**Google Play Store Apps Report**

****

**Data Analyst:** Chaimaa Driouech

**GitHub:**

**Date:** 01/04/2020

**Summary**

[**OVERVIEW** 3](#_Toc29030897)

[Acknowledgements 3](#_Toc29030898)

[Inspiration 3](#_Toc29030899)

[Tasks 3](#_Toc29030900)

[**PYTHON AND ALL IMPORTED LIBRARIES** 4](#_Toc29030901)

[**EXPLORATORY DATA ANALYSIS** 5](#_Toc29030902)

[**DATA COLLECTION AND PRE-PROCESSING** 10](#_Toc29030903)

[**FEATURE ENGINEERING** 15](#_Toc29030904)

[Transforming the Predictive Target (Y) using Label Uncoder 15](#_Toc29030905)

[Method 1: Recursive Feature Elimination 15](#_Toc29030906)

[Method 2: Feature Importance 16](#_Toc29030907)

[**DATA VISUALIZATION** 18](#_Toc29030908)

[App with large number of reviews 18](#_Toc29030909)

[App with the largest size 18](#_Toc29030910)

[App with the largest num of installs 18](#_Toc29030911)

[Paid vs Free 19](#_Toc29030912)

[App which hasn't been updated 19](#_Toc29030913)

[Most popular category 20](#_Toc29030914)

[**DATA SCIENCE PROCESS: BEST PRACTICES** 22](#_Toc29030915)

[**BADIR: Project Process Framework** 22](#_Toc29030916)

[**PREDICTIVE MODELING AND EVALUATION (THE WHOLE PROCESS)** 24](#_Toc29030917)

[Random Forest Regression 26](#_Toc29030918)

[Support Vector Regression 26](#_Toc29030919)

[Linear Regression 27](#_Toc29030920)

[Predictions 27](#_Toc29030921)

[**MODEL SELECTION** 28](#_Toc29030922)

[**CROSS VALIDATION** 29](#_Toc29030923)

# **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

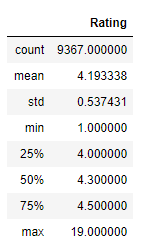
Current Ver 10833 non-null object

Android Ver 10838 non-null object

dtypes: float64(1), object(12)

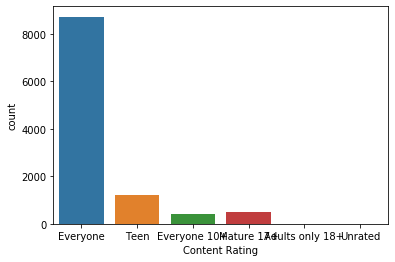
memory usage: 1.1+ MB

df\_apps.describe()

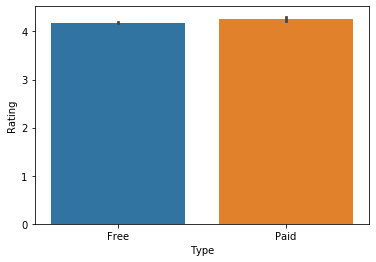
****

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.

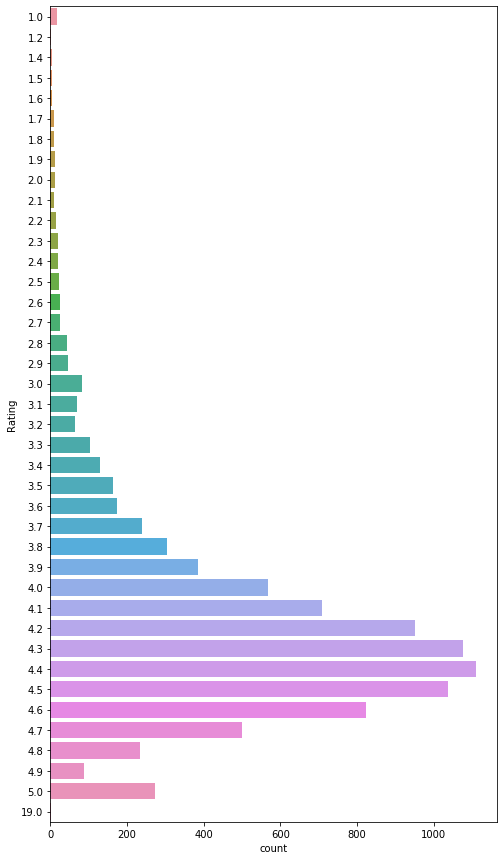
****

Type vs Rating

****

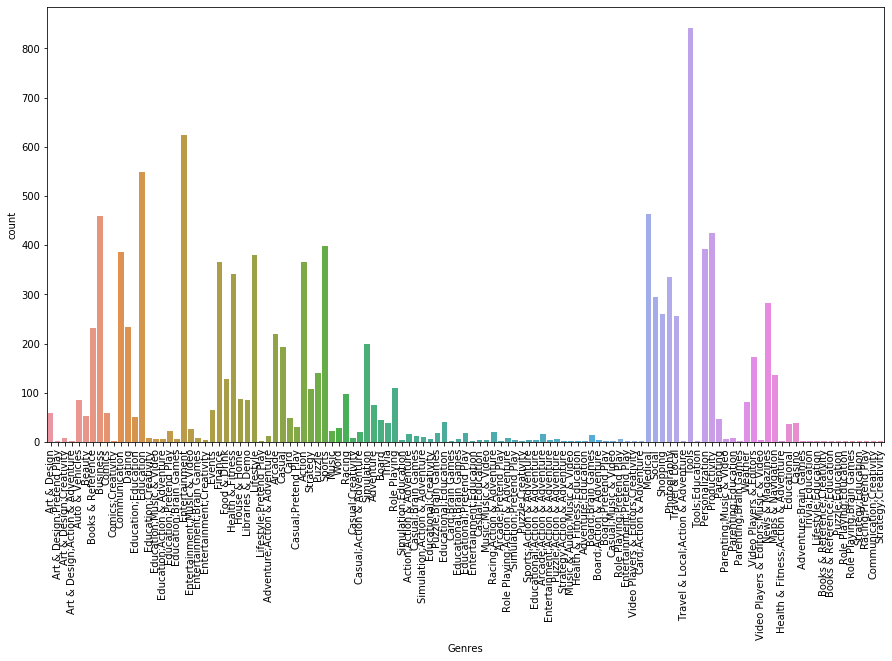
Rating

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

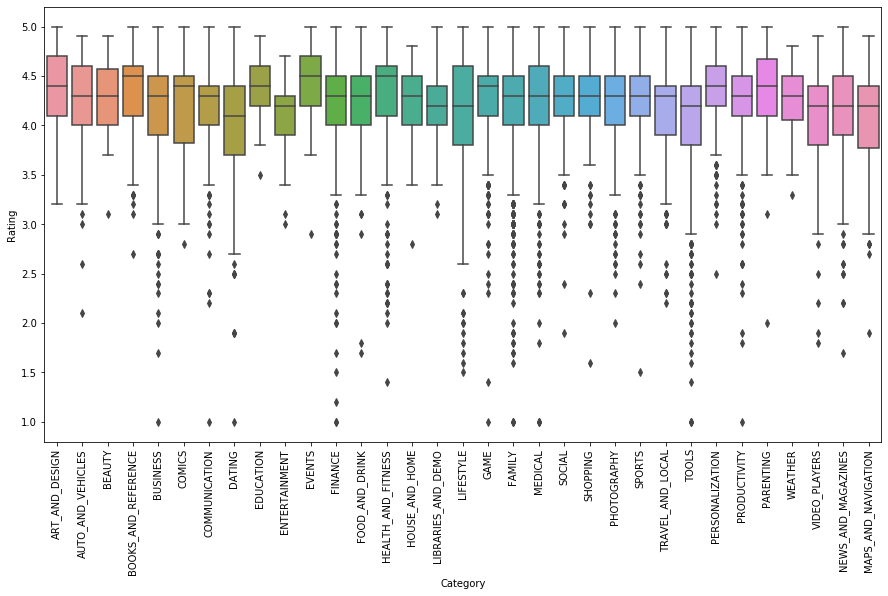
****

Genres

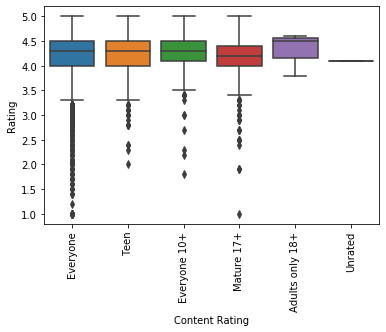
An application can belong to multiple genres (apart from its main category). For eg, a musical family game will belong to



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

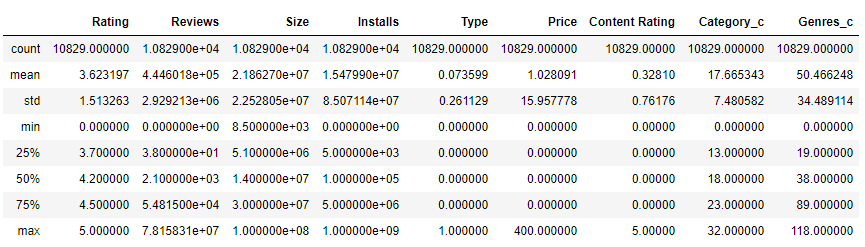
Category\_c 10829 non-null int32

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()



# **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\_split=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)

X.head()



### 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')

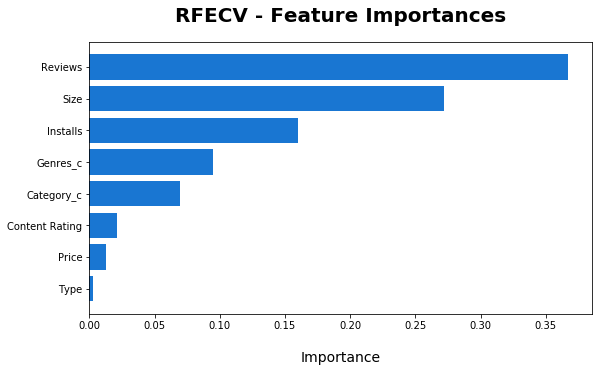
plt.title('RFECV - Feature Importances', fontsize=20, fontweight='bold', pad=20)

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]

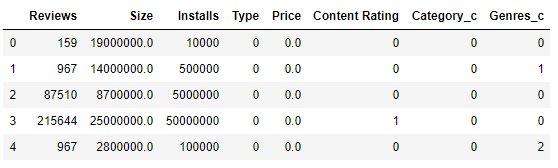




I will be using the Feature Importance method result that shows more important features (comparing to the recursive feature elimination option) and that I have named in the python coding “X\_FI” referring to X Features Importance.

**#Selected Features using Feature Importance**

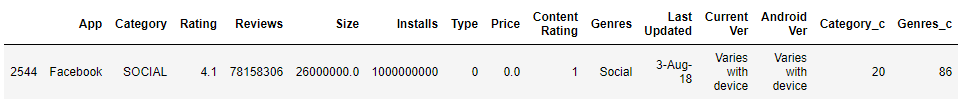
X\_FI.head()



# **DATA VISUALIZATION**

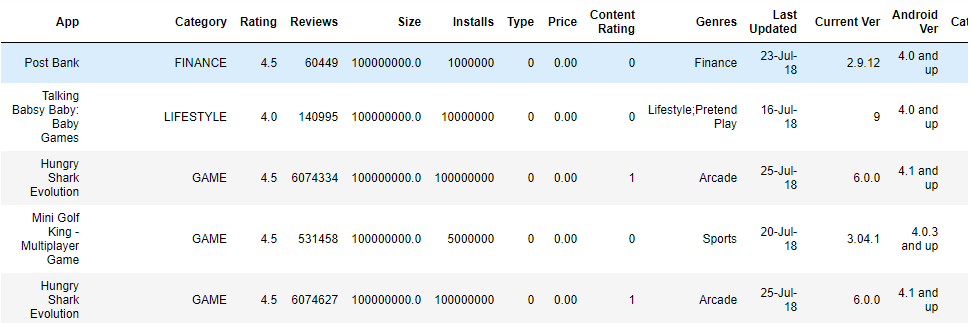
### App with large number of reviews

df\_apps.loc[df\_apps.Reviews == df\_apps.Reviews.max()]



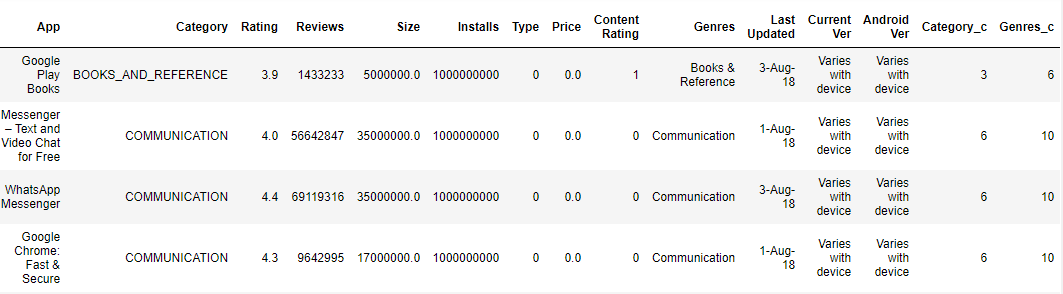
### App with the largest size

df\_apps.loc[df\_apps.Size == df\_apps.Size.max()]



### App with the largest num of installs

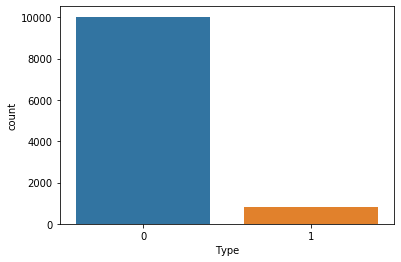
df\_apps.loc[df\_apps.Installs == df\_apps.Installs.max()]



### 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

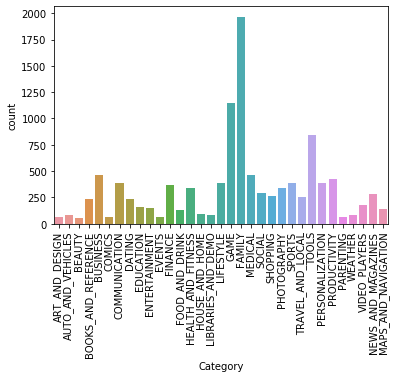
7479 1.5 and up

### 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

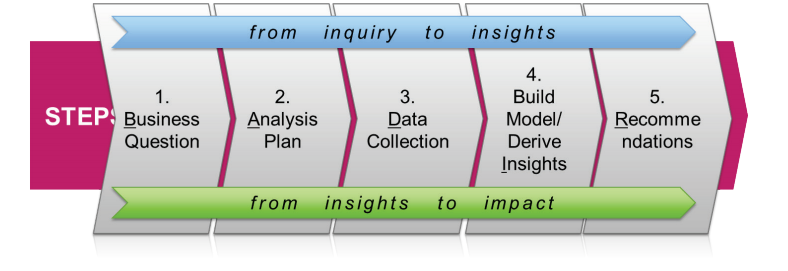
VIDEO\_PLAYERS 175

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



**#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)**

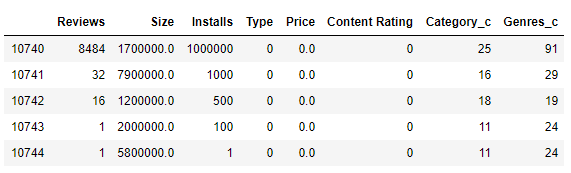
X\_test = XX[-100:]

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



**#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 an indicator of a better performing model

2nd value: The time for fitting the estimator on the train set for each cv 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**

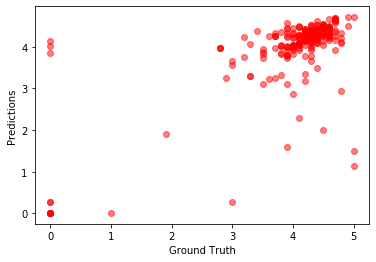
**#R-Squared: 0.922**

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)

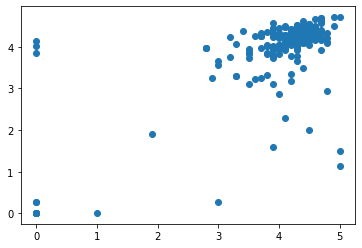
print ("Cross-validated scores:", scores)

Cross-validated scores: [0.36185322 0.44389649 0.56824708 0.46541865 0.46877069 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