Mercari Price Suggestion

1. Business Problem

1.1 Description

It can be hard to know how much something's really worth. Small details can mean big differences in pricing.

Product pricing gets even harder at scale, considering just how many products are sold online. Clothing has strong seasonal pricing trends and is heavily influenced by brand names, while electronics have fluctuating prices based on product specs.

Mercari, Japan's biggest community-powered shopping app, knows this problem deeply. They'd like to offer pricing suggestions to sellers, but this is tough because their sellers are enabled to put just about anything, or any bundle of things, on Mercari's marketplace.

In this case study, we try to build an algorithm that automatically suggests the right product prices. The input consists of user-inputted text descriptions of their products, including details like product category name, brand name, and item condition.

> Credits: Kaggle

Problem Statement:-

- Suggest the price to the product based on the input given by the user.

1.2 Sources/Useful Links

- Source : https://www.kaggle.com/c/mercari-price-suggestion-challenge (https://www.kaggle.com/c/mercari-price-suggestion-challenge)

2. Machine Learning Probelm

2.1 Data

2.1.1 Data Overview

All of the data is in 2 files: Train and Test.

Train.tsv contains 8 columns: train_id, name, item_condition_id, category_name, brand_name, shipping, item_description, price.

Test.tsv contains the same columns but without the price, which is to be predicted.

Size of Train.tsv - 322 MB

Size of Test.tsv - 147 MB

Number of rows in Train.tsv = 1482535

Data Field Explaination

Train Dataset contains 1,482,535 rows.
Test Dataset contains 693,359 rows.
The columns in the table are:

train_id or test_id - the id of the listing

name - the title of the listing

item_condition_id - the condition of the items provided by the seller

category_name - category of the listing

brand_name -

price - the price that the item was sold for. This is the target variable that you will predict. The unit is USD. This column doesn't exist in test.tsv

shipping - boolean value, 1 if shipping fee is paid by seller and 0 by buyer

item_description - the full description of the item

2.1.2 Example Data point

```
train_id    name    item_condition_id    category_name    brand_name
price    shipping    item_description
0    MLB Cincinnati Reds T Shirt Size XL    3    Men/Tops/T-shirts
```

```
10.0    1    No description yet
1    Razer BlackWidow Chroma Keyboard    3    Electronics/Computers & Ta
blets/Components & Parts    Razer    52.0    0    This keyboard is in gr
eat condition and works like it came out of the box. All of the ports ar
e tested and work perfectly. The lights are customizable via the Razer S
ynapse app on your PC.
2    AVA-VIV Blouse    1    Women/Tops & Blouses/Blouse    Target    10.
0    1    Adorable top with a hint of lace and a key hole in the back! T
he pale pink is a 1X, and I also have a 3X available in white!
3    Leather Horse Statues    1    Home/Home Décor/Home Décor Accents
35.0    1    New with tags. Leather horses. Retail for [rm] each. Stand
about a foot high. They are being sold as a pair. Any questions please a
sk. Free shipping. Just got out of storage
4    24K GOLD plated rose    1    Women/Jewelry/Necklaces        44.0
0    Complete with certificate of authenticity
```

2.2 Mapping the real world problem to an ML problem

2.2.1 Type of Machine Leaning Problem

It is a Regression problem, for a given input information about the item we need ti predict the price.

2.2.2 Performance Metric

Source: [https://www.kaggle.com/c/mercari-price-suggestion-challenge/overview/evaluation (https://www.kaggle.com/c/mercari-price-suggestion-challenge/overview/evaluation)](https://www.kaggle.com/c/mercari-price-suggestion-challenge/overview/evaluation)

The evaluation metric for this competition is Root Mean Squared Logarithmic Error.

The RMSLE is calculated as

$$\epsilon = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(p_i + 1) - \log(a_i + 1))^2}$$

Where:

ϵ is the RMSLE value (score)
n is the total number of observations in the data set,
p_i is your prediction of price,

a_i is the actual sale price for i.
log(x) is the natural logarithm of x

Type *Markdown* and LaTeX: α^2

Type *Markdown* and LaTeX: α^2

Type *Markdown* and LaTeX: α^2

3. Exploratory Data Analysis

In [1]:

```
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from wordcloud import WordCloud
import re
import os
import datetime as dt
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem.snowball import SnowballStemmer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from datetime import datetime
```

Loading the data

In [2]:
```
#Loading train data
data_train=pd.read_csv('train.tsv',sep='\t')
print('Shape of the train data is : ',data_train.shape)
data_train.head()
```

```
Shape of the train data is :  (1482535, 8)
```

Out[2]:

	train_id	name	item_condition_id	category_name	brand_name	price	shippi
0	0	MLB Cincinnati Reds T Shirt Size XL	3	Men/Tops/T-shirts	NaN	10.0	
1	1	Razer BlackWidow Chroma Keyboard	3	Electronics/Computers & Tablets/Components & P...	Razer	52.0	
2	2	AVA-VIV Blouse	1	Women/Tops & Blouses/Blouse	Target	10.0	
3	3	Leather Horse Statues	1	Home/Home Décor/Home Décor Accents	NaN	35.0	
4	4	24K GOLD plated rose	1	Women/Jewelry/Necklaces	NaN	44.0	

In [3]:
```
#creating copies of train data
train=data_train.copy()
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1482535 entries, 0 to 1482534
Data columns (total 8 columns):
train_id             1482535 non-null int64
name                 1482535 non-null object
item_condition_id    1482535 non-null int64
category_name        1476208 non-null object
brand_name           849853 non-null object
price                1482535 non-null float64
shipping             1482535 non-null int64
item_description     1482531 non-null object
dtypes: float64(1), int64(3), object(4)
memory usage: 90.5+ MB
```

In [0]:

In [0]:
```
#checking for null values in columns
train.isnull().any()
```

Out[12]:
```
train_id             False
name                 False
item_condition_id    False
category_name         True
brand_name            True
price                False
shipping             False
item_description      True
dtype: bool
```

- The columns category_name , brand_name , item_description have null values

In [4]:
```
#filling null values
train.category_name.fillna(value="Unknown/Unknown/Unknown", inplace = True)
train.brand_name.fillna(value="Unknown", inplace = True)
train.item_description.fillna(value="No description yet", inplace = True)
```

Price

In [0]:
```
train['price'].describe()
```

Out[32]:
```
count    1.482535e+06
mean     2.673752e+01
std      3.858607e+01
min      0.000000e+00
25%      1.000000e+01
50%      1.700000e+01
75%      2.900000e+01
max      2.009000e+03
Name: price, dtype: float64
```

In [0]:
```
#@title Default title text
price=train['price'].values
log_price = np.log1p(price)
```

In [0]:
```
p=np.array(price)
r=np.arange(10,110,10)
q1=np.percentile(p,r)
for i in range(len(r)):
    print(str(r[i])+'th percentile value of price =',q1[i])
```

```
10th percentile value of price = 7.0
20th percentile value of price = 10.0
30th percentile value of price = 12.0
40th percentile value of price = 14.0
50th percentile value of price = 17.0
60th percentile value of price = 20.0
70th percentile value of price = 26.0
80th percentile value of price = 34.0
90th percentile value of price = 51.0
100th percentile value of price = 2009.0
```

In [0]:
```
p=np.array(price)
r=np.arange(90,101,1)
q1=np.percentile(p,r)
for i in range(len(r)):
    print(str(r[i])+'th percentile value of price =',q1[i])
```

```
90th percentile value of price = 51.0
91th percentile value of price = 55.0
92th percentile value of price = 58.0
93th percentile value of price = 62.0
94th percentile value of price = 67.0
95th percentile value of price = 75.0
96th percentile value of price = 85.0
97th percentile value of price = 99.0
98th percentile value of price = 122.0
99th percentile value of price = 170.0
100th percentile value of price = 2009.0
```

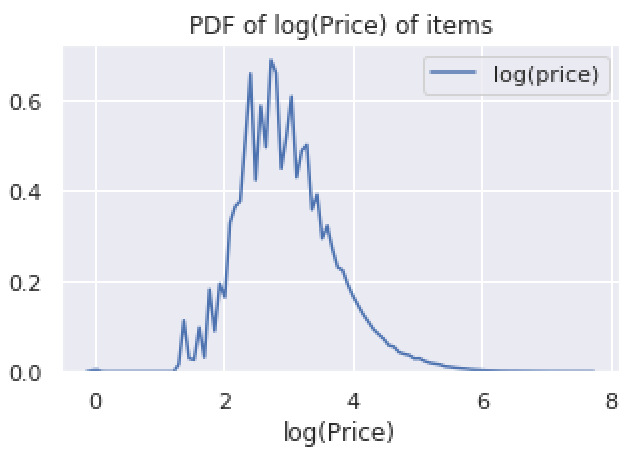
PDF of price :-

In [0]:
```
plt.figure(figsize=(5,3))
sns.distplot(price, hist=False, label="price")
plt.title('PDF of Price of items')
plt.xlabel('Price')
plt.legend()
```

Out[63]: <matplotlib.legend.Legend at 0x7f78d79dafd0>

PDF of log(price) :-

In [0]:
```
plt.figure(figsize=(5,3))
sns.distplot(log_price, hist=False, label="log(price)")
plt.title('PDF of log(Price) of items')
plt.xlabel('log(Price)')
plt.legend()
plt.show()
```

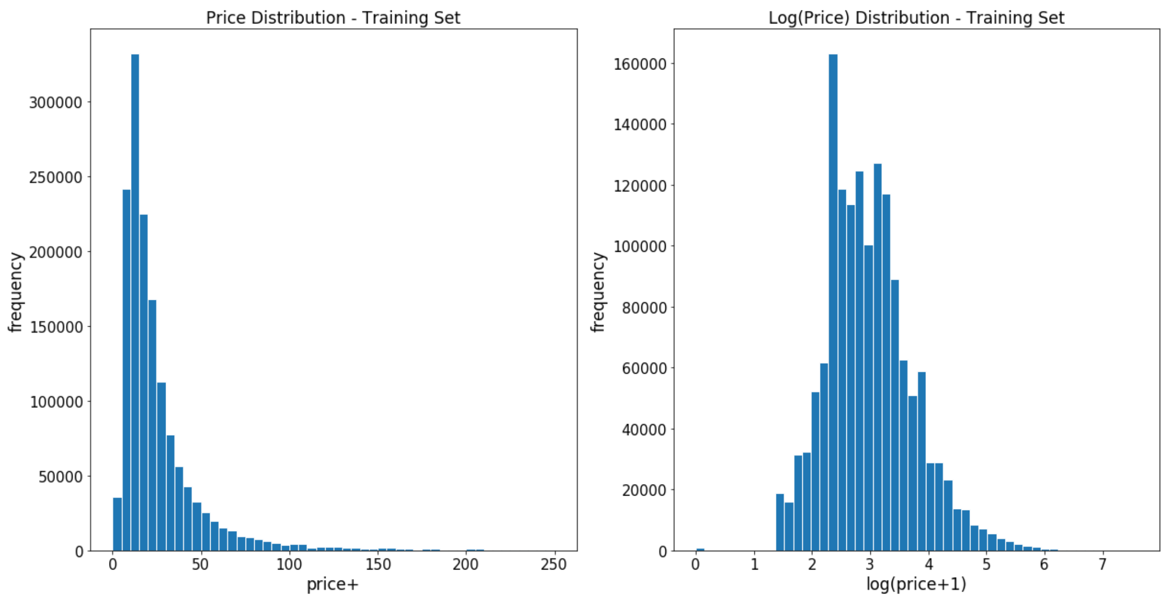


Histograms

In [0]:

```
plt.subplot(1, 2, 1)
(train['price']).plot.hist(bins=50, figsize=(20,10), edgecolor='white',range=
plt.xlabel('price+', fontsize=17)
plt.ylabel('frequency', fontsize=17)
plt.tick_params(labelsize=15)
plt.title('Price Distribution - Training Set', fontsize=17)

plt.subplot(1, 2, 2)
np.log(train['price']+1).plot.hist(bins=50, figsize=(20,10), edgecolor='white
plt.xlabel('log(price+1)', fontsize=17)
plt.ylabel('frequency', fontsize=17)
plt.tick_params(labelsize=15)
plt.title('Log(Price) Distribution - Training Set', fontsize=17)
plt.show()
```



Observations :-

- 90 % of data points have price less than 51$
- Log(price) distribution is more symmetric when compared to price distribution
- That is why we use RMLSE as error metric and not RMSE

Splitting category_name

In [5]:

```
#splitting the category_name column into 3 columns-main_category,sub_cat1,sub
category=list(train['category_name'].values)

main_cat=[]
sub_cat1=[]
sub_cat2=[]
for i in range(len(category)):
    cat=category[i].split("/")
    main_cat.append(cat[0])
    sub_cat1.append(cat[1])
    sub_cat2.append(cat[2])

train['main_category']=main_cat
train['sub_cat1']=sub_cat1
train['sub_cat2']=sub_cat2

#dropping the column category_name
train.drop('category_name', axis=1, inplace=True)
```

In [7]:

```
train.head(2)
```

Out[7]:

	train_id	name	item_condition_id	brand_name	price	shipping	item_description	mai
0	0	MLB Cincinnati Reds T Shirt Size XL	3	Unknown	10.0	1	No description yet	
1	1	Razer BlackWidow Chroma Keyboard	3	Razer	52.0	0	This keyboard is in great condition and works ...	

Main_category

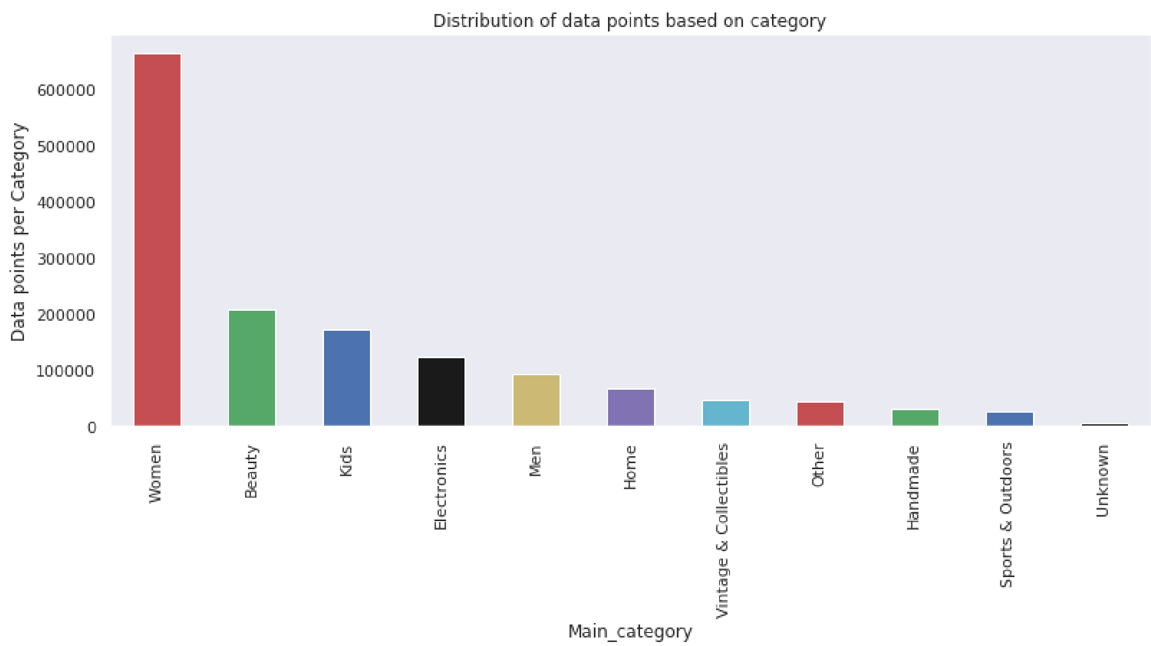
In [0]:

```
main_cat=train['main_category'].value_counts()
print('Number of unique main categories : ',main_cat.size)

my_colors = ['r','g','b','k','y','m','c']
main_cat.plot(kind='bar',color=my_colors)
plt.xlabel('Main_category')
plt.ylabel('Data points per Category')
plt.title('Distribution of data points based on category')
plt.grid()
plt.show()


keys=list(main_cat.keys())
values=list(main_cat.values)
percentage=[]
for i in range(len(main_cat)):
    percent=np.round(float(values[i]/len(train))*100,2)
    percentage.append(percent)
df=pd.DataFrame()
df['Main_Category']=keys
df['data points count']=values
df['%']=percentage
df
```

```
Number of unique main categories :  11
```



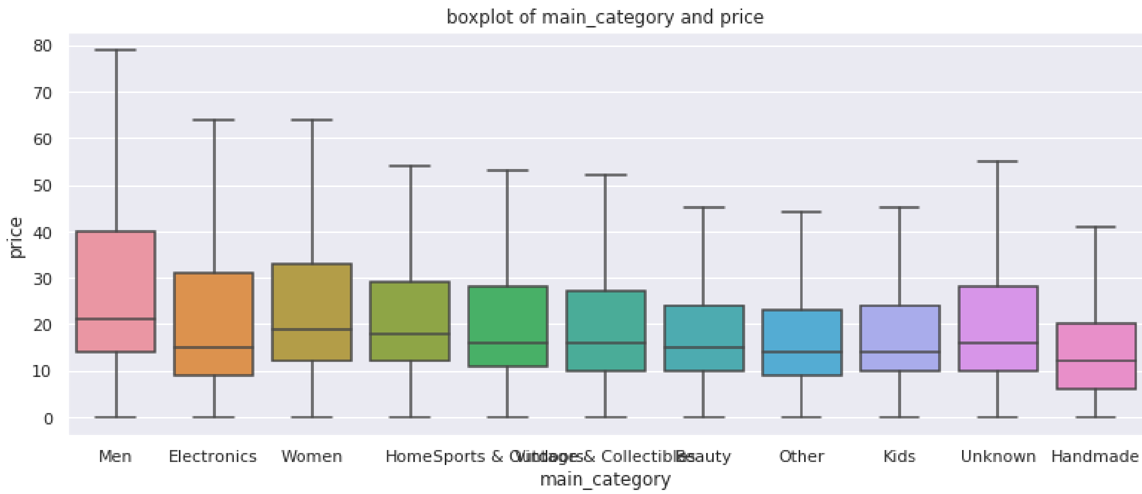
Out[35]:

	Main_Category	data points count	%
0	Women	664385	44.81
1	Beauty	207828	14.02
2	Kids	171689	11.58
3	Electronics	122690	8.28
4	Men	93680	6.32
5	Home	67871	4.58
6	Vintage & Collectibles	46530	3.14
7	Other	45351	3.06
8	Handmade	30842	2.08
9	Sports & Outdoors	25342	1.71
10	Unknown	6327	0.43

Observations :-

- There are a total 11 unique main categories
- Women category itself contains almost 45% of the data points
- The top 3 categories Women,Beauty,Kids contain 60% of data points
- 0.43% of the data do not contain the category information

In [0]:
```
#boxplot of byte files
sns.set(rc={'figure.figsize':(13,5)})
ax = sns.boxplot(x=train['main_category'], y=train['price'],showfliers = Fals
plt.title("boxplot of main_category and price")
plt.plot(figsize=(30,30))
plt.show()
```



In [0]:
```

```

Observations :-

- The data points belonging to Men category have a highest price range when compared to other categories
- The data points belonging to Handmade category have a lowest price range when compared to other categories

sub_cat1

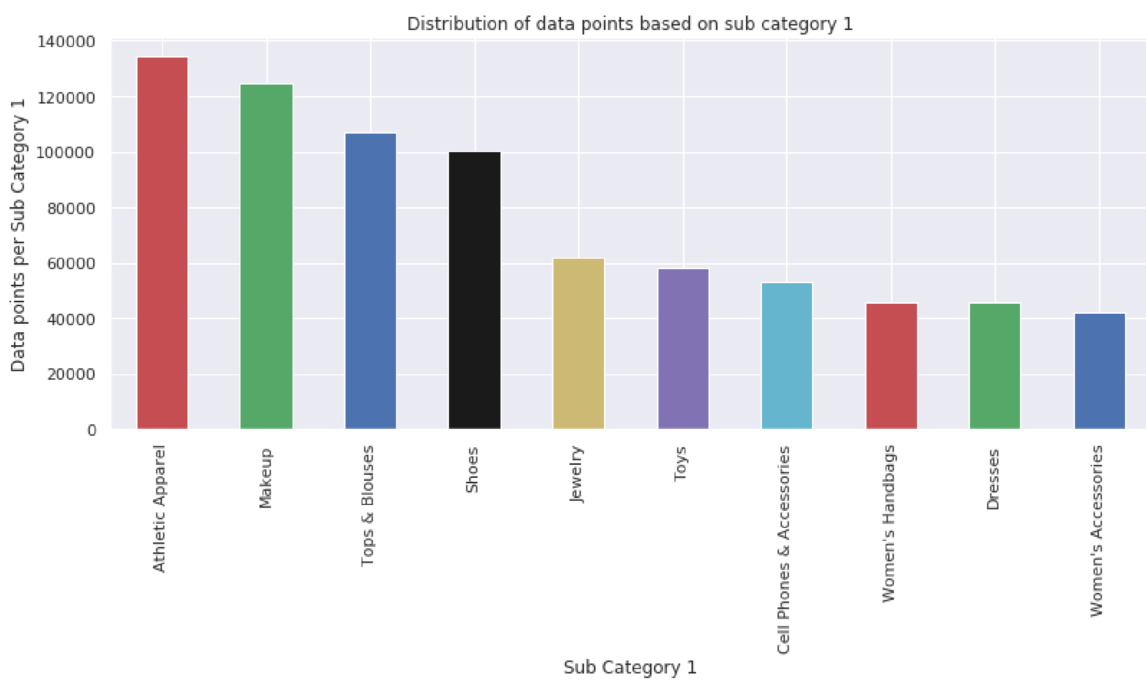
In [0]:

```
sub_cat1=train['sub_cat1'].value_counts()
print('Number of unique sub categories 1 : ',sub_cat1.size)

my_colors = ['r','g','b','k','y','m','c']
sub_cat1[0:10].plot(kind='bar',color=my_colors)
plt.xlabel('Sub Category 1')
plt.ylabel('Data points per Sub Category 1')
plt.title('Distribution of data points based on sub category 1')
plt.show()


keys=list(sub_cat1[0:10].keys())
values=list(sub_cat1[0:10].values)
percentage=[]
for i in range(len(sub_cat1[0:10])):
    percent=np.round(float(values[i]/len(train))*100,2)
    percentage.append(percent)
df=pd.DataFrame()
df['sub_cat1']=keys
df['data points count']=values
df['%']=percentage
df
```

```
Number of unique sub categories 1 :  114
```



Out[34]:

	sub_cat1	data points count	%
0	Athletic Apparel	134383	9.06
1	Makeup	124624	8.41
2	Tops & Blouses	106960	7.21
3	Shoes	100452	6.78
4	Jewelry	61763	4.17
5	Toys	58158	3.92
6	Cell Phones & Accessories	53290	3.59
7	Women's Handbags	45862	3.09
8	Dresses	45758	3.09
9	Women's Accessories	42350	2.86

Observations :-

- There are a total 114 unique sub categories 1
- Athletic Apparel sub category1 contains more number of data points

sub_cat2

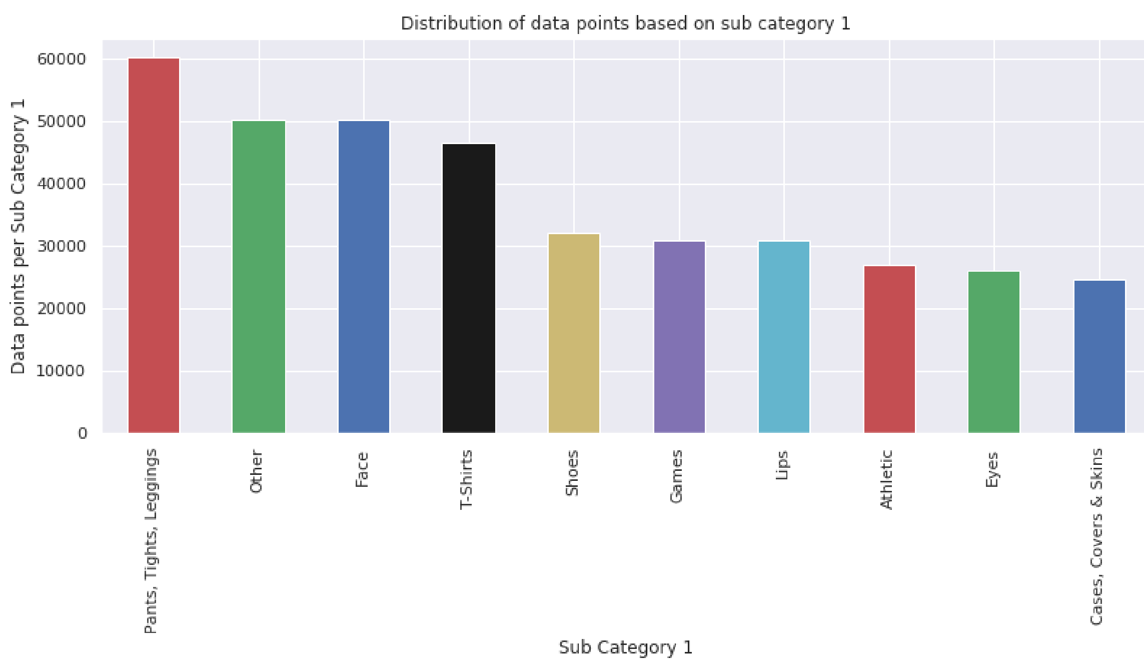
In [0]:

```
sub_cat2=train['sub_cat2'].value_counts()
print('Number of unique sub categories 1 : ',sub_cat2.size)

my_colors = ['r','g','b','k','y','m','c']
sub_cat2[0:10].plot(kind='bar',color=my_colors)
plt.xlabel('Sub Category 1')
plt.ylabel('Data points per Sub Category 1')
plt.title('Distribution of data points based on sub category 1')
plt.show()


keys=list(sub_cat2[0:10].keys())
values=list(sub_cat2[0:10].values)
percentage=[]
for i in range(len(sub_cat2[0:10])):
    percent=np.round(float(values[i]/len(train))*100,2)
    percentage.append(percent)
df=pd.DataFrame()
df['sub_cat2']=keys
df['data points count']=values
df['%']=percentage
df
```

```
Number of unique sub categories 1 :  871
```



Out[36]:

	sub_cat2	data points count	%
0	Pants, Tights, Leggings	60177	4.06
1	Other	50224	3.39
2	Face	50171	3.38
3	T-Shirts	46380	3.13
4	Shoes	32168	2.17
5	Games	30906	2.08

	sub_cat2	data points count	%
6	Lips	30871	2.08
7	Athletic	27059	1.83
8	Eyes	26038	1.76
9	Cases, Covers & Skins	24676	1.66

Observations :-

- There are a total 871 unique sub categories2
- Pants, Tights, Leggings sub category2 contains more number of data points

brand_name

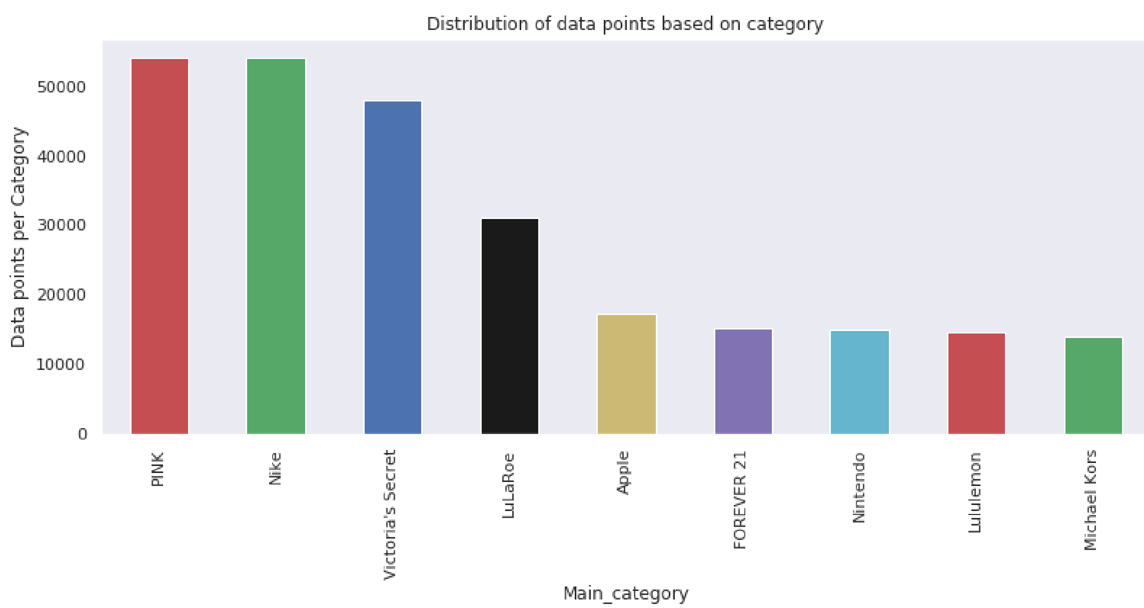
In [0]:

```
brand=train['brand_name'].value_counts()
print('Number of unique brands : ',brand.size)

my_colors = ['r','g','b','k','y','m','c']
brand[1:10].plot(kind='bar',color=my_colors)
plt.xlabel('Main_category')
plt.ylabel('Data points per Category')
plt.title('Distribution of data points based on category')
plt.grid()
plt.show()

keys=list(brand[1:10].keys())
values=list(brand[1:10].values)
percentage=[]
for i in range(len(brand[1:10])):
    percent=np.round(float(values[i]/len(train))*100,2)
    percentage.append(percent)
df=pd.DataFrame()
df['Brand']=keys
df['data points count']=values
df['%']=percentage
df
```

```
Number of unique brands :  4810
```



Out[37]:

	Brand	data points count	%
0	PINK	54088	3.65
1	Nike	54043	3.65
2	Victoria's Secret	48036	3.24
3	LuLaRoe	31024	2.09
4	Apple	17322	1.17
5	FOREVER 21	15186	1.02
6	Nintendo	15007	1.01

	Brand	data points count	%
7	Lululemon	14558	0.98
8	Michael Kors	13928	0.94

Observations :-

- There are a total 4810 unique brands
- PINK and Nike have almost same number of data points

Item_condition_id

In [0]:

```
condition=train['item_condition_id'].value_counts()
condition
```

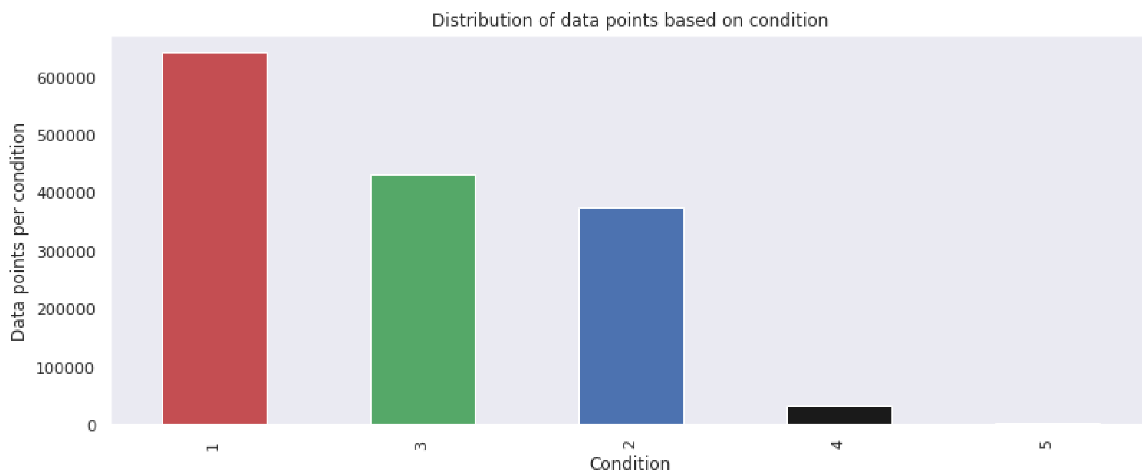
Out[44]:

```
1    640549
3    432161
2    375479
4     31962
5      2384
Name: item_condition_id, dtype: int64
```

In [0]:

```
my_colors = ['r','g','b','k','y']
condition.plot(kind='bar',color=my_colors)
plt.xlabel('Condition')
plt.ylabel('Data points per condition')
plt.title('Distribution of data points based on condition')
plt.grid()
plt.show()

keys=list(condition.keys())
values=list(condition.values)
percentage=[]
for i in range(len(condition)):
    percent=np.round(float(values[i]/len(train))*100,2)
    percentage.append(percent)
df=pd.DataFrame()
df['Item_Condition_Id']=keys
df['data points count']=values
df['%']=percentage
df
```



Out[45]:

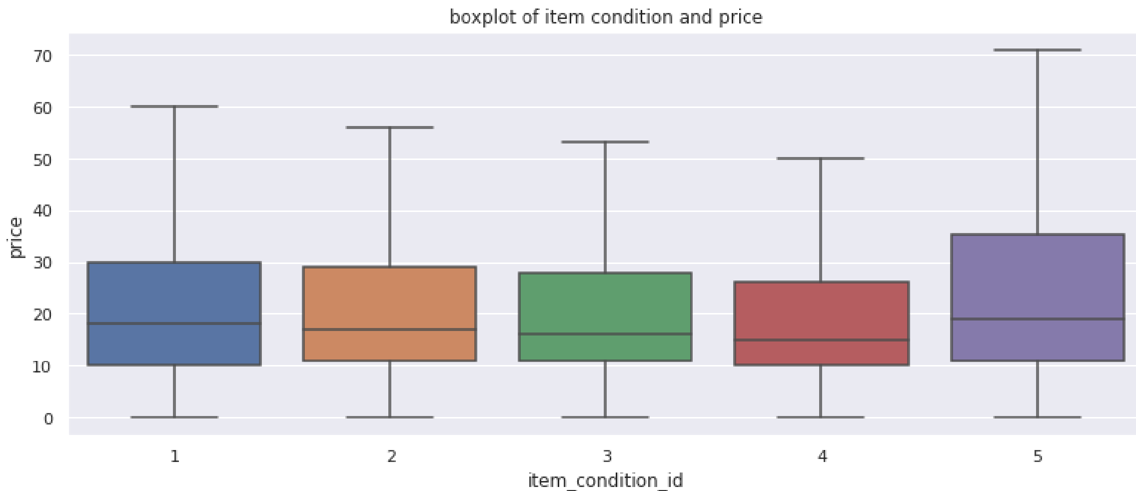
	Item_Condition_Id	data points count	%
0	1	640549	43.21
1	3	432161	29.15
2	2	375479	25.33
3	4	31962	2.16
4	5	2384	0.16

Observations :-

- There are a total 5 Item_Condition_Id's
- Item_Condition_Id '1' has 43.21% of the data points
- The Item_Condition_Id's '4' and '5' have less than 3% of data points

In [0]:

```
#boxplot of byte files
ax = sns.boxplot(x=train['item_condition_id'], y=train['price'],showfliers =
plt.title("boxplot of item condition and price")
plt.show()
```



Observations :-

- The items having Item_Condition_Id '5' have a highest price range when compared to other Item_Condition_Id's

shipping

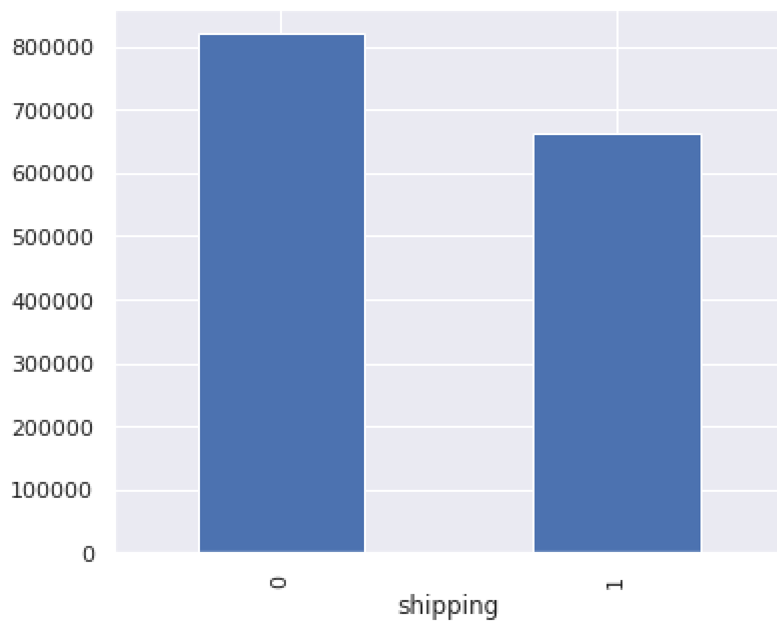
In [0]:

```
shipping=train['shipping'].value_counts()
train.groupby("shipping")['train_id'].count().plot.bar()

keys=list(shipping.keys())
values=list(shipping.values)
percentage=[]
for i in range(len(shipping)):
    percent=np.round(float(values[i]/len(train))*100,2)
    percentage.append(percent)
df=pd.DataFrame()
df['shipping']=keys
df['data points count']=values
df['%']=percentage
df
```

Out[51]:

	shipping	data points count	%
0	0	819435	55.27
1	1	663100	44.73



Observations :-

- Shipping = 0 ,shipping is paid by buyer
- Shipping = 1 ,shipping is paid by seller
- Items with shipping value = 0 are more in number than items with items with shipping value = 1

In [0]:
```
#boxplot of byte files
sns.set(rc={'figure.figsize':(6,5)})
ax = sns.boxplot(x=train['shipping'], y=train['price'],showfliers = False, or
plt.title("boxplot of shipping fee and price")
plt.show()
```

Observations :-

- The price range of items with shipping value = 0 is more when compared to items with items with shipping value = 1

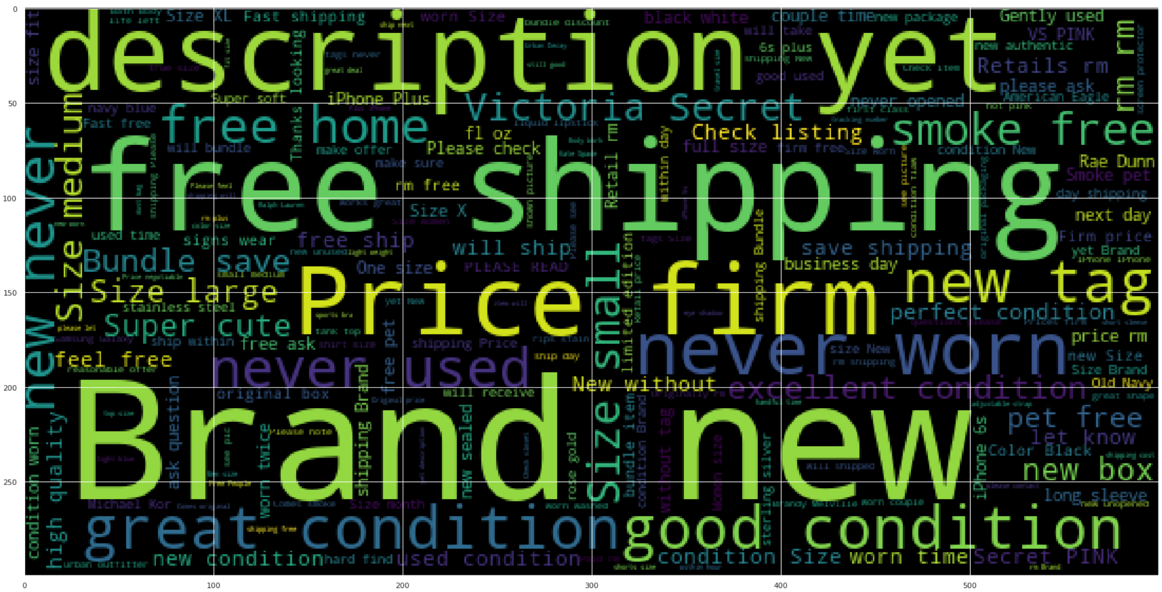
item_description

In [0]:
```
desc=" ".join(train['item_description'].astype(str))
```

In [0]:

```
wordcloud = WordCloud(background_color='black',
                          width=600,
                          height=300,
                    ).generate(desc)

fig = plt.figure(figsize=(30,20))
plt.imshow(wordcloud)
plt.show()
```



Observations :-

- The most frequent words in the item description are price,free shipping,firm,brand new,good condition,great condition,new tag,never worn,never used....

name

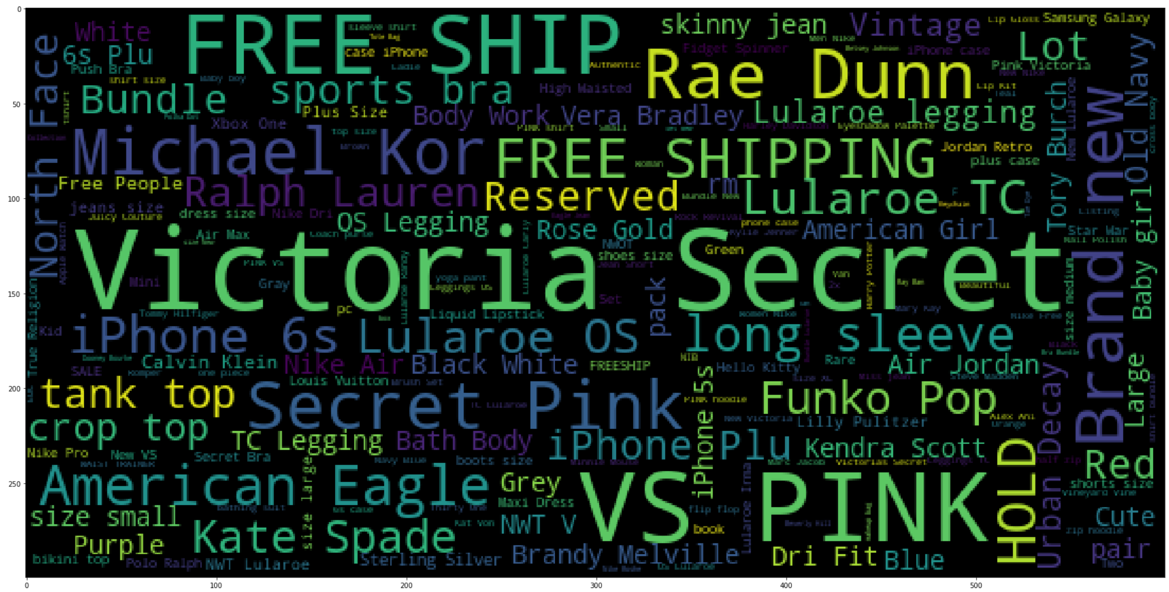
In [0]:

```
name_join=" ".join(train['name'].astype(str))
```

In [0]:

```
wordcloud = WordCloud(background_color='black',
                            width=600,
                            height=300,
                      ).generate(name_join)

fig = plt.figure(figsize=(30,20))
plt.imshow(wordcloud)
plt.show()
```



Observations :-

- The most frequent words in the name are mostly the brand names

In [ ]:

```
```

In [6]:

```
train['text']=train['name'].str.cat(train['item_description'], sep =" ")
train['text']=train['text'].str.cat(train['brand_name'], sep =" ")
```

In [10]:

```
train.drop('name', axis=1, inplace=True)
train.drop('item_description', axis=1, inplace=True)
train.drop('brand_name', axis=1, inplace=True)
```

In [7]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'i
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', '
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have',
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'beca
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into',
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on',
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', '
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'th
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "shoul
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn',
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", '
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shou
            'won', "won't", 'wouldn', "wouldn't"]
```

In [8]:

```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_desc = []
# tqdm is for printing the status bar
for sentance in tqdm(train['text'].values):
    sent = ' '.join(e for e in sentance.split() if e.lower() not in stopwords
    preprocessed_desc.append(sent.lower().strip())
```

```
100%|██████████| 1482535/1482535 [02:03<00:00, 12005.63it/s]
```

In [9]:

```
train.drop('text', axis=1, inplace=True)
train['preprocessed_text']=preprocessed_desc
```

In [11]:

```
train.head()
```

Out[11]:

	train_id	item_condition_id	price	shipping	main_category	sub_cat1	sub_cat2	preproc
0	0	3	10.0	1	Men	Tops	T-shirts	mlb cir sh
1	1	3	52.0	0	Electronics	Computers & Tablets	Components & Parts	razer chron keyb
2	2	1	10.0	1	Women	Tops & Blouses	Blouse	av ador lac
3	3	1	35.0	1	Home	Home Décor	Home Décor Accents	le statue leath
4	4	1	44.0	0	Women	Jewelry	Necklaces	24k ro certi

In [ ]:

```

```

4. Preparing data for models

In [12]:

```
from sklearn.model_selection import train_test_split
import pickle
from scipy.sparse import hstack
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from tqdm import tqdm
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import RandomizedSearchCV
from xgboost import XGBRegressor
```

In [13]:

```
train_price=train['price'].values
train.drop('price', axis=1, inplace=True)
```

4.1 Splitting data into Train, cross validation and test

In [14]:

```
x_1, x_test, y_1, y_test = train_test_split(train, train_price, test_size=0.3
x_train, x_cv, y_train, y_cv = train_test_split(x_1, y_1, test_size=0.3, ranc
```

In [15]:

```
print(x_train.shape)
print(x_cv.shape)
print(x_test.shape)
print(y_train.shape)
print(y_cv.shape)
print(y_test.shape)
```

```
(726441, 7)
(311333, 7)
(444761, 7)
(726441,)
(311333,)
(444761,)
```

4.2 Encoding categorical and text features

Encoding categorical features : main_category

In [16]:
```
vectorizer = CountVectorizer()
vectorizer.fit(x_train['main_category'].values) # fit has to happen only on t

# we use the fitted CountVectorizer to convert the text to vector
X_train_main_cat = vectorizer.transform(x_train['main_category'].values)
X_cv_main_cat = vectorizer.transform(x_cv['main_category'].values)
X_test_main_cat = vectorizer.transform(x_test['main_category'].values)


print(X_train_main_cat.shape)
print(X_cv_main_cat.shape)
print(X_test_main_cat.shape)
```

```
(726441, 13)
(311333, 13)
(444761, 13)
```

Encoding categorical features : sub_cat1

In [17]:
```
vectorizer = CountVectorizer()
vectorizer.fit(x_train['sub_cat1'].values) # fit has to happen only on train

# we use the fitted CountVectorizer to convert the text to vector
X_train_sub_cat1 = vectorizer.transform(x_train['sub_cat1'].values)
X_cv_sub_cat1 = vectorizer.transform(x_cv['sub_cat1'].values)
X_test_sub_cat1 = vectorizer.transform(x_test['sub_cat1'].values)


print(X_train_sub_cat1.shape)
print(X_cv_sub_cat1.shape)
print(X_test_sub_cat1.shape)
```

```
(726441, 142)
(311333, 142)
(444761, 142)
```

Encoding categorical features : sub_cat2

In [18]:
```
vectorizer = CountVectorizer()
vectorizer.fit(x_train['sub_cat2'].values) # fit has to happen only on train

# we use the fitted CountVectorizer to convert the text to vector
X_train_sub_cat2 = vectorizer.transform(x_train['sub_cat2'].values)
X_cv_sub_cat2 = vectorizer.transform(x_cv['sub_cat2'].values)
X_test_sub_cat2 = vectorizer.transform(x_test['sub_cat2'].values)


print(X_train_sub_cat2.shape)
print(X_cv_sub_cat2.shape)
print(X_test_sub_cat2.shape)
```

```
(726441, 937)
(311333, 937)
(444761, 937)
```

Encoding text features : preprocessed_text(BOW)

In [23]:
```
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,2), max_features=2000)
vectorizer.fit(x_train['preprocessed_text'].values)

# we use the fitted CountVectorizer to convert the text to vector
X_train_text_bow = vectorizer.transform(x_train['preprocessed_text'].values)
X_cv_text_bow = vectorizer.transform(x_cv['preprocessed_text'].values)
X_test_text_bow = vectorizer.transform(x_test['preprocessed_text'].values)

print("After vectorizations")
print(X_train_text_bow.shape)
print(X_cv_text_bow.shape)
print(X_test_text_bow.shape)

print("="*100)
```

```
After vectorizations
(726441, 2000)
(311333, 2000)
(444761, 2000)
===========================================================================
=========================
```

Encoding text features : preprocessed_text(TF-IDF)

In [24]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

#Considering max_features=1000 beacuse of system issues while performing Trun
vectorizer = TfidfVectorizer(ngram_range=(1,2),max_features=2000)
vectorizer.fit(x_train['preprocessed_text'].values)

# we use the fitted CountVectorizer to convert the text to vector
X_train_text_tfidf = vectorizer.transform(x_train['preprocessed_text'].values
X_cv_text_tfidf = vectorizer.transform(x_cv['preprocessed_text'].values)
X_test_text_tfidf = vectorizer.transform(x_test['preprocessed_text'].values)

print("After vectorizations")
print(X_train_text_tfidf.shape)
print(X_cv_text_tfidf.shape)
print(X_test_text_tfidf.shape)
print("="*100)
```

```
After vectorizations
(726441, 2000)
(311333, 2000)
(444761, 2000)
===========================================================================
=========================
```

One hot encoding categorical feature : item_condition_id

In [25]:
```
from sklearn.preprocessing import OneHotEncoder

encoder=OneHotEncoder()
X_train_condition=encoder.fit_transform(x_train['item_condition_id'].values.r
X_cv_condition=encoder.transform(x_cv['item_condition_id'].values.reshape(-1,
X_test_condition=encoder.transform(x_test['item_condition_id'].values.reshape
X_train_condition=X_train_condition.toarray()
X_cv_condition=X_cv_condition.toarray()
X_test_condition=X_test_condition.toarray()
```

One Hot Encoding categorical feature : shipping

In [26]:
```
encoder=OneHotEncoder()
X_train_shipping=encoder.fit_transform(x_train['shipping'].values.reshape(-1,
X_cv_shipping=encoder.transform(x_cv['shipping'].values.reshape(-1,1))
X_test_shipping=encoder.transform(x_test['shipping'].values.reshape(-1,1))
X_train_shipping=X_train_shipping.toarray()
X_cv_shipping=X_cv_shipping.toarray()
X_test_shipping=X_test_shipping.toarray()
```

Converting numerical feature price to log(price)

In [27]:
```
y_train_log = np.log1p(y_train)
y_cv_log = np.log1p(y_cv)
y_test_log = np.log1p(y_test)
```

In [ ]:
```

```

4.3 Concatinating all the features

Set 1: categorical + preprocessed_description(BOW) + name (BOW)

In [28]:
```
X1_tr=hstack((X_train_text_bow,X_train_condition,X_train_shipping,X_train_mai

X1_cr=hstack((X_cv_text_bow,X_cv_condition,X_cv_shipping,X_cv_main_cat,X_cv_s

X1_te=hstack((X_test_text_bow,X_test_condition,X_test_shipping,X_test_main_ca
```

In [29]:
```
print(X1_tr.shape, y_train_log.shape)
print(X1_cr.shape, y_cv_log.shape)
print(X1_te.shape, y_test_log.shape)
```

```
(726441, 3099) (726441,)
(311333, 3099) (311333,)
(444761, 3099) (444761,)
```

Set 2: categorical + preprocessed_description(TF-IDF) + name (TF-IDF)

In [30]:
```
X2_tr=hstack((X_train_text_tfidf,X_train_condition,X_train_shipping,X_train_m

X2_cr=hstack((X_cv_text_tfidf,X_cv_condition,X_cv_shipping,X_cv_main_cat,X_cv

X2_te=hstack((X_test_text_tfidf,X_test_condition,X_test_shipping,X_test_main_
```

In [31]:
```
print(X2_tr.shape, y_train_log.shape)
print(X2_cr.shape, y_cv_log.shape)
print(X2_te.shape, y_test_log.shape)
```

```
(726441, 3099) (726441,)
(311333, 3099) (311333,)
(444761, 3099) (444761,)
```

4.4 ML Models

In [32]:
```
def rmsle(y_test, y_pred):
    result = (np.sqrt(((y_test-y_pred)**2).mean())).round(4)
    return result
```

4.4.1 Baseline model

In [0]:
```
#calculating mean of price of train data
y_train_mean = y_train_log.mean()

train_rmsle = rmsle(y_train_log,y_train_mean)
cv_rmsle = rmsle(y_cv_log,y_train_mean)
test_rmsle = rmsle(y_test_log,y_train_mean)

print("Train RMSLE for baseline model =",train_rmsle)
print("CV RMSLE for baseline model =",cv_rmsle)
print("Test RMSLE for baseline model =",test_rmsle)
```

```
Train RMSLE for baseline model = 0.7502
CV RMSLE for baseline model = 0.7496
Test RMSLE for baseline model = 0.7474
```

- Therefore the RMSLE for the ML models should be better than 0.7474

4.4.2 Linear Regression

SET 1:-

In [29]:
```
lr = LinearRegression(n_jobs=-1)
lr.fit(X1_tr,y_train_log)

y_train_pred = lr.predict(X1_tr)
y_cv_pred = lr.predict(X1_cr)
y_test_pred = lr.predict(X1_te)

train_error = rmsle(y_train_log, y_train_pred)
cv_error = rmsle(y_cv_log, y_cv_pred)
test_error = rmsle(y_test_log, y_test_pred)

print("Train RMSLE for baseline model =",train_error)
print("CV RMSLE for baseline model =",cv_error)
print("Test RMSLE for baseline model =",test_error)
```

```
Train RMSLE for baseline model = 0.4806
CV RMSLE for baseline model = 0.4913
Test RMSLE for baseline model = 0.489
```

SET 2:-

In [30]:
```
lr = LinearRegression(n_jobs=-1)
lr.fit(X2_tr,y_train_log)

y_train_pred = lr.predict(X2_tr)
y_cv_pred = lr.predict(X2_cr)
y_test_pred = lr.predict(X2_te)

train_error = rmsle(y_train_log, y_train_pred)
cv_error = rmsle(y_cv_log, y_cv_pred)
test_error = rmsle(y_test_log, y_test_pred)

print("Train RMSLE for baseline model =",train_error)
print("CV RMSLE for baseline model =",cv_error)
print("Test RMSLE for baseline model =",test_error)
```

```
Train RMSLE for baseline model = 0.4802
CV RMSLE for baseline model = 0.4906
Test RMSLE for baseline model = 0.4884
```

4.4.3 Decision Tree Regressor

SET 1:-

Training the model - Hyperparameter tuning

In [0]:

```
max_depth = [5,10,15,20,30]
min_samples_split = [20,30,50,75]

for i in max_depth:
    for j in min_samples_split:
        DT = DecisionTreeRegressor(max_depth=i, min_samples_split=j, random_s
        DT.fit(X1_tr,y_train_log)

        y_train_pred = DT.predict(X1_tr)
        y_cv_pred = DT.predict(X1_cr)

        train_error = rmsle(y_train_log, y_train_pred)
        cv_error = rmsle(y_cv_log, y_cv_pred)

        print('max_depth = '+str(i)+', min_samples_split = '+str(j)+' : Trair
```

```
max_depth = 5, min_samples_split = 20 : Train RMSLE = 0.6803, CV RMSLE =
0.6805
max_depth = 5, min_samples_split = 30 : Train RMSLE = 0.6803, CV RMSLE =
0.6805
max_depth = 5, min_samples_split = 50 : Train RMSLE = 0.6803, CV RMSLE =
0.6805
max_depth = 5, min_samples_split = 75 : Train RMSLE = 0.6803, CV RMSLE =
0.6805
max_depth = 10, min_samples_split = 20 : Train RMSLE = 0.6355, CV RMSLE =
0.6385
max_depth = 10, min_samples_split = 30 : Train RMSLE = 0.6356, CV RMSLE =
0.6384
max_depth = 10, min_samples_split = 50 : Train RMSLE = 0.6358, CV RMSLE =
0.6383
max_depth = 10, min_samples_split = 75 : Train RMSLE = 0.636, CV RMSLE =
0.6383
max_depth = 15, min_samples_split = 20 : Train RMSLE = 0.5999, CV RMSLE =
0.6106
max_depth = 15, min_samples_split = 30 : Train RMSLE = 0.6005, CV RMSLE =
0.6109
max_depth = 15, min_samples_split = 50 : Train RMSLE = 0.6012, CV RMSLE =
0.6106
max_depth = 15, min_samples_split = 75 : Train RMSLE = 0.6022, CV RMSLE =
0.6106
max_depth = 20, min_samples_split = 20 : Train RMSLE = 0.5709, CV RMSLE =
0.5953
max_depth = 20, min_samples_split = 30 : Train RMSLE = 0.5723, CV RMSLE =
0.5952
max_depth = 20, min_samples_split = 50 : Train RMSLE = 0.5743, CV RMSLE =
0.5949
max_depth = 20, min_samples_split = 75 : Train RMSLE = 0.5768, CV RMSLE =
0.5944
max_depth = 30, min_samples_split = 20 : Train RMSLE = 0.5191, CV RMSLE =
0.5762
max_depth = 30, min_samples_split = 30 : Train RMSLE = 0.5227, CV RMSLE =
0.5749
max_depth = 30, min_samples_split = 50 : Train RMSLE = 0.528, CV RMSLE =
0.5733
```

```
max_depth = 30, min_samples_split = 75 : Train RMSLE = 0.5334, CV RMSLE =
0.5726
```

Testing the model with best hyperparameters

In [0]:

```
DT1 = DecisionTreeRegressor(max_depth=30, min_samples_split=75, random_state=
DT1.fit(X1_tr,y_train_log)

y_train_pred = DT1.predict(X1_tr)
y_cv_pred = DT1.predict(X1_cr)
y_test_pred = DT1.predict(X1_te)

train_error = rmsle(y_train_log, y_train_pred)
cv_error = rmsle(y_cv_log, y_cv_pred)
test_error = rmsle(y_test_log, y_test_pred)

print("Train RMSLE for baseline model =",train_error)
print("CV RMSLE for baseline model =",cv_error)
print("Test RMSLE for baseline model =",test_error)
```

```
Train RMSLE for baseline model = 0.5334
CV RMSLE for baseline model = 0.5726
Test RMSLE for baseline model = 0.5714
```

SET 2:-

Training the model - Hyperparameter tuning

In [0]:

```
max_depth = [5,10,15,20,30]
min_samples_split = [20,30,50,75]

for i in max_depth:
    for j in min_samples_split:
        DT = DecisionTreeRegressor(max_depth=i, min_samples_split=j, random_s
        DT.fit(X2_tr,y_train_log)

        y_train_pred = DT.predict(X2_tr)
        y_cv_pred = DT.predict(X2_cr)

        train_error = rmsle(y_train_log, y_train_pred)
        cv_error = rmsle(y_cv_log, y_cv_pred)

        print('max_depth = '+str(i)+', min_samples_split = '+str(j)+' : Trair
```

```
max_depth = 5, min_samples_split = 20 : Train RMSLE = 0.6803, CV RMSLE =
0.6805
max_depth = 5, min_samples_split = 30 : Train RMSLE = 0.6803, CV RMSLE =
0.6805
max_depth = 5, min_samples_split = 50 : Train RMSLE = 0.6803, CV RMSLE =
0.6805
max_depth = 5, min_samples_split = 75 : Train RMSLE = 0.6803, CV RMSLE =
0.6805
max_depth = 10, min_samples_split = 20 : Train RMSLE = 0.6354, CV RMSLE =
0.6387
max_depth = 10, min_samples_split = 30 : Train RMSLE = 0.6355, CV RMSLE =
0.6386
max_depth = 10, min_samples_split = 50 : Train RMSLE = 0.6356, CV RMSLE =
0.6387
max_depth = 10, min_samples_split = 75 : Train RMSLE = 0.6358, CV RMSLE =
0.6384
max_depth = 15, min_samples_split = 20 : Train RMSLE = 0.5995, CV RMSLE =
0.6119
max_depth = 15, min_samples_split = 30 : Train RMSLE = 0.6001, CV RMSLE =
0.6119
max_depth = 15, min_samples_split = 50 : Train RMSLE = 0.6009, CV RMSLE =
0.6117
max_depth = 15, min_samples_split = 75 : Train RMSLE = 0.6018, CV RMSLE =
0.6118
max_depth = 20, min_samples_split = 20 : Train RMSLE = 0.5703, CV RMSLE =
0.5975
max_depth = 20, min_samples_split = 30 : Train RMSLE = 0.5717, CV RMSLE =
0.5973
max_depth = 20, min_samples_split = 50 : Train RMSLE = 0.5738, CV RMSLE =
0.5966
max_depth = 20, min_samples_split = 75 : Train RMSLE = 0.5763, CV RMSLE =
0.5961
max_depth = 30, min_samples_split = 20 : Train RMSLE = 0.5173, CV RMSLE =
0.5808
max_depth = 30, min_samples_split = 30 : Train RMSLE = 0.5209, CV RMSLE =
0.5796
max_depth = 30, min_samples_split = 50 : Train RMSLE = 0.5263, CV RMSLE =
0.5778
```

```
max_depth = 30, min_samples_split = 75 : Train RMSLE = 0.5317, CV RMSLE =
0.5764
```

Testing the model with best hyperparameters

In [0]:

```
DT2 = DecisionTreeRegressor(max_depth=30, min_samples_split=75, random_state=
DT2.fit(X2_tr,y_train_log)

y_train_pred = DT2.predict(X2_tr)
y_cv_pred = DT2.predict(X2_cr)
y_test_pred = DT2.predict(X2_te)

train_error = rmsle(y_train_log, y_train_pred)
cv_error = rmsle(y_cv_log, y_cv_pred)
test_error = rmsle(y_test_log, y_test_pred)

print("Train RMSLE for baseline model =",train_error)
print("CV RMSLE for baseline model =",cv_error)
print("Test RMSLE for baseline model =",test_error)
```

```
Train RMSLE for baseline model = 0.5317
CV RMSLE for baseline model = 0.5764
Test RMSLE for baseline model = 0.5743
```

In [0]:

```
```

4.4.4 Random Forest Regressor

SET 1:-

Training the model - Hyperparameter tuning

In [0]:

```
#parameters = {'n_estimators': [50, 100,500,1000], 'max_depth': [5, 10,20,30,
n_estimators=[100,500,1000,2000]
max_depth= [2,5,10,15,20,30,40,50]
for i in max_depth:
    RF = RandomForestRegressor(max_depth=i, n_jobs=-1)
    RF.fit(X1_tr,y_train_log)

    y_train_pred = RF.predict(X1_tr)
    y_cv_pred = RF.predict(X1_cr)

    train_error = rmsle(y_train_log, y_train_pred)
    cv_error = rmsle(y_cv_log, y_cv_pred)

    print('max_depth = '+str(i)+' : Train RMSLE = '+str(train_error)+', CV RM
```

```
max_depth = 2 : Train RMSLE = 0.7136, CV RMSLE = 0.7135
max_depth = 5 : Train RMSLE = 0.6764, CV RMSLE = 0.6767
max_depth = 10 : Train RMSLE = 0.6283, CV RMSLE = 0.6311
max_depth = 15 : Train RMSLE = 0.5902, CV RMSLE = 0.6009
max_depth = 20 : Train RMSLE = 0.5584, CV RMSLE = 0.5809
max_depth = 30 : Train RMSLE = 0.5017, CV RMSLE = 0.5516
max_depth = 40 : Train RMSLE = 0.4542, CV RMSLE = 0.5345
max_depth = 50 : Train RMSLE = 0.4149, CV RMSLE = 0.5235
```

Testing the model with best hyperparameters

In [0]:

```
RF1 = RandomForestRegressor(max_depth=50, n_jobs=-1)
RF1.fit(X1_tr,y_train_log)

y_train_pred = RF1.predict(X1_tr)
y_cv_pred = RF1.predict(X1_cr)
y_test_pred = RF1.predict(X1_te)

train_error = rmsle(y_train_log, y_train_pred)
cv_error = rmsle(y_cv_log, y_cv_pred)
test_error = rmsle(y_test_log, y_test_pred)

print("Train RMSLE for baseline model =",train_error)
print("CV RMSLE for baseline model =",cv_error)
print("Test RMSLE for baseline model =",test_error)
```

```
Train RMSLE for baseline model = 0.4142
CV RMSLE for baseline model = 0.5231
Test RMSLE for baseline model = 0.5216
```

SET 2:-

Training the model - Hyperparameter tuning

In [0]:

```
#parameters = {'n_estimators': [50, 100,500,1000], 'max_depth': [5, 10,20,30,
n_estimators=[100,500,1000,2000]
max_depth= [2,5,10,15,20,30,40,50]
for i in max_depth:
    RF = RandomForestRegressor(max_depth=i, n_jobs=-1)
    RF.fit(X2_tr,y_train_log)

    y_train_pred = RF.predict(X2_tr)
    y_cv_pred = RF.predict(X2_cr)

    train_error = rmsle(y_train_log, y_train_pred)
    cv_error = rmsle(y_cv_log, y_cv_pred)

    print('max_depth = '+str(i)+' : Train RMSLE = '+str(train_error)+', CV RM
```

```
max_depth = 2 : Train RMSLE = 0.7137, CV RMSLE = 0.7135
max_depth = 5 : Train RMSLE = 0.675, CV RMSLE = 0.6755
max_depth = 10 : Train RMSLE = 0.628, CV RMSLE = 0.6311
max_depth = 15 : Train RMSLE = 0.5905, CV RMSLE = 0.6023
max_depth = 20 : Train RMSLE = 0.5587, CV RMSLE = 0.5827
max_depth = 30 : Train RMSLE = 0.5014, CV RMSLE = 0.5546
max_depth = 40 : Train RMSLE = 0.4524, CV RMSLE = 0.538
max_depth = 50 : Train RMSLE = 0.4142, CV RMSLE = 0.5287
```

Testing the model with best hyperparameters

In [0]:

```
RF2 = RandomForestRegressor(max_depth=50, n_jobs=-1)
RF2.fit(X2_tr,y_train_log)

y_train_pred = RF2.predict(X2_tr)
y_cv_pred = RF2.predict(X2_cr)
y_test_pred = RF2.predict(X2_te)

train_error = rmsle(y_train_log, y_train_pred)
cv_error = rmsle(y_cv_log, y_cv_pred)
test_error = rmsle(y_test_log, y_test_pred)

print("Train RMSLE for baseline model =",train_error)
print("CV RMSLE for baseline model =",cv_error)
print("Test RMSLE for baseline model =",test_error)
```

```
Train RMSLE for baseline model = 0.4142
CV RMSLE for baseline model = 0.5287
Test RMSLE for baseline model = 0.5265
```

In [0]:

4.4.5 XGBOOST Regressor

SET 1:-

Training the model - Hyperparameter tuning

In [0]:

```
n_estimators=[100,500,1000,2000,3000,4000,5000]
train_rmsle=[]
cv_rmsle=[]
for i in n_estimators:
    xgb = XGBRegressor(n_estimators=i, n_jobs=-1)
    xgb.fit(X1_tr,y_train_log)

    y_train_pred = xgb.predict(X1_tr)
    y_cv_pred = xgb.predict(X1_cr)

    train_error = rmsle(y_train_log, y_train_pred)
    cv_error = rmsle(y_cv_log, y_cv_pred)

    train_rmsle.append(train_error)
    cv_rmsle.append(cv_error)

for i in range(len(n_estimators)):
    print('n_estimators = '+str(n_estimators[i])+' : Train RMSLE = '+str(trai
```

```
[07:45:57] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:lin
ear is now deprecated in favor of reg:squarederror.
[07:46:33] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:lin
ear is now deprecated in favor of reg:squarederror.
[07:48:17] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:lin
ear is now deprecated in favor of reg:squarederror.
[07:51:25] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:lin
ear is now deprecated in favor of reg:squarederror.
[07:57:21] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:lin
ear is now deprecated in favor of reg:squarederror.
[08:06:12] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:lin
ear is now deprecated in favor of reg:squarederror.
[08:17:51] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:lin
ear is now deprecated in favor of reg:squarederror.
n_estimators = 100 : Train RMSLE = 0.6143, CV RMSLE = 0.6144
n_estimators = 500 : Train RMSLE = 0.5515, CV RMSLE = 0.5528
n_estimators = 1000 : Train RMSLE = 0.526, CV RMSLE = 0.5286
n_estimators = 2000 : Train RMSLE = 0.5029, CV RMSLE = 0.5079
n_estimators = 3000 : Train RMSLE = 0.4896, CV RMSLE = 0.4969
n_estimators = 4000 : Train RMSLE = 0.4802, CV RMSLE = 0.4898
n_estimators = 5000 : Train RMSLE = 0.4727, CV RMSLE = 0.4844
```

Testing the model with best hyperparameters

In [0]:

```
xgb1 = XGBRegressor(n_estimators=5000, n_jobs=-1)
xgb1.fit(X1_tr,y_train_log)

y_train_pred = xgb1.predict(X1_tr)
y_cv_pred = xgb1.predict(X1_cr)
y_test_pred = xgb1.predict(X1_te)

train_error = rmsle(y_train_log, y_train_pred)
cv_error = rmsle(y_cv_log, y_cv_pred)
test_error = rmsle(y_test_log, y_test_pred)

print("Train RMSLE for baseline model =",train_error)
print("CV RMSLE for baseline model =",cv_error)
print("Test RMSLE for baseline model =",test_error)
```

```
[11:59:53] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:lin
ear is now deprecated in favor of reg:squarederror.
Train RMSLE for baseline model = 0.4727
CV RMSLE for baseline model = 0.4844
Test RMSLE for baseline model = 0.4824
```

SET 2:-

Training the model - Hyperparameter tuning

In [0]:

```
n_estimators=[100,500,1000,2000,3000,4000,5000]
train_rmsle=[]
cv_rmsle=[]
for i in n_estimators:
    xgb2 = XGBRegressor(n_estimators=i, n_jobs=-1)
    xgb2.fit(X2_tr,y_train_log)

    y_train_pred = xgb2.predict(X2_tr)
    y_cv_pred = xgb2.predict(X2_cr)

    train_error = rmsle(y_train_log, y_train_pred)
    cv_error = rmsle(y_cv_log, y_cv_pred)

    train_rmsle.append(train_error)
    cv_rmsle.append(cv_error)

for i in range(len(n_estimators)):
    print('n_estimators = '+str(n_estimators[i])+' : Train RMSLE = '+str(trai
```

```
[11:51:01] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:lin
ear is now deprecated in favor of reg:squarederror.
[11:51:59] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:lin
ear is now deprecated in favor of reg:squarederror.
[11:55:24] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:lin
ear is now deprecated in favor of reg:squarederror.
[11:59:57] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:lin
ear is now deprecated in favor of reg:squarederror.
[12:12:26] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:lin
ear is now deprecated in favor of reg:squarederror.
[12:26:33] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:lin
ear is now deprecated in favor of reg:squarederror.
[12:43:55] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:lin
ear is now deprecated in favor of reg:squarederror.
n_estimators = 100 : Train RMSLE = 0.6148, CV RMSLE = 0.615
n_estimators = 500 : Train RMSLE = 0.551, CV RMSLE = 0.5526
n_estimators = 1000 : Train RMSLE = 0.5253, CV RMSLE = 0.5285
n_estimators = 2000 : Train RMSLE = 0.5015, CV RMSLE = 0.5078
n_estimators = 3000 : Train RMSLE = 0.4873, CV RMSLE = 0.4968
n_estimators = 4000 : Train RMSLE = 0.4772, CV RMSLE = 0.4898
n_estimators = 5000 : Train RMSLE = 0.4694, CV RMSLE = 0.4849
```

Testing the model with best hyperparameters

In [0]:

```
xgb2 = XGBRegressor(n_estimators=5000, n_jobs=-1)
xgb2.fit(X2_tr,y_train_log)

y_train_pred = xgb2.predict(X2_tr)
y_cv_pred = xgb2.predict(X2_cr)
y_test_pred = xgb2.predict(X2_te)

train_error = rmsle(y_train_log, y_train_pred)
cv_error = rmsle(y_cv_log, y_cv_pred)
test_error = rmsle(y_test_log, y_test_pred)

print("Train RMSLE for baseline model =",train_error)
print("CV RMSLE for baseline model =",cv_error)
print("Test RMSLE for baseline model =",test_error)
```

```
[12:10:21] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:lin
ear is now deprecated in favor of reg:squarederror.
Train RMSLE for baseline model = 0.4694
CV RMSLE for baseline model = 0.4849
Test RMSLE for baseline model = 0.4826
```

In [0]:

```
```

Conclusions :-

In [31]:

```
# Please compare all your models using Prettytable library
from prettytable import PrettyTable
x = PrettyTable()

x.field_names = [ "Model","Vectorizer","Train RMSLE","CV RMSLE","Test RMSLE"]

x.add_row(["Linear Regression","BOW",0.4806, 0.4913, 0.4890])
x.add_row(["Linear Regression","TF-IDF",0.4802, 0.4906, 0.4884])
x.add_row(["Decision Tree Regressor","BOW",0.5334, 0.5726, 0.5714])
x.add_row(["Decision Tree Regressor","TF-IDF",0.5317, 0.5764, 0.5743])
x.add_row(["Random Forest Regressor","BOW",0.4142, 0.5231, 0.5216])
x.add_row(["Random Forest Regressor","TF-IDF",0.4142, 0.5287, 0.5265])
x.add_row(["XGBOOST Regressor","BOW",0.4727, 0.4844, 0.4824])
x.add_row(["XGBOOST Regressor","TF-IDF",0.4694, 0.4849, 0.4826])

print(x)
```

```
+-------------------------+------------+-------------+----------+----------
--+
|          Model          | Vectorizer | Train RMSLE | CV RMSLE | Test RMSL
E |
+-------------------------+------------+-------------+----------+----------
--+
|    Linear Regression    |    BOW     |    0.4806   |  0.4913  |   0.489
|
|    Linear Regression    |   TF-IDF   |    0.4802   |  0.4906  |   0.4884
|
| Decision Tree Regressor |    BOW     |    0.5334   |  0.5726  |   0.5714
|
| Decision Tree Regressor |   TF-IDF   |    0.5317   |  0.5764  |   0.5743
|
| Random Forest Regressor |    BOW     |    0.4142   |  0.5231  |   0.5216
|
| Random Forest Regressor |   TF-IDF   |    0.4142   |  0.5287  |   0.5265
|
|    XGBOOST Regressor    |    BOW     |    0.4727   |  0.4844  |   0.4824
|
|    XGBOOST Regressor    |   TF-IDF   |    0.4694   |  0.4849  |   0.4826
|
+-------------------------+------------+-------------+----------+----------
--+
```

4.5 DL Models

In [35]:
```
import tensorflow as tf

import keras
from keras.models import Sequential,Model
from keras.layers import Dense, Dropout, Flatten,concatenate,Input,LSTM
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K
from keras.layers.normalization import BatchNormalization
from keras.layers.convolutional import Convolution2D, MaxPooling2D, ZeroPaddi
from numpy import asarray
from numpy import zeros
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Embedding
from keras.initializers import he_normal
from keras.initializers import RandomNormal
```

```
Using TensorFlow backend.
```

In [34]:
```
X_tr=X2_tr.todense()
X_cr=X2_cr.todense()
X_te=X2_te.todense()
```

In [64]:
```
import keras.backend as K
K.clear_session()
```

In [ ]:
```
def rmsle1(y_true, y_pred):
    result = (np.sqrt(((y_true-y_pred)**2).mean())).round(4)
    return result
```

In [43]:
```
import keras.backend as K

def rmsle(y_true, y_pred):
    result=K.sqrt(K.mean(K.square(y_true-y_pred), axis=-1))
    return result
```

MLP

In [65]:
```
model = Sequential()

model.add(Dense(512, activation='relu', kernel_initializer=RandomNormal(mean=

model.add(Dense(256, activation='relu', kernel_initializer=RandomNormal(mean=

model.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=
model.add(BatchNormalization())

model.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0
model.add(BatchNormalization())


model.add(Dense(output_dim=1))
```

In [66]:
```
model.compile(optimizer='adam', loss='mean_squared_error', metrics=[rmsle])
```

In [67]:
```
istory = model.fit(X_tr, y_train_log, batch_size=256,shuffle="batch", epochs=
```

```
Train on 726441 samples, validate on 311333 samples
Epoch 1/10
726441/726441 [==============================] - 171s 235us/step - loss: 0.
6263 - rmsle: 0.5396 - val_loss: 0.2901 - val_rmsle: 0.4044
Epoch 2/10
726441/726441 [==============================] - 165s 227us/step - loss: 0.
2691 - rmsle: 0.3918 - val_loss: 0.2674 - val_rmsle: 0.3863
Epoch 3/10
726441/726441 [==============================] - 166s 229us/step - loss: 0.
2459 - rmsle: 0.3738 - val_loss: 0.2572 - val_rmsle: 0.3791
Epoch 4/10
726441/726441 [==============================] - 165s 227us/step - loss: 0.
2274 - rmsle: 0.3590 - val_loss: 0.2552 - val_rmsle: 0.3766
Epoch 5/10
726441/726441 [==============================] - 165s 228us/step - loss: 0.
2111 - rmsle: 0.3461 - val_loss: 0.2541 - val_rmsle: 0.3762
Epoch 6/10
726441/726441 [==============================] - 166s 228us/step - loss: 0.
1955 - rmsle: 0.3336 - val_loss: 0.2567 - val_rmsle: 0.3801
Epoch 7/10
726441/726441 [==============================] - 166s 229us/step - loss: 0.
1814 - rmsle: 0.3218 - val_loss: 0.2588 - val_rmsle: 0.3790
Epoch 8/10
726441/726441 [==============================] - 163s 225us/step - loss: 0.
1682 - rmsle: 0.3103 - val_loss: 0.2608 - val_rmsle: 0.3814
Epoch 9/10
726441/726441 [==============================] - 166s 228us/step - loss: 0.
1561 - rmsle: 0.2994 - val_loss: 0.2642 - val_rmsle: 0.3839
Epoch 10/10
726441/726441 [==============================] - 167s 230us/step - loss: 0.
1449 - rmsle: 0.2888 - val_loss: 0.2679 - val_rmsle: 0.3874
```

In [70]:
```
y_test_pred=model.predict(X_te)
```

In [78]:
```
score=model.evaluate(X_te,y_test_log)
```

```
444761/444761 [==============================] - 58s 129us/step
```

In [79]:
```
print('Test RMSLE =',score[1])
```

```
Test RMSLE = 0.38676050305366516
```

In [0]:
```

```

Steps followed in solving the case study:-

- Step 1 :-Exploratory Data Analysis
- Step 2 :-Splitting category into main_category,sub_cat1,sub_cat1

- Step 3 :-Joing name,description and brand features
- Step 4 :-One hot encoding categorical features
- Step 5 :-Bag of Words and TF-IDF on text features
- Step 6 :-Concatenating all features
- Step 7 :-Implementing ML models
- Step 8 :-Implementing MLP deep learning model

In [0]:

In [0]:

In [0]:

In [0]:

In [0]:

In [0]:

In [0]: