Mercari Price Suggestion Challenge

1. Business Problem

1.1. Description

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

Data: Mercari: The Selling App, Japan.

Download train.tsv.7z and test.tsv.7z from Kaggle.

Context :

Source: https://www.kaggle.com/c/mercari-price-suggestion-challenge/discussion/50250 (https://www.kaggle.com/c/mercari-price-suggestion-challenge/discussion/50250)

Problem Statement :

To build an algorithm that automatically suggests the right product prices given the user-inputted text descriptions of the products.

1.2. Source/Useful Links

1.3. Real-world/Business objectives and constraints.

- Low latency requirement.
- Errors can be costly.
- Interpretability is important.
- Probability of Price suggestion for a product can be useful.
- Price suggestion range can be useful.

2. Machine Learning Problem Formulation

2.1. Data

2.1.1. Data Overview

- ***train_id or test_id :*** the id of the listing or product.
- ***name :*** the title of the listing or product.
- ***item_condition_id :*** the condition of the items provided by the seller.
- ***category_name :*** category of the listing or product.
- ***brand_name :*** brand name of the product or listing.
- ***price :*** the price that the item was sold for. This is the target variable that should be predicted. The unit is USD. This column doesn't exist in test.tsv since that is what you I have to predict.
- ***shipping :*** 1 if shipping fee is paid by seller and 0 by buyer
- ***item_description :*** the full description of the item.

  Note: The data has been cleaned to remove text that look like prices (e.g. $20) to avoid leakage. These removed prices are represented as [rm]

2.1.2. Example Data Point

2.2. Mapping the real-world problem to an ML problem

2.2.1. Type of Machine Learning Problem

- Learn the pricing of existing products and predict the pricing of future products. It is a Regression Problem.

2.2.2. Performance Metric

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

The evaluation metric is **Root Mean Squared Logarithmic Error(RMSLE)**.

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 (public/private) 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.

2.2.3. Machine Learing Objectives and Constraints

Objective: Predict the pricing of the products.

Constraints:

- Low latency requirement.
- Interpretability.
- Price range of products with confidence intervals.

2.3. Train and Test Datasets

- Train and Test datasets have been given.
- Test data is given in two stages. Smaller test dataset in the first stage and larger test dataset in the second stage.

3. Exploratory Data Analysis

In [ ]:

```
```

In [ ]:

```
```

In [ ]:

```
```

In [4]:

```
# This Python 3 environment comes with many helpful analytics libraries installe
d
# It is defined by the kaggle/python docker image: https://github.com/kaggle/doc
ker-python
# For example, here's several helpful packages to load in

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list
 the files in the input directory

import os
print(os.listdir("../input"))

# Any results you write to the current directory are saved as output.
```

```
['train.tsv', 'test.tsv']
```

In [5]:

```
import re
import nltk
import string
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt


from nltk.stem.porter import *
from nltk.corpus import stopwords
from sklearn.feature_extraction import stop_words
from nltk.tokenize import word_tokenize, sent_tokenize



from collections import Counter
from wordcloud import WordCloud
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer


import plotly.offline as py
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
import plotly.tools as tls
%matplotlib inline


import warnings
warnings.filterwarnings('ignore')
```

3.1. Reading Data

In [6]:

```
train = pd.read_csv('../input/train.tsv', sep='\t')
test = pd.read_csv('../input/test.tsv', sep='\t')
```

In [7]:

```
# dimensions and features of training dataset
print('\nNumber of data points in training dataset : ', train.shape[0])
print('\nNumber of features in training dataset    : ', train.shape[1])
print('\nTraning Features : ', train.columns.values)

print('\n**********************************************************************
**\n')

# dimensions and features of test dataset
print('\nNumber of data points in test dataset : ', test.shape[0])
print('\nNumber of features in test dataset    : ', test.shape[1])
print('\nTest Features : \n', test.columns.values)

print('\n**********************************************************************
**')
print('\nSample training data\n--------------------')
train.head()
```

```
Number of data points in training dataset :  1482535

Number of features in training dataset    :  8

Traning Features :  ['train_id' 'name' 'item_condition_id' 'category
_name' 'brand_name' 'price'
 'shipping' 'item_description']

**********************************************************************
*****


Number of data points in test dataset :  693359

Number of features in test dataset    :  7

Test Features :
 ['test_id' 'name' 'item_condition_id' 'category_name' 'brand_name'
 'shipping' 'item_description']

**********************************************************************
*****

Sample training data
--------------------

Out[7]:
```

	train_id	name	item_condition_id	category_name	brand_name	pr
0	0	MLB Cincinnati Reds T Shirt Size XL	3	Men/Tops/T-shirts	NaN	10
1	1	Razer BlackWidow Chroma Keyboard	3	Electronics/Computers & Tablets/Components & P...	Razer	52
2	2	AVA-VIV Blouse	1	Women/Tops & Blouses/Blouse	Target	10
3	3	Leather Horse Statues	1	Home/Home Décor/Home Décor Accents	NaN	35
4	4	24K GOLD plated rose	1	Women/Jewelry/Necklaces	NaN	44

In [8]:

```
# different data types in the dataset: categorical (strings) and numeric
train.dtypes
```

Out[8]:

```
train_id              int64
name                 object
item_condition_id      int64
category_name        object
brand_name           object
price               float64
shipping               int64
item_description     object
dtype: object
```

Univariate Analysis

Price

In [9]:

```
train.price.head()
```

Out[9]:

```
0    10.0
1    52.0
2    10.0
3    35.0
4    44.0
Name: price, dtype: float64
```

In [10]:

```
train.price.describe()
```

Out[10]:

```
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
```

- The mean price of the product is 26.72 USD
- Median proce of products is 17
- Min price is 0 USD
- Max price is 2009 USD

In [11]:

```
train.price.isnull().values.any()
```

Out[11]:

```
False
```

- There are no NaN's in the price column.

In [12]:

```
train.name.loc[train['price']==0]
```

Out[12]:

```
1325           Alabama Crimson Tide Quality Lanyard
2544                                  Levi leggings
2712                  Simple Red Lace Lingerie Top
3576              Turquoise 15 Pc Makeup Brush Set
3761                                Girls size 7/8
5237                                   Nursing Bra
6175          Multi Listing Bundled Package, Thanks
7116           Nike Dri-Fit High Power Speed Tights
7622                           Hands free earpiece
10812                           Vampire candy bowl
12738         Beautiful Janie and Jack loafers sz 4
17617                      Coca Cola Stuffed Bears
17831                     Holographic sequin bra
18324             Pura 11 oz straw cup, swirl pink
19909                     Obey Floral Snapback Hat
20986           Sale! Varied sizes - Rock N' Roll t
22876          Quay Australia Sunnies - Barely Worn
24288          Fred Jackson Autographed Mini Helmet
28091                     Jordan Cris Shoes Size 2
28701                   Lace Up Bodysuit * On Hold
29240                      Baby K'Tan Wrap Carrier
29702       Old Navy White Polka Dot Shorts Size 16
29826      Geo Mountains/ Triangles Pillow Set Of 2
33056      Viking compass leathercoinpurse guidance
33176                           Bushnell Golf Watch
33624               Michael Kors Medium Selma Dune
33862                           Sexy cocktail dress
35216                                     VS Bra
38132             Vlone T-shirt Palace Bape Supreme
39519             Jordan shoe's Girls size 12c,used
                           ...
1430798                         Two Tone Leather Bag
1432284                                    Mk Purse
1440173      Brand new Kanye West Life of Pablo Hat
1440325    Galaxy Tab A 8.0 Case/ Windshield Cleane
1442657              Lularoe Disney TC LEGGINGS NWT
1445675    14g Clear on Gold Sml Prong Nipple Rings
1446187      Handcrafted Jadeite Frogs and Crystals
1449089                       REVLON 16hr Colorstay
1449359        PORCELAIN L.A. Girl HD Pro Concealer
1449450              VS Pink Push Up Bralette Large
1452401               american eagle stripped shirt
1456460                     New Frozen Shoes size 6
1458474                  Victoria's Secret PINK bra
1460118         Flowy tank top plus size blouse 2xl
1461712                                  Nike Airmax
1462168     Too Faced Born This Way Foundation Sand
1463021                                faux fur coat
1463174                 Marc Jacobs wristlet/Wallet
1463577                          Soft striped pijama
1464583                  Ardell Lashes Demi Wispies
1465545                     Infant Timberland Boots
1467318    Certified Vegan Card Case New! Blue/Gray
1467698           Rock Revivals Men's Size 29 Jeans
1469806                       Hp 11x Ink cartridge
1471721            Waist Trainer Corset Size S Only
1474172              Homecoming or Dama ivory dress
1474198                          Tie up flannel top
1477958                     Rae Dunn Measuring Pear
```

```
1478519            4 pairs women's wool boots socks
1478813             New Braven HD Waterprf Wireless
Name: name, Length: 874, dtype: object
```

In [13]:

```
train.name.loc[train.price == 0].count()
```

Out[13]:

```
874
```

In [14]:

```
train = train[train['price' ]> 0]
```

In [15]:

```
train.shape
```

Out[15]:

```
(1481661, 8)
```

In [17]:

```
train.price.head()
```

Out[17]:

```
0    10.0
1    52.0
2    10.0
3    35.0
4    44.0
Name: price, dtype: float64
```

In [18]:

```
train.name.loc[train.price == 0].count()
```

Out[18]:

```
0
```

- 874 products have price 0. Those products might be given free for some other products.
- Those products with zero price occur in duplicates with varied prices.
- All the products whose price is zero are removed from training data.

In [19]:

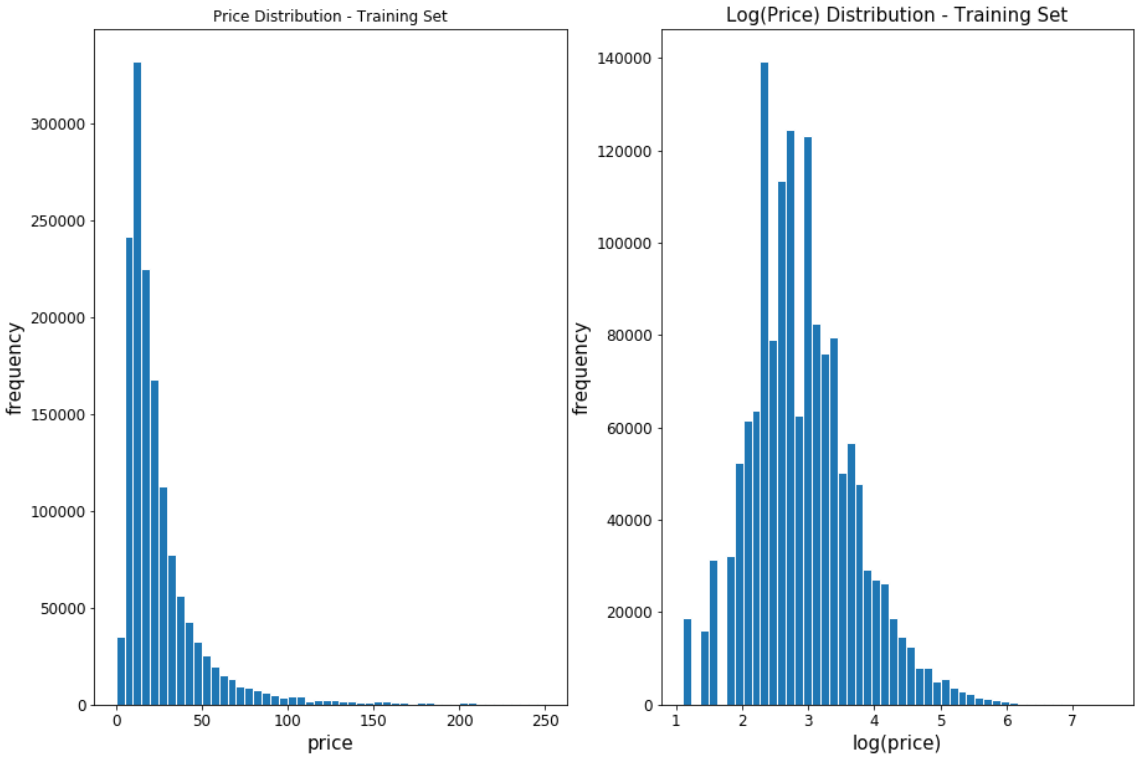
```
(train.price == 0).sum()
```

Out[19]:

```
0
```

In [24]:

```
plt.subplot(1, 2, 1)
train['price'].plot.hist(bins=50, figsize=(15,10), edgecolor='white',range=[0,25
0])
plt.xlabel('price', fontsize=15)
plt.ylabel('frequency', fontsize=15)
plt.tick_params(labelsize=12)
plt.title('Price Distribution - Training Set', fontsize=12)

plt.subplot(1, 2, 2)
np.log(train['price']).plot.hist(bins=50, figsize=(15,10), edgecolor='white')
plt.xlabel('log(price)', fontsize=15)
plt.ylabel('frequency', fontsize=15)
plt.tick_params(labelsize=12)
plt.title('Log(Price) Distribution - Training Set', fontsize=15)
plt.show()
```



In [25]:

```
train.price.describe()
```

Out[25]:

```
count    1.481661e+06
mean     2.675329e+01
std      3.859198e+01
min      3.000000e+00
25%      1.000000e+01
50%      1.700000e+01
75%      2.900000e+01
max      2.009000e+03
Name: price, dtype: float64
```

In [ ]:

```

```

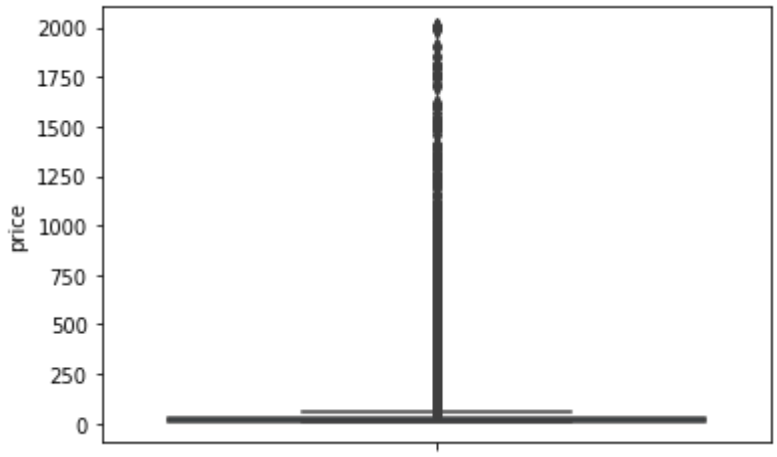
- The mean price of the products is 26.75 USD
- Median proce of products is 17
- Min price is 3 USD
- Max price is 2009 USD

Observations:

- 75% of the products has price less than or equal to 29 USD
- Prices of the products is skewed towards the left i.e., most of the products has low prices and very few products has more price.
- As the graph is skewed, I take log tansform of the prices to get better intuition and visualization of the price distribution over the products.
- From the log transform of price graph we get a better intuition that very few products have larger price and very few products have very less price.

In [28]:

```
sns.boxplot(y=train.price)
plt.show()
```



Shipping

In [30]:

```
train.shipping.value_counts()
```

Out[30]:

```
0    818876
1    662785
Name: shipping, dtype: int64
```

- Shipping is a categorical feature.
- **0 ------> Shipping fee paid by Buyer.**
- **1 ------> Shipping fee paid by Seller.**
- For 818876 products shipping fee is paid by buyer.
- For 662785 products shipping fee is paid by seller.

In [34]:

```
train.shipping.value_counts()/len(train)
```

Out[34]:

```
0    0.552674
1    0.447326
Name: shipping, dtype: float64
```

In [50]:

```
x,y = train.shipping.value_counts()/len(train)
print('Shipping fee of {0:.2f}% of the products is paid by Buyer'.format(x*100))
print('Shipping fee of {0:.2f}% of the products is paid by Seller'.format(y*100
))
```

```
Shipping fee of 55.27% of the products is paid by Buyer
Shipping fee of 44.73% of the products is paid by Seller
```

In [ ]:

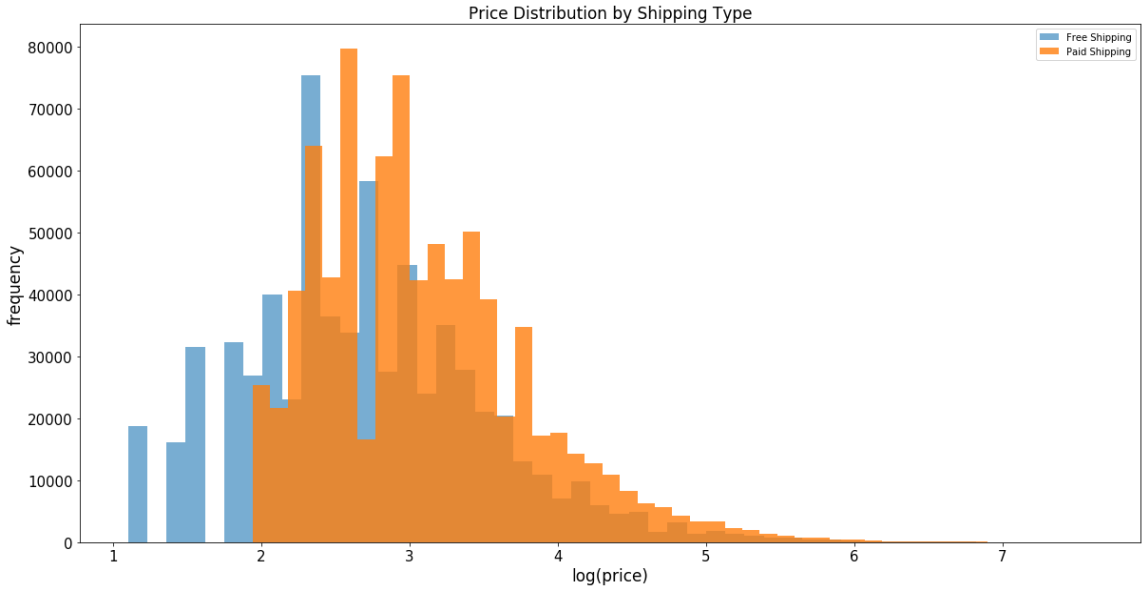
```

```

In [52]:

```
# freeShipping ----> Shipping fee paid by Seller
freeShipping = train.loc[train.shipping==1, 'price']
# paidShipping -----> Shipping fee paid by Buyer
paidShipping = train.loc[train.shipping==0, 'price']
```

In [55]:

```
# plotting Price distribution with respect to shipping type

fig, ax = plt.subplots(figsize=(20,10))
ax.hist(np.log(freeShipping),  alpha=0.6, bins=50,label='Free Shipping')
ax.hist(np.log(paidShipping), alpha=0.8, bins=50,label='Paid Shipping')
ax.set(title='Histogram Comparison', ylabel='% of Dataset in Bin')
plt.xlabel('log(price)', fontsize=17)
plt.ylabel('frequency', fontsize=17)
plt.title('Price Distribution by Shipping Type', fontsize=17)
handles, labels = ax.get_legend_handles_labels()
ax.legend(handles, labels)
plt.tick_params(labelsize=15)
plt.show()
```



In [60]:

```
print('Average price of products with free shipping is %.2f'%freeShipping.mean
())
print('Average price of products with paid shipping is %.2f'%paidShipping.mean
())
```

```
Average price of products with free shipping is 22.58
Average price of products with paid shipping is 30.13
```

- **As mean is prone to outliers, let's check median of shipping types.**

In [61]:

```
print('Median price of products with free shipping is %.2f'%freeShipping.median
())
print('Median price of products with paid shipping is %.2f'%paidShipping.median
())
```

```
Median price of products with free shipping is 14.00
Median price of products with paid shipping is 20.00
```

Observations:

- Generally we might think that, price of the product is somehow correlated with shipping fee.
- If shipping is free, price of the product might be high.
- If shipping is paid,price of the product might be less.
- It is evident from the above graph that **this is not the scenario in this case**.
- **Average price of products is lesser in case of free shipping.** -**Average price of product is higher in case of paid shipping.**

In [62]:

```
train.price.head()
```

Out[62]:

```
0    10.0
1    52.0
2    10.0
3    35.0
4    44.0
Name: price, dtype: float64
```

In [63]:

```
print(paidShipping.describe())
print(freeShipping.describe())
```

```
count    818876.000000
mean         30.132333
std          39.541581
min           5.500000
25%          13.000000
50%          20.000000
75%          33.000000
max        2009.000000
Name: price, dtype: float64
count    662785.000000
mean         22.578452
std          36.961295
min           3.000000
25%           9.000000
50%          14.000000
75%          25.000000
max        2000.000000
Name: price, dtype: float64
```

Item Category

There are about **1,287** unique categories but among each of them, we will always see a main/general category firstly, followed by two more particular subcategories (e.g. Beauty/Makeup/Face or Lips). In adidition, there are about 6,327 items that do not have a category labels. Let's split the categories into three different columns. We will see later that this information is actually quite important from the seller's point of view and how we handle the missing information in the `brand_name` column will impact the model's prediction.

In [64]:

```
print("There are %d unique values in the category column." % train['category_nam
e'].nunique())
```

```
There are 1287 unique values in the category column.
```

In [65]:

```
# TOP 5 RAW CATEGORIES
train['category_name'].value_counts()[:5]
```

Out[65]:

```
Women/Athletic Apparel/Pants, Tights, Leggings    60152
Women/Tops & Blouses/T-Shirts                     46349
Beauty/Makeup/Face                                34320
Beauty/Makeup/Lips                                29901
Electronics/Video Games & Consoles/Games          26547
Name: category_name, dtype: int64
```

In [66]:

```
# missing categories
print("There are %d items that do not have a label." % train['category_name'].is
null().sum())
```

```
There are 6314 items that do not have a label.
```

In [67]:

```
# reference: BuryBuryZymon at https://www.kaggle.com/maheshdadhich/i-will-sell-e
verything-for-free-0-55
def split_cat(text):
    try: return text.split("/")
    except: return ("No Label", "No Label", "No Label")
```

In [68]:

```
# splitting the raw category into main and sub sub categories
train['main_cat'], train['subcat_1'], train['subcat_2'] = \
zip(*train['category_name'].apply(lambda x: split_cat(x)))
# sanity checking the train for new categories
train.head()
```

Out[68]:

	train_id	name	item_condition_id	category_name	brand_name	pr
0	0	MLB Cincinnati Reds T Shirt Size XL	3	Men/Tops/T-shirts	NaN	10
1	1	Razer BlackWidow Chroma Keyboard	3	Electronics/Computers & Tablets/Components & P...	Razer	52
2	2	AVA-VIV Blouse	1	Women/Tops & Blouses/Blouse	Target	10
3	3	Leather Horse Statues	1	Home/Home Décor/Home Décor Accents	NaN	35
4	4	24K GOLD plated rose	1	Women/Jewelry/Necklaces	NaN	44

In [69]:

```
# repeat the same step for the test set
test['general_cat'], test['subcat_1'], test['subcat_2'] = \
zip(*test['category_name'].apply(lambda x: split_cat(x)))
```

In [81]:

```

```

Out[81]:

```
0            Men
1    Electronics
2          Women
3           Home
4          Women
Name: main_cat, dtype: object
```

In [82]:

```
print("There are %d unique main categories." % train['main_cat'].nunique())
print("There are %d unique first sub-categories." % train['subcat_1'].nunique())
print("There are %d unique second sub-categories." % train['subcat_2'].nunique
())
```

```
There are 11 unique main categories.
There are 114 unique first sub-categories.
There are 871 unique second sub-categories.
```

Overall, we have **11 main categories** (114 in the first sub-categories and 871 second sub-categories): women's and beauty items as the two most popular categories (more than 50% of the observations), followed by kids and electronics.

In [ ]:

```
print("There are %d unique first sub-categories." % train['subcat_1'].nunique())
```

In [ ]:

```
print("There are %d unique second sub-categories." % train['subcat_2'].nunique
())
```

In [92]:

```
x = train['main_cat'].value_counts().index.values.astype('str')
y = train['main_cat'].value_counts().values
pct = [("%.2f"%(v*100))+"%"for v in (y/len(train))]
```

In [86]:

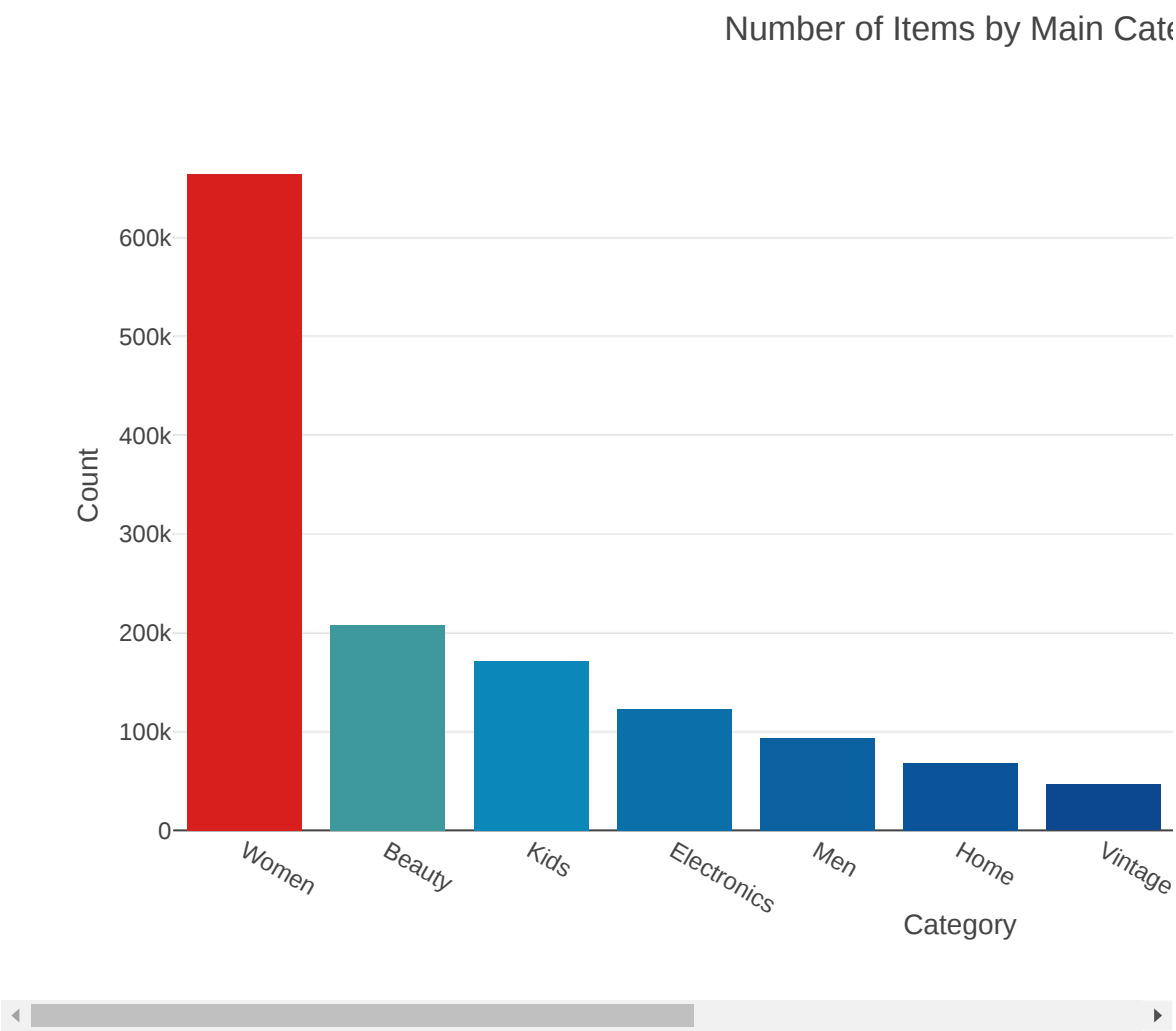
```

```

Out[86]:

```
['44.81%',
 '14.02%',
 '11.58%',
 '8.28%',
 '6.32%',
 '4.58%',
 '3.14%',
 '3.06%',
 '2.08%',
 '1.71%',
 '0.43%']
```

In [93]:

```
trace1 = go.Bar(x=x, y=y, text=pct,
                marker=dict(
                 color = y,colorscale='Portland',showscale=True,
                 reversescale = False
                 ))
layout = dict(title= 'Number of Items by Main Category',
              yaxis = dict(title='Count'),
              xaxis = dict(title='Category'))
fig=dict(data=[trace1], layout=layout)
py.iplot(fig)
```



Observations from Main Category:

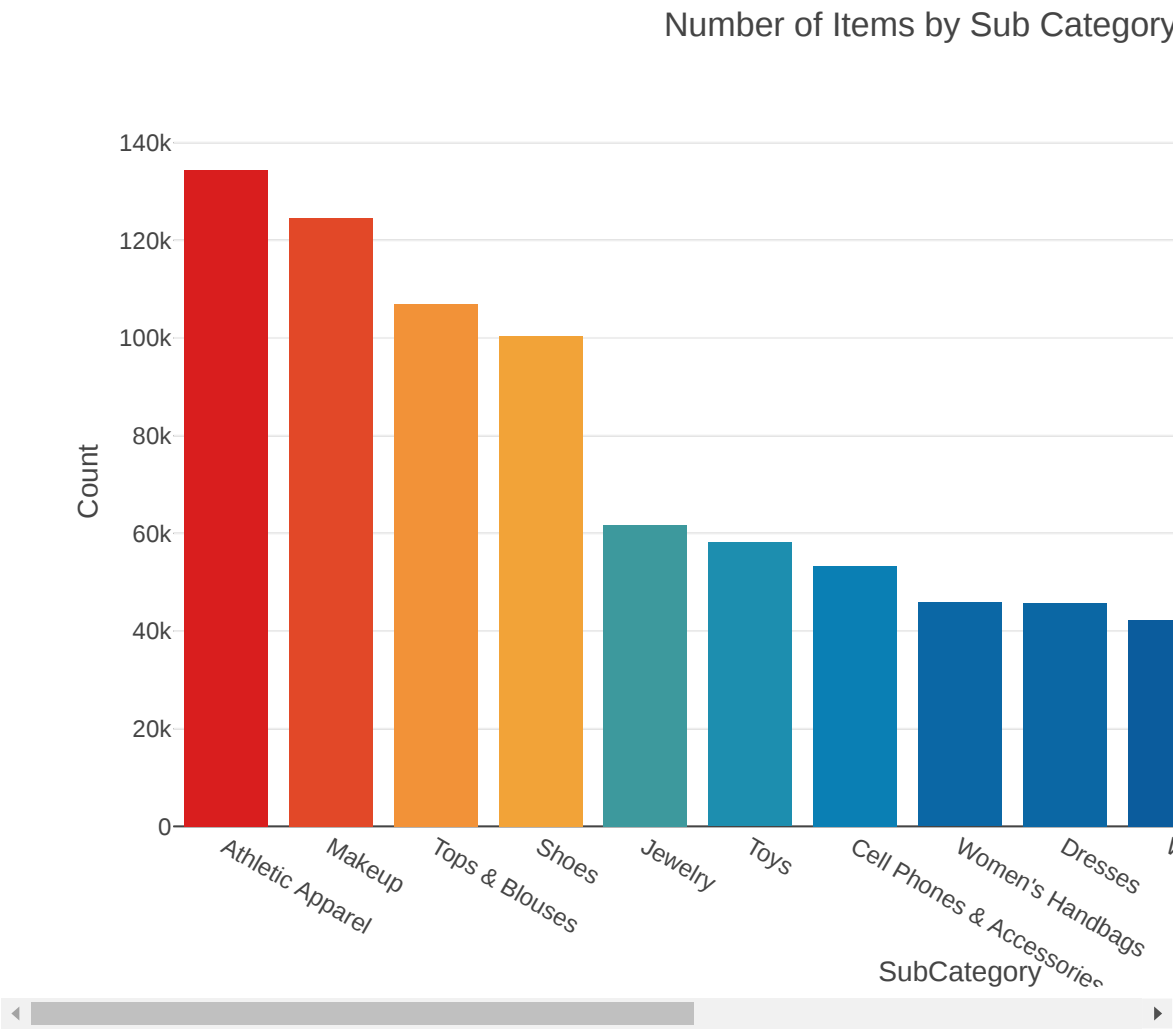
- Women's category is the most popular category constitutes 44.81% of total items followed by Beauty, 14.02% of total items.

In [96]:

```
x = train['subcat_1'].value_counts().index.values.astype('str')[:15]
y = train['subcat_1'].value_counts().values[:15]
pct = [("%.2f"%(v*100))+"%"for v in (y/len(train))][:15]
```

In [95]:

```
trace1 = go.Bar(x=x, y=y, text=pct,
                marker=dict(
                color = y,colorscale='Portland',showscale=True,
                reversescale = False
                ))
layout = dict(title= 'Number of Items by Sub Category (Top 15)',
              yaxis = dict(title='Count'),
              xaxis = dict(title='SubCategory'))
fig=dict(data=[trace1], layout=layout)
py.iplot(fig)
```



Observation from Sub Category 1:

- Athletic Apparel is the most popular category with 9.07% followed by Makeup with 8.41%

From the pricing (log of price) point of view, all the categories are pretty well distributed, with no category with an extraordinary pricing point

In [98]:

```
main_cats = train['main_cat'].unique()
x = [train.loc[train['main_cat']==cat, 'price'] for cat in main_cats]
```

In [100]:

```
data = [go.Box(x=np.log(x[i]+1), name=main_cats[i]) for i in range(len(main_cats
))]
```

In [102]:

```
layout = dict(title="Price Distribution by Main Category",
              yaxis = dict(title='Frequency'),
              xaxis = dict(title='Category'))
fig = dict(data=data, layout=layout)
py.iplot(fig)
```

Brand Name

- Brand Name is a categorical feature.

In [103]:

```
print("There are %d unique brand names in the training dataset." % train['brand_
name'].nunique())
```

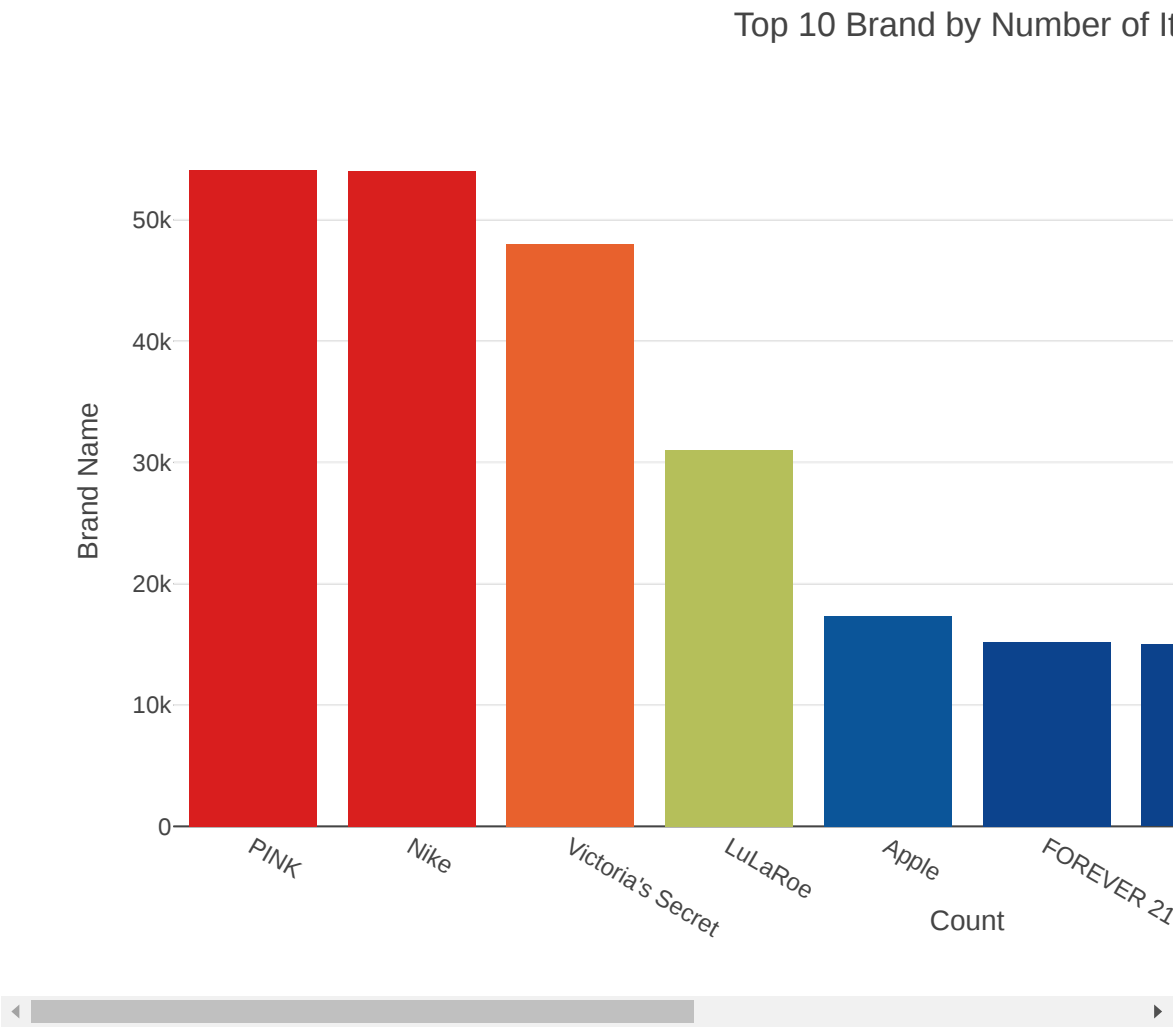
```
There are 4807 unique brand names in the training dataset.
```

In [104]:

```
x = train['brand_name'].value_counts().index.values.astype('str')[:10]
y = train['brand_name'].value_counts().values[:10]
```

In [105]:

```
trace1 = go.Bar(x=x, y=y,
                marker=dict(
                color = y,colorscale='Portland',showscale=True,
                reversescale = False
                ))
layout = dict(title= 'Top 10 Brand by Number of Items',
              yaxis = dict(title='Brand Name'),
              xaxis = dict(title='Count'))
fig=dict(data=[trace1], layout=layout)
py.iplot(fig)
```



Item Description

- Item Description is a text feautre.
- It will be more challenging to parse through this particular item since it's unstructured data.
- We will strip out all punctuations, remove some english stop words (i.e. redundant words such as "a", "the", etc.) and any other words with a length less than 3.
- Does it mean a more detailed and lengthy description will result in a higher bidding price? Let's check it out !!

In [108]:

```
def wordCount(text):
    # convert to lower case and strip regex
    try:
         # convert to lower case and strip regex
        text = text.lower()
        regex = re.compile('[' +re.escape(string.punctuation) + '0-9\\r\\t\\n]')
        txt = regex.sub(" ", text)
        # tokenize
        # words = nltk.word_tokenize(clean_txt)
        # remove words in stop words
        words = [w for w in txt.split(" ") \
                 if not w in stop_words.ENGLISH_STOP_WORDS and len(w)>3]
        return len(words)
    except:
        return 0
```

In [109]:

```
# add a column of word counts to both the training and test set
train['desc_len'] = train['item_description'].apply(lambda x: wordCount(x))
test['desc_len'] = test['item_description'].apply(lambda x: wordCount(x))
```

In [111]:

```
train.head()
```

Out[111]:

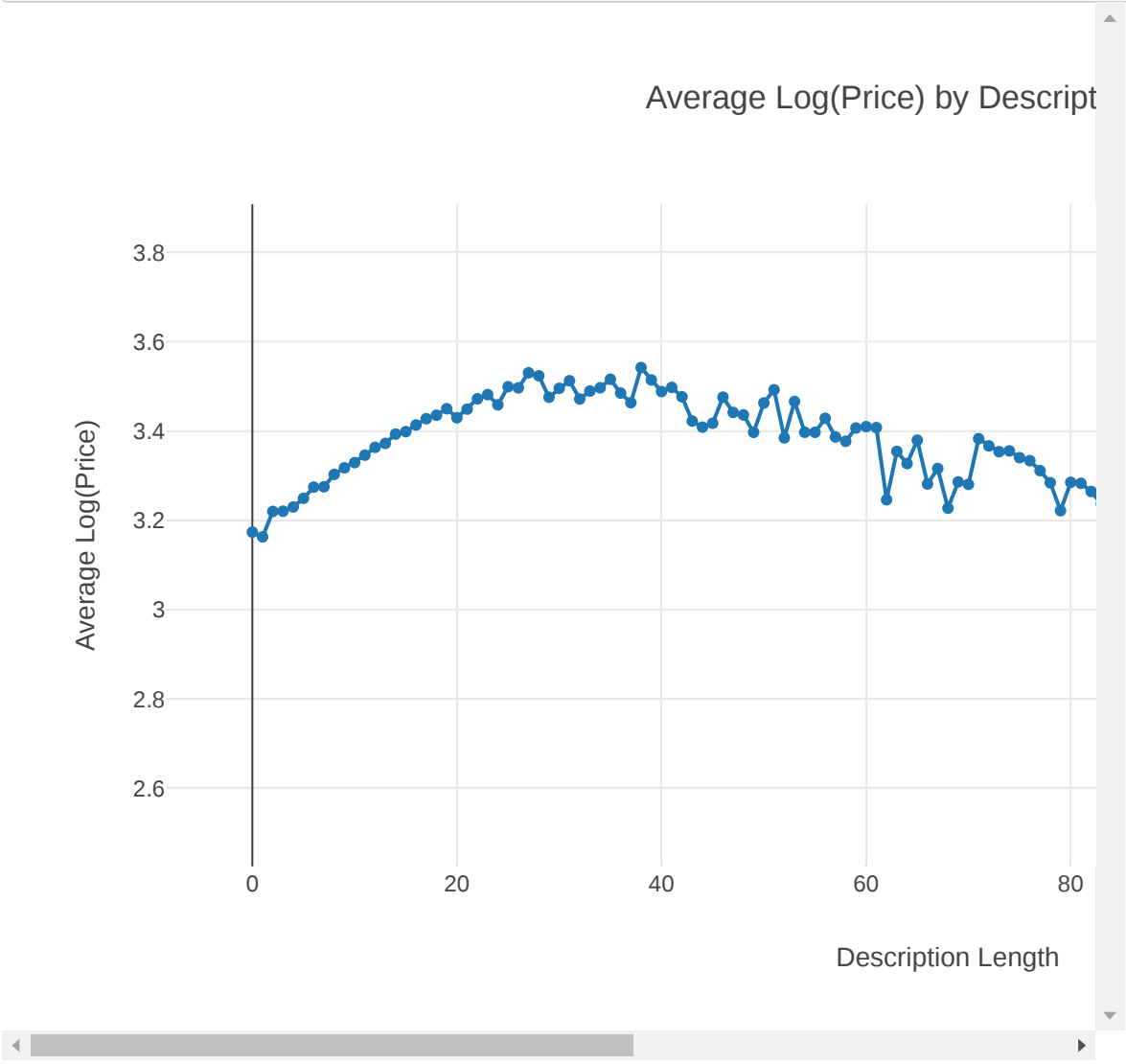
	train_id	name	item_condition_id	category_name	brand_name
0	0	MLB Cincinnati Reds T Shirt Size XL	3	Men/Tops/T-shirts	NaN
1	1	Razer BlackWidow Chroma Keyboard	3	Electronics/Computers & Tablets/Components & P...	Razer
2	2	AVA-VIV Blouse	1	Women/Tops & Blouses/Blouse	Target
3	3	Leather Horse Statues	1	Home/Home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

In [115]:

```
# mean price of unique description lengths
df = train.groupby('desc_len')['price'].mean().reset_index()
```

In [116]:

```
trace1 = go.Scatter(
    x = df['desc_len'],
    y = np.log(df['price']+1),
    mode = 'lines+markers',
    name = 'lines+markers'
)
layout = dict(title= 'Average Log(Price) by Description Length',
              yaxis = dict(title='Average Log(Price)'),
              xaxis = dict(title='Description Length'))
fig=dict(data=[trace1], layout=layout)
py.iplot(fig)
```



We also need to check if there are any missing values in the item description (4 observations don't have a description) andl remove those observations from our training set.

In [117]:

```
train.item_description.isnull().sum()
```

Out[117]:

4

In [118]:

```
# remove missing values in item description
train = train[pd.notnull(train['item_description'])]
```

If we look at the most common words by category, we could also see that, ***size***, ***free*** and ***shipping*** is very commonly used by the sellers, probably with the intention to attract customers, which is contradictory to what we have shown previously that there is little correlation between the two variables `price` and `shipping` (or shipping fees do not account for a differentiation in prices). ***Brand names*** also played quite an important role - it's one of the most popular in all four categories.

Item Description

The following section is based on the tutorial at https://ahmedbesbes.com/how-to-mine-newsfeed-data-and-extract-interactive-insights-in-python.html (https://ahmedbesbes.com/how-to-mine-newsfeed-data-and-extract-interactive-insights-in-python.html)

- Item Description is a text feature.
- Pre-processing has to be done on text featue before we analyze.

In [120]:

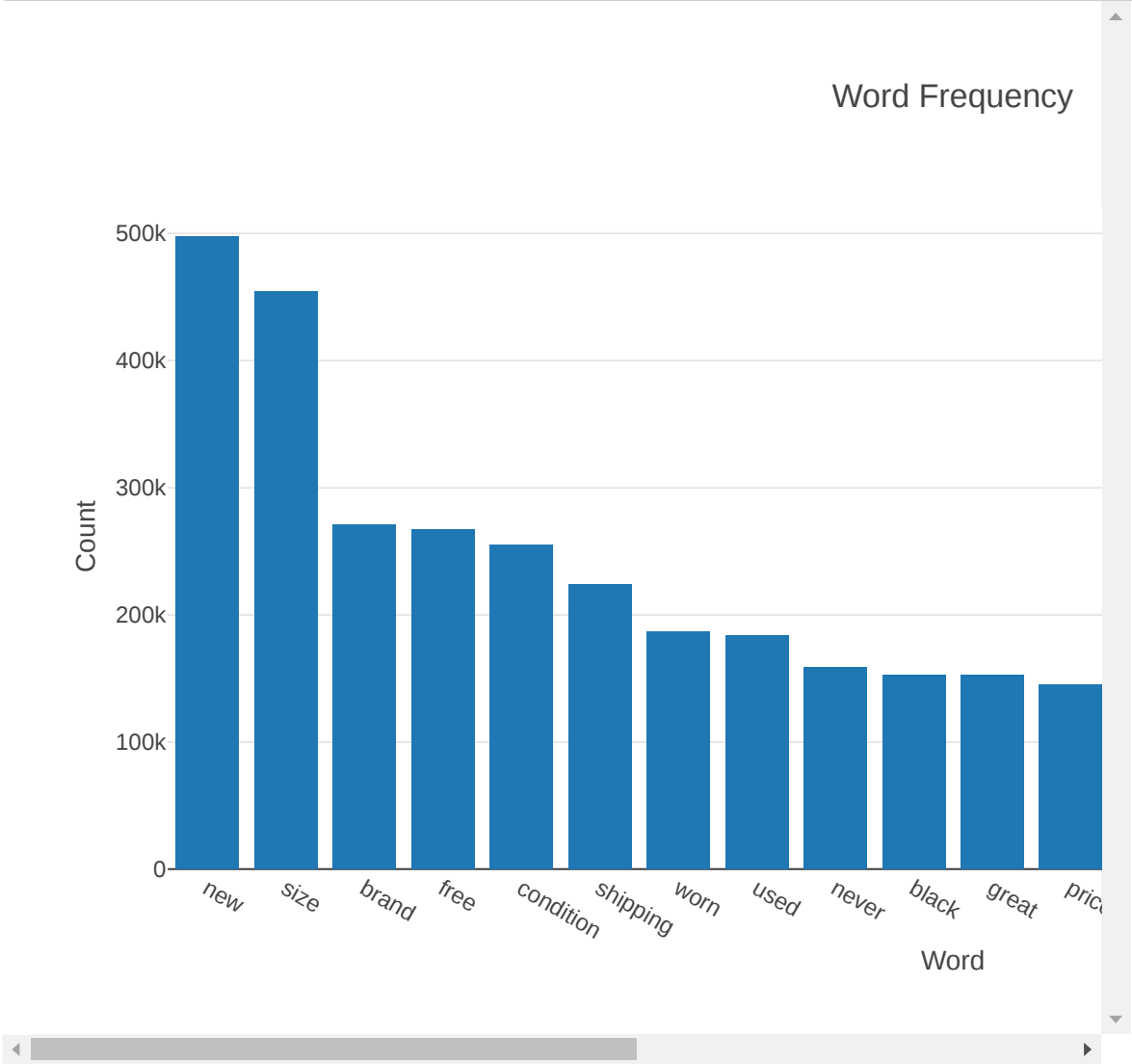
```
stop = set(stopwords.words('english'))
def tokenize(text):
    """
    sent_tokenize(): segment text into sentences
    word_tokenize(): break sentences into words
    """
    try:
        regex = re.compile('[' +re.escape(string.punctuation) + '0-9\\r\\t\\n]')
        text = regex.sub(" ", text) # remove punctuation

        tokens_ = [word_tokenize(s) for s in sent_tokenize(text)]
        tokens = []
        for token_by_sent in tokens_:
            tokens += token_by_sent
        tokens = list(filter(lambda t: t.lower() not in stop, tokens))
        filtered_tokens = [w for w in tokens if re.search('[a-zA-Z]', w)]
        filtered_tokens = [w.lower() for w in filtered_tokens if len(w)>=3]

        return filtered_tokens

    except TypeError as e: print(text,e)
```

In [122]:

```
# create a dictionary of words for each category
cat_desc = dict()
for cat in main_cats:
    text = " ".join(train.loc[train['main_cat']==cat, 'item_description'].values
)
    cat_desc[cat] = tokenize(text)

# flat list of all words combined
flat_lst = [item for sublist in list(cat_desc.values()) for item in sublist]
allWordsCount = Counter(flat_lst)
all_top10 = allWordsCount.most_common(20)
x = [w[0] for w in all_top10]
y = [w[1] for w in all_top10]
```

In [123]:

```
trace1 = go.Bar(x=x, y=y, text=pct)
layout = dict(title= 'Word Frequency',
              yaxis = dict(title='Count'),
              xaxis = dict(title='Word'))
fig=dict(data=[trace1], layout=layout)
py.iplot(fig)
```



Observations:

- ***size***, ***free*** and ***shipping*** are very commonly used by the sellers, probably with the intention to attract customers.
- ***Brand names*** also played quite an important role - it's one of the most popular in all four categories.

Pre-processing

Tokenization

Most of the time, the first steps of an NLP project is to **"tokenize"** your documents, which main purpose is to normalize our texts. The three fundamental stages will usually include:

- break the descriptions into sentences and then break the sentences into tokens.
- remove punctuation and stop words.
- lowercase the tokens.
- herein, I will also only consider words that have length equal to or greater than 3 characters.

In [125]:

```
# apply the tokenizer into the item descriptipn column
train['tokens'] = train['item_description'].map(tokenize)
test['tokens'] = test['item_description'].map(tokenize)
```

In [126]:

```
train.reset_index(drop=True, inplace=True)
test.reset_index(drop=True, inplace=True)
```

Let's look at the examples of if the tokenizer did a good job in cleaning up our descriptions

In [127]:

```
for description, tokens in zip(train['item_description'].head(),
                               train['tokens'].head()):
    print('description:', description)
    print('tokens:', tokens)
    print()
```

```
description: No description yet
tokens: ['description', 'yet']

description: This keyboard is in great condition and works like it c
ame out of the box. All of the ports are tested and work perfectly.
The lights are customizable via the Razer Synapse app on your PC.
tokens: ['keyboard', 'great', 'condition', 'works', 'like', 'came',
'box', 'ports', 'tested', 'work', 'perfectly', 'lights', 'customizab
le', 'via', 'razer', 'synapse', 'app']

description: Adorable top with a hint of lace and a key hole in the
back! The pale pink is a 1X, and I also have a 3X available in whit
e!
tokens: ['adorable', 'top', 'hint', 'lace', 'key', 'hole', 'back',
'pale', 'pink', 'also', 'available', 'white']

description: New with tags. Leather horses. Retail for [rm] each. St
and about a foot high. They are being sold as a pair. Any questions
please ask. Free shipping. Just got out of storage
tokens: ['new', 'tags', 'leather', 'horses', 'retail', 'stand', 'foo
t', 'high', 'sold', 'pair', 'questions', 'please', 'ask', 'free', 's
hipping', 'got', 'storage']

description: Complete with certificate of authenticity
tokens: ['complete', 'certificate', 'authenticity']
```

We could aso use the package `WordCloud` to easily visualize which words has the highest frequencies within each category:

In [129]:

```
# build dictionary with key=category and values as all the descriptions related.
cat_desc = dict()
for cat in main_cats:
    text = " ".join(train.loc[train['main_cat']==cat, 'item_description'].values
)
    cat_desc[cat] = tokenize(text)


# find the most common words for the top 4 categories
women100 = Counter(cat_desc['Women']).most_common(100)
beauty100 = Counter(cat_desc['Beauty']).most_common(100)
kids100 = Counter(cat_desc['Kids']).most_common(100)
electronics100 = Counter(cat_desc['Electronics']).most_common(100)
```

In [139]:

```
def generate_wordcloud(tup):
    wordcloud = WordCloud(background_color='black',
                          max_words=50, max_font_size=30,
                          random_state=36
                         ).generate(str(tup))
    return wordcloud
```

In [140]:

```
fig,axes = plt.subplots(2, 2, figsize=(30, 15))

ax = axes[0, 0]
ax.imshow(generate_wordcloud(women100), interpolation="sinc")
ax.axis('off')
ax.set_title("Women Top 100", fontsize=30)

ax = axes[0, 1]
ax.imshow(generate_wordcloud(beauty100))
ax.axis('off')
ax.set_title("Beauty Top 100", fontsize=30)

ax = axes[1, 0]
ax.imshow(generate_wordcloud(kids100))
ax.axis('off')
ax.set_title("Kids Top 100", fontsize=30)

ax = axes[1, 1]
ax.imshow(generate_wordcloud(electronics100))
ax.axis('off')
ax.set_title("Electronic Top 100", fontsize=30)
```

Out[140]:

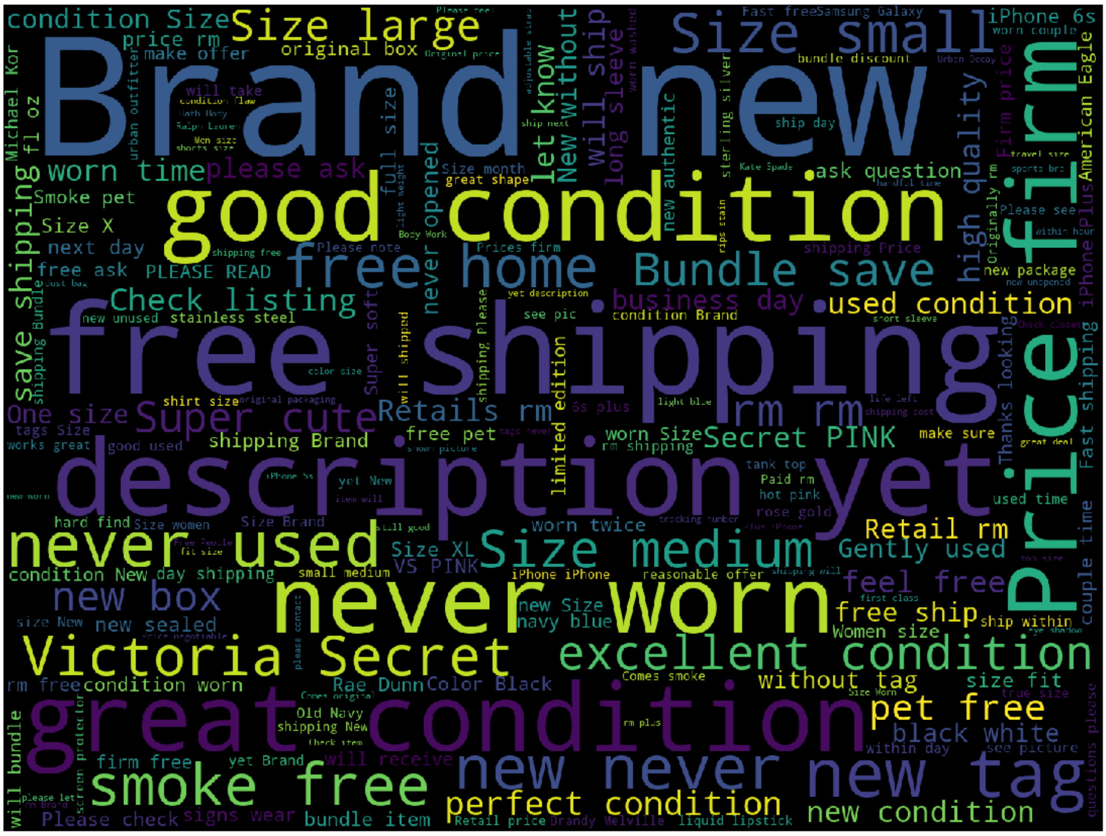
Text(0.5,1,'Electronic Top 100')

In [142]:

```
cloud = WordCloud(width=1440, height=1080).generate(" ".join(train['item_descrip
tion']
.astype(str)))
plt.figure(figsize=(20, 15))
plt.imshow(cloud)
plt.axis('off')
```

Out[142]:

```
(-0.5, 1439.5, 1079.5, -0.5)
```



Pre-processing: tf-idf

tf-idf is the acronym for **Term Frequency–inverse Document Frequency**. It quantifies the importance of a particular word in relative to the vocabulary of a collection of documents or corpus. The metric depends on two factors:

- **Term Frequency**: the occurences of a word in a given document (i.e. bag of words)
- **Inverse Document Frequency**: the reciprocal number of times a word occurs in a corpus of documents

Think about of it this way: If the word is used extensively in all documents, its existence within a specific document will not be able to provide us much specific information about the document itself. So the second term could be seen as a penalty term that penalizes common words such as "a", "the", "and", etc. tf-idf can therefore, be seen as a weighting scheme for words relevancy in a specific document.

In [ ]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10,
                             max_features=180000,
                             tokenizer=tokenize,
                             ngram_range=(1, 2))
```

In [ ]:

```
all_desc = np.append(train['item_description'].values, test['item_description'].
values)
vz = vectorizer.fit_transform(list(all_desc))
```

vz is a tfidf matrix where:

- the number of rows is the total number of descriptions
- the number of columns is the total number of unique tokens across the descriptions

In [ ]:

```
#  create a dictionary mapping the tokens to their tfidf values
tfidf = dict(zip(vectorizer.get_feature_names(), vectorizer.idf_))
tfidf = pd.DataFrame(columns=['tfidf']).from_dict(
                    dict(tfidf), orient='index')
tfidf.columns = ['tfidf']
```

Below is the 10 tokens with the lowest tfidf score, which is unsurprisingly, very generic words that we could not use to distinguish one description from another.

In [ ]:

```
tfidf.sort_values(by=['tfidf'], ascending=True).head(10)
```

Below is the 10 tokens with the highest tfidf score, which includes words that are a lot specific that by looking at them, we could guess the categories that they belong to:

In [ ]:

```
tfidf.sort_values(by=['tfidf'], ascending=False).head(10)
```

Given the high dimension of our tfidf matrix, we need to reduce their dimension using the Singular Value Decomposition (SVD) technique. And to visualize our vocabulary, we could next use t-SNE to reduce the dimension from 50 to 2. t-SNE is more suitable for dimensionality reduction to 2 or 3.

t-Distributed Stochastic Neighbor Embedding (t-SNE)

t-SNE is a technique for dimensionality reduction that is particularly well suited for the visualization of high-dimensional datasets. The goal is to take a set of points in a high-dimensional space and find a representation of those points in a lower-dimensional space, typically the 2D plane. It is based on probability distributions with random walk on neighborhood graphs to find the structure within the data. But since t-SNE complexity is significantly high, usually we'd use other high-dimension reduction techniques before applying t-SNE.

First, let's take a sample from the both training and testing item's description since t-SNE can take a very long time to execute. We can then reduce the dimension of each vector from to n_components (50) using SVD.

In [ ]:

```
trn = train.copy()
tst = test.copy()
trn['is_train'] = 1
tst['is_train'] = 0

sample_sz = 15000

combined_df = pd.concat([trn, tst])
combined_sample = combined_df.sample(n=sample_sz)
vz_sample = vectorizer.fit_transform(list(combined_sample['item_description']))
```

In [ ]:

```
from sklearn.decomposition import TruncatedSVD

n_comp=30
svd = TruncatedSVD(n_components=n_comp, random_state=42)
svd_tfidf = svd.fit_transform(vz_sample)
```

Now we can reduce the dimension from 50 to 2 using t-SNE!

In [ ]:

```
from sklearn.manifold import TSNE
tsne_model = TSNE(n_components=2, verbose=1, random_state=42, n_iter=500)
```

In [ ]:

```
tsne_tfidf = tsne_model.fit_transform(svd_tfidf)
```

It's now possible to visualize our data points. Note that the deviation as well as the size of the clusters imply little information in t-SNE.

In [ ]:

```
import bokeh.plotting as bp
from bokeh.models import HoverTool, BoxSelectTool
from bokeh.models import ColumnDataSource
from bokeh.plotting import figure, show, output_notebook
```

In [ ]:

```
output_notebook()
plot_tfidf = bp.figure(plot_width=700, plot_height=600,
                       title="tf-idf clustering of the item description",
    tools="pan,wheel_zoom,box_zoom,reset,hover,previewsave",
    x_axis_type=None, y_axis_type=None, min_border=1)
```

In [ ]:

```
combined_sample.reset_index(inplace=True, drop=True)
```

In [ ]:

```
tfidf_df = pd.DataFrame(tsne_tfidf, columns=['x', 'y'])
tfidf_df['description'] = combined_sample['item_description']
tfidf_df['tokens'] = combined_sample['tokens']
tfidf_df['category'] = combined_sample['general_cat']
```

In [ ]:

```
plot_tfidf.scatter(x='x', y='y', source=tfidf_df, alpha=0.7)
hover = plot_tfidf.select(dict(type=HoverTool))
hover.tooltips={"description": "@description", "tokens": "@tokens", "category":
"@category"}
show(plot_tfidf)
```

K-Means Clustering

K-means clustering obejctive is to minimize the average squared Euclidean distance of the document / description from their cluster centroids.

In [ ]:

```
from sklearn.cluster import MiniBatchKMeans

num_clusters = 30 # need to be selected wisely
kmeans_model = MiniBatchKMeans(n_clusters=num_clusters,
                               init='k-means++',
                               n_init=1,
                               init_size=1000, batch_size=1000, verbose=0, max_i
ter=1000)
```

In [ ]:

```
kmeans = kmeans_model.fit(vz)
kmeans_clusters = kmeans.predict(vz)
kmeans_distances = kmeans.transform(vz)
```

In [ ]:

```
"""sorted_centroids = kmeans.cluster_centers_.argsort()[:, ::-1]
terms = vectorizer.get_feature_names()

for i in range(num_clusters):
    print("Cluster %d:" % i)
    aux = ''
    for j in sorted_centroids[i, :10]:
        aux += terms[j] + ' | '
    print(aux)
    print() """
```

In order to plot these clusters, first we will need to reduce the dimension of the distances to 2 using tsne: m

In [ ]:

```
# repeat the same steps for the sample
kmeans = kmeans_model.fit(vz_sample)
kmeans_clusters = kmeans.predict(vz_sample)
kmeans_distances = kmeans.transform(vz_sample)
# reduce dimension to 2 using tsne
tsne_kmeans = tsne_model.fit_transform(kmeans_distances)
```

In [ ]:

```
colormap = np.array(["#6d8dca", "#69de53", "#723bca", "#c3e14c", "#c84dc9", "#68
af4e", "#6e6cd5",
"#e3be38", "#4e2d7c", "#5fdfa8", "#d34690", "#3f6d31", "#d44427", "#7fcdd8", "#c
b4053", "#5e9981",
"#803a62", "#9b9e39", "#c88cca", "#e1c37b", "#34223b", "#bdd8a3", "#6e3326", "#c
fbdce", "#d07d3c",
"#52697d", "#194196", "#d27c88", "#36422b", "#b68f79"])
```

In [ ]:

```
#combined_sample.reset_index(drop=True, inplace=True)
kmeans_df = pd.DataFrame(tsne_kmeans, columns=['x', 'y'])
kmeans_df['cluster'] = kmeans_clusters
kmeans_df['description'] = combined_sample['item_description']
kmeans_df['category'] = combined_sample['general_cat']
#kmeans_df['cluster']=kmeans_df.cluster.astype(str).astype('category')
```

In [ ]:

```
plot_kmeans = bp.figure(plot_width=700, plot_height=600,
                        title="KMeans clustering of the description",
    tools="pan,wheel_zoom,box_zoom,reset,hover,previewsave",
    x_axis_type=None, y_axis_type=None, min_border=1)
```

In [ ]:

```
source = ColumnDataSource(data=dict(x=kmeans_df['x'], y=kmeans_df['y'],
                                    color=colormap[kmeans_clusters],
                                    description=kmeans_df['description'],
                                    category=kmeans_df['category'],
                                    cluster=kmeans_df['cluster']))

plot_kmeans.scatter(x='x', y='y', color='color', source=source)
hover = plot_kmeans.select(dict(type=HoverTool))
hover.tooltips={"description": "@description", "category": "@category", "cluste
r":"@cluster" }
show(plot_kmeans)
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: