Mercari Price Suggestion Challenge

Overview

Predicting the price of a product is a tough challenge since very similar products having minute differences such as different brand name, additional specifications, quality of the product, demand of the product, etc. can have very different prizes. For example the price of a pair of running shoes by a very common brand (say Puma) might be around INR 2,500 whereas a similar pair made by Asics can cost around INR 10,000.

Price prediction gets even more difficult when there is a huge range of products, which is common with most of the online shopping platforms. While it might be simpler to predict the price of a particular category of products using some simple criteria, it's highly challenging to predict the price of almost anything that is listed on online platforms. We may have multiple listings of the same product by a large number of sellers priced differently.

Mercari is Japan's biggest community-powered shopping app. Mercari's challenge is to build an algorithm that automatically suggests the right product prices.

Data collection

The data can be downloaded from Kaggle competion page.

We have been provided user-inputted text descriptions of their products, including details like product category name, brand name, and item condition

ML Problem

Using the given data, we have to come up with a model that predicts the price of a product listed on Mercari as accurately as possible.

This is a standard regression problem.

Performance Metric

The performance of the model is measured by Root Mean Squared Logarithmic Error(RMSLE). Lesser the RMSLE, better is our prediction model.

The RMSLE is calculated as \begin{align*} \epsilon = \sqrt{\frac{1}{n} \sum_{i=1}^n (\log(p_i + 1) - \log(a_i+1))^2 } \end{align*}

- ϵ is the RMSLE value (score)
- n is the total number of observations in the (public/private) data set,
- pi is your prediction of price, and
- ai is the actual sale price for i
- log(x) is the natural logarithm of x

In [1]:

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

import os
import shutil
import datetime
import gc
from tqdm import tqdm

import pandas as pd
import numpy as np
from numpy import median

%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style='whitegrid')
```

```python
from sklearn.manifold import TSNE
from sklearn import preprocessing

from collections import Counter

import string
import re
from nltk.corpus import stopwords

import scipy
from scipy import hstack


from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.metrics import mean_squared_error as mse
from math import sqrt
from sklearn.linear_model import Ridge

from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import log_loss

from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint as sp_randint
from scipy.stats import uniform
from sklearn.feature_selection.univariate_selection import SelectKBest, f_regression
```

In [2]:

```python
import tracemalloc
import time
```

In [3]:

```python
tracemalloc.start()

start_time = time.time()
snapshot1 = tracemalloc.take_snapshot()
```

0. Loading data

In [4]:

```python
# https://www.kaggle.com/peterhurford/lgb-and-fm-18th-place-0-40604
def split_cat(text):
    try:
        return text.split("/")
    except:
        return ("No Label", "No Label", "No Label")
```

In [5]:

```python
train = pd.read_csv('train.tsv', sep='\t',
                      dtype={'item_condition_id': 'category', 'shipping': 'category'},
                      converters={'category_name': split_cat})
test = pd.read_csv('test.tsv', sep='\t',
                     dtype={'item_condition_id': 'category', 'shipping': 'category'},
                     converters={'category_name': split_cat})
```

In [6]:

```python
print('Shape of train data: ', train.shape)
print('Shape of test data: ', test.shape)
train.head(5)
```

```
Shape of train data:  (1482535, 8)
Shape of test data:  (3460725, 7)
```

Out[6]:

	train_id	name	item_condition_id	category_name	brand_name	price	shipping	item_description
0	0	MLB Cincinnati Reds T Shirt Size XL	3	[Men, Tops, T-shirts]	NaN	10.0	1	No description yet
1	1	Razer BlackWidow Chroma Keyboard	3	[Electronics, Computers & Tablets, Components ...	Razer	52.0	0	This keyboard is in great condition and works ...
2	2	AVA-VIV Blouse	1	[Women, Tops & Blouses, Blouse]	Target	10.0	1	Adorable top with a hint of lace and a key hol...
3	3	Leather Horse Statues	1	[Home, Home Décor, Home Décor Accents]	NaN	35.0	1	New with tags. Leather horses. Retail for [rm]...
4	4	24K GOLD plated rose	1	[Women, Jewelry, Necklaces]	NaN	44.0	0	Complete with certificate of authenticity

1. Data Overview

Handling missing values

In [7]:

```
train.isnull().any()
```

Out[7]:

```
train_id             False
name                 False
item_condition_id    False
category_name        False
brand_name            True
price                False
shipping             False
item_description      True
dtype: bool
```

In [8]:

```
test.isnull().any()
```

Out[8]:

```
test_id              False
name                 False
item_condition_id    False
category_name        False
brand_name            True
shipping             False
item_description      True
dtype: bool
```

Columns brand_name, item_description have NAs.

NAs in category_name have been replaced by empty lists because of the converter we have used while loading the data.

We will replace NAs, empty lists with 'missing'.

In [9]:

```
# Split category_name by '/' into subcategories and replace nulls with 'missing'
train['gencat_name'] = train['category_name'].str.get(0).replace('', 'missing').astype('category')
train['subcat1_name'] = train['category_name'].str.get(1).fillna('missing').astype('category')
train['subcat2_name'] = train['category_name'].str.get(2).fillna('missing').astype('category')
train.drop('category_name', axis=1, inplace=True)
```

In [10]:

```
# Split category_name by '/' into subcategories and replace nulls with 'missing'
test['gencat_name'] = test['category_name'].str.get(0).replace('', 'missing').astype('category')
test['subcat1_name'] = test['category_name'].str.get(1).fillna('missing').astype('category')
test['subcat2_name'] = test['category_name'].str.get(2).fillna('missing').astype('category')
test.drop('category_name', axis=1, inplace=True)
```

In [11]:

```
train['item_description'].fillna('missing', inplace=True)
train['brand_name'] = train['brand_name'].fillna('missing').astype('category')
```

In [12]:

```
test['item_description'].fillna('missing', inplace=True)
test['brand_name'] = test['brand_name'].fillna('missing').astype('category')
```

Check for duplicate rows, NAs

In [13]:

```
train[train.duplicated()]
```

Out[13]:

train_id	name	item_condition_id	brand_name	price	shipping	item_description	gencat_name	subcat1_name	subcat2_name

No duplicate rows in train data

In [14]:

```
train.isnull().any()
```

Out[14]:

```
train_id             False
name                 False
item_condition_id    False
brand_name           False
price                False
shipping             False
item_description     False
gencat_name          False
subcat1_name         False
subcat2_name         False
dtype: bool
```

Check for rows with invalid price

In [15]:

```
print('Removed {} rows' .format(len(train[train.price<=0])))
train = train[train.price > 0].reset_index(drop=True)
```

```
Removed 874 rows
```

Column-wise overview of data

name

In [16]:

```
train.name.describe()
```

Out[16]:

```
count     1481661
unique    1224596
top        Bundle
freq         2232
Name: name, dtype: object
```

item_condition_id

In [17]:

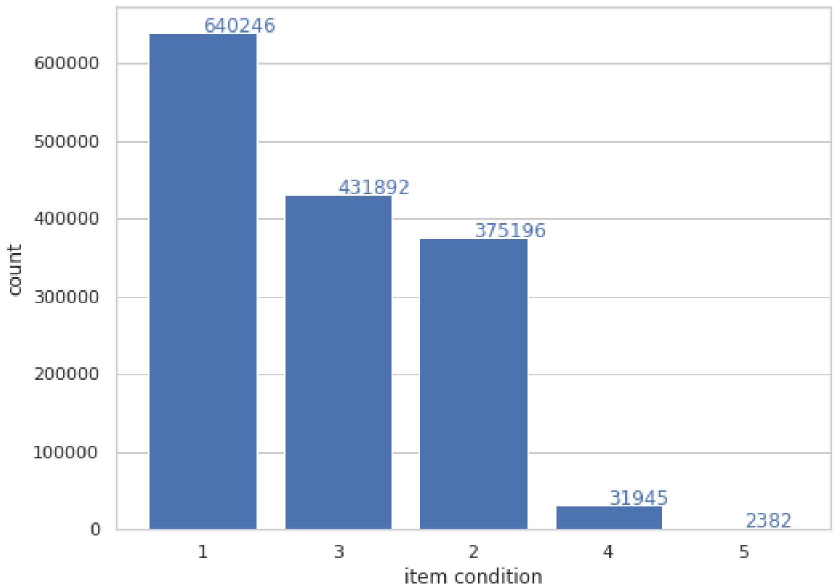
```
train.item_condition_id.describe()
```

Out[17]:

```
count     1481661
unique          5
top             1
freq       640246
Name: item_condition_id, dtype: object
```

In [18]:

```
condition_count = Counter(list(train.item_condition_id))
x, y = zip(*condition_count.most_common())
plt.figure(figsize=[8,6])
plt.bar(x, y, )
for i, val in enumerate(y):
           plt.annotate(val, (x[i], y[i]), color='b')
plt.xlabel('item condition')
plt.ylabel('count')
plt.grid(False, axis='x')
plt.show()
```



Majority of the items are in **condition 1**

brand_name

In [19]:

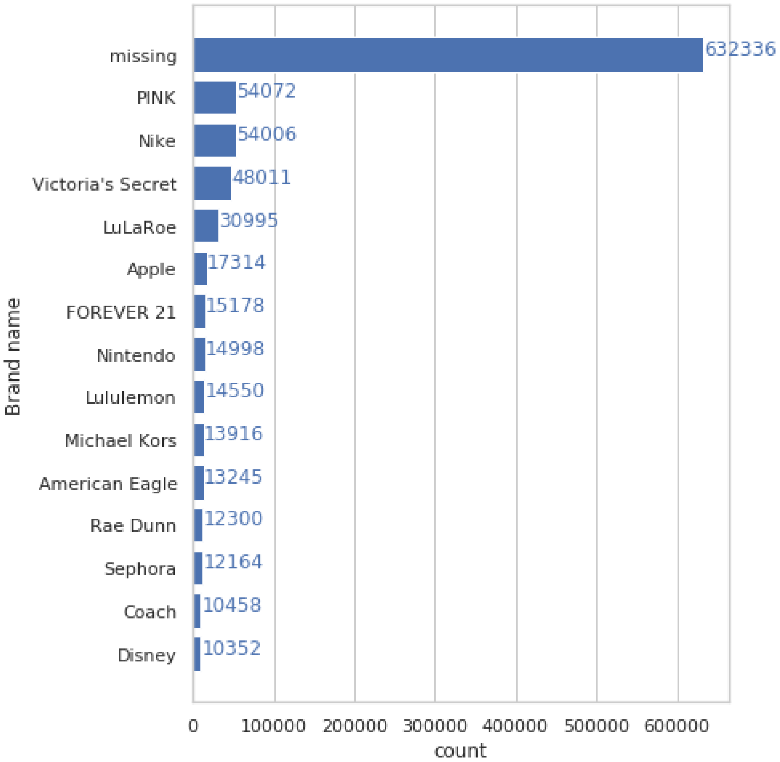
```
train.brand_name.describe()
```

Out[19]:

```
count      1481661
unique        4808
top        missing
freq        632336
Name: brand_name, dtype: object
```

In [20]:

```
brand_count = Counter(list(train.brand_name.values))
x, y = zip(*brand_count.most_common(15))

plt.figure(figsize=[6,8])
plt.barh(x, y)
for i, val in enumerate(y):
           plt.annotate(val, (y[i], x[i]), color='b')
plt.gca().invert_yaxis()
plt.ylabel('Brand name')
plt.xlabel('count')
plt.grid(False, axis='y')
plt.show()
```



In [21]:

```
brand_missing = train[train.brand_name=='missing'].shape[0]
print('Brand name is missing for {} datapoints, i.e. {:.2f} % of train data.'
.format(brand_missing, 100.0*brand_missing/train.shape[0]))
```

```
Brand name is missing for 632336 datapoints, i.e. 42.68 % of train data.
```

gencat_name

In [22]:

```
train.gencat_name.describe()
```

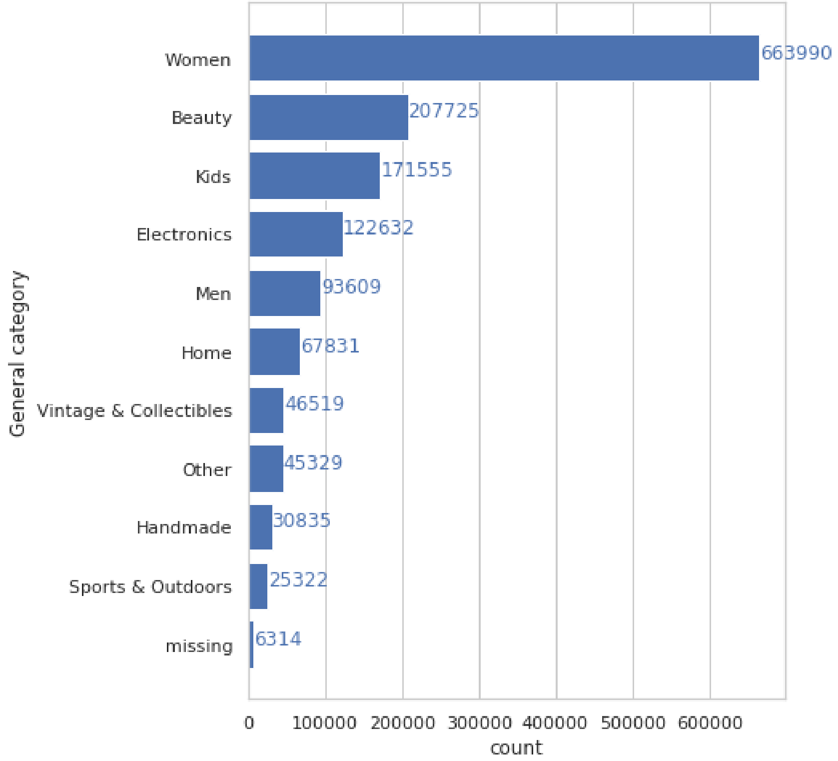
Out[22]:

```
count      1481661
unique          11
top          Women
freq        663990
```

```
Name: gencat_name, dtype: object
```

In [23]:

```
gencat_count = Counter(list(train.gencat_name.values))
x, y = zip(*gencat_count.most_common(15))
plt.figure(figsize=[6,8])
plt.barh(x, y)
for i, val in enumerate(y):
           plt.annotate(val, (y[i], x[i]), color='b')
plt.gca().invert_yaxis()
plt.ylabel('General category')
plt.xlabel('count')
plt.grid(False, axis='y')
plt.show()
```



Majority of the items are from the category **women**

In [24]:

```
gencat_missing = train[train.gencat_name=='missing'].shape[0]
print('category name is missing for {} datapoints, i.e. {:.2f} % of train data.'
.format(gencat_missing, 100.0*gencat_missing/train.shape[0]))
```

```
category name is missing for 6314 datapoints, i.e. 0.43 % of train data.
```

subcat1_name

In [25]:

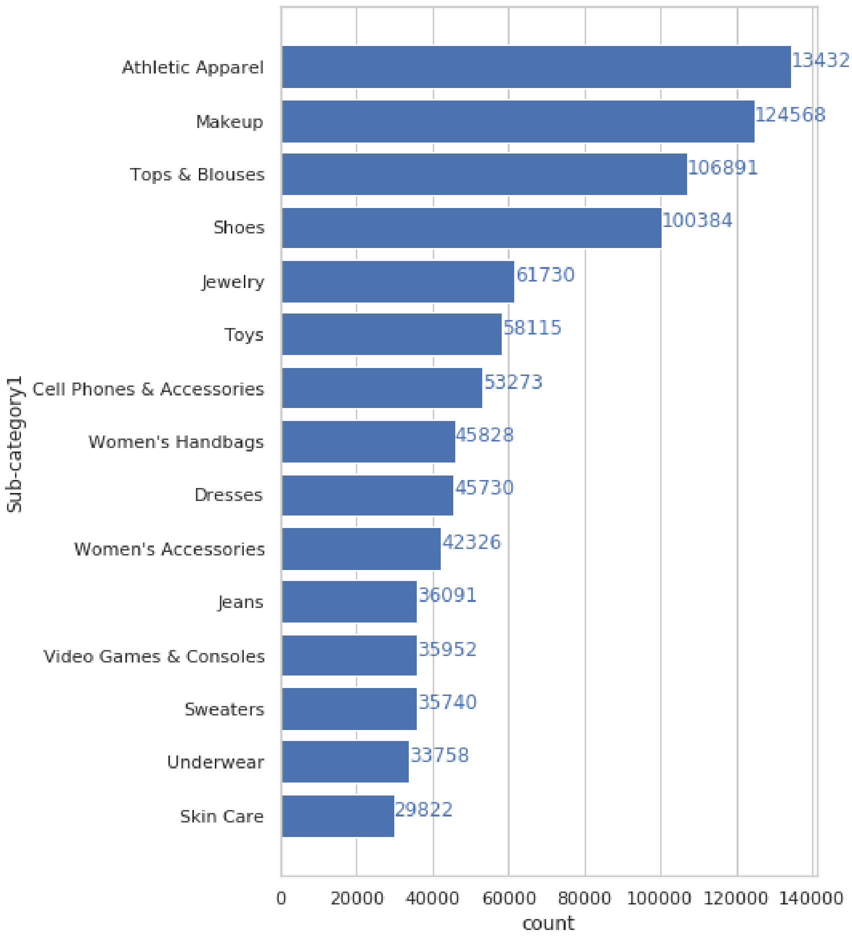
```
train.subcat1_name.describe()
```

Out[25]:

```
count              1481661
unique                 114
top        Athletic Apparel
freq                134321
Name: subcat1_name, dtype: object
```

In [26]:

```
subcat1_count = Counter(list(train.subcat1_name.values))
x, y = zip(*subcat1_count.most_common(15))
plt.figure(figsize=[6,10])
plt.barh(x, y)
for i, val in enumerate(y):
           plt.annotate(val, (y[i], x[i]), color='b')
plt.gca().invert_yaxis()
plt.ylabel('Sub-category1')
plt.xlabel('count')
plt.grid(False, axis='y')
plt.show()
```



In [27]:

```
subcat1_missing = train[train.subcat1_name=='missing'].shape[0]
print('subcategory1 name is missing for {} datapoints, i.e. {:.2f} % of train data.'
.format(subcat1_missing, 100.0*subcat1_missing/train.shape[0]))
```

```
subcategory1 name is missing for 6314 datapoints, i.e. 0.43 % of train data.
```

subcat2_name

In [28]:

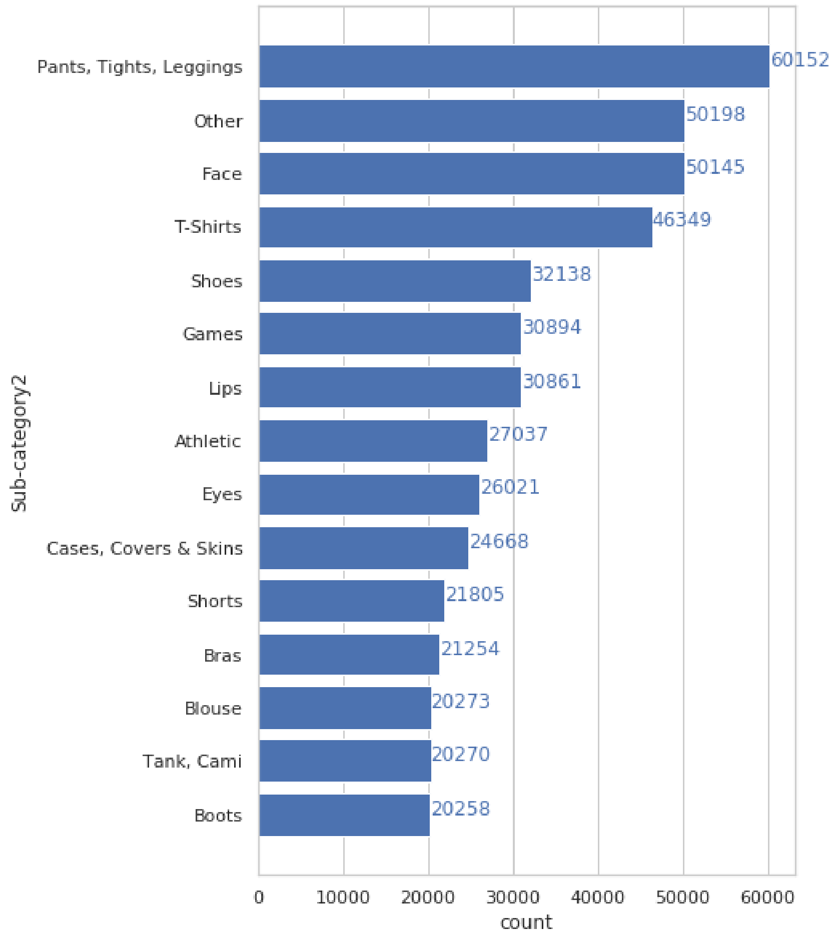
```
train.subcat2_name.describe()
```

Out[28]:

```
count                   1481661
unique                      871
top        Pants, Tights, Leggings
freq                      60152
Name: subcat2_name, dtype: object
```

In [29]:

```
subcat2_count = Counter(list(train.subcat2_name.values))
x, y = zip(*subcat2_count.most_common(15))
plt.figure(figsize=[6,10])
plt.barh(x, y)
for i, val in enumerate(y):
           plt.annotate(val, (y[i], x[i]), color='b')
plt.gca().invert_yaxis()
plt.ylabel('Sub-category2')
plt.xlabel('count')
plt.grid(False, axis='y')
plt.show()
```



In [30]:

```
subcat2_missing = train[train.subcat2_name=='missing'].shape[0]
print('subcategory2 name is missing for {} datapoints, i.e. {:.2f} % of train data.'
.format(subcat2_missing, 100.0*subcat2_missing/train.shape[0]))
```

```
subcategory2 name is missing for 6314 datapoints, i.e. 0.43 % of train data.
```

item_description

In [31]:

```
desc_missing = train[train.item_description=='missing'].shape[0]
print('item description is missing for {} datapoints, i.e. {:.5f} % of train data.'
.format(desc_missing, 100.0*desc_missing/train.shape[0]))
```

```
item description is missing for 4 datapoints, i.e. 0.00027 % of train data.
```

In [32]:

```
train[train.item_description=='missing']
```

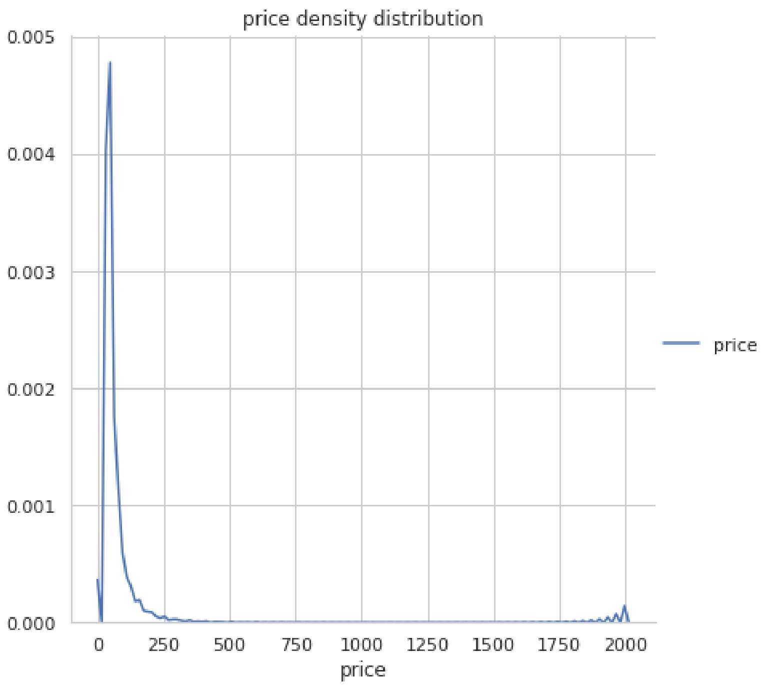
Out[32]:

	train_id	name	item_condition_id	brand_name	price	shipping	item_description	gencat_name	subcat1_name	subcat2_n
511264	511535	Shoes for Michelle	4	missing	9.0	0	missing	Kids	Girls 0-24 Mos	Sh
860756	861230	Lipgloss	4	missing	49.0	0	missing	Beauty	Makeup	
1224217	1224924	Disney Minnie Head band	3	Disney	9.0	0	missing	Women	Women's Accessories	Accesso
1263521	1264242	For Bianca	3	missing	10.0	1	missing	Women	Women's Accessories	Scarv W

price

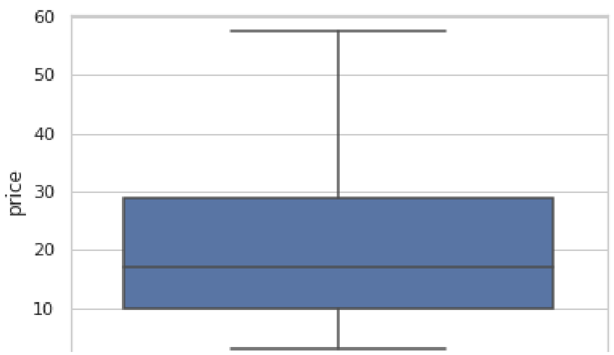
In [33]:

```
sns.FacetGrid(train,size=6) \
    .map(sns.kdeplot,"price") \
    .add_legend();
plt.title('price density distribution')
plt.show();
```



In [34]:

```
sns.boxplot(y='price', data=train, showfliers=False)
plt.show()
```



In [35]:

```
for i in range(0, 100, 10):
    var =train["price"].values
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(i,var[int(len(var)*(float(i)/100))]))
print("100 percentile value is ",var[-1])
```

```
0 percentile value is 3.0
10 percentile value is 7.0
20 percentile value is 10.0
30 percentile value is 12.0
40 percentile value is 14.0
50 percentile value is 17.0
60 percentile value is 20.0
70 percentile value is 26.0
80 percentile value is 34.0
90 percentile value is 51.0
100 percentile value is  2009.0
```

In [36]:

```
for i in range(90, 100, 1):
    var =train["price"].values
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(i,var[int(len(var)*(float(i)/100))]))
print("100 percentile value is ",var[-1])
```

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

97% of datapoints have price less than USD 100. Very few (only 1%) datapoints have price more than USD 170

2. Pre-processing

In [37]:

```
def preprocess_name(text_col):
    preprocessed_names = []
    for sentence in tqdm(text_col.values):
        sent = sentence.replace('\\r', ' ')
        sent = sent.replace('\\"', ' ')
        sent = sent.replace('\\n', ' ')
        sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
        preprocessed_names.append(sent.lower().strip())
    return preprocessed_names

stopwords = stopwords.words('english')
def preprocess_desc(text_col):
    preprocessed_descs = []
    for sentence in tqdm(text_col.values):
        sent = sentence.replace('\\r', ' ')
        sent = sent.replace('\\"', ' ')
        sent = sent.replace('\\n', ' ')
        sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
        sent = ' '.join(e for e in sent.split() if e not in stopwords)
        preprocessed_descs.append(sent.lower().strip())
    return preprocessed_descs
```

In [38]:

```
train['preprocessed_name'] = preprocess_name(train['name'])
test['preprocessed_name'] = preprocess_name(test['name'])

train['preprocessed_description'] = preprocess_desc(train['item_description'])
test['preprocessed_description'] = preprocess_desc(test['item_description'])
```

```
100%|██████████| 1481661/1481661 [00:11<00:00, 124510.07it/s]
100%|██████████| 3460725/3460725 [00:27<00:00, 124025.17it/s]
100%|██████████| 1481661/1481661 [02:33<00:00, 9672.48it/s]
100%|██████████| 3460725/3460725 [05:58<00:00, 9656.55it/s]
```

In [39]:

```
def clean_cat(cat_values):
    '''takes categorical column values as arguments and returns list of cleaned categories'''

    catogories = list(cat_values)

    cat_list = []
    for i in tqdm(catogories):
        i = re.sub('[^A-Za-z0-9]+', ' ', i)
        i = i.replace(' ','')
        i = i.replace('&','_')
        cat_list.append(i.strip())

    return cat_list
```

In [40]:

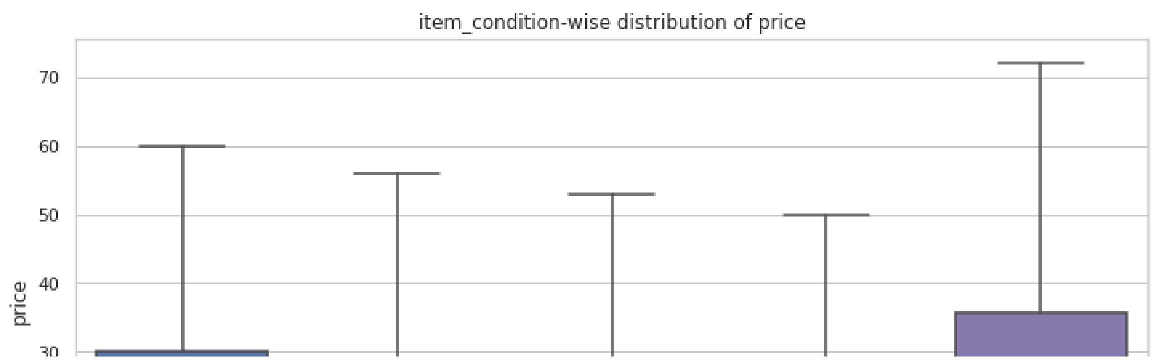
```
train['gencat_name'] = clean_cat(train['gencat_name'].values)
test['gencat_name'] = clean_cat(test['gencat_name'].values)

train['subcat1_name'] = clean_cat(train['subcat1_name'].values)
test['subcat1_name'] = clean_cat(test['subcat1_name'].values)

train['subcat2_name'] = clean_cat(train['subcat2_name'].values)
test['subcat2_name'] = clean_cat(test['subcat2_name'].values)
```

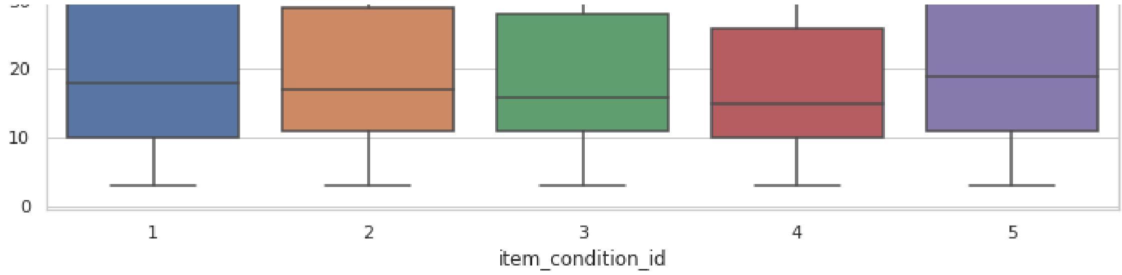
```
100%|██████████| 1481661/1481661 [00:05<00:00, 295231.76it/s]
100%|██████████| 3460725/3460725 [00:11<00:00, 289588.24it/s]
100%|██████████| 1481661/1481661 [00:06<00:00, 219928.05it/s]
100%|██████████| 3460725/3460725 [00:15<00:00, 219610.83it/s]
100%|██████████| 1481661/1481661 [00:06<00:00, 222446.74it/s]
100%|██████████| 3460725/3460725 [00:15<00:00, 221882.62it/s]
```

3. Basic Exploratory Data Analysis

In [38]:

```
sns.set(style='whitegrid')
plt.figure(figsize=(12,6))
sns.boxplot(x='item_condition_id', y='price', data=train, showfliers=False)
plt.title('item_condition-wise distribution of price')
plt.show()
```

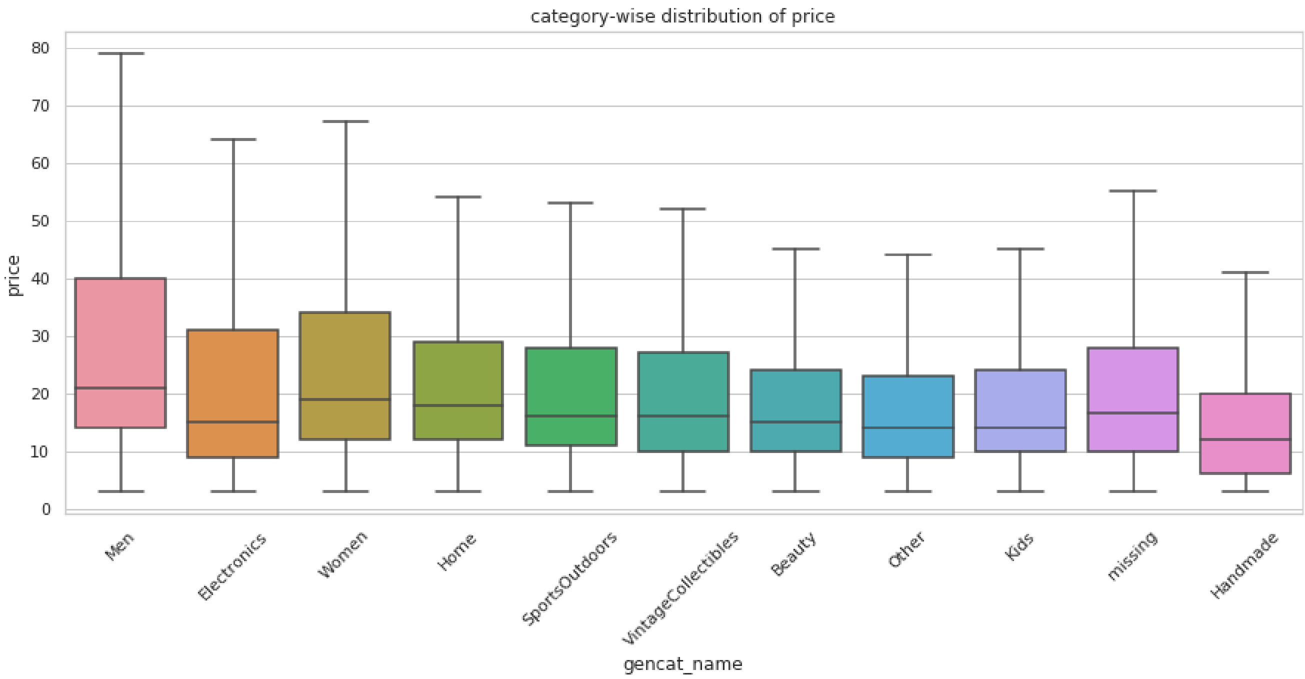




There is slight variation of price based on item condition. Median Price decreases as we go from condition 1 to 4. Items in condition 5 seem to be having higher price, which is a bit strange.

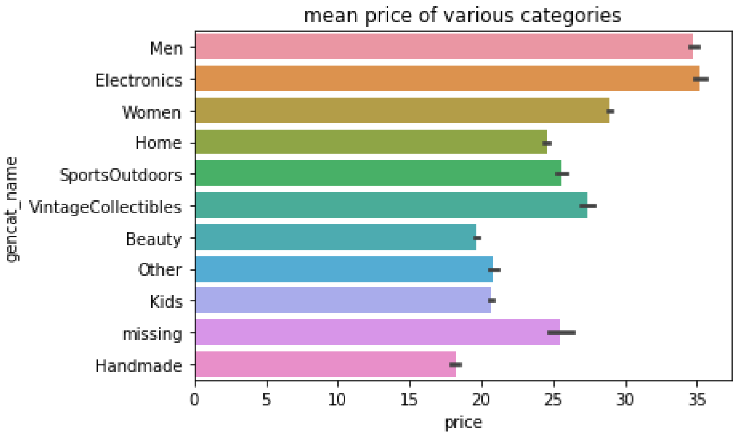
In [39]:

```
plt.figure(figsize=(15,6))
sns.boxplot(y='price', x='gencat_name', data=train, showfliers=False)
plt.xticks(rotation=45)
plt.title('category-wise distribution of price')
plt.show()
```



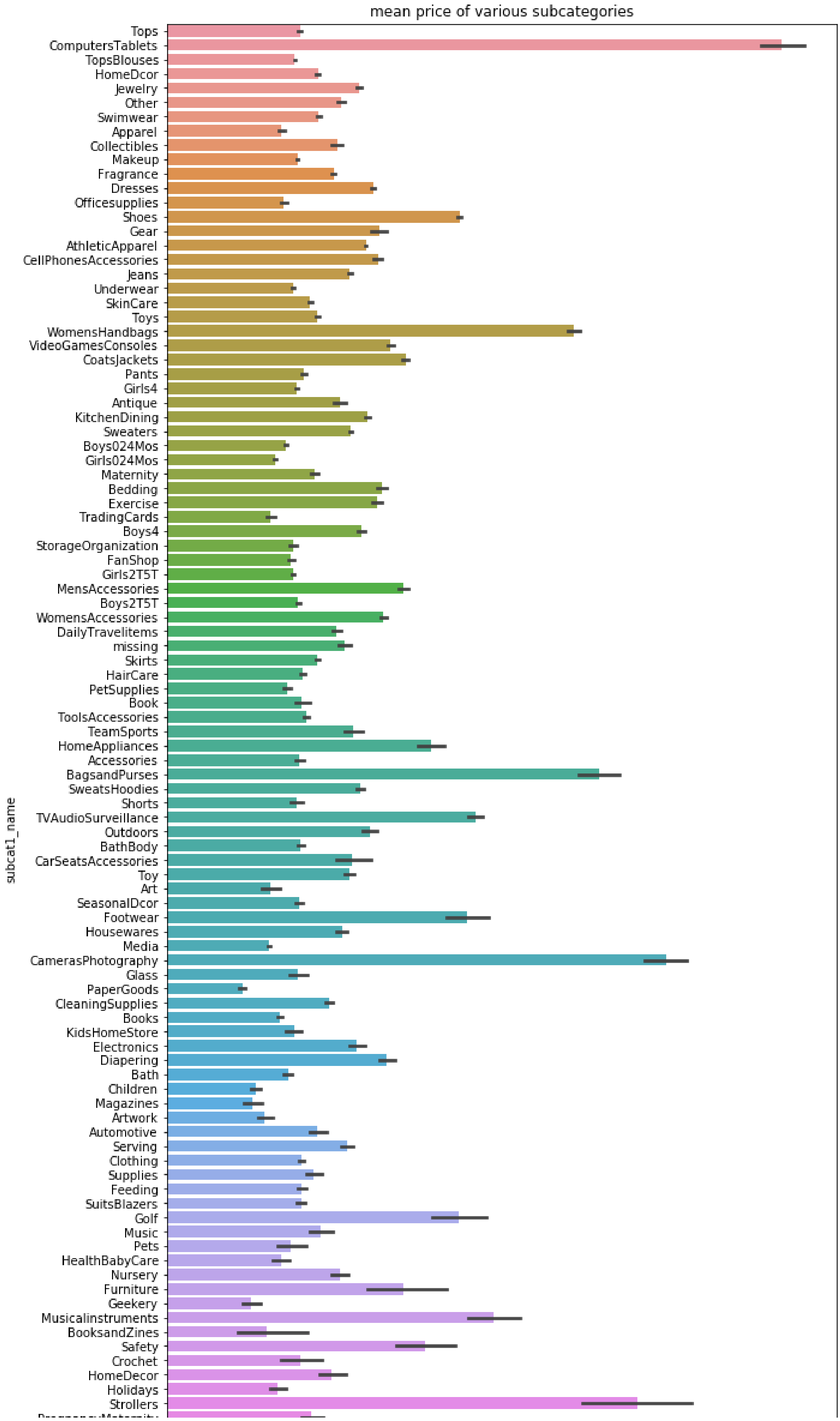
In [38]:

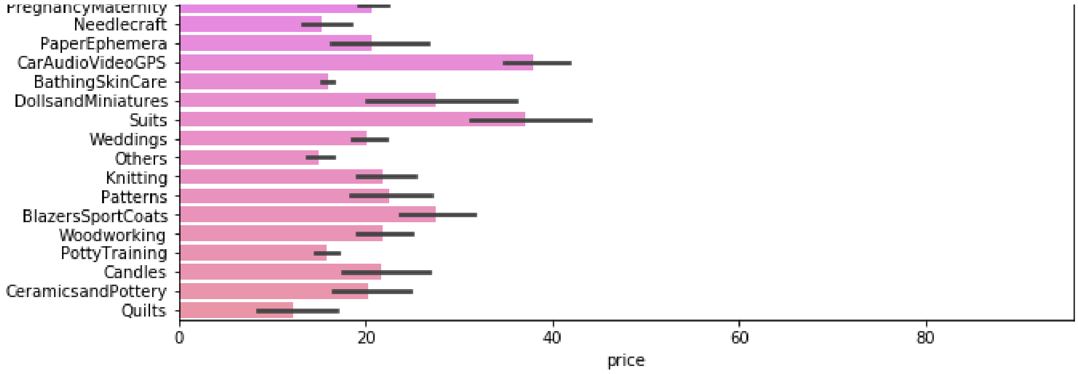
```
sns.barplot(y='gencat_name', x='price', data=train)
plt.title('mean price of various categories')
plt.show()
```



In [39]:

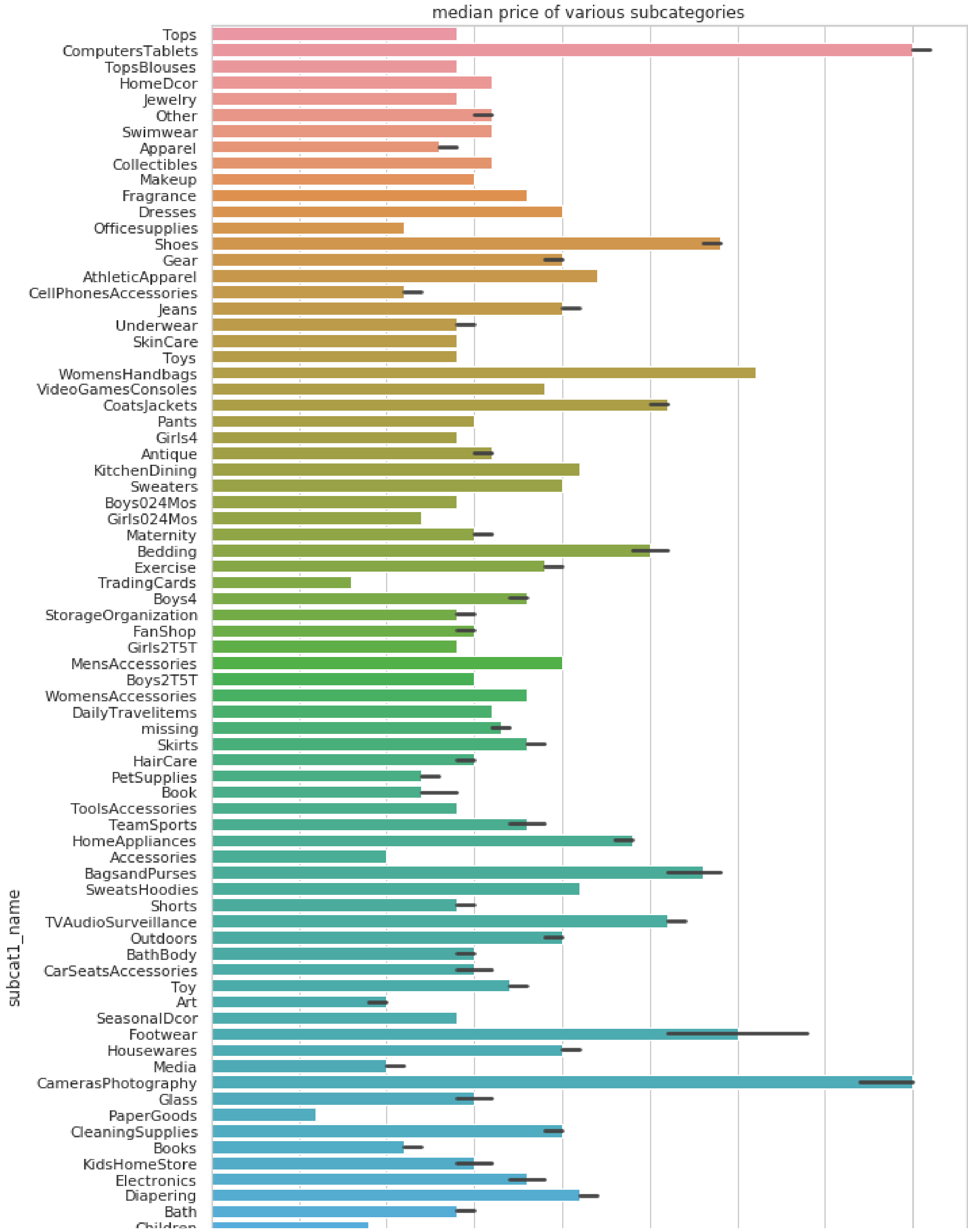
```
plt.figure(figsize=(10,25))
sns.barplot(y='subcat1_name', x='price', data=train)
plt.title('mean price of various subcategories')
plt.show()
```

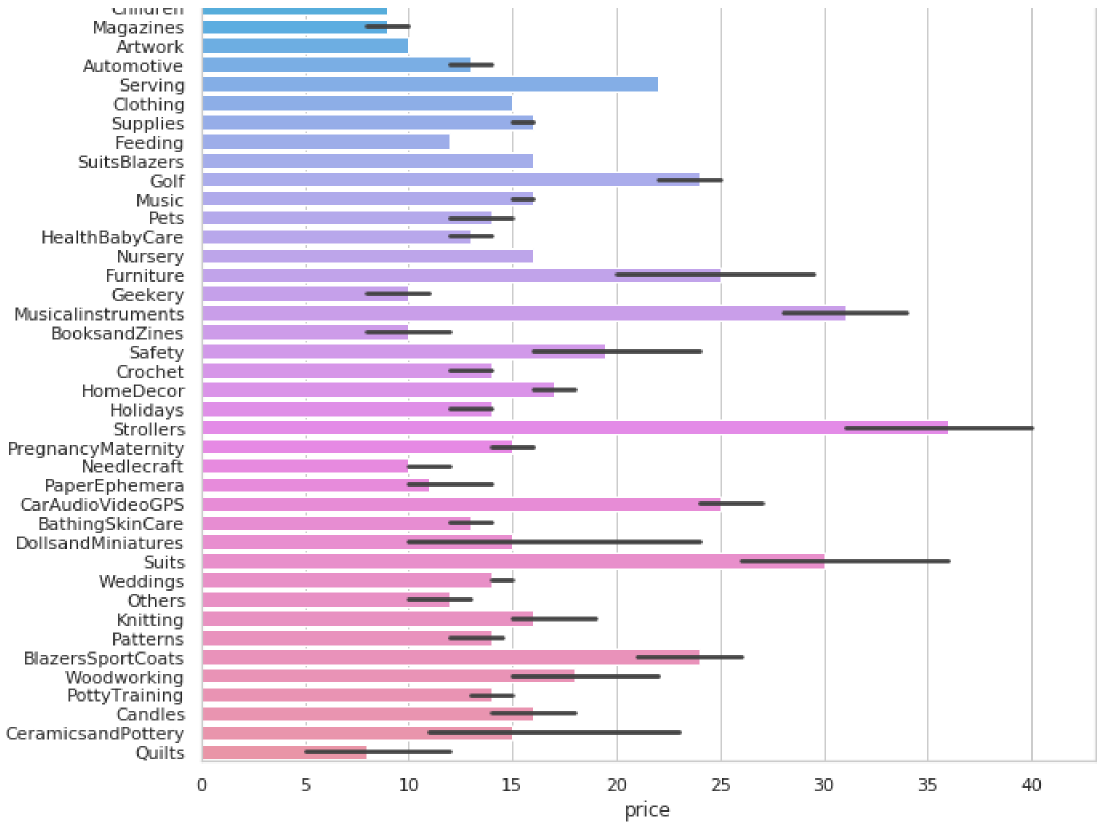




In [41]:

```
plt.figure(figsize=(10,25))
sns.barplot(y='subcat1_name', x='price', data=train, estimator=median)
plt.title('median price of various subcategories')
plt.show()
```





Prices of items belonging to various categories and subcategories vary significantly. This indicates that categories are going to be important features in determining the price of an item.

4. Featurization

4.1. Feature Engineering

name_first: cleaned name first word

name_first_count: count of name in data

gencat_name_count: count of gencat in data

subcat1_name_count: count of subcat1 in data

subcat2_name_count: count of subcat2 in data

brand_name_count: count of brand_name in data

In [41]:

```
def get_name_first(name):

    name =  re.sub('[^A-Za-z0-9]+', ' ', name) .split()
    if len(name):
            return name[0].lower()
    return ''


train['name_first'] = train['name'].apply(get_name_first)
test['name_first'] = test['name'].apply(get_name_first)
```

In [42]:

```
def transform_test(base_col, feat_col):
    '''
    Returns feat_col column of test data by mapping from the values already calculated for the sam
e column in train data
```

```
    Parameters:

    base_col: column based on which a transform(count, mean, median) has been applied

    feat_col: desired feature column after applying the transform
    '''
    #Create dictionary of feature values from train data
    di = pd.Series(train[feat_col].values, index=train[base_col].values).to_dict()

    #Map test data using dictionary and fill NAs with 0

    if base_col == 'item_condition_id':
        #No chance of NAs
        return test[base_col].map(di).astype(float)

    return test[base_col].map(di).fillna(0)
```

In [43]:

```
train['name_first_count'] = train.groupby('name_first')['name_first'].transform('count')
test['name_first_count'] = transform_test('name_first', 'name_first_count')

train['gencat_name_count'] = train.groupby('gencat_name')['gencat_name'].transform('count')
test['gencat_name_count'] = transform_test('gencat_name', 'gencat_name_count')

train['subcat1_name_count'] = train.groupby('subcat1_name')['subcat1_name'].transform('count')
test['subcat1_name_count'] = transform_test('subcat1_name', 'subcat1_name_count')

train['subcat2_name_count'] = train.groupby('subcat2_name')['subcat2_name'].transform('count')
test['subcat2_name_count'] = transform_test('subcat2_name', 'subcat2_name_count')

train['brand_name_count'] = train.groupby('brand_name')['brand_name'].transform('count')
test['brand_name_count'] = transform_test('brand_name', 'brand_name_count')
```

NameLower: # lowercase letters in name

DescriptionLower: # lowercase letters in description

NameUpper: # uppercase letters in name

DescriptionUpper: # uppercase letters in description

name_len: char length of name

des_len: char length of desc

name_desc_len_ratio: name_len / des_len

desc_word_count

mean_des: 10 * desc_word_count / des_len

name_word_count

mean_name: 10 * name_word_count / name_len

desc_letters_per_word: des_len / desc_word_count

name_letters_per_word: name_len / name_word_count

NameLowerRatio: NameLower / name_len

DescriptionLowerRatio: DescriptionLower / des_len

NameUpperRatio: NameUpper / name_len

DescriptionUpperRatio: DescriptionUpper / des_len

In [44]:

```
train['NameLower'] = train.name.str.count('[a-z]')
train['DescriptionLower'] = train.item_description.str.count('[a-z]')
train['NameUpper'] = train.name.str.count('[A-Z]')
train['DescriptionUpper'] = train.item_description.str.count('[A-Z]')
train['name_len'] = train['name'].apply(lambda x: len(x))
```

```
train['name_len'] = train['name'].apply(lambda x: len(x))
train['des_len'] = train['item_description'].apply(lambda x: len(x))
train['name_desc_len_ratio'] = train['name_len']/train['des_len']
train['desc_word_count'] = train['item_description'].apply(lambda x: len(x.split()))
train['mean_des'] = train['item_description'].apply(lambda x: 0 if len(x) == 0 else float(len(x.split())) / len(x)) * 10
train['name_word_count'] = train['name'].apply(lambda x: len(x.split()))
train['mean_name'] = train['name'].apply(lambda x: 0 if len(x) == 0 else float(len(x.split())) / len(x))  * 10
train['desc_letters_per_word'] = train['des_len'] / train['desc_word_count']
train['name_letters_per_word'] = train['name_len'] / train['name_word_count']
train['NameLowerRatio'] = train['NameLower'] / train['name_len']
train['DescriptionLowerRatio'] = train['DescriptionLower'] / train['des_len']
train['NameUpperRatio'] = train['NameUpper'] / train['name_len']
train['DescriptionUpperRatio'] = train['DescriptionUpper'] / train['des_len']
```

In [45]:

```
test['NameLower'] = test.name.str.count('[a-z]')
test['DescriptionLower'] = test.item_description.str.count('[a-z]')
test['NameUpper'] = test.name.str.count('[A-Z]')
test['DescriptionUpper'] = test.item_description.str.count('[A-Z]')
test['name_len'] = test['name'].apply(lambda x: len(x))
test['des_len'] = test['item_description'].apply(lambda x: len(x))
test['name_desc_len_ratio'] = test['name_len']/test['des_len']
test['desc_word_count'] = test['item_description'].apply(lambda x: len(x.split()))
test['mean_des'] = test['item_description'].apply(lambda x: 0 if len(x) == 0 else float(len(x.split())) / len(x)) * 10
test['name_word_count'] = test['name'].apply(lambda x: len(x.split()))
test['mean_name'] = test['name'].apply(lambda x: 0 if len(x) == 0 else float(len(x.split())) / len(x))  * 10
test['desc_letters_per_word'] = test['des_len'] / test['desc_word_count']
test['name_letters_per_word'] = test['name_len'] / test['name_word_count']
test['NameLowerRatio'] = test['NameLower'] / test['name_len']
test['DescriptionLowerRatio'] = test['DescriptionLower'] / test['des_len']
test['NameUpperRatio'] = test['NameUpper'] / test['name_len']
test['DescriptionUpperRatio'] = test['DescriptionUpper'] / test['des_len']
```

NamePunctCount: # punctuations in name

DescriptionPunctCount: # punctuations in desc

NamePunctCountRatio: NamePunctCount / name_word_count

DescriptionPunctCountRatio: DescriptionPunctCount / desc_word_count

NameDigitCount: # digits in name

DescriptionDigitCount: # digits in desc

NameDigitCountRatio: NameDigitCount / name_word_count

DescriptionDigitCountRatio: DescriptionDigitCount / desc_word_count

stopword_ratio_desc: # stopwords in desc / desc_word_count

num_sum: Sum of numbers in desc

weird_characters_desc: # non-alphanumeric, non-punct in desc

weird_characters_name: # non-alphanumeric, non-punct in desc

prices_count: # of [rm] (removed price) in desc

price_in_name: 1 if desc contains [rm]; 0 otherwise

In [46]:

```
from nltk.corpus import stopwords

RE_PUNCTUATION = '|'.join([re.escape(x) for x in string.punctuation])
s_words = {x: 1 for x in stopwords.words('english')} #converting to dictionary for fast look up
non_alphanumpunct = re.compile(u'[^A-Za-z0-9\.?!,; \(\)\[\]\'\"\$]+')
```

In [47]:

```
#https://www.kaggle.com/peterhurford/lgb-and-fm-18th-place-0-40604

def to_number(x):
    try:
        if not x.isdigit():
            return 0
        x = int(x)
        if x > 100:
            return 100
        else:
            return x
    except:
        return 0
```

In [48]:

```
train['NamePunctCount'] = train.name.str.count(RE_PUNCTUATION)
train['DescriptionPunctCount'] = train.item_description.str.count(RE_PUNCTUATION)
train['NamePunctCountRatio'] = train['NamePunctCount'] / train['name_word_count']
train['DescriptionPunctCountRatio'] = train['DescriptionPunctCount'] / train['desc_word_count']
train['NameDigitCount'] = train.name.str.count('[0-9]')
train['DescriptionDigitCount'] = train.item_description.str.count('[0-9]')
train['NameDigitCountRatio'] = train['NameDigitCount'] / train['name_word_count']
train['DescriptionDigitCountRatio'] = train['DescriptionDigitCount']/train['desc_word_count']
train['stopword_ratio_desc'] = train['item_description'].apply(lambda x: len([w for w in x.split()
if w in s_words])) / train['desc_word_count']
train['num_sum'] = train['item_description'].apply(lambda x: sum([to_number(s) for s in x.split()])
)
train['weird_characters_desc'] = train['item_description'].str.count(non_alphanumpunct)
train['weird_characters_name'] = train['name'].str.count(non_alphanumpunct)
train['prices_count'] = train['item_description'].str.count('[rm]')
train['price_in_name'] = train['item_description'].str.contains('[rm]', regex=False).astype('catego
ry')
```

In [49]:

```
test['NamePunctCount'] = test.name.str.count(RE_PUNCTUATION)
test['DescriptionPunctCount'] = test.item_description.str.count(RE_PUNCTUATION)
test['NamePunctCountRatio'] = test['NamePunctCount'] / test['name_word_count']
test['DescriptionPunctCountRatio'] = test['DescriptionPunctCount'] / test['desc_word_count']
test['NameDigitCount'] = test.name.str.count('[0-9]')
test['DescriptionDigitCount'] = test.item_description.str.count('[0-9]')
test['NameDigitCountRatio'] = test['NameDigitCount'] / test['name_word_count']
test['DescriptionDigitCountRatio'] = test['DescriptionDigitCount']/test['desc_word_count']
test['stopword_ratio_desc'] = test['item_description'].apply(lambda x: len([w for w in x.split() if
w in s_words])) / test['desc_word_count']
test['num_sum'] = test['item_description'].apply(lambda x: sum([to_number(s) for s in x.split()]))
test['weird_characters_desc'] = test['item_description'].str.count(non_alphanumpunct)
test['weird_characters_name'] = test['name'].str.count(non_alphanumpunct)
test['prices_count'] = test['item_description'].str.count('[rm]')
test['price_in_name'] = test['item_description'].str.contains('[rm]', regex=False).astype('category
')
```

brand_mean_price: mean price of items by a brand

name_mean_price: mean price of an item by name

gencat_mean_price: mean price of items belonging to gencat

subcat1_mean_price: mean price of items belonging to subcat1

subcat2_mean_price: mean price of items belonging to subcat2

condition_mean_price: mean price of items by condition

brand_median_price: median price of items by a brand

name_median_price: median price of an item by name

gencat_median_price: median price of items belonging to gencat

subcat1_median_price: median price of items belonging to subcat1

subcat2_median_price: median price of items belonging to subcat2

condition_median_price: median price of items by condition

In [50]:

```
train['brand_mean_price'] = train.groupby('brand_name')['price'].transform('mean')
test['brand_mean_price'] = transform_test('brand_name', 'brand_mean_price')

train['name_mean_price'] = train.groupby('name_first')['price'].transform('mean')
test['name_mean_price'] = transform_test('name_first', 'name_mean_price')

train['gencat_mean_price'] = train.groupby('gencat_name')['price'].transform('mean')
test['gencat_mean_price'] = transform_test('gencat_name', 'gencat_mean_price')

train['subcat1_mean_price'] = train.groupby('subcat1_name')['price'].transform('mean')
test['subcat1_mean_price'] = transform_test('subcat1_name', 'subcat1_mean_price')

train['subcat2_mean_price'] = train.groupby('subcat2_name')['price'].transform('mean')
test['subcat2_mean_price'] = transform_test('subcat2_name', 'subcat2_mean_price')

train['condition_mean_price'] = train.groupby('item_condition_id')['price'].transform('mean')
test['condition_mean_price'] = transform_test('item_condition_id', 'condition_mean_price')
```

In [51]:

```
train['brand_median_price'] = train.groupby('brand_name')['price'].transform('median')
test['brand_median_price'] = transform_test('brand_name', 'brand_median_price')

train['name_median_price'] = train.groupby('name_first')['price'].transform('median')
test['name_median_price'] = transform_test('name_first', 'name_median_price')

train['gencat_median_price'] = train.groupby('gencat_name')['price'].transform('median')
test['gencat_median_price'] = transform_test('gencat_name', 'gencat_median_price')

train['subcat1_median_price'] = train.groupby('subcat1_name')['price'].transform('median')
test['subcat1_median_price'] = transform_test('subcat1_name', 'subcat1_median_price')

train['subcat2_median_price'] = train.groupby('subcat2_name')['price'].transform('median')
test['subcat2_median_price'] = transform_test('subcat2_name', 'subcat2_median_price')

train['condition_median_price'] = train.groupby('item_condition_id')['price'].transform('median')
test['condition_median_price'] = transform_test('item_condition_id', 'condition_median_price')
```

In [52]:

```
train.drop(['name', 'item_description'], axis=1, inplace=True)
test.drop(['name', 'item_description'], axis=1, inplace=True)
```

In [53]:

```
print(train.shape, test.shape)
```

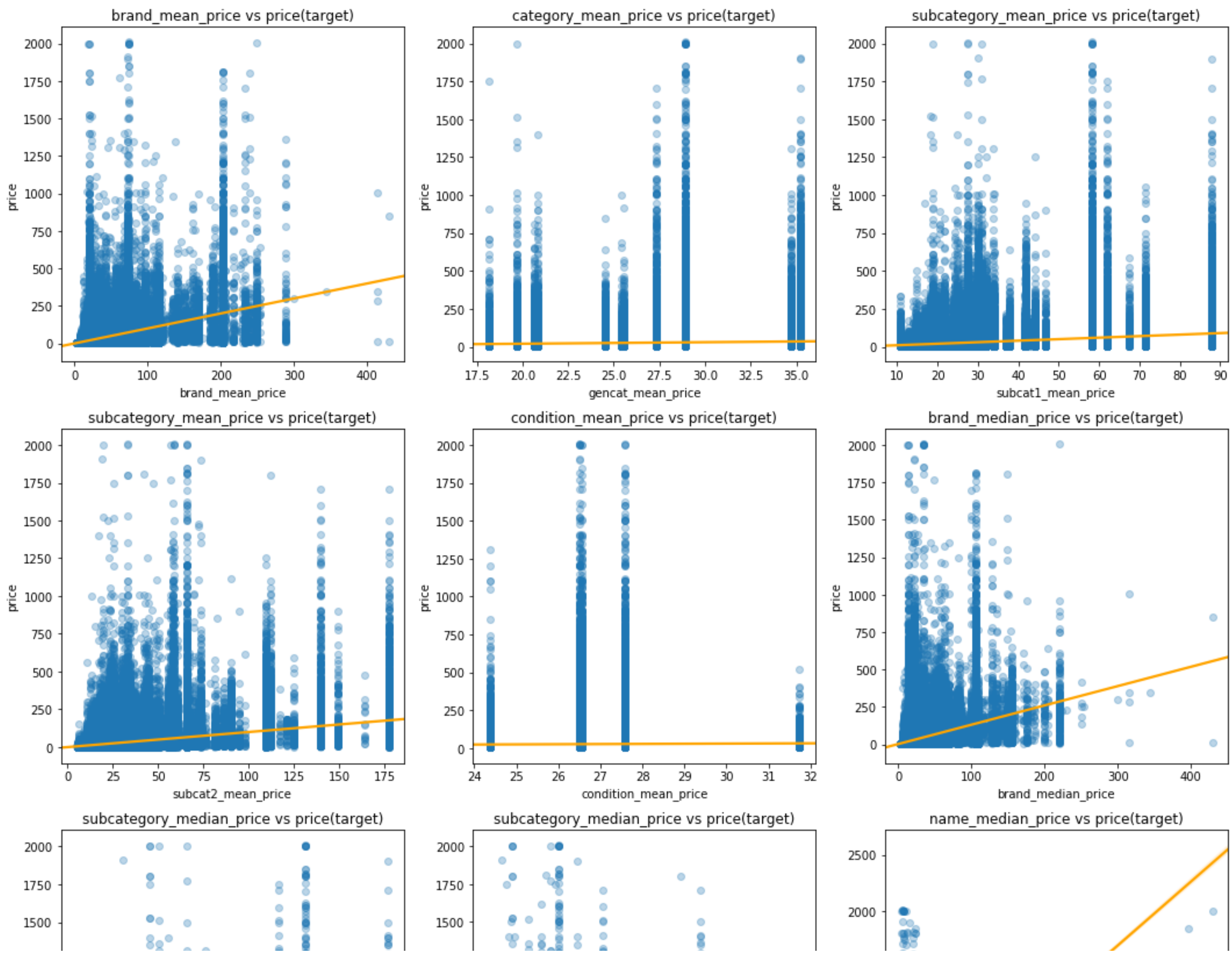
```
(1481661, 59) (3460725, 58)
```

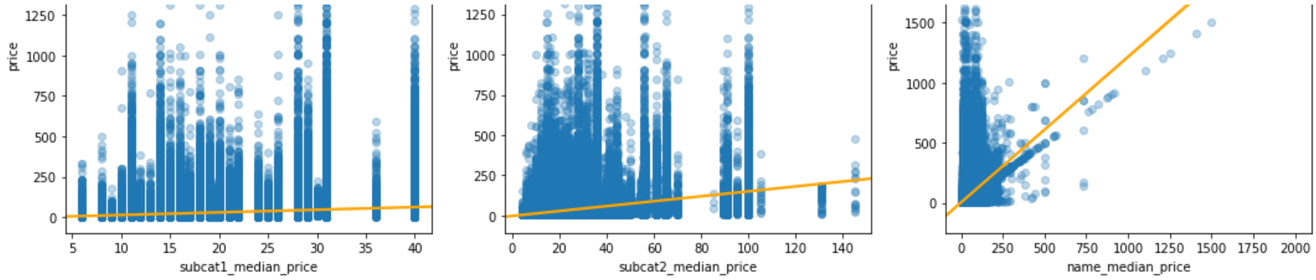
Univariate analysis on the above features

In [61]:

```
plt.figure(figsize=(18,18))

plt.subplot(3,3,1)
sns.regplot(x='brand_mean_price', y='price', data=train, scatter_kws={'alpha':0.3}, line_kws={'colo
r':'orange'})
plt.title('brand_mean_price vs price(target)')

plt.subplot(3,3,2)
sns.regplot(x='gencat_mean_price', y='price', data=train, scatter_kws={'alpha':0.3}, line_kws={'col
or':'orange'})
```

```
plt.title('category_mean_price vs price(target)')

plt.subplot(3,3,3)
sns.regplot(x='subcat1_mean_price', y='price', data=train, scatter_kws={'alpha':0.3}, line_kws={'co
lor':'orange'})
plt.title('subcategory_mean_price vs price(target)')

plt.subplot(3,3,4)
sns.regplot(x='subcat2_mean_price', y='price', data=train, scatter_kws={'alpha':0.3}, line_kws={'co
lor':'orange'})
plt.title('subcategory_mean_price vs price(target)')

plt.subplot(3,3,5)
sns.regplot(x='condition_mean_price', y='price', data=train, scatter_kws={'alpha':0.3}, line_kws={'
color':'orange'})
plt.title('condition_mean_price vs price(target)')

plt.subplot(3,3,6)
sns.regplot(x='brand_median_price', y='price', data=train, scatter_kws={'alpha':0.3}, line_kws={'co
lor':'orange'})
plt.title('brand_median_price vs price(target)')

plt.subplot(3,3,7)
sns.regplot(x='subcat1_median_price', y='price', data=train, scatter_kws={'alpha':0.3}, line_kws={'
color':'orange'})
plt.title('subcategory_median_price vs price(target)')

plt.subplot(3,3,8)
sns.regplot(x='subcat2_median_price', y='price', data=train, scatter_kws={'alpha':0.3}, line_kws={'
color':'orange'})
plt.title('subcategory_median_price vs price(target)')

plt.subplot(3,3,9)
sns.regplot(x='name_median_price', y='price', data=train, scatter_kws={'alpha':0.3}, line_kws={'col
or':'orange'})
plt.title('name_median_price vs price(target)')

plt.show()
```





Features such as **brand_mean_price, brand_median price, subcat2_mean_price, subcat2_median_price** show a linear trend therefore seem to be useful in determinig price of items.

Dropping rows with blank name and description

In [54]:

```
n_rows = train.shape[0]
train = train[train.preprocessed_name != ''].reset_index(drop=True)

print('Dropped {} rows'.format(n_rows - train.shape[0]))
```

```
Dropped 19 rows
```

In [55]:

```
n_rows = train.shape[0]
train = train[train.preprocessed_description != ''].reset_index(drop=True)

print('Dropped {} rows'.format(n_rows - train.shape[0]))

print('Shape of train data: ', train.shape)
```

```
Dropped 567 rows
Shape of train data:  (1481075, 59)
```

4.2. Train, Test split for cross validation

In [56]:

```
from sklearn.model_selection import train_test_split

y_tr = np.log1p(train['price'])
train.drop(['price'], axis=1, inplace=True)

train_df, cv_df , y_train, y_cv = train_test_split(train, y_tr, test_size=0.1, random_state=42)

print('Train size: {}, CV size: {}, Test size: {}' .format(train_df.shape, cv_df.shape, test.shape)
)
```

```
Train size: (1332967, 58), CV size: (148108, 58), Test size: (3460725, 58)
```

In [57]:

```
del train, y_tr
gc.collect()
```

Out[57]:

```
120
```

4.3. Categorical features

One-hot encoding of brand_name, gencat_name, subcat1_name, subcat2_name.

In [58]:

```
#Cleaning brand name before using count vectorizer
# Using same preprocessing as used earlier for categories: 'clean_cat()' function

train_df['brand_name'] = clean_cat(train_df['brand_name'].values)
cv_df['brand_name'] = clean_cat(cv_df['brand_name'].values)
test['brand_name'] = clean_cat(test['brand_name'].values)
```

```
100%|██████████| 1332967/1332967 [00:05<00:00, 255964.09it/s]
100%|██████████| 148108/148108 [00:00<00:00, 251876.77it/s]
100%|██████████| 3460725/3460725 [00:13<00:00, 256942.02it/s]
```

In [59]:

```
vectorizer = CountVectorizer(lowercase=False, binary=True)
train_brand_oneHot = vectorizer.fit_transform(train_df['brand_name'].values)

cv_brand_oneHot = vectorizer.transform(cv_df['brand_name'].values)
test_brand_oneHot = vectorizer.transform(test['brand_name'].values)

print("Shape of matrices after one hot encoding")
print(train_brand_oneHot.shape, "\n", cv_brand_oneHot.shape, "\n", test_brand_oneHot.shape)
```

```
Shape of matrices after one hot encoding
(1332967, 4651)
 (148108, 4651)
 (3460725, 4651)
```

In [60]:

```
vectorizer = CountVectorizer(lowercase=False, binary=True)
train_gencat_oneHot = vectorizer.fit_transform(train_df['gencat_name'].values)

cv_gencat_oneHot = vectorizer.transform(cv_df['gencat_name'].values)
test_gencat_oneHot = vectorizer.transform(test['gencat_name'].values)

print("Shape of matrices after one hot encoding")
print(train_gencat_oneHot.shape, "\n", cv_gencat_oneHot.shape, "\n", test_gencat_oneHot.shape)
```

```
Shape of matrices after one hot encoding
(1332967, 11)
 (148108, 11)
 (3460725, 11)
```

In [61]:

```
vectorizer = CountVectorizer(lowercase=False, binary=True)
train_subcat1_oneHot = vectorizer.fit_transform(train_df['subcat1_name'].values)

cv_subcat1_oneHot = vectorizer.transform(cv_df['subcat1_name'].values)
test_subcat1_oneHot = vectorizer.transform(test['subcat1_name'].values)

print("Shape of matrices after one hot encoding")
print(train_subcat1_oneHot.shape, "\n", cv_subcat1_oneHot.shape, "\n", test_subcat1_oneHot.shape)
```

```
Shape of matrices after one hot encoding
(1332967, 114)
 (148108, 114)
 (3460725, 114)
```

In [62]:

```
vectorizer = CountVectorizer(lowercase=False, binary=True)
train_subcat2_oneHot = vectorizer.fit_transform(train_df['subcat2_name'].values)

cv_subcat2_oneHot = vectorizer.transform(cv_df['subcat2_name'].values)
test_subcat2_oneHot = vectorizer.transform(test['subcat2_name'].values)

print("Shape of matrices after one hot encoding")
```

```
print(train_subcat2_oneHot.shape, "\n", cv_subcat2_oneHot.shape, "\n", test_subcat2_oneHot.shape)
```

```
Shape of matrices after one hot encoding
(1332967, 863)
 (148108, 863)
 (3460725, 863)
```

4.4. Tfidf vectorization on text features

1-3 grams of name
1-3 grams of item_description

In [63]:

```
vectorizer = TfidfVectorizer(ngram_range=(1, 3), min_df=3, max_features=250000)

train_name_tfidf = vectorizer.fit_transform(train_df['preprocessed_name'].values)

cv_name_tfidf = vectorizer.transform(cv_df['preprocessed_name'].values)
test_name_tfidf = vectorizer.transform(test['preprocessed_name'].values)

print("Shape of matrices after vectorization")
print(train_name_tfidf.shape, "\n", cv_name_tfidf.shape, "\n", test_name_tfidf.shape)
```

```
Shape of matrices after vectorization
(1332967, 250000)
 (148108, 250000)
 (3460725, 250000)
```

In [64]:

```
vectorizer = TfidfVectorizer(ngram_range=(1, 3), min_df=5, max_features=500000)

train_description_tfidf = vectorizer.fit_transform(train_df['preprocessed_description'].values)

cv_description_tfidf = vectorizer.transform(cv_df['preprocessed_description'].values)
test_description_tfidf = vectorizer.transform(test['preprocessed_description'].values)

print("Shape of matrices after vectorization")
print(train_description_tfidf.shape, "\n", cv_description_tfidf.shape, "\n", test_description_tfidf
.shape)
```

```
Shape of matrices after vectorization
(1332967, 500000)
 (148108, 500000)
 (3460725, 500000)
```

5. Data preparation

In [65]:

```
submission_df = pd.DataFrame(test['test_id'])
print(submission_df.shape)
submission_df.head()
```

```
(3460725, 1)
```

Out[65]:

	test_id
0	0
1	1
2	2
3	3

4	test_id	4

5.1. Normalize numerical features

In [66]:

```
cols = set(train_df.columns.values) - {'train_id'}
skip_cols = {'preprocessed_name', 'item_condition_id', 'brand_name',
  'shipping', 'preprocessed_description', 'gencat_name',
  'subcat1_name', 'subcat2_name', 'name_first', 'price_in_name'}

cols_to_normalize = cols - skip_cols
print("Normalizing following columns: ", cols_to_normalize)

def normalize(df):
    result1 = df.copy()
    for feature_name in df.columns:
        if (feature_name in cols_to_normalize):
            max_value = df[feature_name].max()
            min_value = df[feature_name].min()
            result1[feature_name] = (df[feature_name] - min_value) / (max_value - min_value)
    return result1
```

```
Normalizing following columns:  {'NameLower', 'name_len', 'name_word_count',
'weird_characters_name', 'NamePunctCount', 'name_median_price', 'name_first_count', 'des_len',
'condition_mean_price', 'subcat1_median_price', 'desc_word_count', 'prices_count', 'num_sum',
'NameLowerRatio', 'condition_median_price', 'desc_letters_per_word', 'DescriptionDigitCount', 'Des
criptionUpperRatio', 'name_letters_per_word', 'subcat2_median_price', 'subcat2_name_count',
'DescriptionLowerRatio', 'NameDigitCountRatio', 'weird_characters_desc', 'brand_name_count',
'gencat_median_price', 'DescriptionLower', 'DescriptionPunctCount', 'NameUpper',
'brand_median_price', 'brand_mean_price', 'gencat_name_count', 'subcat1_name_count',
'name_desc_len_ratio', 'mean_name', 'subcat1_mean_price', 'stopword_ratio_desc',
'DescriptionPunctCountRatio', 'NamePunctCountRatio', 'DescriptionDigitCountRatio', 'mean_des', 'De
scriptionUpper', 'name_mean_price', 'gencat_mean_price', 'NameDigitCount', 'NameUpperRatio',
'subcat2_mean_price'}
```

In [67]:

```
train_normalized = normalize(train_df)
cv_normalized = normalize(cv_df)
test_normalized = normalize(test)
```

In [68]:

```
del train_df, cv_df, test
gc.collect()
```

Out[68]:

```
20
```

5.2. Remove non-features from dataframes

In [69]:

```
#Separating and storing all numerical features

X_tr = train_normalized[list(cols_to_normalize)]
X_val = cv_normalized[list(cols_to_normalize)]
X_te = test_normalized[list(cols_to_normalize)]

X_tr.head(2)
```

Out[69]:

	NameLower	name_len	name_word_count	weird_characters_name	NamePunctCount	name_median_price	name_first_count	c
660536	0.131579	0.166667	0.0625	0.0	0.0	0.008513	0.144537	0.

637076	NameLower 0.789474	name_len 0.928571	name_word_count 0.3125	weird_characters_name 0.0	NamePunctCount 0.0	name_median_price 0.004006	name_first_count 0.000656	0

2 rows × 47 columns

In [70]:

```
from scipy.sparse import csr_matrix

# Storing categorical features to sparse matrix

X_tr_cat = csr_matrix(pd.get_dummies(train_normalized[['item_condition_id', 'shipping', 'price_in_n
ame']], sparse=True).values)

X_cv_cat = csr_matrix(pd.get_dummies(cv_normalized[['item_condition_id', 'shipping', 'price_in_name
']], sparse=True).values)

X_te_cat = csr_matrix(pd.get_dummies(test_normalized[['item_condition_id', 'shipping',
'price_in_name']], sparse=True).values)

print(X_tr_cat.shape, X_cv_cat.shape, X_te_cat.shape)
```

```
(1332967, 9) (148108, 9) (3460725, 9)
```

In [71]:

```
del train_normalized, cv_normalized, test_normalized
gc.collect()
```

Out[71]:

```
20
```

5.3. Consolidate all features to a sparse matrix

In [72]:

```
from scipy.sparse import hstack

# stack all categorical and text sparse matrices

train_sparse = hstack((train_brand_oneHot, train_gencat_oneHot, train_subcat1_oneHot,
train_subcat2_oneHot, \
               train_name_tfidf, train_description_tfidf, X_tr_cat)).tocsr()

cv_sparse = hstack((cv_brand_oneHot, cv_gencat_oneHot, cv_subcat1_oneHot, cv_subcat2_oneHot, \
               cv_name_tfidf, cv_description_tfidf, X_cv_cat)).tocsr()

test_sparse = hstack((test_brand_oneHot, test_gencat_oneHot, test_subcat1_oneHot,
test_subcat2_oneHot, \
               test_name_tfidf, test_description_tfidf, X_te_cat)).tocsr()
```

In [73]:

```
print(train_sparse.shape, cv_sparse.shape, test_sparse.shape)
```

```
(1332967, 755648) (148108, 755648) (3460725, 755648)
```

In [74]:

```
# stack dense feature matrix with categorical and text vectors

X_train = hstack((X_tr.values, train_sparse)).tocsr()

X_cv = hstack((X_val.values, cv_sparse)).tocsr()

X_test = hstack((X_te.values, test_sparse)).tocsr()
```

In [75]:

In [75]:

```
print('Train size: {}, CV size: {}, Test size: {}' .format(X_train.shape, X_cv.shape, X_test.shape)
)
```

```
Train size: (1332967, 755695), CV size: (148108, 755695), Test size: (3460725, 755695)
```

In [76]:

```
del vectorizer
del train_brand_oneHot, train_gencat_oneHot, train_subcat1_oneHot, train_subcat2_oneHot, \
            train_name_tfidf, train_description_tfidf, X_tr_cat

del cv_brand_oneHot, cv_gencat_oneHot, cv_subcat1_oneHot, cv_subcat2_oneHot, cv_name_tfidf,
cv_description_tfidf, X_cv_cat

del test_brand_oneHot, test_gencat_oneHot, test_subcat1_oneHot, test_subcat2_oneHot, \
               test_name_tfidf, test_description_tfidf, X_te_cat
gc.collect()
```

Out[76]:

```
100
```

6. Modeling

In [77]:

```
from sklearn.metrics import mean_squared_error as mse
from math import sqrt
from sklearn.linear_model import Ridge
```

Target variable is ln(1+y), therefore calculating mean square error on ln(1+y) will effectively give us MSLE on y

- MSLE(y^, y) = MSE(ln(1+y^), ln(1+y))

6.1. Ridge Model

Linear least squares with l2 regularization sklearn.linear_model.Ridge

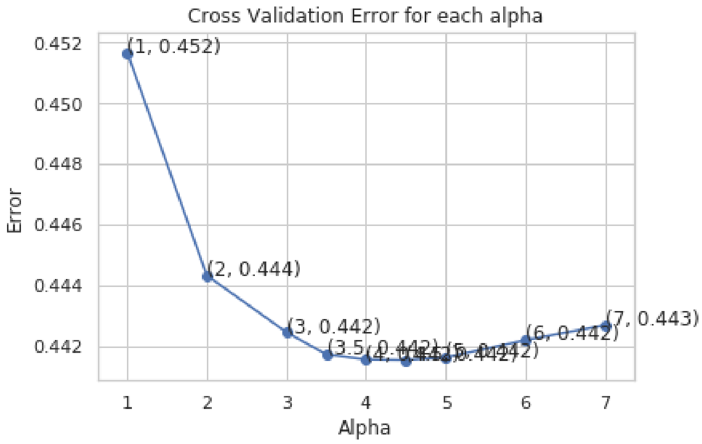
Hyper-parameter tuning

In [79]:

```
alpha = [1, 2, 3, 3.5, 4, 4.5, 5, 6, 7]
cv_rmsle_array=[]
for i in tqdm(alpha):
    model = Ridge(solver="sag", random_state=42, alpha=i)
    model.fit(X_train, y_train)
    preds_cv = model.predict(X_cv)
    cv_rmsle_array.append(sqrt(mse(y_cv, preds_cv)))

for i in range(len(cv_rmsle_array)):
    print ('RMSLE for alpha = ',alpha[i],'is',cv_rmsle_array[i])

best_alpha = np.argmin(cv_rmsle_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_rmsle_array)
ax.scatter(alpha, cv_rmsle_array)
for i, txt in enumerate(np.round(cv_rmsle_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_rmsle_array[i]))

plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha")
plt.ylabel("Error")
plt.show()
```

```
100%|██████████| 9/9 [15:04<00:00, 100.50s/it]
```

```
RMSLE for alpha =  1 is 0.45166601980352833
RMSLE for alpha =  2 is 0.44431546233122515
RMSLE for alpha =  3 is 0.4424425182737627
RMSLE for alpha =  3.5 is 0.44171501703551286
RMSLE for alpha =  4 is 0.44154616529145424
RMSLE for alpha =  4.5 is 0.4415286167861061
RMSLE for alpha =  5 is 0.44161632764828285
RMSLE for alpha =  6 is 0.4421832813533032
RMSLE for alpha =  7 is 0.44267468278758176
```



Training using best hyper-parameters and testing

In [81]:

```python
print("Best alpha: ",  alpha[best_alpha])
model = Ridge(solver="sag", random_state=42, alpha=alpha[best_alpha])
model.fit(X_train, y_train)
ridge_preds_tr = model.predict(X_train)
ridge_preds_cv = model.predict(X_cv)
ridge_preds_te = model.predict(X_test)

print('Train RMSLE:', sqrt(mse(y_train, ridge_preds_tr)))

ridge_rmsle = sqrt(mse(y_cv, ridge_preds_cv))
print("Cross validation RMSLE: ", ridge_rmsle)
```

```
Best alpha:  4.5
Train RMSLE: 0.3834498984020693
Cross validation RMSLE:  0.4415286167861061
```

6.2. Multinomial Naive Bayes: Regression using Classification

sklearn.naive_bayes.MultinomialNB

This model has been used as a correction for the tendency of the Ridge model to underestimate. The predicted values are saved for use as features in a later model. The authors of 18th place solution kernel improved their RMSLE by 0.003 using this.

In [82]:

```python
from sklearn.naive_bayes import MultinomialNB

model = MultinomialNB(alpha=0.01)
model.fit(X_train, y_train>= 4)

mnb_preds_tr = model.predict_proba(X_train)[:, 1]
mnb_preds_cv = model.predict_proba(X_cv)[:, 1]
mnb_preds_te = model.predict_proba(X_test)[:, 1]
```

SelectKBest: Selecting top 48k features from categorical and text features

In [83]:

```
# from sklearn.feature_selection.univariate_selection import SelectKBest, f_regression

fselect = SelectKBest(f_regression, k=48000)
train_features = fselect.fit_transform(train_sparse, y_train)

cv_features = fselect.transform(cv_sparse)
test_features = fselect.transform(test_sparse)
```

In [84]:

```
print('Shapes after SelectKBest:', train_features.shape, cv_features.shape, test_features.shape)
```

```
Shapes after SelectKBest: (1332967, 48000) (148108, 48000) (3460725, 48000)
```

In [85]:

```
# stack feature matrix with Ridge, MNB model predictions, engineered features
X_train = hstack((X_tr.values, ridge_preds_tr.reshape(-1,1), mnb_preds_tr.reshape(-1,1), train_feat
ures)).tocsr()

X_cv = hstack((X_val.values, ridge_preds_cv.reshape(-1,1), mnb_preds_cv.reshape(-1,1), cv_features)
).tocsr()

X_test = hstack((X_te.values, ridge_preds_te.reshape(-1,1), mnb_preds_te.reshape(-1,1), test_featur
es)).tocsr()

print('Train size: {}, CV size: {}, Test size: {}' .format(X_train.shape, X_cv.shape, X_test.shape)
)
```

```
Train size: (1332967, 48049), CV size: (148108, 48049), Test size: (3460725, 48049)
```

In [86]:

```
del train_features, cv_features
gc.collect()
```

Out[86]:

```
2685
```

In [87]:

```
print('Time taken: ', time.time()-start_time)
snapshot2 = tracemalloc.take_snapshot()
top_stats = snapshot2.compare_to(snapshot1, 'lineno')

print("[ Top 10 ]")
for stat in top_stats[:10]:
    print(stat)
```

```
Time taken:  5293.345500946045
[ Top 10 ]
<__array_function__ internals>:6: size=6529 MiB (+6529 MiB), count=520 (+520), average=12.6 MiB
/opt/conda/lib/python3.7/site-packages/pandas/core/algorithms.py:1657: size=1772 MiB (+1772 MiB),
count=13 (+13), average=136 MiB
/opt/conda/lib/python3.7/site-packages/scipy/sparse/compressed.py:755: size=668 MiB (+668 MiB), co
unt=5 (+5), average=134 MiB
<ipython-input-4-1155efdb4829>:4: size=485 MiB (+485 MiB), count=7401015 (+7401015), average=69 B
/opt/conda/lib/python3.7/site-packages/pandas/io/parsers.py:2037: size=400 MiB (+400 MiB),
count=2803386 (+2803386), average=150 B
/opt/conda/lib/python3.7/site-packages/scipy/sparse/compressed.py:754: size=334 MiB (+334 MiB), co
unt=4 (+4), average=83.5 MiB
/opt/conda/lib/python3.7/site-packages/pandas/core/internals/managers.py:1848: size=94.3 MiB (+94.
3 MiB), count=11 (+11), average=8776 KiB
/opt/conda/lib/python3.7/site-packages/sklearn/naive_bayes.py:117: size=75.4 MiB (+75.4 MiB), coun
t=7 (+7), average=10.8 MiB
```

```
/opt/conda/lib/python3.7/site-packages/scipy/sparse/compressed.py:469: size=38.8 MiB (+38.8 MiB), count=10 (+10), average=3977 KiB
/opt/conda/lib/python3.7/site-packages/pandas/core/algorithms.py:484: size=38.3 MiB (+38.3 MiB), count=160292 (+160292), average=250 B
```

In [88]:

```
submission_df['price'] = np.exp(ridge_preds_te) - 1

submission_df.to_csv('ridge_submission.csv', index=False)
```

In [89]:

```
scipy.sparse.save_npz("cv_final.npz", X_cv)
np.save('y_cv', y_cv)

del X_cv, y_cv
gc.collect()
```

Out[89]:

```
40
```

In [90]:

```
scipy.sparse.save_npz("train_final.npz", X_train)
np.save('y_train', y_train)

del X_train, y_train
gc.collect()
```

Out[90]:

```
20
```

In [91]:

```
scipy.sparse.save_npz("test_final.npz", X_test)

del X_test
gc.collect()
```

Out[91]:

```
20
```