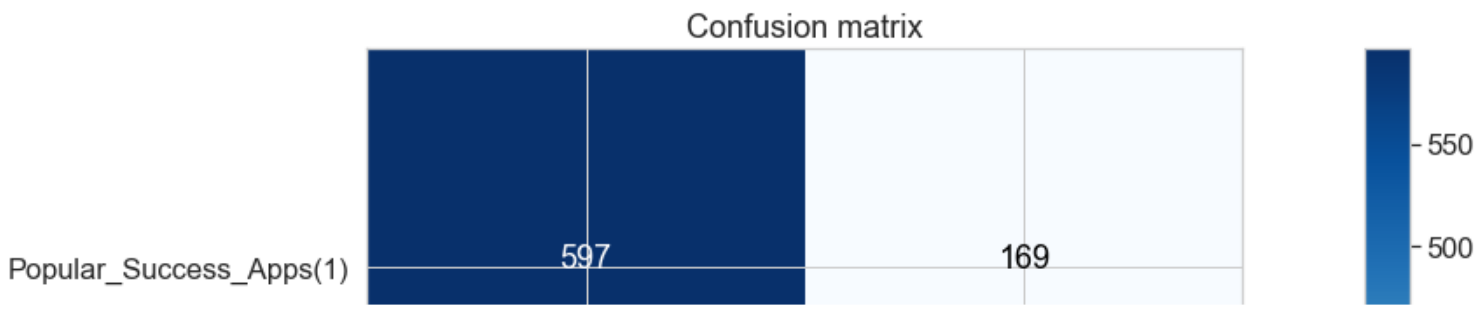
evaluation

In [593]:

In [594]:

In [595]:

```
              precision    recall  f1-score   support

           0       0.56      0.78      0.65       766
           1       0.76      0.54      0.63      1010

   micro avg       0.64      0.64      0.64      1776
   macro avg       0.66      0.66      0.64      1776
weighted avg       0.68      0.64      0.64      1776

Confusion matrix, without normalization
[[597 169]
 [467 543]]
```



In [596]:

Out[596]:

```
0.6418918918918919
```

Find out what caused higher popularity

If different apps with the same app sizes are compared, we can see that the Category and the Genres columns are the only parameters that differ when determining popularity. Shown below, the 1 in the "Popular?" column may be an outlier, so as a whole, given all columns below, we can predict with ~72% accuracy the success of an app.

In [597]:

Out[597]:

	Category	Size	Type	Price	Content Rating	Genres	Popular?
112	11	3600000.0	0	0.00	1	50	0
616	12	3600000.0	0	0.00	1	58	0
1297	19	3600000.0	0	0.00	1	68	1
1310	31	3600000.0	0	0.00	4	110	0
1352	23	3600000.0	1	0.99	1	78	0

When running the kernel, the Accuracy of this Decision Tree Classifier will be about 95% (IF INCLUDING REVIEWS & RATINGS). When not including the rating and reviews features, the Classifier has around 72% Accuracy. This shows that **given the Size, Type, Price, Content Rating, and Genre of an app, we can predict within 72% certainty if an app will have more than 100,000 installs and be a hit on the Google Play Store.**

Conclusion

For Innovation - Developers should focus in on apps with a category of Auto and Vehicles and Entertainment, as there are not many highly rated apps in these categories.

For Revenue - Marketers should advertise on the top 40 most installed apps list above, in order to reach the maximum viewing of their advertisements.

For Popularity - Everyone building apps should consider that the Category and Genre of an app may strongly dictate if an app will be popular or not. However, the Size, Type, Price, Content Rating, and Genre features should all be used to most accurately determine if an app will gain maximum installs.

Model Limitations Better performance for predicting app success would come by using alternative ML models. Random Forests, Logistic Regression, and K Nearest-Neighbors would be great models to test against Decision Trees. More than likely, we could achieve up to 80% accuracy using one of these models rather than the current ML model.

Thank you everyone !

In [ ]: