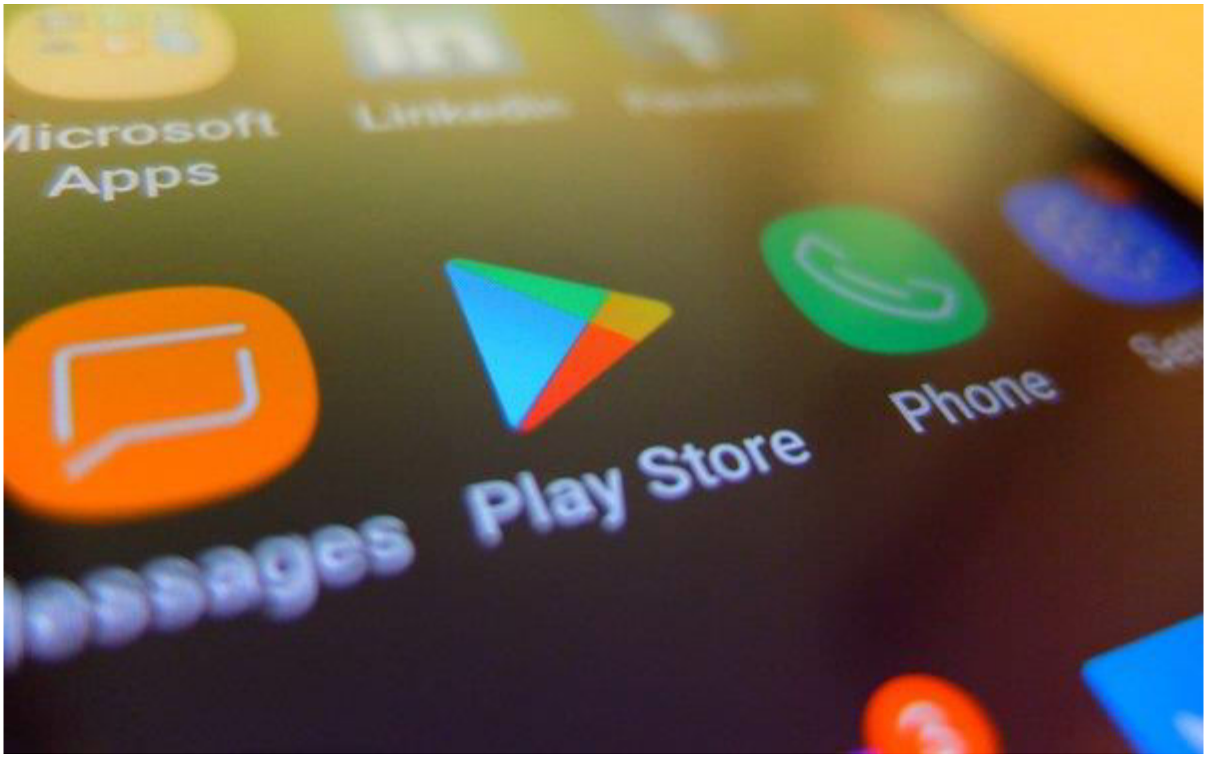
Google Play Store Apps Report



Data Analyst: Chaimaa Driouech
GitHub:
Date: 01/04/2020

Summary

OVERVIEW

While many public datasets (on Kaggle and the like) provide Apple App Store data, there are not many counterpart datasets available for Google Play Store apps anywhere on the web. On digging deeper, I found out that iTunes App Store page deploys a nicely indexed appendix-like structure to allow for simple and easy web scraping. On the other hand, Google Play Store uses sophisticated modern-day techniques (like dynamic page load) using JQuery making scraping more challenging.

Acknowledgements

This information is scraped from the Google Play Store. This app information would not be available without it.

Inspiration

The Play Store apps data has enormous potential to drive app-making businesses to success. Actionable insights can be drawn for developers to work on and capture the Android market!

Tasks

- GitHub
- Python and all libraries needed to solve the problem
- Exploratory Data Analysis
- Data collection, pre-processing and feature engineering
- Data Visualization
- Data science process: Best practices
- Predictive Modeling and Evaluation (the whole process)
- Model selection
- Cross validation

PYTHON AND ALL IMPORTED LIBRARIES

```
#imports
#numpy,pandas,scipy, math, matplotlib
import numpy as np
import pandas as pd
import scipy
from math import sqrt
import seaborn as sns
import matplotlib.pyplot as plt

#estimators Regression
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn import linear_model

#estimators Classification
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVR
from sklearn import svm
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler, LabelEncoder

#model metrics Regression
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.model_selection import cross_val_score
#model metrics Classification
from sklearn.metrics import confusion_matrix, classification_report

#cross validation
from sklearn.model_selection import train_test_split

import warnings
warnings.filterwarnings('ignore')
pd.options.display.max_columns = None

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.feature_selection import RFECV

%matplotlib inline
```

EXPLORATORY DATA ANALYSIS

df_apps.dtypes.index

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

df_apps.info()

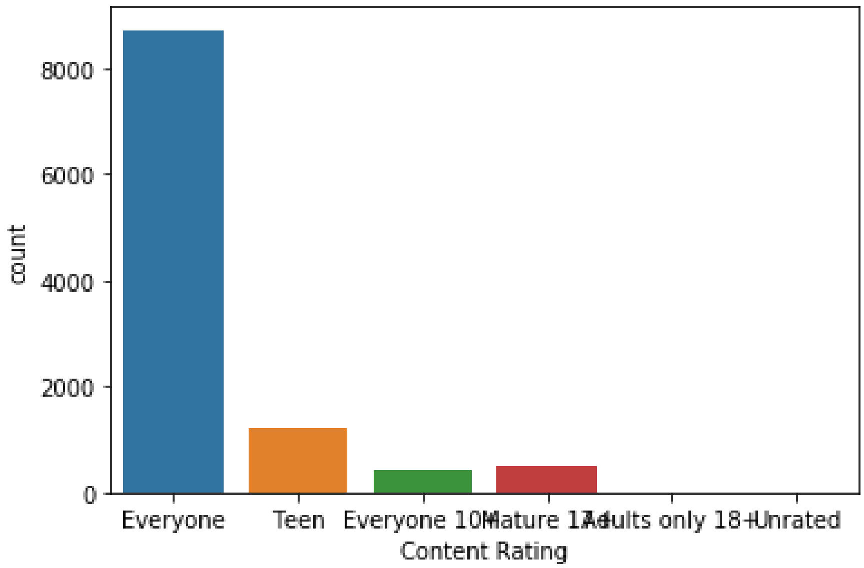
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10841 entries, 0 to 10840
Data columns (total 13 columns):
App                  10841 non-null object
Category             10841 non-null object
Rating               9367 non-null float64
Reviews              10841 non-null object
Size                 10841 non-null object
Installs             10841 non-null object
Type                 10840 non-null object
Price                10841 non-null object
Content Rating       10840 non-null object
Genres               10841 non-null object
Last Updated         10841 non-null object
Current Ver          10833 non-null object
Android Ver          10838 non-null object
dtypes: float64(1), object(12)
memory usage: 1.1+ MB
```

df_apps.describe()

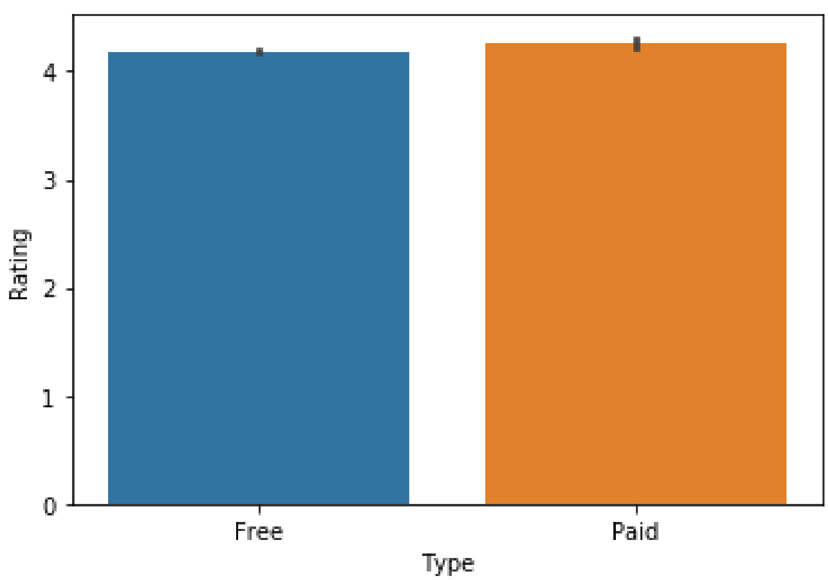
	Rating
count	9367.000000
mean	4.193338
std	0.537431
min	1.000000
25%	4.000000
50%	4.300000
75%	4.500000
max	19.000000

Content Rating

Out of the 10840 apps stored in our Dataset, 80 % of the applications are targeting all age groups from children, mature 21+ to adult as shown in the graph below.

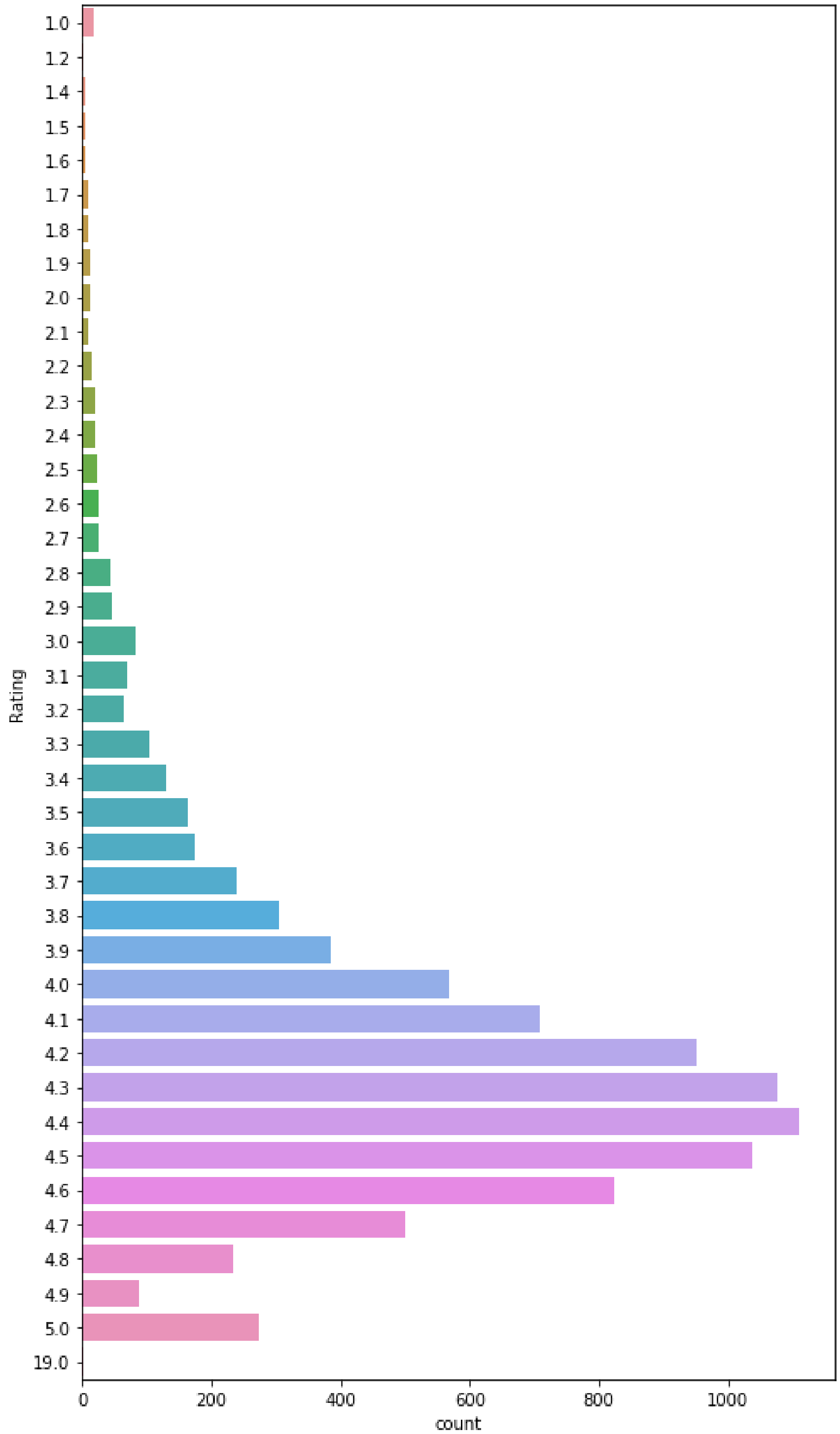


Type vs Rating



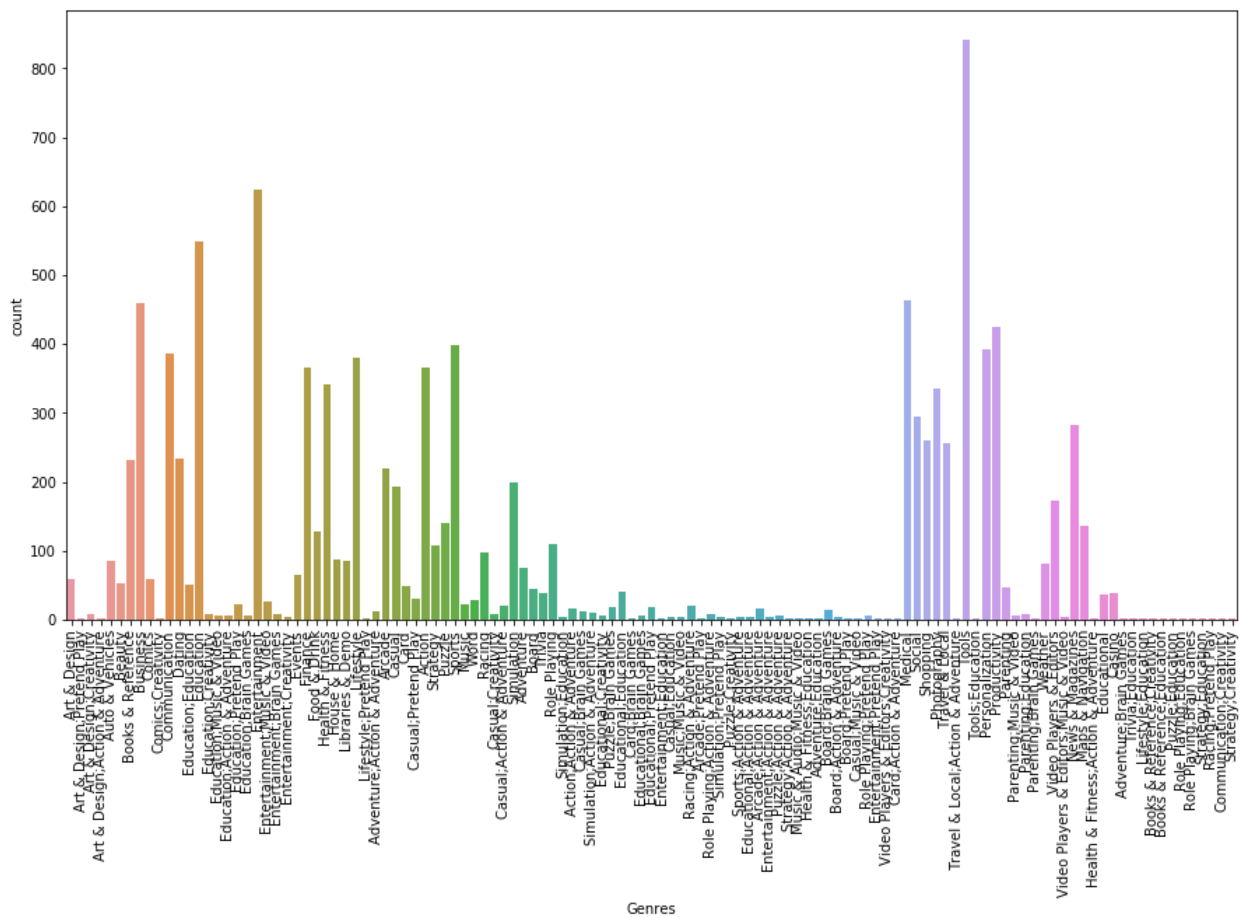
Rating

Overall user rating of the app (as when scraped) shows in the following figure over 1000 apps were rated 4.4 as highest rating.

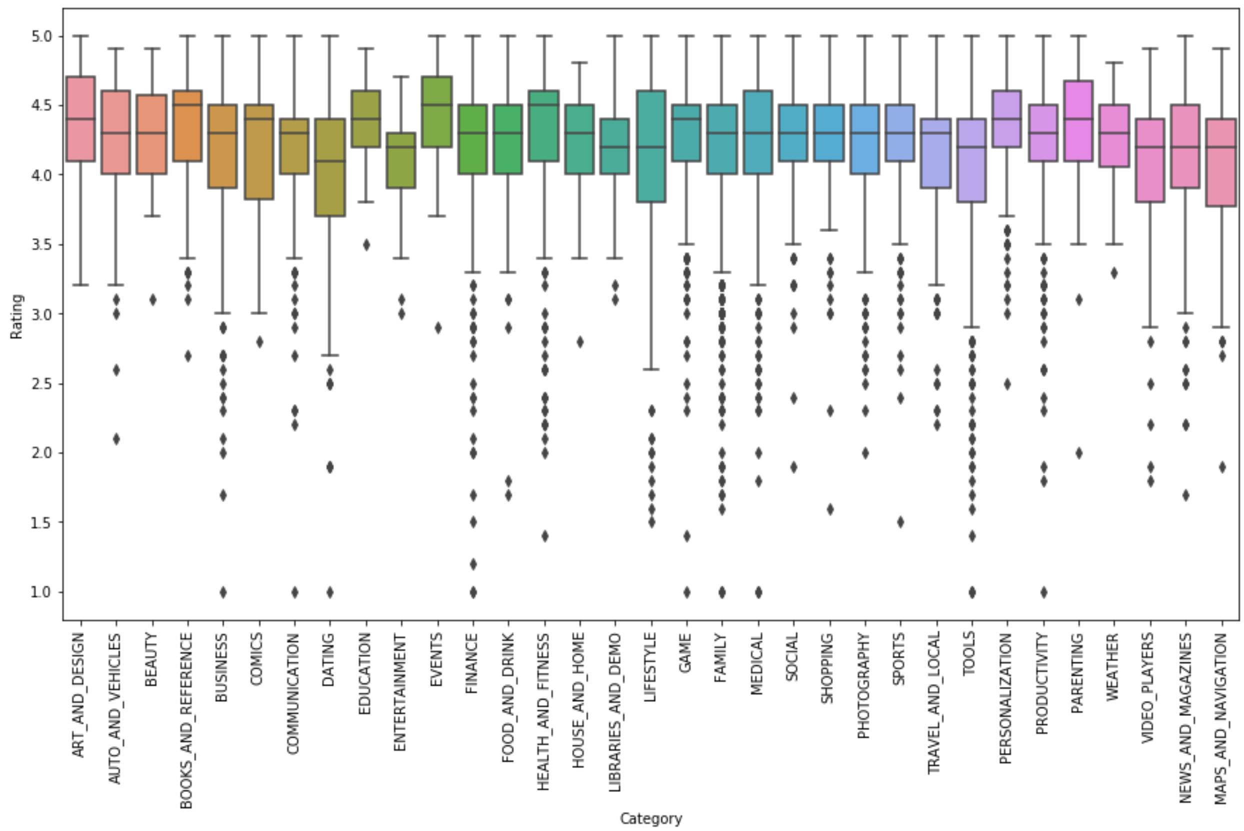


Genres

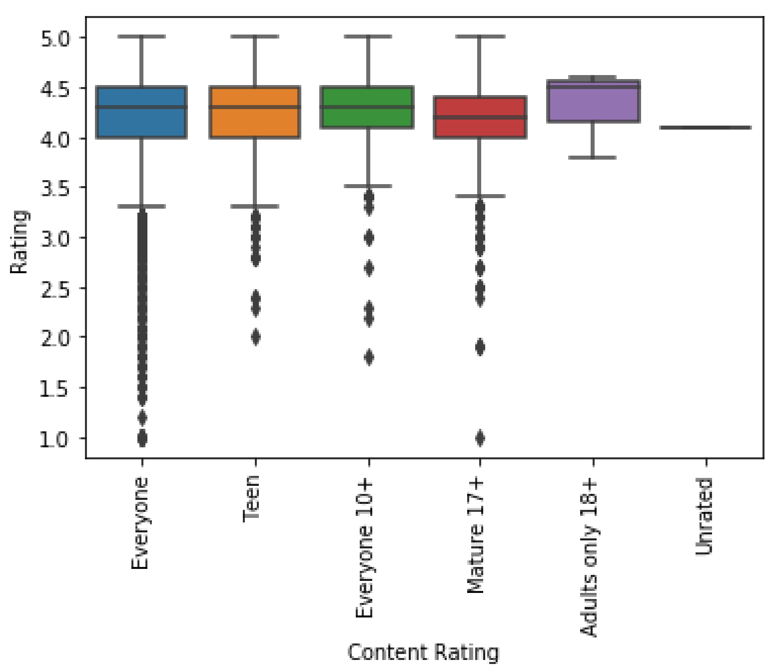
An application can belong to multiple genres (apart from its main category). For eg, a musical family game will belong to



Category vs Rating



Content Rating vs Rating



DATA COLLECTION AND PRE-PROCESSING

Data Collection

This information was scraped from the ***Google Play Store***. Using the ***Web Scraping*** technique *(also termed Screen Scraping, Web Data Extraction, Web Harvesting etc.)* it is employed to extract large amounts of data from websites whereby the data is extracted and saved to a local file in your computer or to a database in table (spreadsheet) format.

Data displayed by most websites can only be viewed using a web browser. They do not offer the functionality to save a copy of this data for personal use. The only option then is to manually copy and paste the data - a very tedious job which can take many hours or sometimes days to complete. Web Scraping is the technique of automating this process, so that instead of manually copying the data from websites, the Web Scraping software will perform the same task within a fraction of the time.

There are 13 features in our Data (as when scraped):

1. **App:** Application name
2. **Category:** Category the app belongs to
3. **Rating:** Overall user rating of the app
4. **Reviews:** Number of user reviews for the app
5. **Size:** Size of the app
6. **Installs:** Number of user downloads/installs for the app
7. **Type:** Paid or Free
8. **Price:** Price of the app
9. **Content Rating:** Age group the app is targeted at - Children / Mature 21+ / Adult
10. **Genres:** An app can belong to multiple genres (apart from its main category). For eg, a musical family game will belong to Music, Game, Family genres.
11. **Last Updated:** Date when the app was last updated on Play Store
12. **Current Ver:** Current version of the app available on Play Store
13. **Android Ver:** Min required Android version

Data Pre-processing

#Detection of Missing Values

df_apps.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10841 entries, 0 to 10840
Data columns (total 13 columns):
App                 10841 non-null object
Category            10841 non-null object
Rating              9367 non-null float64
Reviews             10841 non-null object
Size                10841 non-null object
Installs            10841 non-null object
Type                10840 non-null object
Price               10841 non-null object
Content Rating      10840 non-null object
Genres              10841 non-null object
Last Updated        10841 non-null object
Current Ver         10833 non-null object
Android Ver         10838 non-null object
dtypes: float64(1), object(12)
memory usage: 1.1+ MB
```

df_apps.isnull().sum()

```
App                  0
Category             0
Rating            1474
Reviews              0
Size                 0
Installs             0
Type                 1
Price                0
Content Rating       1
Genres               0
Last Updated         0
Current Ver          8
Android Ver          3
dtype: int64
```

#Missing values in Rating should be filled with the integer value of 0

df_apps['Rating'] = df_apps['Rating'].fillna(int(0))
df_apps.dropna(inplace = True)

df_apps.info()

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10829 entries, 0 to 10839
Data columns (total 13 columns):
App               10829 non-null object
Category          10829 non-null object
Rating            10829 non-null float64
Reviews           10829 non-null int64
Size              10829 non-null object
Installs          10829 non-null object
Type              10829 non-null object
Price             10829 non-null object
Content Rating    10829 non-null object
Genres            10829 non-null object
Last Updated      10829 non-null datetime64[ns]
Current Ver       10829 non-null object
Android Ver       10829 non-null object
dtypes: datetime64[ns](1), float64(1), int64(1), object(10)
memory usage: 1.2+ MB
```

```
# Cleaning Categories into integers
CategoryString = df_apps["Category"]
categoryVal = df_apps["Category"].unique()
categoryValCount = len(categoryVal)
category_dict = {}
for i in range(0,categoryValCount):
  category_dict[categoryVal[i]] = i
df_apps["Category_c"] = df_apps["Category"].map(category_dict).astype(int)

#scaling and cleaning size of installation
def change_size(size):
  if 'M' in size:
    x = size[:-1]
    x = float(x)*1000000
    return(x)
  elif 'k' == size[-1:]:
    x = size[:-1]
    x = float(x)*1000
    return(x)
  else:
    return None

df_apps["Size"] = df_apps["Size"].map(change_size)

#filling Size which had NA
df_apps.Size.fillna(method = 'ffill', inplace = True)
```

```
#Converting Type classification into binary
def type_cat(types):
    if types == 'Free':
        return 0
    else:
        return 1

df_apps['Type'] = df_apps['Type'].map(type_cat)

#Cleaning of genres
GenresL = df_apps.Genres.unique()
GenresDict = {}
for i in range(len(GenresL)):
    GenresDict[GenresL[i]] = i
df_apps['Genres_c'] = df_apps['Genres'].map(GenresDict).astype(int)

#Cleaning prices
def price_clean(price):
    if price == '0':
        return 0
    else:
        price = price[1:]
        price = float(price)
        return price

df_apps['Price'] = df_apps['Price'].map(price_clean).astype(float)

# convert reviews to numeric
df_apps['Reviews'] = df_apps['Reviews'].astype(int)

#Cleaning of content rating classification
RatingL = df_apps['Content Rating'].unique()
RatingDict = {}
for i in range(len(RatingL)):
    RatingDict[RatingL[i]] = i
df_apps['Content Rating'] = df_apps['Content Rating'].map(RatingDict).astype(int)

#Cleaning no of installs classification
df_apps['Installs'] = [int(i[:-1].replace(',','')) for i in df_apps['Installs']]

df_apps.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10829 entries, 0 to 10839
Data columns (total 15 columns):
App               10829 non-null object
Category          10829 non-null object
Rating            10829 non-null float64
Reviews           10829 non-null int32
Size              10829 non-null float64
Installs          10829 non-null int64
Type              10829 non-null int64
Price             10829 non-null float64
Content Rating    10829 non-null int32
Genres            10829 non-null object
Last Updated      10829 non-null datetime64[ns]
Current Ver       10829 non-null object
Android Ver       10829 non-null object
Category_c        10829 non-null int32
Genres_c          10829 non-null int32
dtypes: datetime64[ns](1), float64(3), int32(4), int64(2), object(5)
memory usage: 1.2+ MB
```

df_apps.describe()

	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Category_c	Genres_c
count	10829.000000	1.082900e+04	1.082900e+04	1.082900e+04	10829.000000	10829.000000	10829.00000	10829.000000	10829.000000
mean	3.623197	4.446018e+05	2.186270e+07	1.547990e+07	0.073599	1.028091	0.32810	17.665343	50.466248
std	1.513263	2.929213e+06	2.252805e+07	8.507114e+07	0.261129	15.957778	0.76176	7.480582	34.489114
min	0.000000	0.000000e+00	8.500000e+03	0.000000e+00	0.000000	0.000000	0.00000	0.000000	0.000000
25%	3.700000	3.800000e+01	5.100000e+06	5.000000e+03	0.000000	0.000000	0.00000	13.000000	19.000000
50%	4.200000	2.100000e+03	1.400000e+07	1.000000e+05	0.000000	0.000000	0.00000	18.000000	38.000000
75%	4.500000	5.481500e+04	3.000000e+07	5.000000e+06	0.000000	0.000000	0.00000	23.000000	89.000000
max	5.000000	7.815831e+07	1.000000e+08	1.000000e+09	1.000000	400.000000	5.00000	32.000000	118.000000

FEATURE ENGINEERING

from sklearn import preprocessing
from sklearn import utils

Transforming the Predictive Target (Y) using Label Uncoder

from sklearn import preprocessing
from sklearn import utils
Y= df_apps['Rating']
X = df_apps.drop('Rating', axis=1)
lab_enc = preprocessing.LabelEncoder()
rating_encoded = lab_enc.fit_transform(Y)
print(rating_encoded)
print(utils.multiclass.type_of_target(Y))
print(utils.multiclass.type_of_target(Y.astype('int')))
print(utils.multiclass.type_of_target(rating_encoded))

```
[30 28 36 ...  0 34 34]
continuous
multiclass
multiclass
```

Method 1: Recursive Feature Elimination

# Recursive Feature Elimination
from sklearn import datasets
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestRegressor

X = df_apps.drop('Rating', axis=1)

rfc = RandomForestClassifier(random_state=101)
rfecv = RFECV(estimator=rfc, step=1, cv=StratifiedKFold(10), scoring='accuracy')
rfecv.fit(X, rating_encoded)

```
RFECV(cv=StratifiedKFold(n_splits=10, random_state=None, shuffle=False),
      estimator=RandomForestClassifier(bootstrap=True, class_weight=None,
                                       criterion='gini', max_depth=None,
                                       max_features='auto', max_leaf_nodes=
None,
                                       min_impurity_decrease=0.0,
                                       min_impurity_split=None,
                                       min_samples_leaf=1, min_samples_spli
t=2,
                                       min_weight_fraction_leaf=0.0,
                                       n_estimators='warn', n_jobs=None,
                                       oob_score=False, random_state=101,
                                       verbose=0, warm_start=False),
      min_features_to_select=1, n_jobs=None, scoring='accuracy', step=1,
      verbose=0)
```

print('Optimal number of features: {}'.format(rfecv.n_features_))

```
Optimal number of features: 1
```

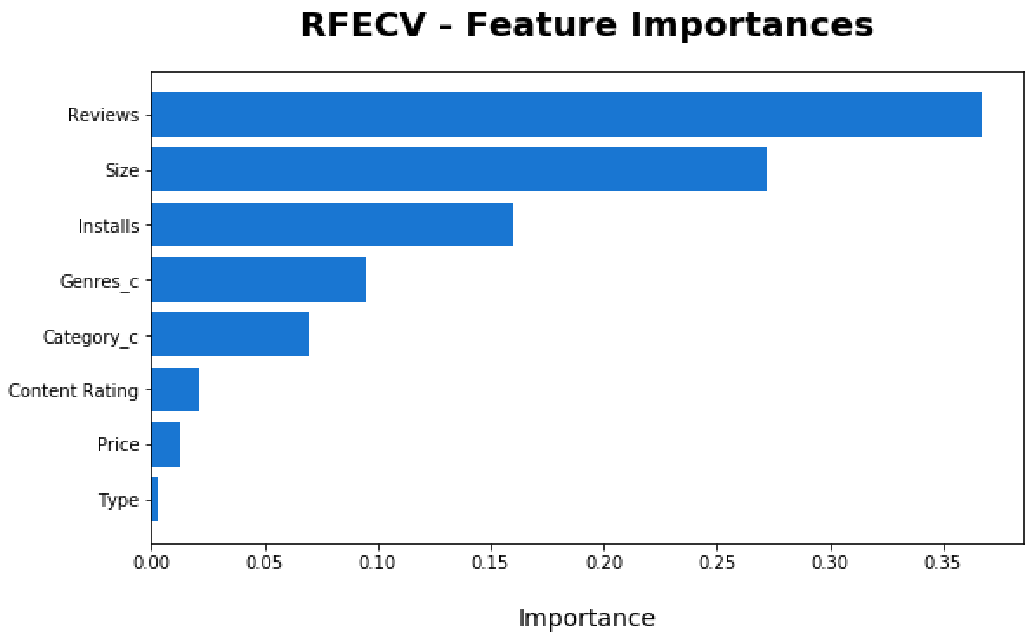
```
X.drop(X.columns[np.where(rfecv.support_ == False)[0]], axis=1,
inplace=True)
```

	Reviews
0	159
1	967
2	87510
3	215644
4	967

Method 2: Feature Importance

```
# Feature Importance
from sklearn import datasets
from sklearn import metrics
from sklearn.ensemble import ExtraTreesClassifier

XX = df_apps.drop('Rating', axis=1)

# fit an Extra Trees model to the data
model = ExtraTreesClassifier()
model.fit(XX, rating_encoded)
# display the relative importance of each attribute
print(model.feature_importances_)
dset = pd.DataFrame()
dset['attr'] = XX.columns
dset['importance'] = model.feature_importances_

dset = dset.sort_values(by='importance', ascending=True)

plt.figure(figsize=(9, 5))
plt.barh(y=dset['attr'], width=dset['importance'], color='#1976D2')
plt.title('RFECV - Feature Importances', fontsize=20, fontweight='bold', pad=20)
plt.xlabel('Importance', fontsize=14, labelpad=20)
plt.show()
```

```
[0.36677685 0.2721426  0.1601974  0.00272684 0.01309127 0.02086396
 0.06932111 0.09487996]
```



I will be using the Feature Importance method result that shows more important features (comparing to the recursive feature elimination option) and that I have named in the python coding “X_FI” referring to X Features Importance.

#Selected Features using Feature Importance
X_FI.head()

	Reviews	Size	Installs	Type	Price	Content Rating	Category_c	Genres_c
0	159	19000000.0	10000	0	0.0	0	0	0
1	967	14000000.0	500000	0	0.0	0	0	1
2	87510	8700000.0	5000000	0	0.0	0	0	0
3	215644	25000000.0	50000000	0	0.0	1	0	0
4	967	2800000.0	100000	0	0.0	0	0	2

DATA VISUALIZATION

App with large number of reviews

df_apps.loc[df_apps.Reviews == df_apps.Reviews.max()]

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres	Last Updated	Current Ver	Android Ver	Category_c	Genres_c
2544	Facebook	SOCIAL	4.1	78158306	26000000.0	1000000000	0	0.0	1	Social	3-Aug-18	Varies with device	Varies with device	20	86

App with the largest size

df_apps.loc[df_apps.Size == df_apps.Size.max()]

App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres	Last Updated	Current Ver	Android Ver	Cat
Post Bank	FINANCE	4.5	60449	100000000.0	1000000	0	0.00	0	Finance	23-Jul-18	2.9.12	4.0 and up	
Talking Babsy Baby: Baby Games	LIFESTYLE	4.0	140995	100000000.0	10000000	0	0.00	0	Lifestyle;Pretend Play	16-Jul-18	9	4.0 and up	
Hungry Shark Evolution	GAME	4.5	6074334	100000000.0	100000000	0	0.00	1	Arcade	25-Jul-18	6.0.0	4.1 and up	
Mini Golf King - Multiplayer Game	GAME	4.5	531458	100000000.0	5000000	0	0.00	0	Sports	20-Jul-18	3.04.1	4.0.3 and up	
Hungry Shark Evolution	GAME	4.5	6074627	100000000.0	100000000	0	0.00	1	Arcade	25-Jul-18	6.0.0	4.1 and up	

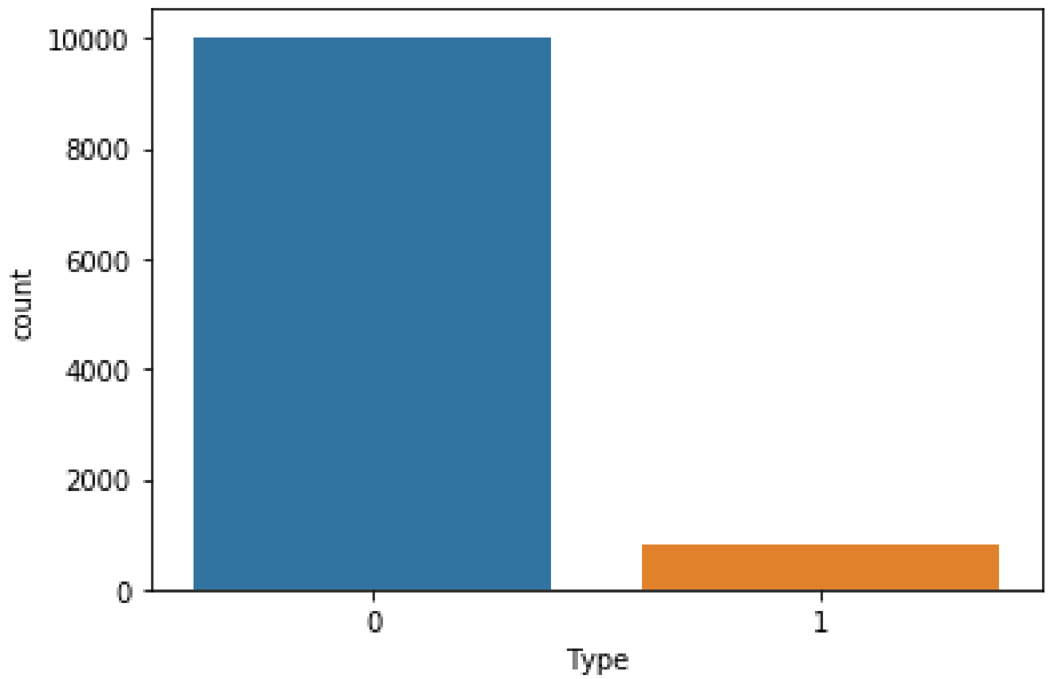
App with the largest num of installs

df_apps.loc[df_apps.Installs == df_apps.Installs.max()]

App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres	Last Updated	Current Ver	Android Ver	Category_c	Genres_c
Google Play Books	BOOKS_AND_REFERENCE	3.9	1433233	5000000.0	1000000000	0	0.0	1	Books & Reference	3-Aug-18	Varies with device	Varies with device	3	6
Messenger – Text and Video Chat for Free	COMMUNICATION	4.0	56642847	35000000.0	1000000000	0	0.0	0	Communication	1-Aug-18	Varies with device	Varies with device	6	10
WhatsApp Messenger	COMMUNICATION	4.4	69119316	35000000.0	1000000000	0	0.0	0	Communication	3-Aug-18	Varies with device	Varies with device	6	10
Google Chrome: Fast & Secure	COMMUNICATION	4.3	9642995	17000000.0	1000000000	0	0.0	0	Communication	1-Aug-18	Varies with device	Varies with device	6	10

Paid vs Free

sns.countplot(df_apps['Type'],label="Count")
plt.show()



df_apps.groupby('Type')['Type'].count()

```
Type
0    10032
1      797
Name: Type, dtype: int64
```

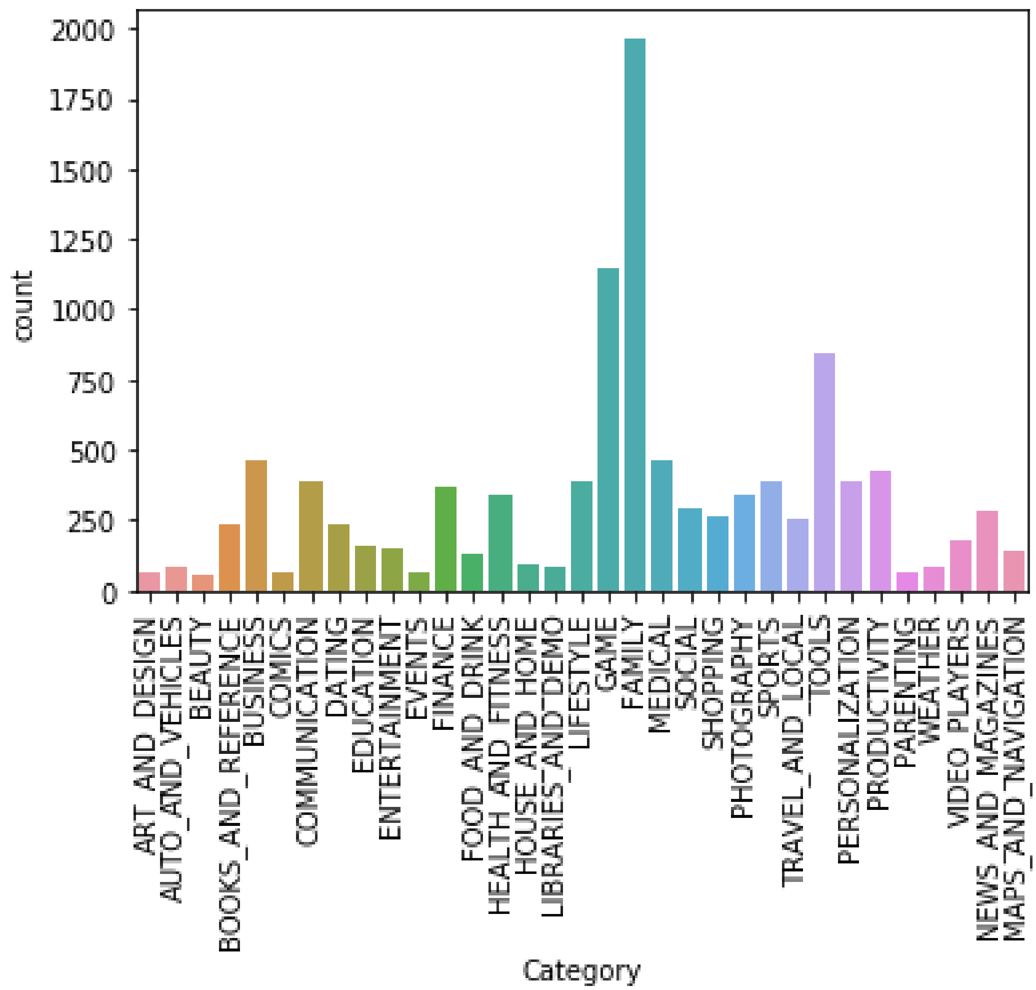
App which hasn't been updated

# App which hasn't been updated
min(df_apps['Last Updated'])
print(df_apps[df_apps['Last Updated'] == df_apps['Last Updated'].min()])

```
                App                     Category  Rating  Reviews  Size  Installs  \
7479  FML F*ck my life + widget   FAMILY     4.2     1415  209k  100,000+

      Type   Price Content Rating      Genres     Last Updated Current Ver  \
7479  Free      0        Everyone   Entertainment   2010-05-21         3.1

     Android Ver
7479  1.5 and up
```

Most popular category

```
sns.countplot(df_apps['Category'],label="Count")
plt.xticks(rotation='vertical')
plt.show()
```

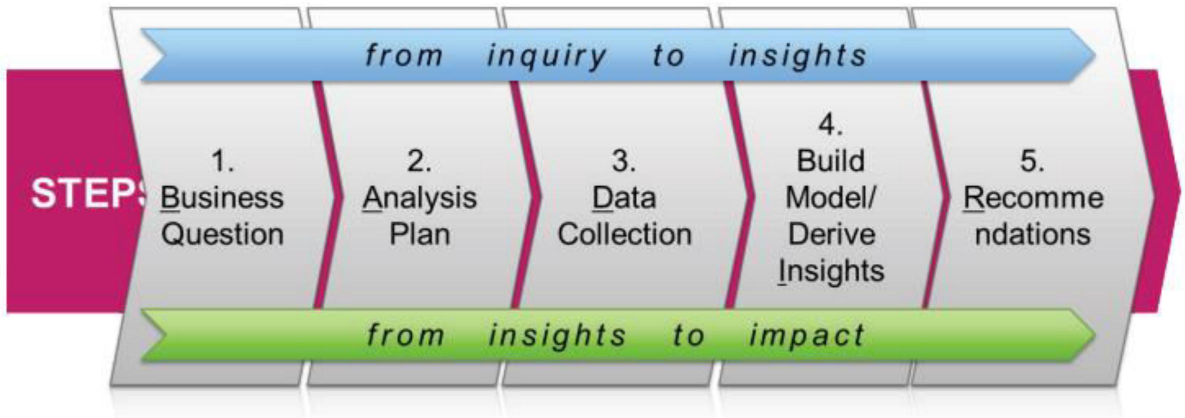


df_apps.groupby('Category')['Category'].count()

```
Category
ART_AND_DESIGN           64
AUTO_AND_VEHICLES        85
BEAUTY                   53
BOOKS_AND_REFERENCE     230
BUSINESS                460
COMICS                   60
COMMUNICATION           387
DATING                  234
EDUCATION               156
ENTERTAINMENT           149
EVENTS                   64
FAMILY                 1968
FINANCE                 366
FOOD_AND_DRINK          127
GAME                   1144
HEALTH_AND_FITNESS      341
HOUSE_AND_HOME           88
LIBRARIES_AND_DEMO       84
LIFESTYLE               382
MAPS_AND_NAVIGATION     137
MEDICAL                 463
NEWS_AND_MAGAZINES      283
PARENTING                60
PERSONALIZATION         390
PHOTOGRAPHY             335
PRODUCTIVITY            424
SHOPPING                260
SOCIAL                  295
SPORTS                  384
TOOLS                   841
TRAVEL_AND_LOCAL        258
VIDEO_PLAYERS           175
WEATHER                  82
Name: Category, dtype: int64
```

DATA SCIENCE PROCESS: BEST PRACTICES

BADIR: Project Process Framework



- Business Question

  The Play Store apps data has enormous potential to drive app-making businesses to success. Actionable insights can be drawn for developers to work on and capture the Android market!

- Analysis Plan

Analysis Goal	Have a better understanding of the Android market.
Hypothesis	The social and games apps are the best rated
Methodology	Predictive Analysis Methods using Regression Algorithms like Random Forest Regression, Support Vector Regression and Linear Regression.
Project Plan	1. **Business & Data Understanding** (BADIR Framework) 2. **Data Preparation** (cleaning, Selecting and transformation) 3. **Modeling** 4. **Evaluation**: evaluate the result in the context of the business goal + new business requirements can pop up, due to the new patterns discovered during the data evaluation. 5. **Deployment:** The final report needs to summarize the project insights and outcomes and review the project to see what needs to be improved upon.

- Data Collection
    - The data was scraped from the Google Play Store. Using the Web Scraping technique (for more details go back to Data Collection section in this file).
    - In order to clean and validate the data, I used the following preprocesses:
        - Remove all Missing Data
        - Feature Selection
        - Transformation of variables
    - 80% Time spent on Data Munging.

- Build Model / Derive Insights
    - Try more than one machine learning technique.
    - Fine-Tune parameters.
    - Assess Model Performance.
    - Avoid Over-fitting.
    - 20% Time spent.
- Recommendations
    - Visualizations of derived insights presenting each predictor's relationship with target "*Rating*": Histogram, scatter plots, Heat map…
    - Review existing business rules/model.
    - Target customers that are likely to default less.
    - Tracking model: Test, Measure and Improve.

PREDICTIVE MODELING AND EVALUATION (THE WHOLE PROCESS)

#XX refers to the data frame with the Important features only

XX.head()
depVar= df_apps['Rating']

#estimators
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn import linear_model

#model metrics
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.model_selection import cross_val_score

#cross validation
from sklearn.model_selection import train_test_split

#Training Set (Feature Space: X Training)
X_train = XX[: 1000]
X_train_count = len(X_train.index)
print('The number of observations in the Y training set are:',str(X_train_count))
X_train.head()

```
The number of observations in the Y training set are: 1000
```

	Reviews	Size	Installs	Type	Price	Content Rating	Category_c	Genres_c
0	159	19000000.0	10000	0	0.0	0	0	0
1	967	14000000.0	500000	0	0.0	0	0	1
2	87510	8700000.0	5000000	0	0.0	0	0	0
3	215644	25000000.0	50000000	0	0.0	1	0	0
4	967	2800000.0	100000	0	0.0	0	0	2

#Dependent Variable Training Set (y Training)

```
y_train = depVar[: 1000]
y_train_count = len(y_train.index)
#y_train_count = len(y_train)
print('The number of observations in the Y training set are:',str(y_train_count))
y_train.head()
```

```
The number of observations in the Y training set are: 1000
Out[71]:
0    4.1
1    3.9
2    4.7
3    4.5
4    4.3
Name: Rating, dtype: float64
```

#Testing Set (X Testing)

```
X_test = XX[-100:]
X_test_count = len(X_test.index)
print('The number of observations in the feature testing set is:',str(X_test_count))
X_test.head()
```

```
The number of observations in the feature testing set is: 100
```

	Reviews	Size	Installs	Type	Price	Content Rating	Category_c	Genres_c
10740	8484	1700000.0	1000000	0	0.0	0	25	91
10741	32	7900000.0	1000	0	0.0	0	16	29
10742	16	1200000.0	500	0	0.0	0	18	19
10743	1	2000000.0	100	0	0.0	0	11	24
10744	1	5800000.0	1	0	0.0	0	11	24

#Ground Truth (y_test)

```
y_test = depVar[-100:]
y_test_count = len(y_test.index)
#y_test_count = len(y_test)
print('The number of observations in the Y training set are:',str(y_test_count))
y_test.head()
```

```
The number of observations in the Y training set are: 100
Out[73]:
10740    4.2
10741    5.0
10742    3.4
10743    0.0
10744    0.0
Name: Rating, dtype: float64
```

X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, test_size=0.25, random_state=0)

X_train.shape, y_train.shape, X_test.shape, y_test.shape

```
((750, 8), (750,), (250, 8), (250,))
```

#Models
modelSVR = SVR()
modelRF = RandomForestRegressor()
modelLR = LinearRegression()

Random Forest Regression

from sklearn.model_selection import cross_val_score
modelRF.fit(X_train,y_train)
print(cross_val_score(modelRF, X_train, y_train))

print('These values correspond to the the following: ')
print('1st value: The score array for test scores on each cv split. (Higher is an indicator of a better performing model')
print('2nd value: The time for fitting the estimator on the train set for each cv split.')
print('3rd Value: The time for scoring the estimator on the test set for each cv split.')
print('R-Squared: %.3f' % modelRF.score(X_train,y_train))

```
[0.43065548 0.56608103 0.31868752]
These values correspond to the the following:
1st value:  The score array for test scores on each cv split. (Higher is a
n indicator of a better performing model
2nd value:  The time for fitting the estimator on the train set for each c
v split.
3rd Value:  The time for scoring the estimator on the test set for each cv
split.
R-Squared:  0.922
```

Support Vector Regression

modelSVR.fit(X_train,y_train)
print(cross_val_score(modelSVR, X_train, y_train))
print('R-Squared: %.3f' % modelSVR.score(X_train,y_train))

```
[-0.01695971 -0.01677     0.01178045]
R-Squared:  0.496
```

Linear Regression

```
modelLR.fit(X_train,y_train)
print(cross_val_score(modelLR, X_train, y_train))
print('R-Squared:  %.3f' % modelLR.score(X_train,y_train))
```

```
[-0.01257082 -0.03493534  0.02720635]
R-Squared:  0.094
```

Predictions

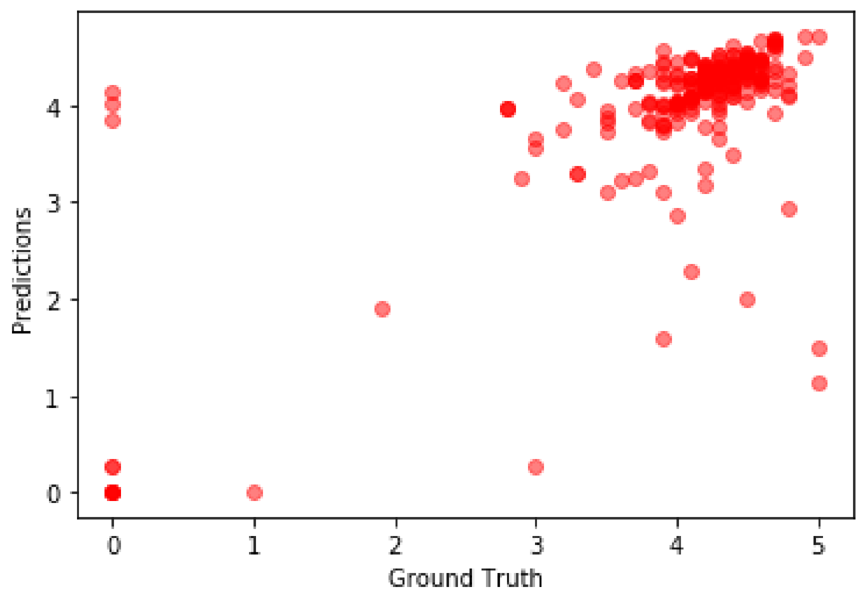
```
#RandomForest Regression Model Predictions

predRF = modelRF.predict(X_test)
predRF_Rsquared = r2_score(y_test,predRF)
rmseRF = sqrt(mean_squared_error(y_test, predRF))
print('RandomForest Regression Model Predictions:')
print('RMSE: %.3f' % rmseRF)

#Support Vector Regression Model Predictions

predSVR = modelSVR.predict(X_test)
predSVR_Rsquared = r2_score(y_test,predSVR)
rmseSVR = sqrt(mean_squared_error(y_test, predSVR))
print('Support Vector Regression Model Predictions:')
print('RMSE: %.3f' % rmseSVR)

#Linear Regression Model Predictions

predLR = modelLR.predict(X_test)
predLR_Rsquared = r2_score(y_test,predLR)
rmseLR = sqrt(mean_squared_error(y_test, predLR))
print('Linear Regression Model Predictions:')
print('RMSE: %.3f' % rmseLR)
```

```
RandomForest Regression Model Predictions:
RMSE: 0.717
Support Vector Regression Model Predictions:
RMSE: 0.942
Linear Regression Model Predictions:
RMSE: 0.928
```

MODEL SELECTION

#Random Forest Selected Model with best performance
#RMSE: 0.717
#R-Squared: 0.922

```
plt.scatter(y_test, predRF, c='r', alpha = 0.5)
plt.xlabel('Ground Truth')
plt.ylabel('Predictions')
plt.show();
```



CROSS VALIDATION

 # Necessary imports

from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn import metrics

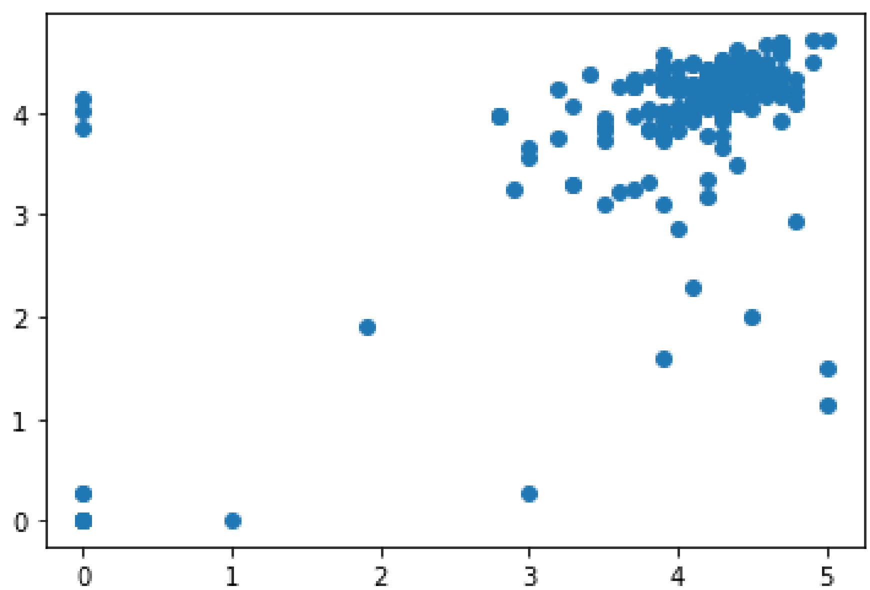
 # Perform 6-fold cross validation

scores = cross_val_score(modelRF, XX, depVar, cv=6)
print ("Cross-validated scores:", scores)

```
Cross-validated scores: [0.36185322 0.44389649 0.56824708 0.46541865 0.4687
7069 0.46052085]
```

 # Make cross validated predictions

predictions = cross_val_predict(modelRF, XX, depVar, cv=6)
plt.scatter(y_test, predRF)



accuracy = metrics.r2_score(y_test, predRF)
print ("Cross-Predicted Accuracy:", accuracy)

```
Cross-Predicted Accuracy: 0.45928695245513584
```