Binoy Dutt/Priya Sanodia

GroUP 12

Pokemon GO Analytics

Contents

[1 Introduction 2](#_Toc481634638)

[2 Data Description 3](#_Toc481634639)

[3 Project Instruction 4](#_Toc481634640)

[3.1 Web Scrapping: 4](#_Toc481634641)

[3.1.1 Android: 4](#_Toc481634642)

[3.1.2 IOS: 4](#_Toc481634643)

[3.2 Data Organization 4](#_Toc481634644)

[3.3 Data Exploration 5](#_Toc481634645)

[3.3.1 Scatter Matrix 7](#_Toc481634646)

[3.3.2 Pearson Coefficient: 8](#_Toc481634647)

[3.3.3 Time Series Plot 9](#_Toc481634648)

[3.4 Prediction Model 11](#_Toc481634649)

[3.4.1 Android Prediction: 11](#_Toc481634650)

[3.4.2 iOS Prediction: 13](#_Toc481634651)

[4 Deep Learning 15](#_Toc481634652)

[4.1 Step 1: Web Scrapping 15](#_Toc481634653)

[4.2 Step 2: Download Images 15](#_Toc481634654)

[4.3 Step 3: Deep Learning 15](#_Toc481634655)

[4.4 Results: 15](#_Toc481634656)

[4.4.1 Android Images 15](#_Toc481634657)

[4.4.2 IOS Images 18](#_Toc481634658)

# Introduction

Pokemon Go! became a very famous augmented reality (AR) game in 2016 summer. In

this project, we want to understand the success of the mobile app game. Specifically, the

purposes of this project are (1) to do web scraping using BeautifulSoup, (2) to

construct a Pandas dataframe, (3) to explore/visualize the numeric data using

matplotlib or seaborn, and finally (4) to use sklearn to build machine learning

models to predict the app’s review counts. The 5 best teams will get extra credits. Finally,

for more extra credits, you can (5) analyze the app’s screenshot images using deep

learning with tensorflow.

# Data Description

For this project, I have downloaded app pages of Pokemon Go! from Google Play Store

and Apple App Store from July 21 2016 to October 31 2016:

* https://play.google.com/store/apps/details?id=com.nianticlabs.pokemongo&hl=en
* https://itunes.apple.com/us/app/pok%C3%A9mongo/id1094591345?mt=8

The webpages were downloaded every ten minutes. This means that there are 144

(=24x6) HTML files for a given day and a given platform. You can download the zip file

from the following link:

* http://diamond.mccombs.utexas.edu/insy5378/pokemon\_5378.zip

Once you extract the ZIP file, you will see 103 date folders under “data” folder. Each

date folder contains HTML files downloaded in the specified date. Each HTML file name

is formatted as “HH\_MM\_pokemon\_PLATFORM.html”, where HH is hour, MM is

minute, and PLATFORM is either “android” or “ios”. Note that due to intermittent

connection errors, some HTML files may not be properly downloaded.

# Project Instruction

## Web Scrapping:

**File - dataextraction.py**

In order to navigate the folders OS module was used with 2 loops where in the first loop was navigating the folders and the second loop was navigating the files.

Web Scrapping was performed using Beautiful Soup and the following variables were extracted.

Error list for each of the 11 variables along with the file names is mentioned in the following json file



### Android:

average rating (in the scale between 1.0 and 5.0) (*android\_avg\_rating*)

number of total ratings (*android\_total\_ratings*)

number of ratings for 1-5 stars (*android\_ratings\_1*, *android\_ratings\_2*, … , *android\_ratings\_5*) file size in MB (*android\_file\_size*)

### IOS:

number of customer ratings in the Current Version (let’s call it *ios\_current\_ratings*)

number of customer ratings in All Versions (*ios\_all\_ratings*)

file size in MB (*ios\_file\_size*)

In all 11 variables were extracted.

As there were many values that were missing the scrapping portion of the code were enclosed in a try catch block with the missing values set as 0.

## Data Organization

The collected data was stored in a dictionary as below

datetime(2016, 7, 21, 0, 0, 0) =

{‘ios\_current\_ratings’: 4688,

‘ios\_all\_ratings’: 106508,

‘ios\_file\_size’: 110,

‘android\_avg\_rating’: 3.9,

‘android\_total\_ratings’: 1281802,

‘android\_rating\_1’: 199974,

‘android\_rating\_2’: 71512,

‘android\_rating\_3’: 117754,

‘android\_rating\_4’: 165956,

‘android\_rating\_5’: 726597,

‘android\_file\_size’: 58}

The date time values were extracted from the working directory and the file names for the web page using simple string manipulation.

The dictionary created was then converted into a pandas.

**File - dataframe.py**

The data was further pre-processed where in all the missing value that were set as 0 were dealt with using the following rules:

If the first value of an attribute is 0 the missing value is replaced with the 2nd value

If the last values of an attribute is 0 then the missing value is replaced with the 2nd last value

For all the rest of the missing values in an attribute an estimation by simple average of the next and previous values.

**File - dataprocess.py**

The pre-processed data frame was then saved in json, csv, excel formats using built in pandas method *.to\_json(Filename), .to\_csv(Filename), .to\_excelFilename)* respectively.

## Data Exploration

Using the pre-processed and raw DataFrame a describe method resulted in the following output

**File - dataexploration.py**

**Raw Data:**

android\_avg\_rating android\_file\_size android\_rating\_1 \

count 14810.000000 14810.000000 14810.000000

mean 4.040074 55.912154 720337.732208

std 0.177924 26.105101 229050.372204

min 0.000000 0.000000 0.000000

25% 4.000000 58.000000 627242.000000

50% 4.100000 61.000000 752846.000000

75% 4.100000 76.000000 909636.000000

max 4.100000 77.000000 982631.000000

android\_rating\_2 android\_rating\_3 android\_rating\_4 android\_rating\_5 \

count 14810.000000 14810.000000 14810.000000 1.481000e+04

mean 220940.611209 406204.605199 650649.112154 3.274879e+06

std 62104.040636 120682.713630 203898.888126 1.091526e+06

min 0.000000 0.000000 0.000000 0.000000e+00

25% 204299.000000 373913.000000 596010.000000 2.977746e+06

50% 240452.000000 447650.000000 716201.000000 3.633064e+06

75% 267621.000000 496153.000000 804331.000000 4.099775e+06

max 285115.000000 528687.000000 856213.000000 4.352574e+06

android\_total\_ratings ios\_all\_ratings ios\_current\_ratings \

count 1.481000e+04 14810.000000 14810.000000

mean 5.277365e+06 202859.866036 7048.311411

std 1.695682e+06 33352.105141 8911.959360

min 1.281802e+06 106508.000000 0.000000

25% 4.779210e+06 201533.000000 1662.000000

50% 5.790213e+06 215355.000000 3457.000000

75% 6.577516e+06 223336.000000 9166.000000

max 7.005220e+06 230601.000000 46692.000000

ios\_file\_size

count 14810.000000

mean 196.718096

std 67.164680

min 104.000000

25% 110.000000

50% 211.000000

75% 258.000000

max 260.000000

**Pre- Processed Data**

android\_avg\_rating android\_file\_size android\_rating\_1 \

count 14810.000000 14810.000000 14810.000000

mean 4.046550 67.968805 720981.584267

std 0.071877 8.191252 227578.490699

min 3.900000 58.000000 199974.000000

25% 4.000000 61.000000 627242.000000

50% 4.100000 61.000000 752846.000000

75% 4.100000 77.000000 909636.000000

max 4.100000 77.000000 982631.000000

android\_rating\_2 android\_rating\_3 android\_rating\_4 android\_rating\_5 \

count 14810.000000 14810.000000 14810.000000 1.481000e+04

mean 221147.901958 406554.456111 651182.477583 3.277482e+06

std 61577.562465 119814.578100 202622.936471 1.085615e+06

min 71521.000000 117754.000000 165956.000000 7.265970e+05

25% 204299.000000 373913.000000 596010.000000 2.977746e+06

50% 240452.000000 447650.000000 716201.000000 3.633064e+06

75% 267621.000000 496153.000000 804331.000000 4.099775e+06

max 285115.000000 528687.000000 856213.000000 4.352574e+06

android\_total\_ratings ios\_all\_ratings ios\_current\_ratings \

count 1.481000e+04 14810.000000 14810.000000

mean 5.277365e+06 202859.866036 7424.409858

std 1.695682e+06 33352.105141 9107.281048

min 1.281802e+06 106508.000000 29.000000

25% 4.779210e+06 201533.000000 1865.000000

50% 5.790213e+06 215355.000000 3676.000000

75% 6.577516e+06 223336.000000 9609.000000

max 7.005220e+06 230601.000000 46692.000000

ios\_file\_size

count 14810.000000

mean 196.718096

std 67.164680

min 104.000000

25% 110.000000

50% 211.000000

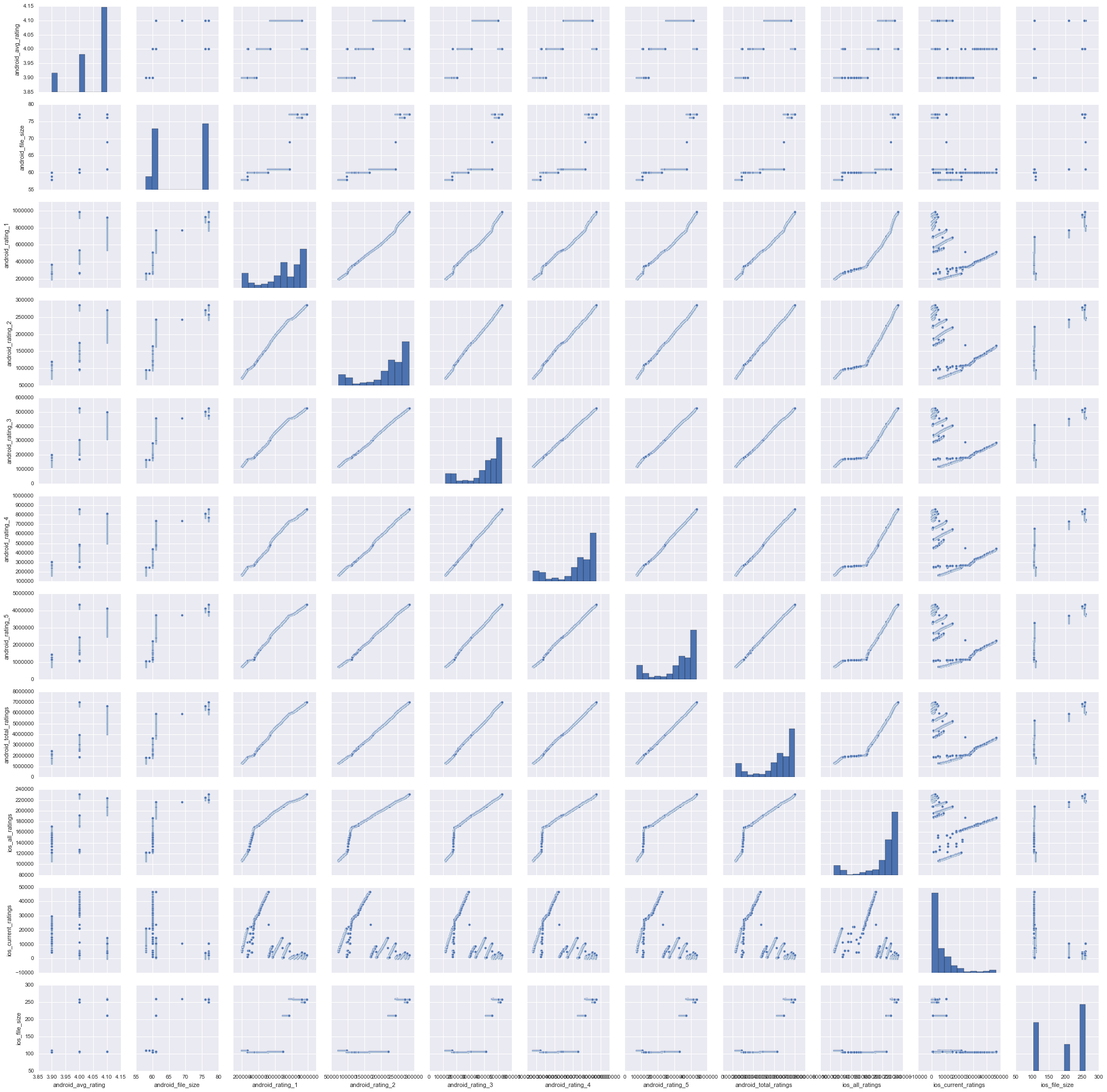
75% 258.000000

max 260.000000

As we can see that in the pre-processed results we have no 0 values and hence we see consistency in the data.

Seaborn was used in order to plot the scatter matrix which resulted in the following output

### Scatter Matrix



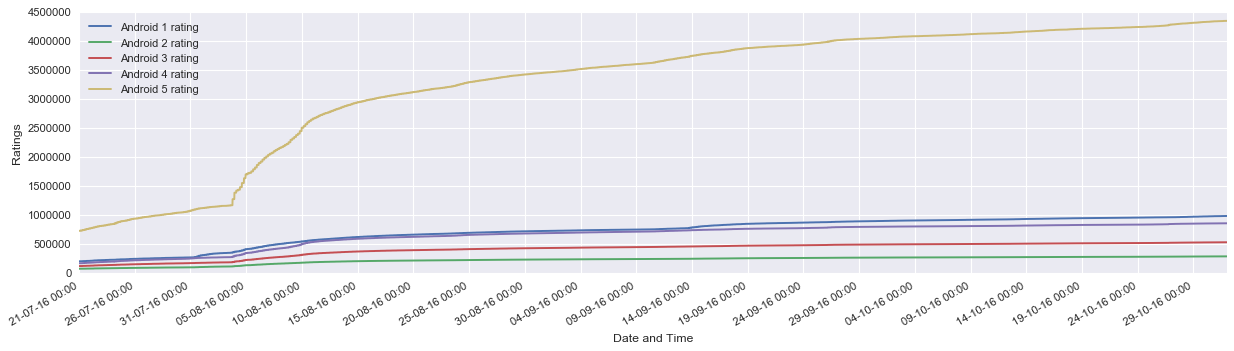
### Pearson Coefficient:

Perarson’s Coefficient was calculated for all the 55 combinations of the variables and keeping a threshold of +0.6 or -0.6, we have shortlisted to 49 pairs of variables.

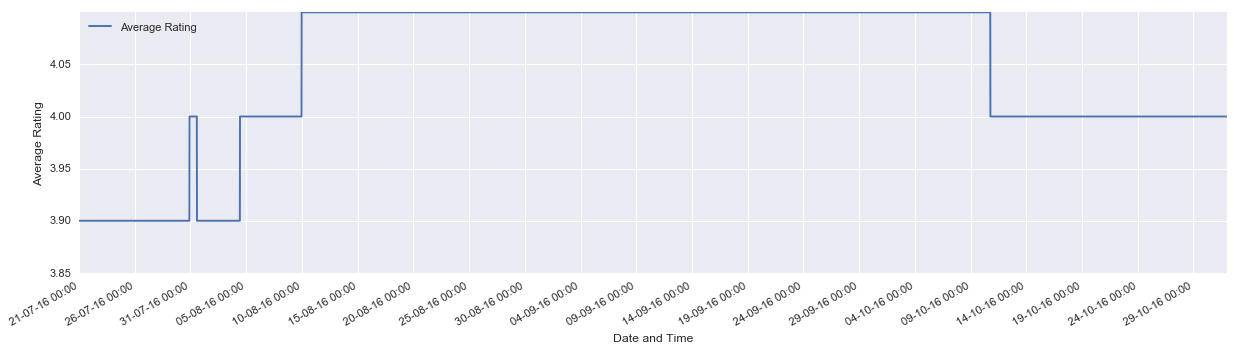
|  |  |
| --- | --- |
| Variable Combination | **Pearsons Coefficient** |
| **android\_avg\_rating + android\_rating\_2** | 0.624979764 |
| **android\_avg\_rating + android\_rating\_3** | 0.63144075 |
| **android\_avg\_rating + android\_rating\_4** | 0.631472791 |
| **android\_avg\_rating + android\_rating\_5** | 0.636710817 |
| **android\_avg\_rating + android\_total\_ratings** | 0.625247911 |
| **android\_avg\_rating + ios\_all\_ratings** | 0.698173832 |
| **android\_file\_size + android\_rating\_1** | 0.825892052 |
| **android\_file\_size + android\_rating\_2** | 0.76776074 |
| **android\_file\_size + android\_rating\_3** | 0.763082055 |
| **android\_file\_size + android\_rating\_4** | 0.76877924 |
| **android\_file\_size + android\_rating\_5** | 0.767654818 |
| **android\_file\_size + android\_total\_ratings** | 0.775967623 |
| **android\_file\_size + ios\_all\_ratings** | 0.666937275 |
| **android\_file\_size + ios\_file\_size** | 0.857201365 |
| **android\_rating\_1 + android\_rating\_2** | 0.994258048 |
| **android\_rating\_1 + android\_rating\_3** | 0.992427325 |
| **android\_rating\_1 + android\_rating\_4** | 0.9930505 |
| **android\_rating\_1 + android\_rating\_5** | 0.99285072 |
| **android\_rating\_1 + android\_total\_ratings** | 0.994734941 |
| **android\_rating\_1 + ios\_all\_ratings** | 0.950041779 |
| **android\_rating\_1 + ios\_current\_ratings** | -0.645667548 |
| **android\_rating\_1 + ios\_file\_size** | 0.871197048 |
| **android\_rating\_2 + android\_rating\_3** | 0.999493688 |
| **android\_rating\_2 + android\_rating\_4** | 0.999406688 |
| **android\_rating\_2 + android\_rating\_5** | 0.999457922 |
| **android\_rating\_2 + android\_total\_ratings** | 0.999660779 |
| **android\_rating\_2 + ios\_all\_ratings** | 0.967513259 |
| **android\_rating\_2 + ios\_current\_ratings** | -0.645666193 |
| **android\_rating\_2 + ios\_file\_size** | 0.84373554 |
| **android\_rating\_3 + android\_rating\_4** | 0.999891386 |
| **android\_rating\_3 + android\_rating\_5** | 0.999591062 |
| **android\_rating\_3 + android\_total\_ratings** | 0.999574905 |
| **android\_rating\_3 + ios\_all\_ratings** | 0.962866667 |
| **android\_rating\_3 + ios\_current\_ratings** | -0.659800236 |
| **android\_rating\_3 + ios\_file\_size** | 0.847384635 |
| **android\_rating\_4 + android\_rating\_5** | 0.999683613 |
| **android\_rating\_4 + android\_total\_ratings** | 0.999719937 |
| **android\_rating\_4 + ios\_all\_ratings** | 0.96221853 |
| **android\_rating\_4 + ios\_current\_ratings** | -0.664616146 |
| **android\_rating\_4 + ios\_file\_size** | 0.849746793 |
| **android\_rating\_5 + android\_total\_ratings** | 0.999838851 |
| **android\_rating\_5 + ios\_all\_ratings** | 0.964007589 |
| **android\_rating\_5 + ios\_current\_ratings** | -0.655635959 |
| **android\_rating\_5 + ios\_file\_size** | 0.848018682 |
| **android\_total\_ratings + ios\_all\_ratings** | 0.962825148 |
| **android\_total\_ratings + ios\_current\_ratings** | -0.655868754 |
| **android\_total\_ratings + ios\_file\_size** | 0.851885687 |
| **ios\_all\_ratings + ios\_file\_size** | 0.738353401 |
| **ios\_current\_ratings + ios\_file\_size** | -0.666957068 |

### Time Series Plot

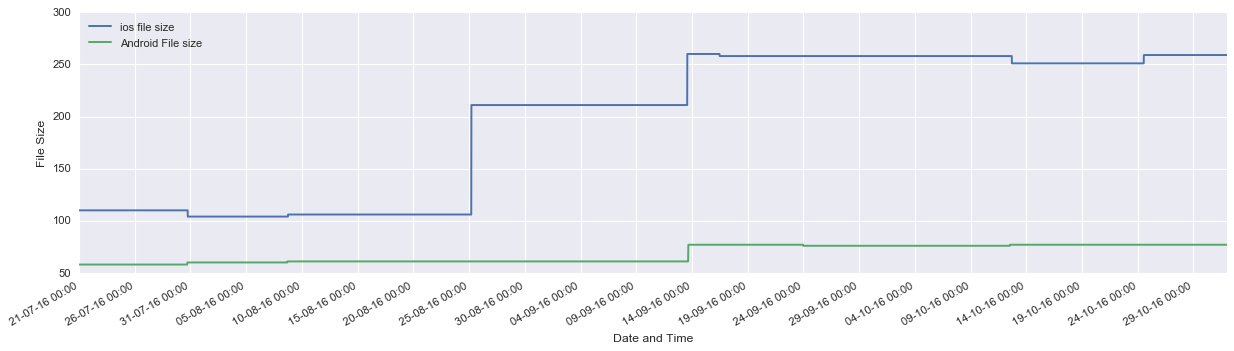
**Plot - Date and Time vs Android ratings 1/2/3/4/5**



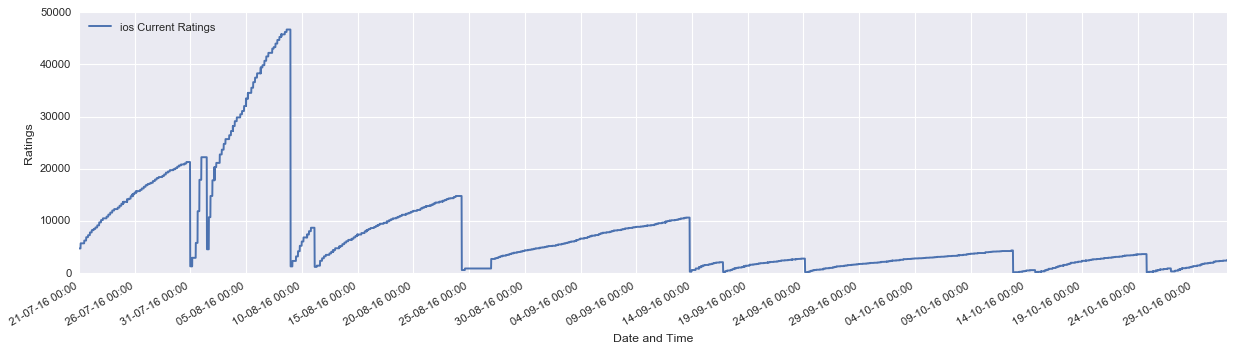
**Plot – Date and Time vs Android Average Rating**



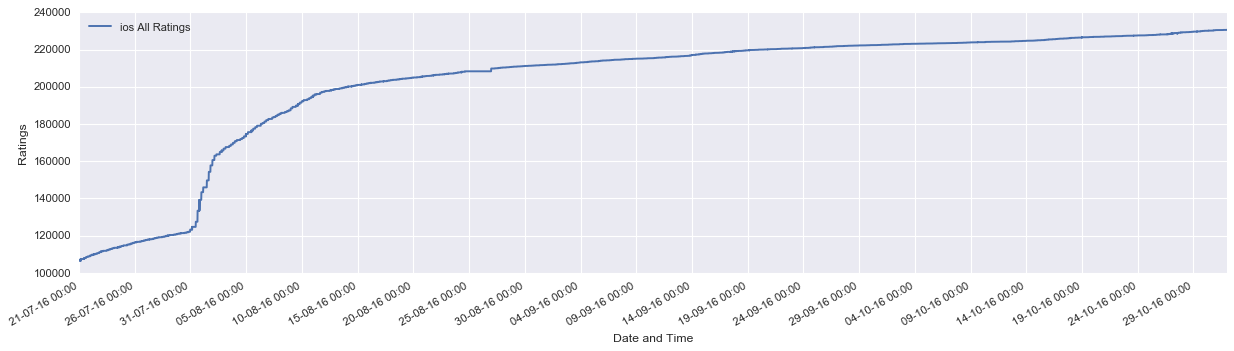
**Plot – Date and Time vs File Size**



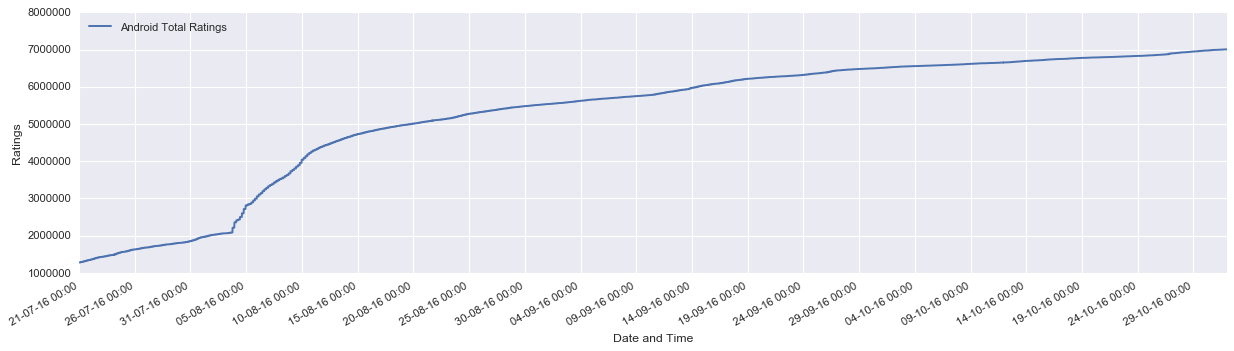
**Plot – Date and Time vs IOS current Ratings**



**Plot – Date and Time vs IOS all Ratings**



**Plot – Data and Time vs Android Total Ratings**



## Prediction Model

**File - modelbuilding.py**

In order to predict the Total Android Rating and Total IOS rating for **2016/11/01 11:50 PM** we have used 4 regression models ie Linear Regression, Ridge Regression, Lasso Regression and Elastic Net.

The performance of each model was measured using SkLearn Cross Validation Scores

As the models required an input in numerfic form the models were not excepting data objects as independent variable leaving us with 2 option.

Create dummy variable for each combination of data and time -This would create 14809 variables which was a bad idea as the dimensionality of the data would become very high and still we would not be able to give an ordinal touch to data- time combination which is critical

Convert the date and time in to single numeric entry – This option seemed feasible and in order to achieve the same we converted each datatime object to minutes away from the first date using *to\_seconds()/60*

As there is high co-relation between the dependent variables we required some regulation in order to contain the impact.

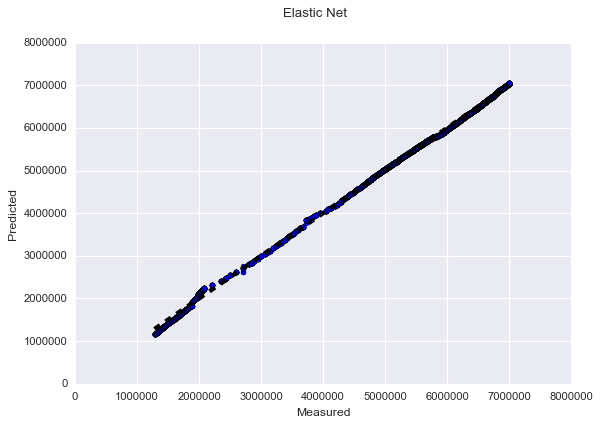
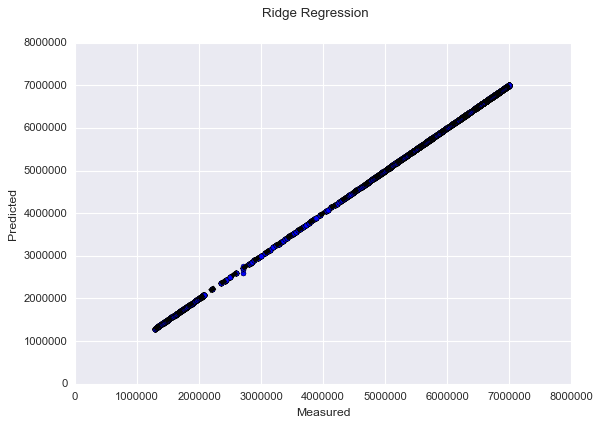
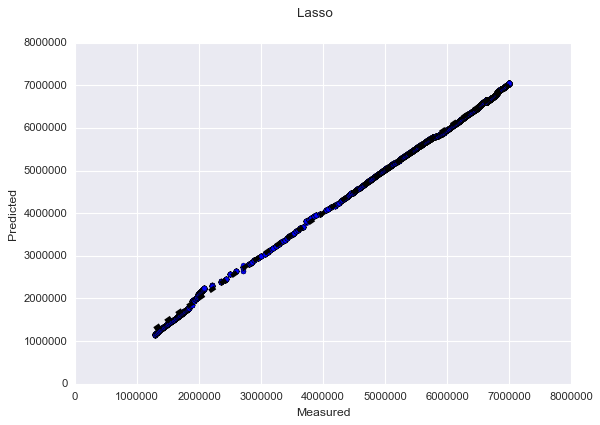
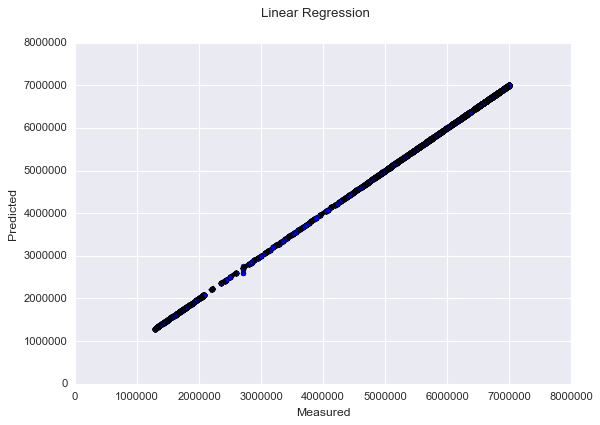
### Android Prediction:

In case of android prediction if we study the data the increase in Total Rating is slowing going down with time. We created all the 4 models using various combination of models in order to come up with the best suited model

The initial choice of model was based on the R2 and the Mean Square Error but when we have a higher R2 and low MSE we were basically dealing with an overfitted model. Hence more analysis on the data lead us to a decent and reasonable prediction.

Models were re-built using various suitable combination of variables and with changing the alpha values

#### Model Plots



#### Model Results:

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **R2** | **MSE** | **Prediction** |
| **Linear Regression** | 0.999991361732 | -2194150.24319 | 7005279.90704478 |
| **Ridge Regression** | 0.999991364115 | -2194146.64321 | 7005279.89715651 |
| **Lasso Regression** | 0.892904487674 | -1834346616.8 | 7034067.93989456 |
| **Elastic Net** | 0.910478487111 | -1717607207.93 | 7027585.25503652 |

Elastic net using both L1 and L2 regularization giving the best result with the hyper parameters as *alpha =0.01,fit\_intercept= True, l1\_ratio=0.7*

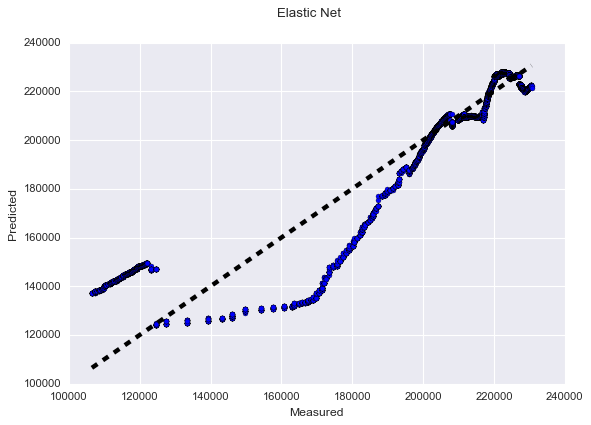
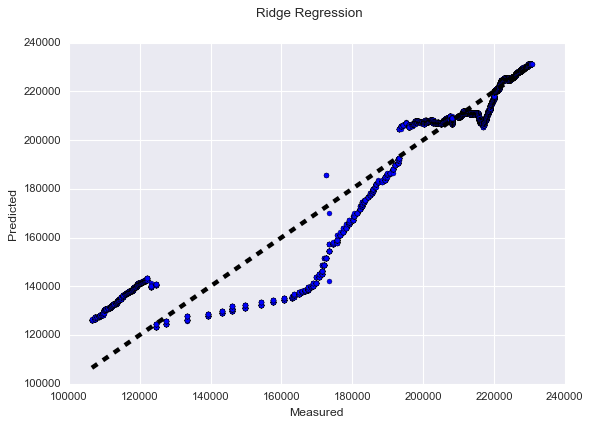
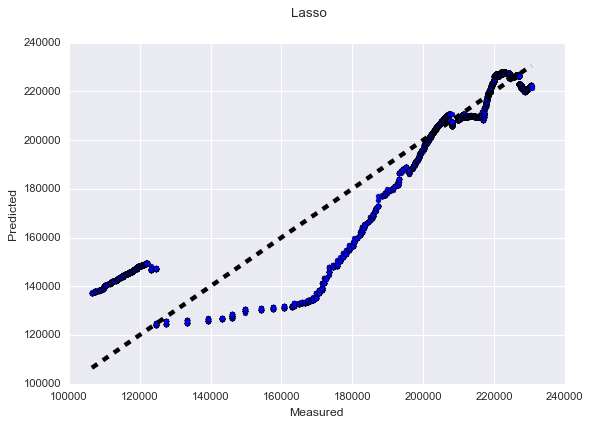
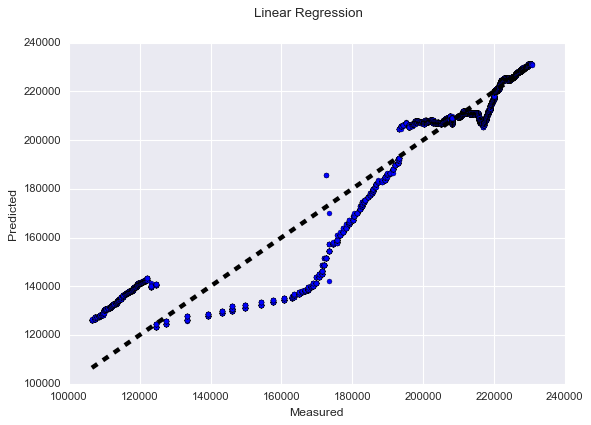
#### Final Prediction

**Android Total Ratings** for **2016/11/01 11:50 PM** is **7027585.255** using the variables *Android\_Avg\_Rating,* *Android\_Rating\_1, Android\_Rating\_2, Android\_Rating\_3, Android\_Rating\_4, Android\_Rating\_5 and IOS\_Current\_Rating.*

### iOS Prediction:

In generating the model for iOS there was no decent co-relation that could be found and hence obtained negative R2 values for all the model. Further analysis into the data helped in coming up with a decent model. In case of IOS prediction the situation was to find the best among not so good models.

#### Model Plots



#### Model Results:

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **R2** | **MSE** | **Prediction** |
| **Linear Regression** | -6.77882142885 | -82162341.9906 | 230028.1274333 |
| **Ridge Regression** | -6.77877619338 | -82162649.3867 | 230028.12743333 |
| **Lasso Regression** | -24.1205891346 | -152637009.195 | 225906.58476661 |
| **Elastic Net** | -24.1299022073 | -152331903.003 | 225903.42884409 |

In case of IOS prediction Ridge Regression and Linear Regression proved to give better resulted compared to Lasso and Elastic Net.

#### Final Prediction

**IOS Total Ratings** for **2016/11/01 11:50 PM** is **230028.127** using the variables *Android\_Total\_Ratings*,*Android\_Rating\_1, Android\_Rating\_2, Android\_Rating\_3, Android\_Rating\_4, Android\_Rating\_5 and IOS\_Current\_Rating.*

In case of IOS prediction the predicted values is less than the last reported values of **230601** as of **2016-10-31 23:50:00.**

However, considering we are dealing with sample and considering the Margin of Error the prediction can be decent enough as the increase in IOS total rating per day has also dropped considerably.

# Deep Learning

Tensor flow image recognition was used as a tool for getting the top 5 image tags along with the probability.

## Step 1: Web Scrapping

**File - imagedata\_android.py, imagedata\_ios.py**

All the android and IOS images web scrapped using Beautiful soup and OS module and Image urls were retrieved

The URLs were then added to a set so that we are left with Unique application screen shots

## Step 2: Download Images

**File - downloadimage.py**

The set containing unique screenshot of Android and IOS were then downloaded using *urllib.urlretrieve*  method.

The downloaded images of Android were not in the required image format ie JPEG as tensor flow requires the input in JPEG format we have to preprocess them

Android Images were converted to appropriate JPEG formats using Image module were in all the images were coverted into RGB and then into JPEG. The challenge faced while using image module was that it was not supported in a 64 bit system.

**File - Covert\_JPEG.py**

Hence we had to arrange a 32 bit system to convert the images into JPEG using Python Image module.

## Step 3: Deep Learning

**File - classify\_image1.py**

Tensor flow image classification Image net pre-trained model was used to get the image classification

Each image was iterated over to get the tags and their respective probabilities

## Results:

Total Android Unique Images – **5**

Total IOS Unique Images – **17**

### Android Images

**android\_image\_0**

Web site, website, internet site, site (score = 0.59768)

sunglasses, dark glasses, shades (score = 0.04691)

electric fan, blower (score = 0.03147)

sunglass (score = 0.02155)

comic book (score = 0.02090)



**android\_image\_1**

web site, website, internet site, site (score = 0.69588),

television, television system (score = 0.02657),

monitor (score = 0.01863),

pool table, billiard table, snooker table (score = 0.01564),

screen, CRT screen (score = 0.01539)



**android\_image\_2**

lawn mower, mower (score = 0.30521),

golf ball (score = 0.06549),

bow (score = 0.03530),

croquet ball (score = 0.02764),

barrow, garden cart, lawn cart, wheelbarrow (score = 0.02662)



**android\_image\_3**

web site, website, internet site, site (score = 0.54017),

monitor (score = 0.05432),

notebook, notebook computer (score = 0.03848),

television, television system (score = 0.03007),

maillot (score = 0.01110)



**android\_image\_4**

ant, emmet, pismire (score = 0.13091),

monitor (score = 0.08763),

aircraft carrier, carrier, flattop, attack aircraft carrier (score = 0.07103),

wing (score = 0.05027),

web site, website, internet site, site (score = 0.04293)



### IOS Images

**ios\_image\_0**

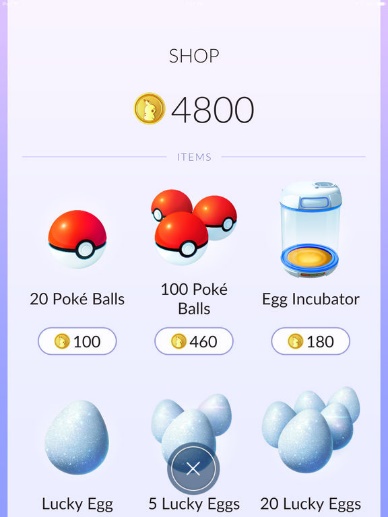
web site, website, internet site, site (score = 0.88357),

menu (score = 0.00803),

slot, one-armed bandit (score = 0.00404),

washer, automatic washer, washing machine (score = 0.00371),

hand-held computer, hand-held microcomputer (score = 0.00296)



**ios\_image\_1**

comic book (score = 0.19361),

maze, labyrinth (score = 0.19330),

web site, website, internet site, site (score = 0.05236),

monitor (score = 0.02957),

book jacket, dust cover, dust jacket, dust wrapper (score = 0.02767)



**ios\_image\_2**

comic book (score = 0.19361),

maze, labyrinth (score = 0.19330),

web site, website, internet site, site (score = 0.05236),

monitor (score = 0.02957),

book jacket, dust cover, dust jacket, dust wrapper (score = 0.02767)



**ios\_image\_3**

web site, website, internet site, site (score = 0.22753),

envelope (score = 0.09163),

Band Aid (score = 0.03712),

pinwheel (score = 0.02946),

airship, dirigible (score = 0.02486)



**ios\_image\_4**

laptop, laptop computer (score = 0.49859),

web site, website, internet site, site (score = 0.10646),

monitor (score = 0.06384),

screen, CRT screen (score = 0.02985),

notebook, notebook computer (score = 0.02801)



**ios\_image\_5**

web site, website, internet site, site (score = 0.36619),

safety pin (score = 0.02004),

sunglasses, dark glasses, shades (score = 0.01677),

toilet seat (score = 0.01562),

washer, automatic washer, washing machine (score = 0.01438)



**ios\_image\_6**

web site, website, internet site, site (score = 0.89077),

menu (score = 0.00364),

monitor (score = 0.00185),

screen, CRT screen (score = 0.00184),

analog clock (score = 0.00177)



**ios\_image\_7**

web site, website, internet site, site (score = 0.11637),

laptop, laptop computer (score = 0.08080),

notebook, notebook computer (score = 0.05349),

joystick (score = 0.04791),

monitor (score = 0.04169)



**ios\_image\_8**

web site, website, internet site, site (score = 0.11637),

laptop, laptop computer (score = 0.08080),

notebook, notebook computer (score = 0.05349),

joystick (score = 0.04791),

monitor (score = 0.04169)



**ios\_image\_9**

web site, website, internet site, site (score = 0.60886),

television, television system (score = 0.05665),

monitor (score = 0.01996),

notebook, notebook computer (score = 0.01607),

iPod (score = 0.01180)



**ios\_image\_10**

web site, website, internet site, site (score = 0.58624),

monitor (score = 0.07197),

television, television system (score = 0.05955),

comic book (score = 0.04756),

teapot (score = 0.01425)



**ios\_image\_11**

web site, website, internet site, site (score = 0.12342),

maze, labyrinth (score = 0.07149),

comic book (score = 0.04789),

joystick (score = 0.04421),

television, television system (score = 0.03758)



**ios\_image\_12**

fountain (score = 0.20303),

carousel, carrousel, merry-go-round, roundabout, whirligig (score = 0.08314),

comic book (score = 0.05171),

toyshop (score = 0.03343),

monitor (score = 0.03227)



**ios\_image\_13**

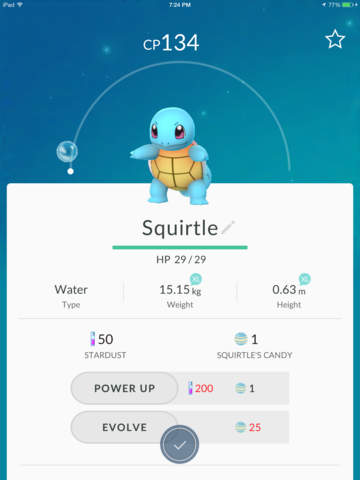
web site, website, internet site, site (score = 0.94092),

analog clock (score = 0.00367),

envelope (score = 0.00291),

monitor (score = 0.00225),

screen, CRT screen (score = 0.00217)



**ios\_image\_14**

web site, website, internet site, site (score = 0.36779),

envelope (score = 0.16914),

binder, ring-binder (score = 0.05812),

tray (score = 0.01764),

monitor (score = 0.01721)



**ios\_image\_15**

space shuttle (score = 0.23042),

joystick (score = 0.05992),

racer, race car, racing car (score = 0.05626),

scoreboard (score = 0.04957),

airliner (score = 0.04576)



**ios\_image\_16**

aircraft carrier, carrier, flattop, attack aircraft carrier (score = 0.09968),

pole (score = 0.03657),

wing (score = 0.02655),

lakeside, lakeshore (score = 0.02437),

magnetic compass (score = 0.02396)

