Nulogy_DataScience_Pricing_Assignment

August 22, 2019

0.1 Instructions

- Please complete the following assignment using (preferrably) Python. I am using Python 3.7 v.
- If you feel like you have the time and skill set to complete all 2 exercises, feel free to do so.
- Please push the assignment onto a free, publicly available repository for review (e.g. GitHub or Gitlab).
- You are free to use any open source library or package to complete the exercises.

0.2 Assumptions

- Since there is no one 'price' column in the formulation of the problem + dataset, I am defining the 'price' target variable as being the mean of amountMin and amountMax prices, both converted to USD. ~0.03% of rows in the dataset have amountMin != amountMax, therefore it was a reasonable assumption to make. i.e. we are assuming that the bias introduced by the currency conversion of the price is negligeable. Furthermore, I left the currency column in the model as an categorical variable to see if it has much feature importance on the target variable, price.

- Assuming that each row represents a product-price_range per set of listing URLs: more explicitly each row represents a range of prices, captured by prices.amountMin and prices.amountMax, which were each seen on the prices.sourceURLs of that row (there may be multiple sources on which those prices were seen).

1 Introduction

Given the time constraint of this assignment, the focus of this exercise was on using a simple but reliable model to generate a reasonably good performance (using MSE, RMSE, R2 as my evaluation metric(s)) and comparing it to a more complex supervised learning model. The accuracy of the model saw the greatest improvement by developing the "Data Cleaning", "Feature Engineering" and "Model Implementation" sections.

This notebook outlines the following components of my analysis:

```
1) Read Data
2) Data Cleaning
3) Feature Engineering
4) EDA
```

5) Benchmarking
6) Model Implementation (LightGBM)
7) Future Improvements

In [1]:
```python
# import libraries required for analysis

import os
import sys

import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

import math
import datetime as dt
from dateutil.relativedelta import relativedelta

import tldextract as tldextract
# from currency_converter import CurrencyConverter

import category_encoders as ce
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix
from imblearn.over_sampling import SMOTE
from bayes_opt import BayesianOptimization
from sklearn import metrics
from sklearn.model_selection import KFold
from sklearn.decomposition import PCA

from sklearn.linear_model import LinearRegression

import lightgbm as lgb
```

```
/anaconda3/lib/python3.7/site-packages/lightgbm/__init__.py:46: UserWarning: Starting from vers
This means that in case of installing LightGBM from PyPI via the ``pip install lightgbm`` comma
Instead of that, you need to install the OpenMP library, which is required for running LightGBI
You can install the OpenMP library by the following command: ``brew install libomp``.
  "You can install the OpenMP library by the following command: ``brew install libomp``.", Use
```

In [2]:
```python
# initialize necessary directories and params
os.chdir('/Users/estevanfalcone/Nulogy/Nulogy_DataScience_Pricing_Assignment/')
```

2 1. Read Data

- Investigate problematic rows at import (hypothesizing that it's JSON column foratting interfering with column splitting).

In [3]:
```python
# limits # of rows to read (None to read entire dataset)
nRowsRead = None

data = pd.read_csv('product_data_schema.csv', nrows = nRowsRead, low_memory = False,
                   error_bad_lines = False)
print('Dataset dimensions:', data.shape)
data.head()
```

```
Dataset dimensions: (19387, 48)
```

Out[3]:
```
                     id asins                brand  \
0  AVpfHrJ6ilAPnD_xVXOI   NaN                Josmo
1  AVpfHrJ6ilAPnD_xVXOI   NaN                Josmo
2  AVpfHsWP1cnluZ0-eVZ7   NaN  SERVUS BY HONEYWELL
3  AVpfHsWP1cnluZ0-eVZ7   NaN  SERVUS BY HONEYWELL
4  AVpfHsWP1cnluZ0-eVZ7   NaN  SERVUS BY HONEYWELL

                                   categories colors  count  \
0  Clothing,Shoes,Men's Shoes,All Men's Shoes    NaN    NaN
1  Clothing,Shoes,Men's Shoes,All Men's Shoes    NaN    NaN
2  All Men's Shoes,Shoes,Men's Shoes,Clothing    NaN    NaN
3  All Men's Shoes,Shoes,Men's Shoes,Clothing    NaN    NaN
4  All Men's Shoes,Shoes,Men's Shoes,Clothing    NaN    NaN

              dateAdded           dateUpdated  \
0  2016-11-07T00:45:12Z  2016-11-07T00:45:12Z
1  2016-11-07T00:45:12Z  2016-11-07T00:45:12Z
2  2016-06-14T04:29:57Z  2016-07-09T20:26:48Z
3  2016-06-14T04:29:57Z  2016-07-09T20:26:48Z
4  2016-06-14T04:29:57Z  2016-07-09T20:26:48Z

                                        descriptions dimension  ...    \
0  [{"dateSeen":["2016-11-07T00:45:12Z"],"sourceU...       NaN  ...
1  [{"dateSeen":["2016-11-07T00:45:12Z"],"sourceU...       NaN  ...
2  [{"dateSeen":["2016-07-09T20:26:48Z"],"sourceU...       NaN  ...
3  [{"dateSeen":["2016-07-09T20:26:48Z"],"sourceU...       NaN  ...
4  [{"dateSeen":["2016-07-09T20:26:48Z"],"sourceU...       NaN  ...

  prices.warranty quantities  reviews sizes  skus  \
0             NaN        NaN      NaN   NaN   NaN
1             NaN        NaN      NaN   NaN   NaN
2             NaN        NaN      NaN   NaN   NaN
3             NaN        NaN      NaN   NaN   NaN
```

```
4             NaN        NaN      NaN    NaN    NaN

                                                sourceURLs           upc vin  \
0  https://www.walmart.com/ip/Josmo-8190-Plain-In...  699302044036 NaN
1  https://www.walmart.com/ip/Josmo-8190-Plain-In...  699302044036 NaN
2  http://www.walmart.com/ip/Studs-Shoe-Large-Pr-...           NaN NaN
3  http://www.walmart.com/ip/Studs-Shoe-Large-Pr-...           NaN NaN
4  http://www.walmart.com/ip/Studs-Shoe-Large-Pr-...           NaN NaN

  websiteIDs weight
0        NaN    NaN
1        NaN    NaN
2        NaN    NaN
3        NaN    NaN
4        NaN    NaN

[5 rows x 48 columns]
```

3 2. Data Cleaning

3.0.1 Data Type and Missing Data Cleaning

In [4]:
```
data.dtypes
```

Out[4]:
```
id                         object
asins                      object
brand                      object
categories                 object
colors                     object
count                     float64
dateAdded                  object
dateUpdated                object
descriptions               object
dimension                  object
ean                        object
features                   object
flavors                   float64
imageURLs                  object
isbn                      float64
keys                       object
manufacturer               object
manufacturerNumber         object
merchants                  object
name                       object
prices.amountMin          float64
prices.amountMax          float64
prices.availability        object
prices.color               object
```

```
prices.condition        object
prices.count            object
prices.currency         object
prices.dateAdded        object
prices.dateSeen         object
prices.flavor           object
prices.isSale           object
prices.merchant         object
prices.offer            object
prices.returnPolicy     object
prices.shipping         object
prices.size             object
prices.source          float64
prices.sourceURLs       object
prices.warranty        float64
quantities             float64
reviews                 object
sizes                   object
skus                    object
sourceURLs              object
upc                     object
vin                    float64
websiteIDs             float64
weight                  object
dtype: object
```

First, start by verifying if there are any duplicated rows in the data and remove them:

```
In [5]: f'duplicate rows: {data.duplicated().sum()}'
        data.drop_duplicates(inplace=True)
        print(data.shape)
```

```
(19383, 48)
```

Remove non-ASCII char entries from dataset:

```
In [6]: objects_columns = data.select_dtypes(object)
        objects_columns = objects_columns.drop(['prices.availability','prices.count',
                                        'prices.flavor','prices.isSale'],axis=1)
        data[objects_columns.columns] = objects_columns.apply(lambda x: x.str.encode('ascii',
                                                              .str.decode('ascii'))
```

Find columns with no data (i.e. number of missing values == number of rows of dataset) and remove them:

```
In [7]: # find columns with entirely missing data
        null_sums_per_col = data.isna().sum()
        null_columns = list(null_sums_per_col[null_sums_per_col == data.shape[0]].index)
        print(null_columns)
```

```
['count', 'flavors', 'isbn', 'prices.source', 'prices.warranty', 'quantities', 'vin', 'website
```

```
In [8]: # remove these null columns from the dataset
        data1 = data.drop(null_columns, 'columns')
        print(data1.shape)
```

```
(19383, 40)
```

Determine how much data (quantity and proportion) is missing per column:

```
In [9]: # print quantity/ratio of missing values
        missing_data_metrics = {'missing_quantity': data1.isnull().sum(),
            'missing_ratio': data1.isnull().sum() * 100 / len(data1)}

        missing_data_metrics_df = pd.DataFrame(data = missing_data_metrics)
        print(missing_data_metrics_df.sort_values(by = ['missing_ratio'], ascending = False))
```

```
                     missing_quantity  missing_ratio
prices.count                    19371      99.938090
prices.flavor                   19356      99.860703
prices.availability             19314      99.644018
prices.size                     18730      96.631068
prices.color                    18610      96.011969
weight                          18526      95.578600
prices.returnPolicy             18358      94.711861
reviews                         17709      91.363566
asins                           16815      86.751277
dimension                       16369      84.450291
prices.shipping                 13682      70.587628
sizes                           13391      69.086313
prices.offer                    13329      68.766445
manufacturer                    12684      65.438787
skus                            10637      54.877986
descriptions                     9493      48.975907
ean                              9381      48.398081
upc                              8605      44.394573
colors                           8366      43.161533
prices.condition                 6735      34.746943
prices.merchant                  5542      28.592065
features                         5415      27.936852
merchants                        5346      27.580870
manufacturerNumber               4213      21.735541
imageURLs                        1072       5.530620
brand                             258       1.331063
prices.amountMax                   52       0.268276
prices.dateSeen                    52       0.268276
prices.isSale                      52       0.268276
```

```
prices.sourceURLs                  46      0.237321
sourceURLs                         32      0.165093
keys                               27      0.139297
dateUpdated                        27      0.139297
prices.amountMin                   25      0.128979
prices.dateAdded                   25      0.128979
prices.currency                    11      0.056751
categories                          0      0.000000
dateAdded                           0      0.000000
name                                0      0.000000
id                                  0      0.000000
```

```
In [10]: f'There are {len(missing_data_metrics_df[missing_data_metrics_df.missing_ratio >= 45]
```

```
Out[10]: 'There are 17 of 40 features missing 45% or more of their data.'
```

Many of these columns (including some of the ones with less missing data) will likely end up being dropped from the analysis in the interest of time. *See 'Future Improvements' for further analysis that could be done to some of said columns given additional time.*

```
In [11]: data1['colors'].value_counts()
         data1['colors'].isna().sum()
```

```
Out[11]: 8366
```

3.0.2 Gender

Filter out any Female Gender products erroneously in the data (given we want only Men's Shoe Prices). - By assumption, we will keep the Unisex products.

```
In [12]: data1['gender'] = data1.features.str.split(pat='"key":"Gender","value":\["',expand=Tr
         data1['gender'] = data1['gender'].str.replace("'","")
         data1['gender'] = data1['gender'].str.replace('"','')
         data1['gender'] = data1['gender'].str.strip()
         data1['gender'].value_counts()
```

```
Out[12]: Men                    7520
         Unisex                 1144
         Boys                    144
         Male                    142
         Women                   105
         Mens                    102
         Girls                    71
         Women , Men              15
         Male,Men                 11
         Men,Unisex               11
         Unisex Adults            10
         Men,Women                 9
```

```
Men,Mens                  8
Men,Boys                  7
Adult Unisex              7
Unisex, Mens, Womens      6
Women,Men                 6
Men / Women               5
Female                    4
Mens, Womens, Unisex      4
Boys,Men                  3
Men,Adult Unisex          3
Male,Mens                 3
Men,mens                  2
Unisex Adult              2
Womens                    2
Women,Unisex              1
Mens Womens               1
Does not apply            1
Name: gender, dtype: int64
```

```
In [13]: # remove rows that are not Mens, Boys or Unisex
         data1 = data1.loc[(data1.gender!='Womens') & (data1.gender!='Girls') & (data1.gender!=
         print(data1.shape)
```

```
(19201, 41)
```

```
In [14]: # map to fewer encodings based on description
         gender_map = {
             'Male':'Men',
             'Mens':'Men',
             'Women , Men':'Unisex',
             'Men,Unisex':'Unisex',
             'Male,Men':'Men',
             'Unisex Adults':'Unisex',
             'Men,Women':'Unisex',
             'Men,Mens':'Men',
             'Men,Boys':'Men',
             'Adult Unisex':'Unisex',
             'Unisex, Mens, Womens':'Unisex',
             'Women,Men':'Unisex',
             'Men / Women':'Unisex',
             'Mens, Womens, Unisex':'Unisex',
             'Men,Adult Unisex':'Unisex',
             'Male,Mens':'Men',
             'Boys,Men':'Men',
             'Men,mens':'Men',
             'Unisex Adult':'Unisex',
             'Mens Womens':'Unisex',
```

```
    'Women,Unisex':'Unisex'
}
data1['gender'] = data1['gender'].replace(gender_map)

# upon inspection, remove since *not* men's shoes
index = data1['gender'] == 'Does not apply'
data1 = data1[-index]
```

3.0.3 Categories

Sanity-check that the data contains 'Shoes' in Categories (and not other Category products).

```
In [15]: data1['categories'] = data1['categories'].str.lower()
         index = data1['categories'].str.contains('shoe')
         index.value_counts()

Out[15]: True     19173
         False       27
         Name: categories, dtype: int64

In [16]: data1 = data1[index == True]
         print(data1.shape)

(19173, 41)
```

3.0.4 isSale Flag

Upon close inspection, this flag does accurately show whether a particular line price has been discounted or not. We will simply encode the binary feature. Since the percentage of missing values for prices.isSale col is small, we will impute the missing values with the categorical mode (notably, isSale = False).

```
In [17]: # fill missing values with mode (False) and convert to int
         data1['prices.isSale'] = data1['prices.isSale'].fillna(False)
         data1['isSale'] = data1['prices.isSale'].astype(int)
         data1['isSale'].value_counts()

Out[17]: 0    13564
         1     5609
         Name: isSale, dtype: int64

In [18]: # drop the redundant isSale column
         data1 = data1.drop('prices.isSale','columns')
         data1.columns

Out[18]: Index(['id', 'asins', 'brand', 'categories', 'colors', 'dateAdded',
                'dateUpdated', 'descriptions', 'dimension', 'ean', 'features',
                'imageURLs', 'keys', 'manufacturer', 'manufacturerNumber', 'merchants',
                'name', 'prices.amountMin', 'prices.amountMax', 'prices.availability',
```

```
 'prices.color', 'prices.condition', 'prices.count', 'prices.currency',
 'prices.dateAdded', 'prices.dateSeen', 'prices.flavor',
 'prices.merchant', 'prices.offer', 'prices.returnPolicy',
 'prices.shipping', 'prices.size', 'prices.sourceURLs', 'reviews',
 'sizes', 'skus', 'sourceURLs', 'upc', 'weight', 'gender', 'isSale'],
dtype='object')
```

3.0.5 Date Features

- Convert any date-time columns to appropriate date-time type.
- Extract Month and Year from Date variables of interest; in particular, prices.dateSeen. Remove rows with missing dates - considered using an average date between min and max, but formatting errors exist in other rows for these data points (i.e. safer to remove them).

```
In [19]: # remove NA prices.dateSeen rows and rename 'dateSeen' col with it
         index = data1['prices.dateSeen'].isna()
         data1 = data1[-index]
         data1 = data1.drop(['dateAdded','dateUpdated','prices.dateAdded'],'columns')
         data1['dateSeen'] = data1['prices.dateSeen']
         data1 = data1.drop(['prices.dateSeen'],'columns')

         data1.columns
```

```
Out[19]: Index(['id', 'asins', 'brand', 'categories', 'colors', 'descriptions',
               'dimension', 'ean', 'features', 'imageURLs', 'keys', 'manufacturer',
               'manufacturerNumber', 'merchants', 'name', 'prices.amountMin',
               'prices.amountMax', 'prices.availability', 'prices.color',
               'prices.condition', 'prices.count', 'prices.currency', 'prices.flavor',
               'prices.merchant', 'prices.offer', 'prices.returnPolicy',
               'prices.shipping', 'prices.size', 'prices.sourceURLs', 'reviews',
               'sizes', 'skus', 'sourceURLs', 'upc', 'weight', 'gender', 'isSale',
               'dateSeen'],
              dtype='object')
```

```
In [20]: # convert dateSeen to date-time format
         data1 = data1.assign(
             dateSeen = pd.to_datetime(pd.to_datetime(data1.dateSeen).dt.date)
         )
         print(data1.shape)
```

```
(19148, 38)
```

3.0.6 Currency (cleaning)

Filter out erroneous currencies (it's imperative to have correct currencies for prices - desired target variable - to make sense).

```
In [21]: data1['prices.currency'].value_counts()
```

```
Out[21]: USD    18379
         AUD      337
         CAD      303
         EUR      107
         GBP       22
         Name: prices.currency, dtype: int64
```

```
In [22]: curr = list(['USD','AUD','CAD','GBP','EUR'])
         data1 = data1[data1['prices.currency'].isin(curr)]
         print(data1.shape)
         data1['prices.currency'].value_counts()
```

```
(19148, 38)
```

```
Out[22]: USD    18379
         AUD      337
         CAD      303
         EUR      107
         GBP       22
         Name: prices.currency, dtype: int64
```

3.0.7 Brands

- This could potentially be made more intelligent using techniques like fuzzy string matching, but in the interest of time, I made a map for the cases I could detect visually.

```
In [23]: data1['brand'] = data1['brand'].str.strip().str.lower()
         data1['brand'].value_counts().sort_values(ascending=False)
```

```
Out[23]: nike                              1763
         ralph lauren                       700
         puma                               651
         vans                               386
         new balance                        364
         reebok                             271
         adidas                             255
         jordan                             193
         superior glove works               187
         fuse lenses                        174
         fossa apparel                      174
         skechers                           164
         converse                           144
         dickies                            144
         unique bargains                    142
         unbranded                          140
         berne apparel                      126
         asics                              122
         crocs                              120
```

```
kinco                            110
toms                             108
national safety apparel inc      107
under armour                     105
gameday boots                    103
carhartt                          99
stacy adams                       98
scully                            96
georgia boot                      94
carrera                           93
polo ralph lauren                 92
                                ...
concepts sports                    1
ridge footwear                     1
greatlookz fashion                 1
hulkamania                         1
tailian                            1
jbw                                1
under armour outerwear             1
deroyal                            1
wood n stream                      1
silver lilly                       1
gitzo                              1
onitsuka tiger by asics            1
e2 sport balance                   1
emu                                1
miko lotti                         1
jed north                          1
msa                                1
soda                               1
calcutta                           1
carrera sole                       1
adidas crazy1 kobe bb shoes        1
elastogel                          1
navali                             1
eros                               1
wilsons leather                    1
akadema                            1
us toy                             1
basics                             1
shock doctor                       1
earrings-midwestjewellery          1
Name: brand, Length: 1805, dtype: int64
```

In [24]:
```python
# brand names to change (note: must be certain of brand mapping fixes)
brand_map = {
    '3drose llc':'3drose',
    '3n2':'3n2 sports',
```

```
'3m (formerly aearo)':'3m',
'academie gear':'academie',
'adidas outdoors':'adidas',
'adidas performance':'adidas',
'adidas crazy1 kobe bb shoes':'adidas',
'adidas crazy 8 ny knicks':'adidas',
'aearo':'3m',
'alexander mcqueen by puma':'alexander mcqueen',
'alexanders costumes':'alexanders',
'american eagle outfitters':'american eagle',
'and 1':'and1',
'augusta sportswear':'augusta',
'baffin inc':'baffin',
'berne apparel':'berne',
'blackhawk!':'blackhawk',
'boss hugo boss':'hugo boss',
'california costume':'california costumes',
'calvin klein jeans':'calvin klein',
'champ':'champion',
'cufflink aficionado':'cufflinks',
'cufflink inc':'cufflinks',
'dan post boots':'dan post',
'dc shoes':'dc',
'diamondback fitness':'diamondback',
'diesel black gold':'diesel',
'dije california':'dije',
'dolce e gabbana':'dolce gabbana',
'dolce & gabbana':'dolce gabbana',
'dr. martens air wair':'dr. martens',
'dr. martens work':'dr. martens',
'ellie shoes':'ellie',
'fitflop.':'fitflop',
'forever collectibles':'forever collectible',
'fox outdoor products':'fox outdoor',
'franco american novelties':'franco',
'ganesha handicrafts':'ganesha handicraft',
'ganeshahandicraft':'ganesha handicraft',
"Ganesha Handicraft \",\"Clothing, Shoes & Accessories,Men's Accessories,Backpacks
'gearonic tm':'gearonic',
'generic':'unbranded',
'generic / unbranded':'unbranded',
'generic surplus':'surplus',
'genuine dickies':'dickies',
'georgia boot':'georgia',
'gifts infinity':'gifts',
'gola classics':'gola',
'handmadecart':'handmadecraft',
'hao_bo':'hao-bo',
```

```
'holloway sportswear':'holloway',
"hot chilly's":'hot chillys',
'hugo by hugo boss':'hugo boss',
'incharacter costumes':'incharacter',
'j`s awake':"j's awake",
'jewelrywe':'jewelryweb',
"joe's":'joes',
'justin boots':'justin',
'justin original work boots':'justin',
'justin original workboots':'justin',
'k- swiss':'k-swiss',
'kenneth cole new yor':'kenneth cole',
'kenneth cole new york':'kenneth cole',
'kenneth cole ny':'kenneth cole',
'key apparel':'key',
'lacrosse footwear':'lacrosse',
'lauren ralph lauren':'ralph lauren',
'levi strauss & co.':'levis',
'levi strauss':'levis',
"levi's":'levis',
"levy's":'levis',
'magid glove and safety (industrial)':'magid glove and safety',
'majestic glove':'majestic',
'mancini leather goods':'mancini leather',
'marc new york by andrew marc':'marc new york',
'marc ny':'marc new york',
'maui jim 142-10':'maui jim',
'mg:dakota':'mg',
'mg:gn':'mg',
'michael michael kors':'michael kors',
'micro flex':'microflex',
'milkoloti':'miko lotti',
'milwaukee electric tool':'milwaukee',
'minnetonka men':'minnetonka',
'minnetonka moccasin company, inc.':"minnetonka",
'minx nyamp44 inc':'minx ny',
'mln':'mlb',
'muck boot team j':'muck boot',
'muck boots':'muck boot',
'muckboot':'muck boot',
'national safety apparel inc':'national safety apparel',
'new balance numeric':'new balance',
'newbalance':'new balance',
'nike - kobe':'nike',
'nike air jordan i':'nike air jordan',
'nike jordan future low':'nike air jordan',
'nike lunarglide 7':'nike',
'nike sb':'nike',
```

```
'norcross safety':'norcross',
'norcross safety prod':'norcross',
'norcross safety products':'norcross',
'north safety / honeywell':'honeywell',
"o'neill":"o'neal",
'onguard industries':'onguard',
'original s.w.a.t.':'original swat',
'otto:dakota':'otto',
'otto:gn':'otto',
'palmbeach jewelry':'palm beach jewelry',
'pearl izumi - run':'pearl izumi',
'perry ellis portfolio':'perry ellis',
'pf-flyers':'pf flyers',
'polaroid sunglasses':'polaroid',
'polo ralph lauren golf':'polo ralph lauren',
'polo sport ralph lauren':'polo ralph lauren',
'prada sport':'prada',
'principle plastics inc':'principle plastics',
'propet':'proper',
'ralph lauren black label':'ralph lauren',
'ralph lauren polo':'ralph lauren',
'ralph lauren purple label':'ralph lauren',
'ralph lauren rlx':'ralph lauren',
'ralph lauren rrl':'ralph lauren',
'ralph lauren yacht':'ralph lauren',
'ranger by honeywell':'ranger',
'ray-ban':'rayban',
'ray ban':'rayban',
'reaction kenneth cole':'kenneth cole reaction',
'red wing heritage':'red wing',
'red wing shoes':'red wing',
'reebok nfl equipment':'reebok',
'reef footwear':'reef',
'ridge footwear':'ridge',
'ridge outdoors':'ridge',
'river city clocks':'river city',
'river city garments':'river city',
'rockport works':'rockport',
'rockport xcs':'rockport',
'salomon, salomon':'salomon',
'serengeti eyewear':'serengeti',
'signature by levi strauss & co.':'levis',
'signature by levi strauss &amp; co':'levis',
'slipperooz by deer stags':'slipperooz',
"smiffy's":'smiffys',
'solo, solo':'solo',
'sondooongmart':'sondoongmart',
'sperry top sider':'sperry',
```

```
    'sperry top-sider':'sperry',
    'team beans, l.l.c':'team beans',
    'the original muck boot co.':'the original muck boot company',
    'the original swat footwear co':'the original swat footwear',
    'timberland boot company':'timberland',
    'tingley rubber corp.':'tingley rubber',
    'toms footwear':'toms',
    'tony lama boot co.':'tony lama',
    'totes isotoner':'totes',
    'u.s polo assn.':'u.s. polo assn.',
    'u.s. polo assn. classic':'u.s. polo assn.',
    'u.s. polo association':'u.s. polo assn.',
    'ugg australia':'ugg',
    'ugg mens':'ugg',
    'unbrand':'unbranded',
    'unbranded/generic':'unbranded',
    'under armour':'under armor',
    'under armour outerwear':'under armor',
    'us polo':'u.s. polo assn.',
    'us polo assn':'u.s. polo assn.',
    'vans av78':'vans',
    'vans footwear':'vans',
    'vasque footwear':'vasque',
    'venum':'venom',
    'vionic by orthaheel':'vionic',
    'vionic with orthaheel technology':'vionic',
    'vogue code':'vogue',
    'vonzipper':'von zipper',
    'walleva (not an oakley product)':'walleva',
    'westchester':'west chester',
    'wings & horns':'wings + horns',
    "wood n' stream":'wood n stream',
    'wrangler riggs':'wrangler',
    'zoot sports':'zoot'
}

data1['brand_clean'] = data1.brand.replace(brand_map)

data1['brand_clean'].value_counts()
```

Out[24]:

```
nike                      1767
ralph lauren               712
puma                       651
vans                       394
new balance                371
reebok                     273
adidas                     259
jordan                     193
```

superior glove works	187
unbranded	184
fuse lenses	174
fossa apparel	174
skechers	164
dickies	154
converse	144
unique bargains	142
justin	141
berne	137
asics	122
crocs	120
national safety apparel	119
kinco	110
toms	110
under armor	107
onguard	103
gameday boots	103
kenneth cole	100
carhartt	99
georgia	98
stacy adams	98
	...
sga	1
zlyc	1
rcs	1
ballcap buddy	1
whose lemon	1
alleson athletic	1
cubavera	1
deadman wonderland	1
tempur-pedic	1
seven color cotton	1
walk over	1
randolph	1
c1rca	1
mitchell & ness	1
pentair	1
otz shoes	1
dc comics	1
aristocrat homewares	1
lyceem	1
iceblink	1
loro piana	1
croft & barrow (kohl's)	1
j'colour	1
rlx ralph lauren	1
acne studios	1

```
mountain gear                   1
d555                            1
party favors plus               1
montblanc                       1
snickers next generation        1
Name: brand_clean, Length: 1633, dtype: int64
```

In [25]:
```python
index = data1['brand'].isna()
data1 = data1[-index]
data1.shape
```

Out[25]:
```
(18890, 39)
```

4 3. Feature Engineering

For feature engineering, I'll attempt to create additional features from the existing data that I believe will assist in my model's performance. Some additional cleaning and/or imputing may be required along the way.

4.0.1 freeShipping Flag

- 1 for offers free shipping and/or returns, 0 for doesn't; 2 for 'missing' flag

Note: I've lumped free Shipping and free Returns together, could be improved on in 'Future Improvements'.

In [26]:
```python
# create categorical variable for free shipping or not; include a missing flag
data1['prices.shipping'] = data1['prices.shipping'].str.strip().str.lower()

data1['prices.shipping'] = data1['prices.shipping'].fillna('missing')

conditions = [
    data1['prices.shipping'] == 'missing',
    data1['prices.shipping'].str.contains('free')]
choices = [2,1]
data1['freeShipping'] = np.select(conditions, choices, default=0)

data1['freeShipping'].value_counts()
```

Out[26]:
```
2    13468
1     3799
0     1623
Name: freeShipping, dtype: int64
```

4.0.2 isBid Flag

- 1 for whether the purchase was a winning bid, 0 for wasn't; 2 for 'missing' flag

In [27]:
```
# create categorical variable for winning bid or not; include a missing flag
data1['prices.offer'] = data1['prices.offer'].str.strip().str.lower()

data1['prices.offer'] = data1['prices.offer'].fillna('missing')

conditions = [
    data1['prices.offer'] == 'missing',
    data1['prices.offer'].str.contains('winning bid')]
choices = [2,1]
data1['isBid'] = np.select(conditions, choices, default=0)

data1['isBid'].value_counts()
```

Out[27]:
```
2    13026
0     4618
1     1246
Name: isBid, dtype: int64
```

4.0.3 isNew Flag

- 1 for whether the purchase was new, 0 for wasn't; 2 for 'missing' flag

In [28]:
```
# create categorical variable for isNew or not; include a missing flag
data1['prices.condition'] = data1['prices.condition'].str.strip().str.lower()

index = data1['prices.condition'] == 'amazon.com'
data1 = data1[-index]

data1['isNew'] = data1['prices.condition'].fillna('missing')

condition_map = {
    'brand new':'new',
    'new with box':'new',
    'new with tags':'new',
    'new without box':'new',
    'new without tags':'new',
    'pre-owned':'used'
}

data1['isNew'] = data1['isNew'].replace(condition_map)

data1['isNew'].value_counts()
```

Out[28]:
```
new                 12013
missing              6664
used                  164
new with defects       49
Name: isNew, dtype: int64
```

4.0.4 withBox Flag

- 1 for whether the purchase had a box, 0 for didn't; 2 for 'missing' flag

```
In [29]: data1['withBox'] = data1['prices.condition'].fillna('missing')

         conditions = [
             data1['withBox'] == 'missing',
             data1['withBox'].str.contains('with box'),
             data1['withBox'].str.contains('without box')
         ]
         choices = ['missing','with_box','without_box']
         data1['withBox'] = np.select(conditions, choices, default='missing')

         data1['withBox'].value_counts()

Out[29]: missing        17468
         with_box        1063
         without_box      359
         Name: withBox, dtype: int64
```

4.0.5 withTags Flag

- 1 for whether the purchase had tags, 0 for didn't; 2 for 'missing' flag

```
In [30]: data1['withTags'] = data1['prices.condition'].fillna('missing')

         conditions = [
             data1['withTags'] == 'missing',
             data1['withTags'].str.contains('with tags'),
             data1['withTags'].str.contains('without tags')
         ]
         choices = ['missing','with_tags','without_tags']
         data1['withTags'] = np.select(conditions, choices, default='missing')

         data1['withTags'].value_counts()

Out[30]: missing         17064
         with_tags        1683
         without_tags      143
         Name: withTags, dtype: int64
```

4.0.6 Datetime Features

Create the following new features from the 'dateSeen' column (i.e. when a particular price was seen): - duration_since_dateSeen_month: *Duration in months since a particular price (row) was seen.* - dateSeen_month: *Month when a particular price (row) was seen.* - dateSeen_year: *Year when a particular price (row) was seen.*

In [31]:
```
# create new Date Features
data1 = data1.assign(
    duration_since_dateSeen_month = ((dt.datetime.now() - data1.dateSeen)/np.timedelta
    dateSeen_month = data1.dateSeen.dt.month,
    dateSeen_year = data1.dateSeen.dt.year
)
data1.head()
```

Out[31]:
```
                     id asins                brand  \
0  AVpfHrJ6ilAPnD_xVXOI   NaN               josmo
1  AVpfHrJ6ilAPnD_xVXOI   NaN               josmo
2  AVpfHsWP1cnluZ0-eVZ7   NaN  servus by honeywell
3  AVpfHsWP1cnluZ0-eVZ7   NaN  servus by honeywell
4  AVpfHsWP1cnluZ0-eVZ7   NaN  servus by honeywell

                              categories colors  \
0  clothing,shoes,men's shoes,all men's shoes    NaN
1  clothing,shoes,men's shoes,all men's shoes    NaN
2  all men's shoes,shoes,men's shoes,clothing    NaN
3  all men's shoes,shoes,men's shoes,clothing    NaN
4  all men's shoes,shoes,men's shoes,clothing    NaN

                                        descriptions dimension            ean  \
0  [{"dateSeen":["2016-11-07T00:45:12Z"],"sourceU...       NaN  0699302044036
1  [{"dateSeen":["2016-11-07T00:45:12Z"],"sourceU...       NaN  0699302044036
2  [{"dateSeen":["2016-07-09T20:26:48Z"],"sourceU...       NaN            NaN
3  [{"dateSeen":["2016-07-09T20:26:48Z"],"sourceU...       NaN            NaN
4  [{"dateSeen":["2016-07-09T20:26:48Z"],"sourceU...       NaN            NaN

                                            features  \
0  [{"key":"Gender","value":["Men"]},{"key":"Shoe...
1  [{"key":"Gender","value":["Men"]},{"key":"Shoe...
2  [{"key":"Gender","value":["Men"]},{"key":"Colo...
3  [{"key":"Gender","value":["Men"]},{"key":"Colo...
4  [{"key":"Gender","value":["Men"]},{"key":"Colo...

                                           imageURLs      ...        dateSeen  \
0  https://i5.walmartimages.com/asr/13ac3d61-003c...      ...      2016-11-05
1  https://i5.walmartimages.com/asr/13ac3d61-003c...      ...      2016-11-05
2  http://i5.walmartimages.com/dfw/dce07b8c-5844/...      ...      2016-03-08
3  http://i5.walmartimages.com/dfw/dce07b8c-5844/...      ...      2015-11-30
4  http://i5.walmartimages.com/dfw/dce07b8c-5844/...      ...      2016-04-29

           brand_clean freeShipping isBid    isNew  withBox  withTags  \
0                josmo            2     0  missing  missing   missing
1                josmo            2     0      new  missing   missing
2  servus by honeywell            2     2      new  missing   missing
3  servus by honeywell            2     2      new  missing   missing
```

```
4  servus by honeywell            2     2      new  missing   missing

  duration_since_dateSeen_month dateSeen_month dateSeen_year
0                     33.534298             11          2016
1                     33.534298             11          2016
2                     41.485180              3          2016
3                     44.737814             11          2015
4                     39.776726              4          2016

[5 rows x 47 columns]
```

4.0.7 URL Features

Using basic string manipulations, I want to see whether I can extract some meaningful categorical labels for websites on which the stored prices were seen. - A discount store like Walmart or a bidding site like eBay (often used goods) might price products lower than a luxury fashion retailer like SSENSE. - I'm also curious to see just how many different sources (base URLs) are being scraped for prices.

In [32]:
```python
data1['sourceURLs'] = data1['sourceURLs'].fillna('missing')
data1['prices.sourceURLs'] = data1['prices.sourceURLs'].fillna('missing')

conditions = [
    data1['sourceURLs'] != 'missing',
    data1['prices.sourceURLs'] != 'missing']
choices = [data1['sourceURLs'],data1['prices.sourceURLs']]
data1['URL_clean'] = np.select(conditions, choices, default='missing')

data1['URL_domain_clean'] = data1['URL_clean'].apply(lambda url: tldextract.extract(u
data1['URL_domain_clean'].value_counts()
```

Out[32]:
```
walmart                                                  8648
ebay                                                     3578
sears                                                    3471
amazon                                                   1916
ralphlauren                                               674
sportsauthority                                           169
nordstrom                                                 110
newegg                                                     68
macys                                                      58
shoes                                                      48
puma                                                       40
gandermountain                                             33
kmart                                                      20
overstock                                                  17
sunglassesshop                                             16
lowes                                                      10
homedepot                                                   7
```

```
calvinklein                                                    4
missing                                                        2
 Please use a view that flattens this field to see this data   1
Name: URL_domain_clean, dtype: int64
```

```
In [33]: # upon close inspection, remove sunglasses anomalies
         index = data1['URL_domain_clean'] == 'sunglassesshop'
         data1 = data1[-index]

In [34]: index = data1['URL_domain_clean'] == 'missing'
         data1[index]
```

```
Out[34]:                           id                   asins     brand  \
         1390  AVpfCauiilAPnD_xTlFQ  B009E2JPBK,B009E2JK8S  burberry
         1392  AVpfCauiilAPnD_xTlFQ  B009E2JPBK,B009E2JK8S  burberry

                                                   categories colors descriptions  \
         1390  accessories,sunglasses,men,shoes,clothing, sho...    NaN          NaN
         1392  accessories,sunglasses,men,shoes,clothing, sho...    NaN          NaN

              dimension  ean features  \
         1390       NaN  NaN      NaN
         1392       NaN  NaN      NaN

                                                    imageURLs      ...          \
         1390  http://ecx.images-amazon.com/images/I/31PBPB8N...      ...
         1392  http://ecx.images-amazon.com/images/I/31PBPB8N...      ...

              freeShipping isBid    isNew  withBox withTags  \
         1390            0     2  missing  missing  missing
         1392            0     2  missing  missing  missing

               duration_since_dateSeen_month  dateSeen_month dateSeen_year URL_clean  \
         1390                      41.419471               3          2016   missing
         1392                      41.386616               3          2016   missing

              URL_domain_clean
         1390          missing
         1392          missing

         [2 rows x 49 columns]
```

Since both missing URLs are Burberry brand products, let's see what domain sells the most Burberry brand (i.e. mode).

```
In [35]: index = data1['brand'] == 'burberry'
         data1[index]['URL_domain_clean'].value_counts()

Out[35]: amazon     32
         missing     2
```

```
ebay        1
Name: URL_domain_clean, dtype: int64
```

```
In [36]: # replace the missing URL domain with the mode for brand 'Burberry'
         URL_map = {
             'missing':'amazon'
         }
         data1['URL_domain_clean'] = data1['URL_domain_clean'].replace(URL_map)
```

Out of curiosity, let's see what domains have 'winning bids' for prices.

```
In [37]: # check what websites you can bid on
         cond_one = data1['isBid'] == 1
         data1[cond_one]['URL_domain_clean'].value_counts()
```

```
Out[37]: ebay       1215
         walmart      27
         sears         2
         amazon        2
         Name: URL_domain_clean, dtype: int64
```

As expected, most of the 'winning bid' prices were on eBay. Unsure of whether the rest are just anomalous, but will leave them in for the time being (another 'Future Improvement' to investigate).

4.0.8 Currency (conversion)

For this analysis, we will convert all prices to USD - i.e. use USD price as target variable. - Alternative would be to predict prices in each currency, at which point the target variable would be imbalanced in the dataset since there is an overwhelming majority of USD price data points in the dataset. (Added to 'Future Improvements' section.)

```
In [38]: from currency_converter import CurrencyConverter
         EUR = CurrencyConverter().convert(1, currency= 'EUR', new_currency= 'USD');
         CAD = CurrencyConverter().convert(1, currency= 'CAD', new_currency= 'USD');
         AUD = CurrencyConverter().convert(1, currency= 'AUD', new_currency= 'USD');
         GBP = CurrencyConverter().convert(1, currency= 'GBP', new_currency= 'USD');

         # just to have a conversion column
         data1['conversion'] = pd.DataFrame(np.ones(data1.shape[1]))
         for index, row in data1.iterrows():
             if (data1['prices.currency'][index]=='EUR'): data1['conversion']= EUR
             if (data1['prices.currency'][index]=='CAD'): data1['conversion']= CAD
             if (data1['prices.currency'][index]=='AUD'): data1['conversion']= AUD
             if (data1['prices.currency'][index]=='GBP'): data1['conversion']= GBP
```

```
In [39]: f"{EUR},{CAD},{AUD},{GBP}"
```

```
Out[39]: '1.1209,0.7434009815625414,0.7039944730561486,1.2951643653590617'
```

```
In [40]: data1['prices.amountMin_converted'] = data1['prices.amountMin'].copy()
         data1['prices.amountMax_converted'] = data1['prices.amountMax'].copy()


         data1.loc[data1['prices.currency']=='EUR', 'prices.amountMin_converted'] = data1['pri
         data1.loc[data1['prices.currency']=='EUR', 'prices.amountMax_converted'] = data1['pri
         data1.loc[data1['prices.currency']=='CAD', 'prices.amountMin_converted'] = data1['pri
         data1.loc[data1['prices.currency']=='CAD', 'prices.amountMax_converted'] = data1['pri
         data1.loc[data1['prices.currency']=='AUD', 'prices.amountMin_converted'] = data1['pri
         data1.loc[data1['prices.currency']=='AUD', 'prices.amountMax_converted'] = data1['pri
         data1.loc[data1['prices.currency']=='GBP', 'prices.amountMin_converted'] = data1['pri
         data1.loc[data1['prices.currency']=='GBP', 'prices.amountMax_converted'] = data1['pri
```

5 4. Exploratory Data Analysis

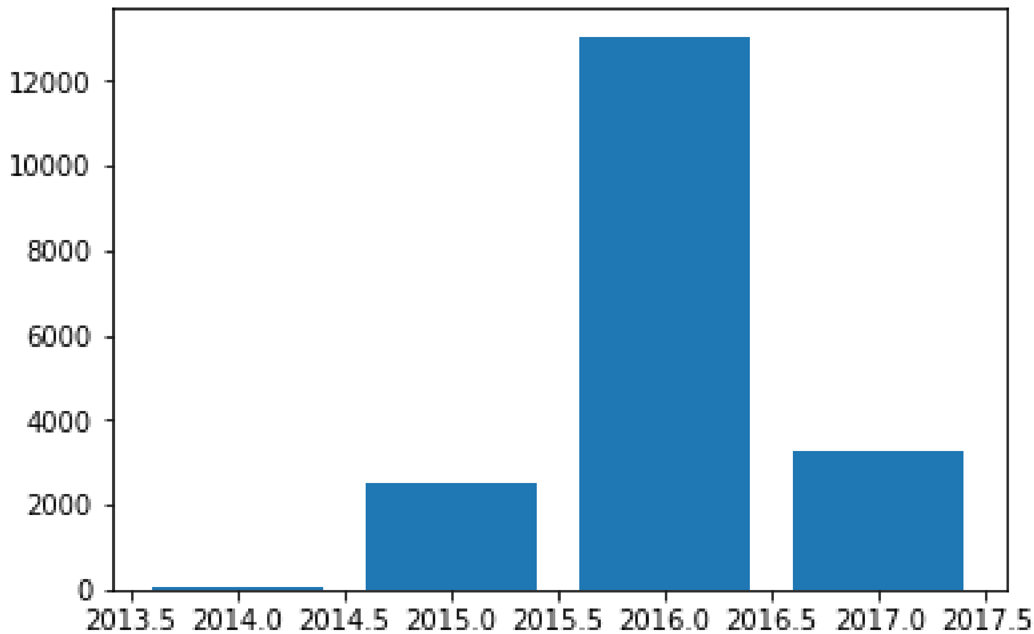
Note that the EDA for this Notebook was performed before (as well as in tandem with) the previous Feature Engineering step. I found myself cycling between steps 2, 3 and 4, which is fairly common place.

5.0.1 Temporal Exploration

```
In [41]: year_counts = (data1['dateSeen_year'].value_counts()
                            .reset_index()
                            .sort_values(by=['index'], ascending=True)
                            .rename(columns={'index': 'year', 'dateSeen_year': 'count'})
                        )
         print('These are the following counts by year:')
         print(year_counts)
         plt.bar(year_counts['year'], year_counts['count'])
```

```
These are the following counts by year:
   year  count
3  2014     27
2  2015   2505
0  2016  13049
1  2017   3293
```

```
Out[41]: <BarContainer object of 4 artists>
```



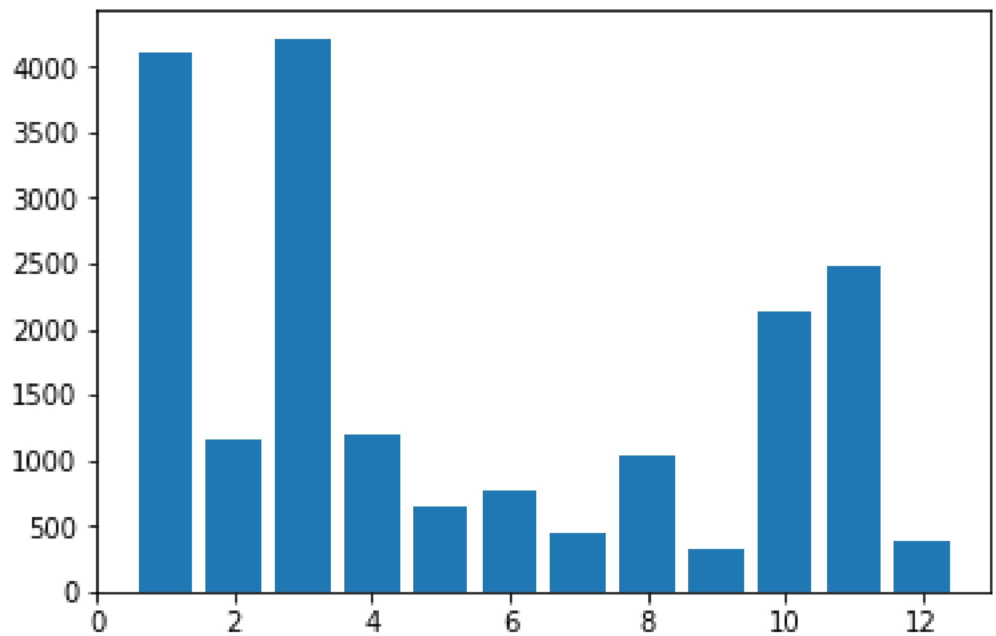
The majority of prices were collected during the year of 2016. For 'Further Investigation', it might be worth looking (at least high level) at whether any macroeconomic trends (e.g. recession) may have affected prices (per region) during that year with respect to the others, notably 2014, 2015 and 2017.

```
In [42]: month_counts = (data1['dateSeen_month'].value_counts()
                                  .reset_index()
                                  .sort_values(by=['index'], ascending=True)
                                  .rename(columns={'index': 'month', 'dateSeen_month': 'count'})
                              )
         print('These are the following counts by month:')
         print(month_counts)
         plt.bar(month_counts['month'], month_counts['count'])
```

```
These are the following counts by month:
    month  count
1       1   4113
5       2   1153
0       3   4215
4       4   1187
8       5    648
7       6    776
9       7    445
6       8   1033
11      9    318
3      10   2125
2      11   2486
```

```
10      12     375
```

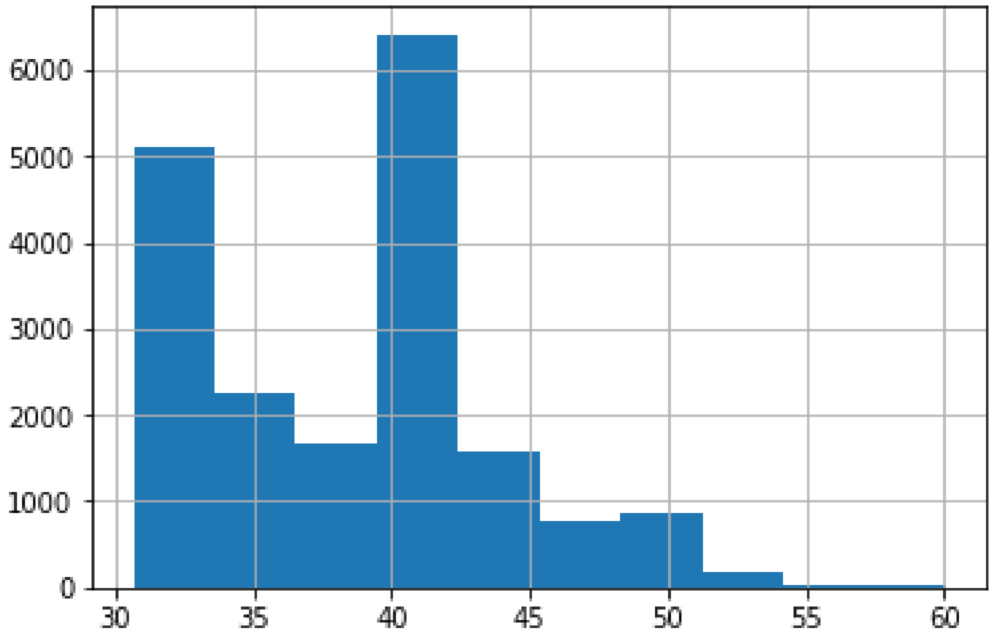
```
Out[42]: <BarContainer object of 12 artists>
```



The distribution of months during which prices were collected is highly non-uniform, with peaks in January and March, as well as growth in October and November (possibly coinciding with festive/sales events such as Christmas/New Years, Black Friday, etc). That being said, following that train of thought, one might expect more price-tracking in July and December, so it's best not to assume anything other than a non-uniform collection of data points temporally.

```
In [43]: data1['duration_since_dateSeen_month'].hist()
```

```
Out[43]: <matplotlib.axes._subplots.AxesSubplot at 0x1a166b2550>
```

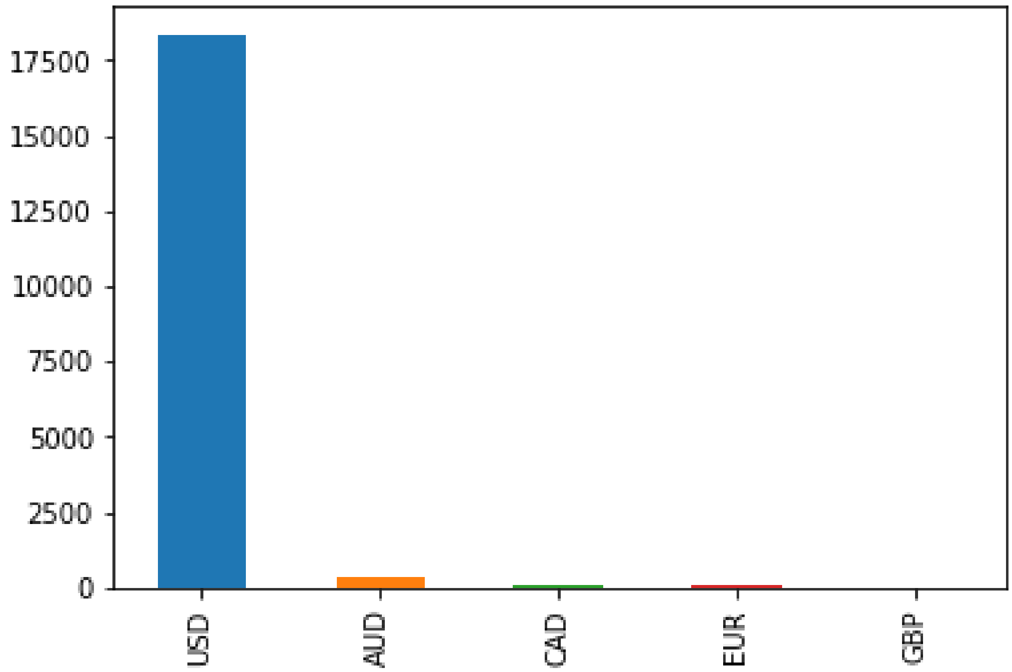


5.0.2 Currency Exploration

```
In [44]: print('There are', data1['prices.currency'].nunique(), 'different currencies in this d
         print(data1['prices.currency'].value_counts())
         data1['prices.currency'].value_counts().plot(kind="bar")
```

```
There are 5 different currencies in this dataset.
USD    18348
AUD      335
CAD      103
EUR       66
GBP       22
Name: prices.currency, dtype: int64
```

```
Out[44]: <matplotlib.axes._subplots.AxesSubplot at 0x1a173e1240>
```



The prices in the dataset are clearly dominated by USD pricing. It would probably make sense to convert all prices to USD (as done in 'Feature Engineering' section), since there just wouldn't be a lot of data points for the model to learn to predict in different currencies (+ possible bias in the target variable).

Even if prices are converted, it might still be worth encoding currencies as a categorical variable in case pricing differs per country/currency, even after converting the prices to USD. Time permitted (see 'Further Improvements'), I would try building out *both* models - 1) for predicting prices converted to USD, and 2) for predicting prices based on the currency of interest.

5.0.3 Price Exploration

This section deep-dives into the disparity between prices.amountMin and prices.amount Max, as well as possible outliers therein. - As outlined in the introduction, each row represents a product-price tuple: each product can occur in multiple rows if it is listed at multiple different prices on different website listings (sourceURLs). - Furthermore, the prices on those listings can change over time (temporal element), so: 1. prices.amountMin = MINIMUM price listed for the product on those same sourceURLs 2. prices.amountMax = MAXIMUM price listed for the product on those same sourceURLs

```
In [45]: # number of rows where prices.amountMin > prices.amountMax
         len(data1[data1['prices.amountMin'] > data1['prices.amountMax']])
```

```
Out[45]: 0
```

Note: all values of prices.amountMin are indeed < prices.amountMax.

```
In [46]: # number of rows where prices.amountMin < prices.amountMax
         print('There are',len(data1[data1['prices.amountMin'] < data1['prices.amountMax']]),
         print('These rows represent',len(data1[data1['prices.amountMin'] < data1['prices.amou
```

```
There are 640 rows with amountMin < amountMax
These rows represent 0.03390908127582918 of the data.
```

Create a new column to investigate the percentage difference between amountMin and amountMax when they differ.

```
In [47]: data1['percent_difference'] = (data1['prices.amountMax'] - data1['prices.amountMin'])
```

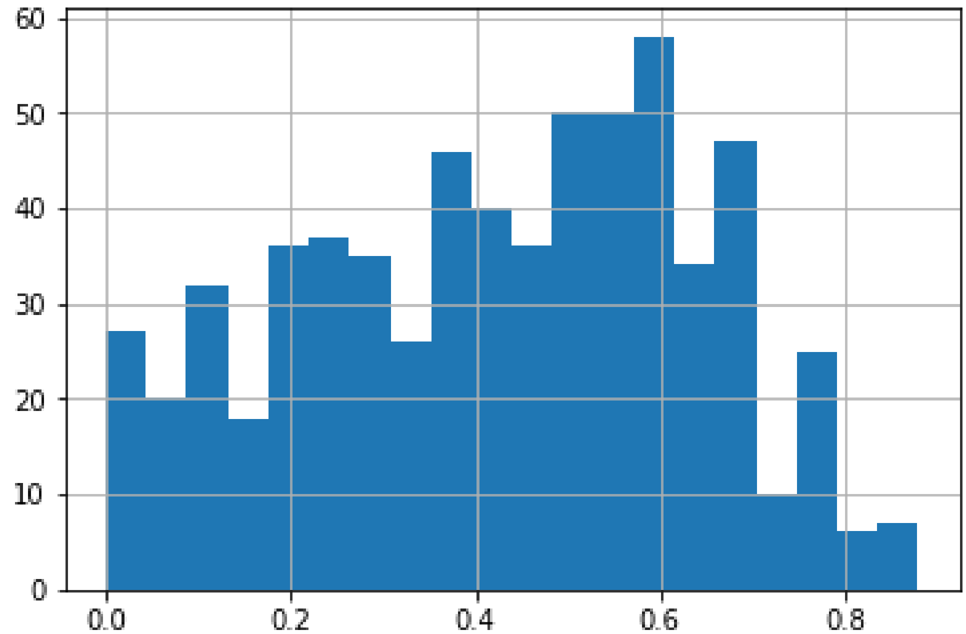
```
In [48]: amountMin_less_than_amountMax = data1[data1['prices.amountMin'] < data1['prices.amoun
         amountMin_less_than_amountMax[['id','isSale','prices.offer','prices.amountMin','price
```

```
Out[48]:                        id  isSale prices.offer  prices.amountMin  \
         109  AVpfEdjqLJeJML431trL       0      missing             84.85
         189  AVpfG4VH1cnluZ0-eEOn       0      missing             96.44
         190  AVpfG4VH1cnluZ0-eEOn       0      missing            109.00
         191  AVpfG4VH1cnluZ0-eEOn       0      missing            103.01
         192  AVpfG4VH1cnluZ0-eEOn       0      missing            110.00
         236  AVpfCDi91cnluZ0-cabr       0      missing             34.99
         417  AVpe-HGdilAPnD_xSDuY       0      missing             28.95
         465  AVpfDlggLJeJML431aAm       0      missing            114.99
         466  AVpfDlggLJeJML431aAm       0      missing            114.99
         471  AVpfDYmVLJeJML431VWs       0      missing             29.50

              prices.amountMax
         109            179.99
         189            165.00
         190            179.99
         191            165.00
         192            179.99
         236             42.00
         417             37.95
         465            175.00
         466            174.95
         471             45.00
```

```
In [49]: amountMin_less_than_amountMax['percent_difference'].hist(bins=20)
```

```
Out[49]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1d7fd9b0>
```



The distribution of the percent difference between amountMin and amountMax (for the products where they're not equal) is actually "somewhat" normal, albeit a bit right skewed. We will proceed nevertheless with using the mean of USD prices as the target variable in our model (since the potentially biased data points represent less than 1% of the data points in the dataset).

```
In [50]: # define a column for mean of min and max price converted to USD, and get snapshot of
         data1['mean_price_usd'] = data1[['prices.amountMin_converted', 'prices.amountMax_conve
         data1[data1['prices.amountMin'] < data1['prices.amountMax']].head()
```

```
Out[50]:                         id                    asins   brand  \
         109  AVpfEdjqLJeJML431trL               B00242PMIC  jordan
         189  AVpfG4VH1cnluZ0-eE0n  B005DXAS2E,B005DXB0LM    geox
         190  AVpfG4VH1cnluZ0-eE0n  B005DXAS2E,B005DXB0LM    geox
         191  AVpfG4VH1cnluZ0-eE0n  B005DXAS2E,B005DXB0LM    geox
         192  AVpfG4VH1cnluZ0-eE0n  B005DXAS2E,B005DXB0LM    geox

                                                   categories colors descriptions  \
         109  shoes,clothing, shoes & jewelry,men,athletic,t...    NaN          NaN
         189  loafers & slip-ons,men,shoes,clothing, shoes &...    NaN          NaN
         190  loafers & slip-ons,men,shoes,clothing, shoes &...    NaN          NaN
         191  loafers & slip-ons,men,shoes,clothing, shoes &...    NaN          NaN
         192  loafers & slip-ons,men,shoes,clothing, shoes &...    NaN          NaN

             dimension  ean features  \
         109       NaN  NaN      NaN
         189       NaN  NaN      NaN
         190       NaN  NaN      NaN
```

```
191        NaN   NaN       NaN
192        NaN   NaN       NaN

                                              imageURLs       ...         \
109  http://ecx.images-amazon.com/images/I/41dfRe0b...        ...
189                                                 NaN       ...
190                                                 NaN       ...
191                                                 NaN       ...
192                                                 NaN       ...

    duration_since_dateSeen_month dateSeen_month dateSeen_year  \
109                     45.559186             11          2015
189                     45.756315             10          2015
190                     50.027450              6          2015
191                     45.822025             10          2015
192                     47.629044              9          2015

                                              URL_clean URL_domain_clean  \
109  http://www.amazon.com/Jordan-Rising-White-Infr...           amazon
189  http://www.amazon.com/Geox-Monet-Plain-Vamp-Le...           amazon
190  http://www.amazon.com/Geox-Monet-Plain-Vamp-Le...           amazon
191  http://www.amazon.com/Geox-Monet-Plain-Vamp-Le...           amazon
192  http://www.amazon.com/Geox-Monet-Plain-Vamp-Le...           amazon

     conversion  prices.amountMin_converted prices.amountMax_converted  \
109    0.743401                       84.85                     179.99
189    0.743401                       96.44                     165.00
190    0.743401                      109.00                     179.99
191    0.743401                      103.01                     165.00
192    0.743401                      110.00                     179.99

    percent_difference mean_price_usd
109           0.528585        132.420
189           0.415515        130.720
190           0.394411        144.495
191           0.375697        134.005
192           0.388855        144.995

[5 rows x 54 columns]
```

5.0.4 Names and Categories

Do some preliminary string cleaning on names and categories (i.e. basic string manipulations - ideally should be lemmatizing and stemming).

In [51]:
```
data1['name'] = data1['name'].str.lower().str.strip()
data1['categories'] = data1['categories'].str.lower().str.strip()
```

```
data1['name'] = data1['name'].str.replace("shoes","shoe")
data1['name'] = data1['name'].str.replace("boots","boot")
data1['name'] = data1['name'].str.replace("sandals","sandal")
data1['name'] = data1['name'].str.replace("sneakers","sneaker")
data1['name'] = data1['name'].str.replace("loafers","loafer")
data1['name'] = data1['name'].str.replace("slippers","slipper")

data1['categories'] = data1['categories'].str.replace("shoes","shoe")
data1['categories'] = data1['categories'].str.replace("boots","boot")
data1['categories'] = data1['categories'].str.replace("sandals","sandal")
data1['categories'] = data1['categories'].str.replace("sneakers","sneaker")
data1['categories'] = data1['categories'].str.replace("loafers","loafer")
data1['categories'] = data1['categories'].str.replace("slippers","slipper")
```

Use TF-iDF to try extracting a score for relevant category in the product name. *Note: 'Future Improvement', compare the performance when using CountVectorizer for name as well.*

```
In [52]: from sklearn.feature_extraction.text import TfidfVectorizer

         tfidf_vectorizer = TfidfVectorizer()
         values = tfidf_vectorizer.fit_transform(data1['name'])

         # Show the Model as a pandas DataFrame
         feature_names = tfidf_vectorizer.get_feature_names()
         nlp_name = pd.DataFrame(values.toarray(), columns = feature_names)
         nlp_name = nlp_name[['boot','shoe','footwear','sandal','running',
                              'sneaker','loafer','slipper','moccasin']]

         nlp_name.columns = ['nlp_name_' + str(col) for col in nlp_name.columns]
```

Use Count Vectorization to try extracting a score for relevant category in the product name.

```
In [53]: from sklearn.feature_extraction.text import CountVectorizer

         count_vectorizer = CountVectorizer()
         values = count_vectorizer.fit_transform(data1['categories'])

         # Show the Model as a pandas DataFrame
         feature_names = count_vectorizer.get_feature_names()
         nlp_categories = pd.DataFrame(values.toarray(), columns = feature_names)
         nlp_categories = nlp_categories[['boot','shoe','footwear','sandal',
                              'sneaker','loafer','slipper','moccasin','running']]

         nlp_categories.columns = ['nlp_cat_' + str(col) for col in nlp_categories.columns]
```

```
In [54]: data1.shape
         nlp_name.shape
         #nlp_categories.shape
```

Out[54]: (18874, 9)

In [55]:
```
data1.reset_index(drop=True,inplace=True)
nlp_name.reset_index(drop=True,inplace=True)
nlp_categories.reset_index(drop=True,inplace=True)

data1 = pd.concat([data1, nlp_name, nlp_categories],axis=1)
```

In [56]:
```
data1.shape
```

Out[56]: (18874, 72)

Define the shoe_name_score to be the NLP-generated column that most closely (max score) matches the name in the data.

In [57]:
```
data1['shoe_name_score'] = data1[nlp_name.columns].max(axis=1)
data1['shoe_name_score'] = data1['shoe_name_score'].fillna(0)
data1.head(2)
```

Out[57]:
```
                    id asins  brand                              categories  \
0  AVpfHrJ6ilAPnD_xVXOI   NaN  josmo  clothing,shoe,men's shoe,all men's shoe
1  AVpfHrJ6ilAPnD_xVXOI   NaN  josmo  clothing,shoe,men's shoe,all men's shoe

  colors                                       descriptions dimension  \
0    NaN  [{"dateSeen":["2016-11-07T00:45:12Z"],"sourceU...       NaN
1    NaN  [{"dateSeen":["2016-11-07T00:45:12Z"],"sourceU...       NaN

            ean                                           features  \
0  0699302044036  [{"key":"Gender","value":["Men"]},{"key":"Shoe...
1  0699302044036  [{"key":"Gender","value":["Men"]},{"key":"Shoe...

                                           imageURLs      ...         \
0  https://i5.walmartimages.com/asr/13ac3d61-003c...      ...
1  https://i5.walmartimages.com/asr/13ac3d61-003c...      ...

  nlp_cat_boot nlp_cat_shoe nlp_cat_footwear nlp_cat_sandal nlp_cat_sneaker  \
0            0            3                0              0               0
1            0            3                0              0               0

   nlp_cat_loafer  nlp_cat_slipper nlp_cat_moccasin nlp_cat_running  \
0               0                0                0               0
1               0                0                0               0

  shoe_name_score
0         0.14022
1         0.14022

[2 rows x 73 columns]
```

Define the shoe_cat_score to be the NLP-generated column that most closely (max score) matches the category in the data.

```
In [58]: data1['shoe_cat_score'] = data1[nlp_categories.columns].max(axis=1)
         data1['shoe_cat_score'] = data1['shoe_cat_score'].fillna(0)
         data1.head(2)
```

```
Out[58]:                      id asins  brand                              categories  \
         0  AVpfHrJ6ilAPnD_xVXOI   NaN  josmo  clothing,shoe,men's shoe,all men's shoe
         1  AVpfHrJ6ilAPnD_xVXOI   NaN  josmo  clothing,shoe,men's shoe,all men's shoe

           colors                                       descriptions dimension  \
         0    NaN  [{"dateSeen":["2016-11-07T00:45:12Z"],"sourceU...       NaN
         1    NaN  [{"dateSeen":["2016-11-07T00:45:12Z"],"sourceU...       NaN

                      ean                                           features  \
         0  0699302044036  [{"key":"Gender","value":["Men"]},{"key":"Shoe...
         1  0699302044036  [{"key":"Gender","value":["Men"]},{"key":"Shoe...

                                                     imageURLs      ...       \
         0  https://i5.walmartimages.com/asr/13ac3d61-003c...      ...
         1  https://i5.walmartimages.com/asr/13ac3d61-003c...      ...

           nlp_cat_shoe nlp_cat_footwear nlp_cat_sandal nlp_cat_sneaker nlp_cat_loafer  \
         0            3                0              0               0              0
         1            3                0              0               0              0

            nlp_cat_slipper  nlp_cat_moccasin nlp_cat_running shoe_name_score  \
         0                0                 0               0         0.14022
         1                0                 0               0         0.14022

           shoe_cat_score
         0              3
         1              3

         [2 rows x 74 columns]
```

```
In [59]: cond_1 = data1['shoe_name_score'] < 0.20
         cond_2 = data1['shoe_cat_score'] <= 1
         data1[['categories','name','shoe_name_score','shoe_cat_score']][cond_1 & cond_2].head
```

```
Out[59]:                                             categories  \
         18  clothing, shoe, accessories,bags, briefcases,m...
         22  men's halloween costumes,adult halloween costu...
         23  men's halloween costumes,adult halloween costu...
         24  men's halloween costumes,adult halloween costu...
         25  clothing, shoe & accessories,men's clothing,un...
         32  clothing, shoe & accessories,men's clothing,sh...
         35       clothing, shoe, accessories,men's sunglasses
```

```
36  clothing, shoe & accessories,men's clothing,sh...
37  all men's clothing,men's clothing,men's outerw...
38  all men's clothing,men's clothing,men's outerw...

                                              name  shoe_name_score  \
18  men's faux leather business handbag messenger ...              0.0
22  rubies costume adult mens regency plush santa ...              0.0
23  rubies costume adult mens regency plush santa ...              0.0
24  rubies costume adult mens regency plush santa ...              0.0
25  men boxer underwear shorts modal male underpan...              0.0
32  american fighter by affliction north creek boa...              0.0
35  polarized sunglasses maui jim bamboo forest 41...              0.0
36  venum men's koi compression pants spats mma bl...              0.0
37  azzuro cozy fit notched lapel long sleeve blaz...              0.0
38  azzuro cozy fit notched lapel long sleeve blaz...              0.0

    shoe_cat_score
18               1
22               1
23               1
24               1
25               1
32               1
35               1
36               1
37               1
38               1
```

In [60]:
```
data1['shoe_name_score'].isna().sum()
```

Out[60]: 0

At first glance, some of these NLP scores are problematic because some data points which are clearly shoes (i.e. contain the words that we're defining to be part of the 'shoe' bag of words). I will utilize the scores as features derived from the 'categories' and 'name' columns, and input them into the model as is, but they clearly need more tweaking (see 'Future Improvements').

5.0.5 Correlation of Features

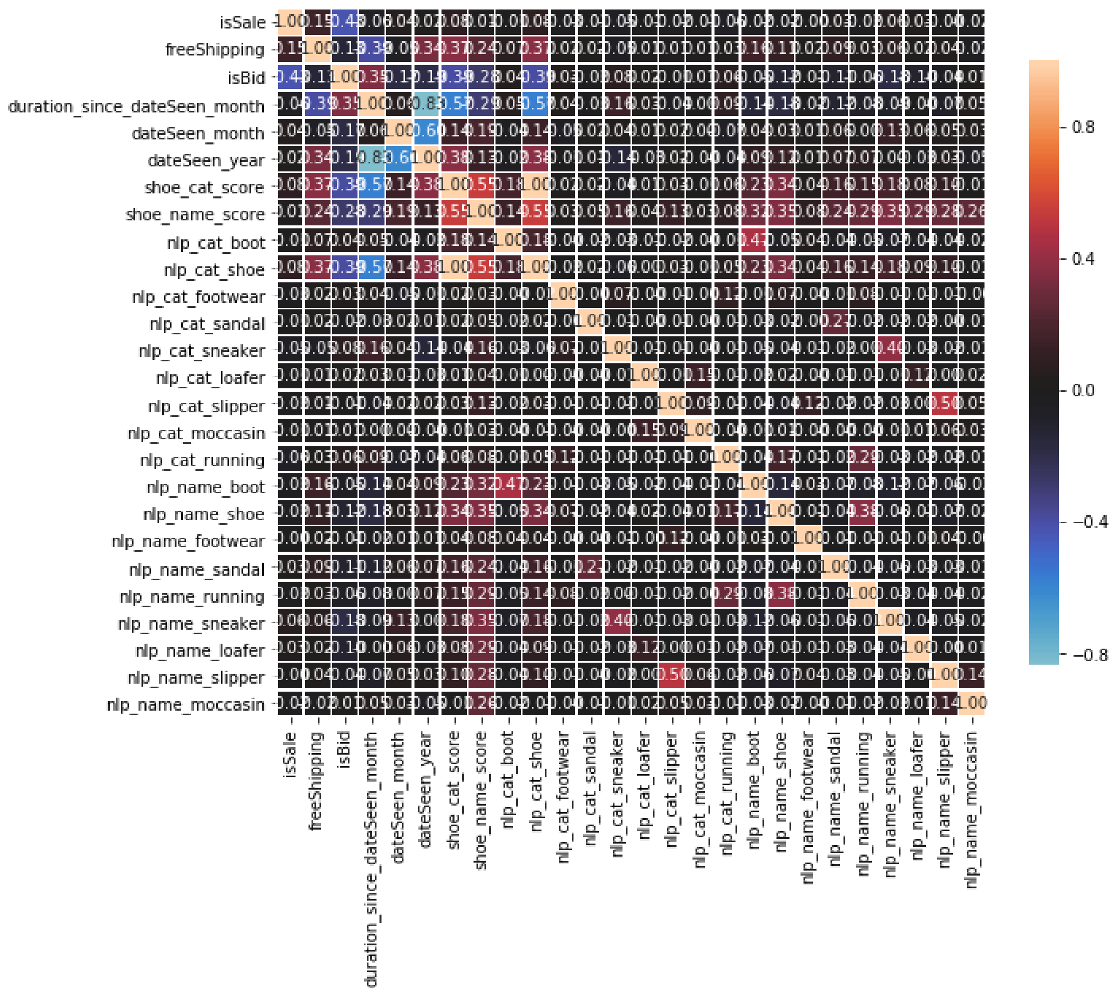
In [61]:
```
def correlation_heatmap(train):
    correlations = train.corr()

    fig, ax = plt.subplots(figsize=(10,10))
    sns.heatmap(correlations, vmax=1.0, center=0, fmt='.2f',
                square=True, linewidths=.5, annot=True, cbar_kws={"shrink": .70})
    plt.show();

correlation_columns = ['brand_clean', 'gender', 'prices.currency', 'isSale', 'freeShi
            'duration_since_dateSeen_month','URL_domain_clean','dateSeen_month','dateS
```

```
                'isNew','withTags','withBox','shoe_cat_score','shoe_name_score']
correlation_columns = correlation_columns + nlp_categories.columns.tolist() + nlp_name

correlation_heatmap(data1[correlation_columns])
```



The most problematic columns with respect to correlation are 'dateSeen_year' and 'nlp_cat_shoe'. We will consider removing these features to avoid colinearity in the model (will test to see difference in model's performance with and without them present).

6 5. Benchmarking

I will use both a *measure of central tendency comparison* and *linear regression model* as benchmarks to compare my final model to. A linear regression model is arguably the most basic/displaying of the least competitive accuracy in comparison to other more complex supervised machine learning regression algorithms, so it is a worthy option as a benchmark. It will be contrasted to the worse-case benchmark of using mean and/or median of the target variable as predictions.

Preprocessing Data for Linear Model I will select specific features from the existing + engineered ones in the dataset, and scale them using MinMaxScaler for the linear model. The model will be split into train and test sets respectively. I will use cross validation (GridSearchcv) on the train set to pick the best K fold. Then, I will run predictions on the test set using the model, and compare the predictions to the true values in the test set to get the benchmarking metrics.

Model Performance MSE, RMSE and R2 will be used to evaluate model performance.

6.1 Pre-processing

6.1.1 Feature Selection

The following features were used as inputs for the linear model:

```
In [62]: data1.columns
```

```
Out[62]: Index(['id', 'asins', 'brand', 'categories', 'colors', 'descriptions',
               'dimension', 'ean', 'features', 'imageURLs', 'keys', 'manufacturer',
               'manufacturerNumber', 'merchants', 'name', 'prices.amountMin',
               'prices.amountMax', 'prices.availability', 'prices.color',
               'prices.condition', 'prices.count', 'prices.currency', 'prices.flavor',
               'prices.merchant', 'prices.offer', 'prices.returnPolicy',
               'prices.shipping', 'prices.size', 'prices.sourceURLs', 'reviews',
               'sizes', 'skus', 'sourceURLs', 'upc', 'weight', 'gender', 'isSale',
               'dateSeen', 'brand_clean', 'freeShipping', 'isBid', 'isNew', 'withBox',
               'withTags', 'duration_since_dateSeen_month', 'dateSeen_month',
               'dateSeen_year', 'URL_clean', 'URL_domain_clean', 'conversion',
               'prices.amountMin_converted', 'prices.amountMax_converted',
               'percent_difference', 'mean_price_usd', 'nlp_name_boot',
               'nlp_name_shoe', 'nlp_name_footwear', 'nlp_name_sandal',
               'nlp_name_running', 'nlp_name_sneaker', 'nlp_name_loafer',
               'nlp_name_slipper', 'nlp_name_moccasin', 'nlp_cat_boot', 'nlp_cat_shoe',
               'nlp_cat_footwear', 'nlp_cat_sandal', 'nlp_cat_sneaker',
               'nlp_cat_loafer', 'nlp_cat_slipper', 'nlp_cat_moccasin',
               'nlp_cat_running', 'shoe_name_score', 'shoe_cat_score'],
              dtype='object')
```

```
In [63]: input_features = ['brand_clean', 'gender', 'prices.currency', 'isSale', 'freeShipping
                     'duration_since_dateSeen_month','URL_domain_clean','dateSeen_month',
                     'isNew','withTags','withBox','shoe_cat_score','shoe_name_score']

         target_feature = ['mean_price_usd']

         X = data1[input_features]
         y = data1[target_feature]

         X.head(2)
```

```
Out[63]:   brand_clean gender prices.currency  isSale  freeShipping  isBid  \
        0       josmo    Men             USD       1             2      0
        1       josmo    Men             USD       0             2      0

          duration_since_dateSeen_month URL_domain_clean  dateSeen_month    isNew  \
        0                     33.534298          walmart              11  missing
        1                     33.534298          walmart              11      new

          withTags  withBox  shoe_cat_score  shoe_name_score
        0  missing  missing               3          0.14022
        1  missing  missing               3          0.14022
```

Note: I've explicitly removed 'dateSeen_year' column due to high correlation with columns 'dateSeen_month' and 'duration_since_dateSeen_month'. Further, I've omitted most of the NLP columns safe for the MAX scores due to significant negative impact on the linear model performance; 'nlp_cat_shoe' column in particular due to its perfect correlation with the 'shoe_cat_score' column (to be expected giving the cleaning I did in sections 2. and 3.).

6.1.2 Preprocessing Features

- one hot encode object features EXCEPT 'brand_clean' (high cardinality categorical feature)
- for 'brand_clean', we will use hash encoding since its a nominal feature with high cardinality

```
In [64]: X_dummies = pd.get_dummies(X[X.columns.difference(['brand_clean'])])
         X_dummies = X_dummies.drop('URL_domain_clean_ Please use a view that flattens this fie
         X_dummies.head(2)
```

```
Out[64]:    dateSeen_month  duration_since_dateSeen_month  freeShipping  isBid  isSale  \
        0              11                      33.534298             2      0       1
        1              11                      33.534298             2      0       0

           shoe_cat_score  shoe_name_score  URL_domain_clean_amazon  \
        0               3          0.14022                        0
        1               3          0.14022                        0

           URL_domain_clean_calvinklein  URL_domain_clean_ebay          ...            \
        0                             0                      0          ...
        1                             0                      0          ...

           prices.currency_CAD  prices.currency_EUR  prices.currency_GBP  \
        0                    0                    0                    0
        1                    0                    0                    0

           prices.currency_USD  withBox_missing  withBox_with_box  \
        0                    1                1                 0
        1                    1                1                 0

           withBox_without_box  withTags_missing  withTags_with_tags  \
        0                    0                 1                   0
```

```
1                      0                  1                     0

   withTags_without_tags
0                      0
1                      0

[2 rows x 42 columns]
```

```
In [65]: brand_clean_hash = ce.HashingEncoder(cols = ['brand_clean'])
         brand_clean_encoded = brand_clean_hash.fit_transform(X['brand_clean'], y)
         brand_clean_encoded.columns = ['brand_clean_' + str(col) for col in brand_clean_encode

         brand_clean_encoded.head(2)
```

```
Out[65]:    brand_clean_col_0  brand_clean_col_1  brand_clean_col_2  brand_clean_col_3  \
         0                  1                  0                  0                  0
         1                  1                  0                  0                  0

            brand_clean_col_4  brand_clean_col_5  brand_clean_col_6  brand_clean_col_7
         0                  0                  0                  0                  0
         1                  0                  0                  0                  0
```

```
In [66]: X_dummies = pd.concat([X_dummies,brand_clean_encoded],axis=1)
         X_dummies.head(2)
```

```
Out[66]:    dateSeen_month  duration_since_dateSeen_month  freeShipping  isBid  isSale  \
         0              11                      33.534298             2      0       1
         1              11                      33.534298             2      0       0

            shoe_cat_score  shoe_name_score  URL_domain_clean_amazon  \
         0               3          0.14022                        0
         1               3          0.14022                        0

            URL_domain_clean_calvinklein  URL_domain_clean_ebay        ...          \
         0                             0                      0        ...
         1                             0                      0        ...

            withTags_with_tags  withTags_without_tags  brand_clean_col_0  \
         0                   0                      0                  1
         1                   0                      0                  1

            brand_clean_col_1  brand_clean_col_2  brand_clean_col_3  brand_clean_col_4  \
         0                  0                  0                  0                  0
         1                  0                  0                  0                  0

            brand_clean_col_5  brand_clean_col_6  brand_clean_col_7
         0                  0                  0                  0
         1                  0                  0                  0
```

```
[2 rows x 50 columns]
```

Use MinMaxScaler on the feature set to standardize the input data.

```
In [67]: scaler = MinMaxScaler()

         X_scaled = pd.DataFrame(scaler.fit_transform(X_dummies), columns=X_dummies.columns)
         X_scaled.head(2)
```

```
Out[67]:    dateSeen_month  duration_since_dateSeen_month  freeShipping  isBid  isSale  \
         0        0.909091                       0.098324           1.0    0.0     1.0
         1        0.909091                       0.098324           1.0    0.0     0.0

            shoe_cat_score  shoe_name_score  URL_domain_clean_amazon  \
         0        0.333333         0.192687                      0.0
         1        0.333333         0.192687                      0.0

            URL_domain_clean_calvinklein  URL_domain_clean_ebay        ...          \
         0                           0.0                    0.0        ...
         1                           0.0                    0.0        ...

            withTags_with_tags  withTags_without_tags  brand_clean_col_0  \
         0                 0.0                    0.0                1.0
         1                 0.0                    0.0                1.0

            brand_clean_col_1  brand_clean_col_2  brand_clean_col_3  brand_clean_col_4  \
         0                0.0                0.0                0.0                0.0
         1                0.0                0.0                0.0                0.0

            brand_clean_col_5  brand_clean_col_6  brand_clean_col_7
         0                0.0                0.0                0.0
         1                0.0                0.0                0.0

         [2 rows x 50 columns]
```

```
In [68]: y = np.array(y).ravel()
         y.shape
```

```
Out[68]: (18874,)
```

```
In [69]: X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.20, rand
         print('Shape of training set is:', X_train.shape)
         print('Shape of testing set is:', X_test.shape)
```

```
Shape of training set is: (15099, 50)
Shape of testing set is: (3775, 50)
```

6.1.3 Train Linear Regression model

- Use gridsearchcv to try out different ranges of k on the training set.
- **NOTE: There was ultimately no real reason to use GridSearchCV since there were no LinearRegression parameters that I felt needed iterating through. I could've instead used a more basic cross-validation method/function to see how splitting the data affected the mean score (GridSearchCV is a bit overkill in this scenario), but if it was another type of problem - say classification - it may still have come in hand.**

```
In [70]: linear_model = LinearRegression()
         linear_model_params = {}
```

```
In [71]: %%time
         linear_model_cv = GridSearchCV(linear_model, param_grid = linear_model_params, cv=4, v
         linear_model_cv.fit(X_train,y_train)
```

```
Fitting 4 folds for each of 1 candidates, totalling 4 fits
[CV]  ...
[CV] ... , score=0.205, total=   0.0s
[CV]  ...
[CV] ... , score=0.071, total=   0.0s
[CV]  ...
[CV] ... , score=0.271, total=   0.0s
[CV]  ...
[CV] ... , score=0.077, total=   0.0s
CPU times: user 179 ms, sys: 93.9 ms, total: 273 ms
Wall time: 84.7 ms


[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.1s finished
```

```
In [72]: print('The best R2 / variance score is:', linear_model_cv.best_score_)
         print('The best k value is:', linear_model_cv.best_params_)  # overkill for linear re
```

```
The best R2 / variance score is: 0.1560419756604791
The best k value is: {}
```

```
In [73]: # compare model performance to simply using the average of y_train
         y_mean = y_train.mean()
         y_median = np.median(y_train)
         y_mean_predicted = pd.Series(np.tile(y_mean, len(y_test)))

         # compute RMSE and R2
         mse = metrics.mean_squared_error(y_test, y_mean_predicted)
```

```
r2 = metrics.r2_score(y_test, y_mean_predicted)

# printing values
print('Mean squared error: ', mse)
print('Root mean squared error: ', math.sqrt(mse))
print('R2 score: ', r2)
```

```
Mean squared error:  104964.15278670496
Root mean squared error:  323.9817167475735
R2 score:  -1.404270285831899e-06
```

In [74]:
```
# have a quick peek at y_mean_pred vs y_test side-by-side
pd.DataFrame({'y_test_true':y_test,'y_mean_predicted':y_mean_predicted}).head(10)
```

Out[74]:

	y_test_true	y_mean_predicted
0	160.000000	113.032981
1	59.220000	113.032981
2	110.000000	113.032981
3	110.730000	113.032981
4	49.272573	113.032981
5	74.690000	113.032981
6	17.990000	113.032981
7	17.650000	113.032981
8	74.990000	113.032981
9	82.990000	113.032981

As expected, using metrics of central tendency as y-value baseline predictions (mean and median alike) gives an **R2 score near 0**, i.e. your regression is no better than taking the mean value and you are **not using any information from the other variables**.

Furthermore, the mean squared error score (metric used for scoring our linear regression model) is on the order of 10ˆ5, RMSE on the order of 10ˆ2. Since RMSE is measured in the units of the target variables (USD), it represents a dollar amount by which our baseline predictions are off.

At least we can rest easy knowing that the basic linear model, however poor its performance, is still performing better than using just a measure of central tendency! Granted, all signs point to the fact that more work likely needs to be done on the feature engineering front.

6.1.4 Testing Linear model

- Apply the linear model on full test set and evaluate the model performance using evaluation metrics MSE, RMSE and R2.

In [75]:
```
%%time
linear_test = LinearRegression()
linear_test.fit(X_train,y_train)
linear_test_predictions = linear_test.predict(X_test)
```

```
CPU times: user 37.5 ms, sys: 16 ms, total: 53.5 ms
Wall time: 14.6 ms
```

```
In [76]: print("MSE: ",metrics.mean_squared_error(y_test, linear_test_predictions))
         print("RMSE: ",math.sqrt(metrics.mean_squared_error(y_test, linear_test_predictions)))
         print("R2 :",metrics.r2_score(y_test, linear_test_predictions))
```

```
MSE:  95770.81557033962
RMSE:  309.46860191356996
R2 : 0.08758421312594278
```

6.1.5 Results Summary

Unfortunately, it appears the basic linear model **doesn't perform much better than our worse-case measure of central tendency comparison**; notably, R2 is still very low (close to 0, although a bit greater than without the features' information) and the RMSE score is of the same order of magnitued (10ˆ2 in USD).

Though I won't due to time constraints, it's also worth **checking adjusted R2** to make sure the small improvements we're seeing in evaluation metrics (R2 in particular) are simply due to adding new features.

Although I will proceed with trying a more complex supervised regression model, this is **usually a sign we need to re-evaluate our input features + try deriving more valuable features from the data that we have**; or alternatively, **try reframing the task/problem itself** given the data that we have.

7 6.0 Model Implementation (LightGBM)

I will try using a boosting (LightGBM) model for the following reasons: - Efficient on a large datasets (note on this in 'Future Improvements') - Handles categorical variables without the need to encode - Better performance from an evaluation perspective compared to Random Forest and SVM; also less computation intensive than neural networks - There are many parameters that can be tuned to improve evaluation performance

```
In [77]: nlp_name.columns.tolist()
```

```
Out[77]: ['nlp_name_boot',
          'nlp_name_shoe',
          'nlp_name_footwear',
          'nlp_name_sandal',
          'nlp_name_running',
          'nlp_name_sneaker',
          'nlp_name_loafer',
          'nlp_name_slipper',
          'nlp_name_moccasin']
```

```
In [78]: # Note: dateSeen_year removed due to high correlation with dateSeen_month and duratio
         # led to negligeable impact on the model

         numeric_features = ['isSale', 'freeShipping', 'isBid',
                             'duration_since_dateSeen_month','dateSeen_month',
```

```
                         'shoe_cat_score','shoe_name_score']

nlp_cat_subset = ['nlp_cat_boot',
 'nlp_cat_shoe',
 'nlp_cat_sneaker']

nlp_name_subset = ['nlp_name_boot',
 'nlp_name_shoe',
 'nlp_name_sandal',
 'nlp_name_running',
 'nlp_name_sneaker',
 'nlp_name_loafer',
 'nlp_name_slipper',
 'nlp_name_moccasin']

numeric_features = numeric_features + nlp_cat_subset + nlp_name_subset

categorical_features = ['brand_clean', 'gender', 'isNew', 'withTags', 'withBox',
                        'prices.currency', 'URL_domain_clean']

target_feature = ['mean_price_usd']

y = data1[target_feature]
```

In [79]:
```
numerical_data = data1[numeric_features]

scaler = MinMaxScaler()
numerical_data_scaled = pd.DataFrame(scaler.fit_transform(numerical_data), columns = r
```

In [80]:
```
categorical_data = data1[categorical_features]
for col in categorical_data.columns:
     categorical_data[col] = categorical_data[col].astype('category')

print('Converting object variables to type category for lightgbm: ')
categorical_data.dtypes
```

```
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
  This is separate from the ipykernel package so we can avoid doing imports until


Converting object variables to type category for lightgbm:
```

Out[80]:
```
brand_clean          category
gender               category
```

```
isNew                 category
withTags              category
withBox               category
prices.currency       category
URL_domain_clean      category
dtype: object
```

```
In [81]: X_lgbm = pd.concat([numerical_data_scaled.reset_index(drop=True), categorical_data.res
         print('Features selected for lightgbm model based on feature importance and trial and
         X_lgbm.columns
```

```
Features selected for lightgbm model based on feature importance and trial and error:
```

```
Out[81]: Index(['isSale', 'freeShipping', 'isBid', 'duration_since_dateSeen_month',
                'dateSeen_month', 'shoe_cat_score', 'shoe_name_score', 'nlp_cat_boot',
                'nlp_cat_shoe', 'nlp_cat_sneaker', 'nlp_name_boot', 'nlp_name_shoe',
                'nlp_name_sandal', 'nlp_name_running', 'nlp_name_sneaker',
                'nlp_name_loafer', 'nlp_name_slipper', 'nlp_name_moccasin',
                'brand_clean', 'gender', 'isNew', 'withTags', 'withBox',
                'prices.currency', 'URL_domain_clean'],
               dtype='object')
```

```
In [82]: X_data, X_test, y_data, y_test = train_test_split(X_lgbm, y, test_size=0.2, random_sta
         print('Shape of train set is:', X_data.shape)
         print('Shape of testing set is:', X_test.shape)
```

```
Shape of train set is: (15099, 25)
Shape of testing set is: (3775, 25)
```

```
In [83]: def lgbm_model(X_data, y_data, params, n_folds):
             n_folds = KFold(n_splits=n_folds)
             X_data = X_data.reset_index(drop=True)
             y_data = y_data.reset_index(drop=True)



             train_mse = []
             train_rmse = []
             train_r2 = []
             validation_mse = []
             validation_rmse = []
             validation_r2 = []

             for train_index, validate_index in n_folds.split(X_data):

                 X_train, X_validate = X_data.iloc[train_index], X_data.iloc[validate_index]
                 y_train, y_validate = y_data.iloc[train_index], y_data.iloc[validate_index]
```

```
            lgb_model = lgb.LGBMRegressor(**params, n_jobs=5)

            # train the model
            lgb_model.fit(X_train, y_train, early_stopping_rounds=50,
                          eval_set=(X_validate, y_validate), eval_metric='l2')

            # get train rmse and r2
            y_train_pred = lgb_model.predict(X_train, num_iteration=lgb_model.best_iterat

            # get validation rmse and r2
            y_pred = lgb_model.predict(X_validate, num_iteration=lgb_model.best_iteration

            train_mse.append(metrics.mean_squared_error(y_train, y_train_pred))
            train_rmse.append(math.sqrt(metrics.mean_squared_error(y_train, y_train_pred)
            train_r2.append(metrics.r2_score(y_train, y_train_pred))

            validation_mse.append(metrics.mean_squared_error(y_validate, y_pred))
            validation_rmse.append(math.sqrt(metrics.mean_squared_error(y_validate, y_pre
            validation_r2.append(metrics.r2_score(y_validate, y_pred))


        return [pd.DataFrame(train_mse).mean().values[0],
                pd.DataFrame(train_rmse).mean().values[0],
                pd.DataFrame(train_r2).mean().values[0],
                pd.DataFrame(validation_mse).mean().values[0],
                pd.DataFrame(validation_rmse).mean().values[0],
                pd.DataFrame(validation_r2).mean().values[0]], lgb_model
```

In [84]:
```
params = {'learning_rate':0.08,
          'boosting_type': 'gbdt',
          'num_leaves':20,
          'objective':'regression',
          'metric': 'l1',
          'max_depth': 4,
          'max_bin': 50,
          'feature_fraction': 0.5,
          'reg_alpha': 0.5,
          'reg_lambda': 0.5,
          'min_gain_to_split': 0.2,
          'n_estimators': 100,
          'num_iterations': 90,
          'min_split_gain': 0.08,
          'min_child_samples': 5}
```

In [85]:
```
model_eval, result_model = lgbm_model(X_data, y_data,params,  5)
```

```
[1]        valid_0's l2: 39403.5        valid_0's l1: 86.3906
```

```
Training until validation scores don't improve for 50 rounds.
[2]	valid_0's l2: 37833.7	valid_0's l1: 83.8002
[3]	valid_0's l2: 36299.4	valid_0's l1: 81.2633
[4]	valid_0's l2: 35055.7	valid_0's l1: 79.6185
[5]	valid_0's l2: 34138	valid_0's l1: 78.46
[6]	valid_0's l2: 33267.5	valid_0's l1: 77.2665
[7]	valid_0's l2: 32392	valid_0's l1: 75.1701
[8]	valid_0's l2: 31749.7	valid_0's l1: 74.2724
[9]	valid_0's l2: 31261.3	valid_0's l1: 73.2828
[10]	valid_0's l2: 30704.8	valid_0's l1: 72.3628
[11]	valid_0's l2: 30101.2	valid_0's l1: 70.6852
[12]	valid_0's l2: 29558	valid_0's l1: 69.1741
[13]	valid_0's l2: 29171	valid_0's l1: 68.198
[14]	valid_0's l2: 28818.5	valid_0's l1: 67.2476
[15]	valid_0's l2: 28642.3	valid_0's l1: 66.9385
[16]	valid_0's l2: 28447.5	valid_0's l1: 66.3351
[17]	valid_0's l2: 28302.2	valid_0's l1: 65.976
[18]	valid_0's l2: 28103	valid_0's l1: 65.3637
[19]	valid_0's l2: 27864.4	valid_0's l1: 64.6103
[20]	valid_0's l2: 27628.3	valid_0's l1: 64.012
[21]	valid_0's l2: 27418.8	valid_0's l1: 63.4101
[22]	valid_0's l2: 27380.1	valid_0's l1: 63.3069
[23]	valid_0's l2: 27226.1	valid_0's l1: 62.7823
[24]	valid_0's l2: 27072.9	valid_0's l1: 62.5697
[25]	valid_0's l2: 26901.2	valid_0's l1: 62.3202
[26]	valid_0's l2: 26777.1	valid_0's l1: 61.9807
[27]	valid_0's l2: 26722.3	valid_0's l1: 61.837
[28]	valid_0's l2: 26628.6	valid_0's l1: 61.582
[29]	valid_0's l2: 26560.1	valid_0's l1: 61.4005
[30]	valid_0's l2: 26513.9	valid_0's l1: 61.2665
[31]	valid_0's l2: 26445.8	valid_0's l1: 61.0992
[32]	valid_0's l2: 26411.1	valid_0's l1: 60.8439
[33]	valid_0's l2: 26394.6	valid_0's l1: 60.7637
[34]	valid_0's l2: 26348.3	valid_0's l1: 60.5823
[35]	valid_0's l2: 26301.2	valid_0's l1: 60.4341
[36]	valid_0's l2: 26228.2	valid_0's l1: 60.1348
[37]	valid_0's l2: 26176.8	valid_0's l1: 59.9299
[38]	valid_0's l2: 26128.7	valid_0's l1: 59.6945
[39]	valid_0's l2: 26074.4	valid_0's l1: 59.5054
[40]	valid_0's l2: 26081.4	valid_0's l1: 59.4722
[41]	valid_0's l2: 26045.1	valid_0's l1: 59.3395
[42]	valid_0's l2: 26007.9	valid_0's l1: 59.1907
[43]	valid_0's l2: 25979.7	valid_0's l1: 59.0821
[44]	valid_0's l2: 25954	valid_0's l1: 58.9777
[45]	valid_0's l2: 25916.2	valid_0's l1: 58.8466
[46]	valid_0's l2: 25899.3	valid_0's l1: 58.738
[47]	valid_0's l2: 25927.4	valid_0's l1: 58.6148
[48]	valid_0's l2: 25915.8	valid_0's l1: 58.5201
```

```
[49]	valid_0's l2: 25923	valid_0's l1: 58.5147
[50]	valid_0's l2: 25903.3	valid_0's l1: 58.4177
[51]	valid_0's l2: 25888.6	valid_0's l1: 58.3402
[52]	valid_0's l2: 25873.6	valid_0's l1: 58.2381
[53]	valid_0's l2: 25903.1	valid_0's l1: 58.2372
[54]	valid_0's l2: 25878.3	valid_0's l1: 58.11
[55]	valid_0's l2: 25871	valid_0's l1: 58.0461
[56]	valid_0's l2: 25847.9	valid_0's l1: 57.8845
[57]	valid_0's l2: 25832.4	valid_0's l1: 57.7872
[58]	valid_0's l2: 25812.2	valid_0's l1: 57.6656
[59]	valid_0's l2: 25801.6	valid_0's l1: 57.6354
[60]	valid_0's l2: 25782.8	valid_0's l1: 57.5686
[61]	valid_0's l2: 25773.8	valid_0's l1: 57.5294
[62]	valid_0's l2: 25795.4	valid_0's l1: 57.495
[63]	valid_0's l2: 25775.1	valid_0's l1: 57.3613
[64]	valid_0's l2: 25797.2	valid_0's l1: 57.35
[65]	valid_0's l2: 25790.3	valid_0's l1: 57.2414
[66]	valid_0's l2: 25811.7	valid_0's l1: 57.2305
[67]	valid_0's l2: 25869	valid_0's l1: 57.2777
[68]	valid_0's l2: 25856.3	valid_0's l1: 57.2048
[69]	valid_0's l2: 25840.4	valid_0's l1: 57.1627
[70]	valid_0's l2: 25826.8	valid_0's l1: 57.1193
[71]	valid_0's l2: 25811.4	valid_0's l1: 57.0477
[72]	valid_0's l2: 25803.9	valid_0's l1: 57.0059
[73]	valid_0's l2: 25834.8	valid_0's l1: 57.0282
[74]	valid_0's l2: 25831.2	valid_0's l1: 57.0123
[75]	valid_0's l2: 25816.6	valid_0's l1: 56.9837
[76]	valid_0's l2: 25802	valid_0's l1: 56.9171
[77]	valid_0's l2: 25788.8	valid_0's l1: 56.8434
[78]	valid_0's l2: 25751.8	valid_0's l1: 56.8064
[79]	valid_0's l2: 25746	valid_0's l1: 56.7877
[80]	valid_0's l2: 25733.7	valid_0's l1: 56.7273
[81]	valid_0's l2: 25727.6	valid_0's l1: 56.6834
[82]	valid_0's l2: 25714.2	valid_0's l1: 56.6507
[83]	valid_0's l2: 25704.4	valid_0's l1: 56.5988
[84]	valid_0's l2: 25703.9	valid_0's l1: 56.5755
[85]	valid_0's l2: 25697.1	valid_0's l1: 56.5508
[86]	valid_0's l2: 25690	valid_0's l1: 56.4977
[87]	valid_0's l2: 25683.9	valid_0's l1: 56.4767
[88]	valid_0's l2: 25675.9	valid_0's l1: 56.4303
[89]	valid_0's l2: 25669.9	valid_0's l1: 56.3944
[90]	valid_0's l2: 25666	valid_0's l1: 56.3639
Did not meet early stopping. Best iteration is:
[90]	valid_0's l2: 25666	valid_0's l1: 56.3639
```

```
/anaconda3/lib/python3.7/site-packages/lightgbm/engine.py:118: UserWarning: Found `num_iteratio
  warnings.warn("Found `{}` in params. Will use it instead of argument".format(alias))
```

```
/anaconda3/lib/python3.7/site-packages/lightgbm/engine.py:118: UserWarning: Found `num_iteratio
  warnings.warn("Found `{}` in params. Will use it instead of argument".format(alias))
```

```
[1]	valid_0's l2: 117195	valid_0's l1: 88.9688
Training until validation scores don't improve for 50 rounds.
[2]	valid_0's l2: 115830	valid_0's l1: 86.5413
[3]	valid_0's l2: 114322	valid_0's l1: 83.9777
[4]	valid_0's l2: 113159	valid_0's l1: 82.3151
[5]	valid_0's l2: 112191	valid_0's l1: 81.122
[6]	valid_0's l2: 111418	valid_0's l1: 80.0316
[7]	valid_0's l2: 110619	valid_0's l1: 77.9809
[8]	valid_0's l2: 109908	valid_0's l1: 76.9983
[9]	valid_0's l2: 109358	valid_0's l1: 75.9764
[10]	valid_0's l2: 108802	valid_0's l1: 74.9903
[11]	valid_0's l2: 108272	valid_0's l1: 73.3814
[12]	valid_0's l2: 107779	valid_0's l1: 72.0775
[13]	valid_0's l2: 107362	valid_0's l1: 71.2055
[14]	valid_0's l2: 107037	valid_0's l1: 70.1954
[15]	valid_0's l2: 106662	valid_0's l1: 69.7819
[16]	valid_0's l2: 106423	valid_0's l1: 69.0633
[17]	valid_0's l2: 106119	valid_0's l1: 68.694
[18]	valid_0's l2: 105879	valid_0's l1: 67.9948
[19]	valid_0's l2: 105691	valid_0's l1: 67.2697
[20]	valid_0's l2: 105477	valid_0's l1: 66.653
[21]	valid_0's l2: 105294	valid_0's l1: 66.0782
[22]	valid_0's l2: 105138	valid_0's l1: 65.9248
[23]	valid_0's l2: 105004	valid_0's l1: 65.4203
[24]	valid_0's l2: 104830	valid_0's l1: 65.0985
[25]	valid_0's l2: 104670	valid_0's l1: 64.7622
[26]	valid_0's l2: 104552	valid_0's l1: 64.3157
[27]	valid_0's l2: 104417	valid_0's l1: 64.0382
[28]	valid_0's l2: 104272	valid_0's l1: 63.7261
[29]	valid_0's l2: 104198	valid_0's l1: 63.5494
[30]	valid_0's l2: 104138	valid_0's l1: 63.3609
[31]	valid_0's l2: 104031	valid_0's l1: 63.1323
[32]	valid_0's l2: 103948	valid_0's l1: 62.7927
[33]	valid_0's l2: 103716	valid_0's l1: 62.7462
[34]	valid_0's l2: 103658	valid_0's l1: 62.539
[35]	valid_0's l2: 103588	valid_0's l1: 62.3468
[36]	valid_0's l2: 103525	valid_0's l1: 61.9963
[37]	valid_0's l2: 103481	valid_0's l1: 61.7257
[38]	valid_0's l2: 103428	valid_0's l1: 61.4617
[39]	valid_0's l2: 103389	valid_0's l1: 61.2943
[40]	valid_0's l2: 103347	valid_0's l1: 61.2145
[41]	valid_0's l2: 103317	valid_0's l1: 61.0673
[42]	valid_0's l2: 103271	valid_0's l1: 60.9092
[43]	valid_0's l2: 103234	valid_0's l1: 60.768
```

```
[44]	valid_0's l2: 103222	valid_0's l1: 60.6892
[45]	valid_0's l2: 103190	valid_0's l1: 60.5751
[46]	valid_0's l2: 103165	valid_0's l1: 60.4569
[47]	valid_0's l2: 102997	valid_0's l1: 60.3727
[48]	valid_0's l2: 102984	valid_0's l1: 60.2767
[49]	valid_0's l2: 102975	valid_0's l1: 60.2482
[50]	valid_0's l2: 102943	valid_0's l1: 60.1248
[51]	valid_0's l2: 102877	valid_0's l1: 60.07
[52]	valid_0's l2: 102872	valid_0's l1: 59.9455
[53]	valid_0's l2: 102832	valid_0's l1: 59.9383
[54]	valid_0's l2: 102826	valid_0's l1: 59.8933
[55]	valid_0's l2: 102813	valid_0's l1: 59.7941
[56]	valid_0's l2: 102800	valid_0's l1: 59.6851
[57]	valid_0's l2: 102770	valid_0's l1: 59.5544
[58]	valid_0's l2: 102735	valid_0's l1: 59.4557
[59]	valid_0's l2: 102698	valid_0's l1: 59.4111
[60]	valid_0's l2: 102630	valid_0's l1: 59.377
[61]	valid_0's l2: 102615	valid_0's l1: 59.3471
[62]	valid_0's l2: 102611	valid_0's l1: 59.2866
[63]	valid_0's l2: 102537	valid_0's l1: 59.0907
[64]	valid_0's l2: 102515	valid_0's l1: 59.0713
[65]	valid_0's l2: 102475	valid_0's l1: 58.9417
[66]	valid_0's l2: 102467	valid_0's l1: 58.8839
[67]	valid_0's l2: 102462	valid_0's l1: 58.8127
[68]	valid_0's l2: 102438	valid_0's l1: 58.7114
[69]	valid_0's l2: 102416	valid_0's l1: 58.6706
[70]	valid_0's l2: 102369	valid_0's l1: 58.6384
[71]	valid_0's l2: 102350	valid_0's l1: 58.5883
[72]	valid_0's l2: 102291	valid_0's l1: 58.5544
[73]	valid_0's l2: 102276	valid_0's l1: 58.5176
[74]	valid_0's l2: 102270	valid_0's l1: 58.4222
[75]	valid_0's l2: 102262	valid_0's l1: 58.3906
[76]	valid_0's l2: 102120	valid_0's l1: 58.3972
[77]	valid_0's l2: 102104	valid_0's l1: 58.3334
[78]	valid_0's l2: 102098	valid_0's l1: 58.2571
[79]	valid_0's l2: 102086	valid_0's l1: 58.2021
[80]	valid_0's l2: 102069	valid_0's l1: 58.1334
[81]	valid_0's l2: 102054	valid_0's l1: 58.0666
[82]	valid_0's l2: 102042	valid_0's l1: 58.0223
[83]	valid_0's l2: 102017	valid_0's l1: 58.0033
[84]	valid_0's l2: 102009	valid_0's l1: 57.9829
[85]	valid_0's l2: 101988	valid_0's l1: 57.9491
[86]	valid_0's l2: 101984	valid_0's l1: 57.9291
[87]	valid_0's l2: 101976	valid_0's l1: 57.8909
[88]	valid_0's l2: 101980	valid_0's l1: 57.9179
[89]	valid_0's l2: 101982	valid_0's l1: 57.8453
[90]	valid_0's l2: 101970	valid_0's l1: 57.8121
Did not meet early stopping. Best iteration is:
```

```
[90]        valid_0's l2: 101970       valid_0's l1: 57.8121
[1]        valid_0's l2: 29981.5       valid_0's l1: 81.6252
Training until validation scores don't improve for 50 rounds.
[2]        valid_0's l2: 28753.6       valid_0's l1: 78.9882
[3]        valid_0's l2: 27549.8       valid_0's l1: 76.5825
[4]        valid_0's l2: 26605.6       valid_0's l1: 75.0281
[5]        valid_0's l2: 25913.3       valid_0's l1: 73.9985
[6]        valid_0's l2: 25181.1       valid_0's l1: 73.0122
[7]        valid_0's l2: 24424       valid_0's l1: 70.9432
[8]        valid_0's l2: 23927.3       valid_0's l1: 70.2026
[9]        valid_0's l2: 23403.8       valid_0's l1: 69.3796
[10]        valid_0's l2: 22955.8       valid_0's l1: 68.4613
[11]        valid_0's l2: 22484.8       valid_0's l1: 66.9474
[12]        valid_0's l2: 22095.7       valid_0's l1: 65.7493
[13]        valid_0's l2: 21753       valid_0's l1: 64.8116
[14]        valid_0's l2: 21454.7       valid_0's l1: 63.819
[15]        valid_0's l2: 21328       valid_0's l1: 63.613
[16]        valid_0's l2: 21112.8       valid_0's l1: 62.8831
[17]        valid_0's l2: 21013.5       valid_0's l1: 62.6332
[18]        valid_0's l2: 20793.1       valid_0's l1: 61.9384
[19]        valid_0's l2: 20610.9       valid_0's l1: 61.294
[20]        valid_0's l2: 20456.1       valid_0's l1: 60.7655
[21]        valid_0's l2: 20278.3       valid_0's l1: 60.1708
[22]        valid_0's l2: 20234.9       valid_0's l1: 60.1244
[23]        valid_0's l2: 20111.2       valid_0's l1: 59.6372
[24]        valid_0's l2: 19968.2       valid_0's l1: 59.3043
[25]        valid_0's l2: 19826.6       valid_0's l1: 58.9368
[26]        valid_0's l2: 19731.8       valid_0's l1: 58.5692
[27]        valid_0's l2: 19661.1       valid_0's l1: 58.3891
[28]        valid_0's l2: 19621.2       valid_0's l1: 58.191
[29]        valid_0's l2: 19572.1       valid_0's l1: 58.0341
[30]        valid_0's l2: 19544.7       valid_0's l1: 57.927
```

```
/anaconda3/lib/python3.7/site-packages/lightgbm/engine.py:118: UserWarning: Found `num_iteratio
  warnings.warn("Found `{}` in params. Will use it instead of argument".format(alias))
/anaconda3/lib/python3.7/site-packages/lightgbm/engine.py:118: UserWarning: Found `num_iteratio
  warnings.warn("Found `{}` in params. Will use it instead of argument".format(alias))
```

```
[31]        valid_0's l2: 19480       valid_0's l1: 57.7295
[32]        valid_0's l2: 19407.8       valid_0's l1: 57.4764
[33]        valid_0's l2: 19377.9       valid_0's l1: 57.542
[34]        valid_0's l2: 19338.9       valid_0's l1: 57.3846
[35]        valid_0's l2: 19282       valid_0's l1: 57.1718
[36]        valid_0's l2: 19211.6       valid_0's l1: 56.8603
[37]        valid_0's l2: 19163.8       valid_0's l1: 56.6155
[38]        valid_0's l2: 19116.6       valid_0's l1: 56.3982
```

```
[39]	valid_0's l2: 19084.1	valid_0's l1: 56.2535
[40]	valid_0's l2: 19096.8	valid_0's l1: 56.2189
[41]	valid_0's l2: 19079.6	valid_0's l1: 56.1215
[42]	valid_0's l2: 19038.5	valid_0's l1: 55.9586
[43]	valid_0's l2: 19006.2	valid_0's l1: 55.8479
[44]	valid_0's l2: 18987.8	valid_0's l1: 55.7297
[45]	valid_0's l2: 18966.5	valid_0's l1: 55.6579
[46]	valid_0's l2: 18955.2	valid_0's l1: 55.5718
[47]	valid_0's l2: 18934	valid_0's l1: 55.476
[48]	valid_0's l2: 18921.7	valid_0's l1: 55.4187
[49]	valid_0's l2: 18912.5	valid_0's l1: 55.3407
[50]	valid_0's l2: 18883.7	valid_0's l1: 55.2195
[51]	valid_0's l2: 18903	valid_0's l1: 55.271
[52]	valid_0's l2: 18887.5	valid_0's l1: 55.1602
[53]	valid_0's l2: 18900.9	valid_0's l1: 55.2413
[54]	valid_0's l2: 18878.2	valid_0's l1: 55.1542
[55]	valid_0's l2: 18866.6	valid_0's l1: 55.0818
[56]	valid_0's l2: 18838	valid_0's l1: 54.9349
[57]	valid_0's l2: 18823.8	valid_0's l1: 54.8522
[58]	valid_0's l2: 18814.3	valid_0's l1: 54.7644
[59]	valid_0's l2: 18814.9	valid_0's l1: 54.7501
[60]	valid_0's l2: 18812.5	valid_0's l1: 54.7281
[61]	valid_0's l2: 18805.4	valid_0's l1: 54.7197
[62]	valid_0's l2: 18790.8	valid_0's l1: 54.6899
[63]	valid_0's l2: 18749.3	valid_0's l1: 54.5614
[64]	valid_0's l2: 18739.9	valid_0's l1: 54.5014
[65]	valid_0's l2: 18712.7	valid_0's l1: 54.3791
[66]	valid_0's l2: 18707.4	valid_0's l1: 54.3178
[67]	valid_0's l2: 18918.2	valid_0's l1: 54.4743
[68]	valid_0's l2: 18914	valid_0's l1: 54.4538
[69]	valid_0's l2: 18904.1	valid_0's l1: 54.3812
[70]	valid_0's l2: 18911	valid_0's l1: 54.3855
[71]	valid_0's l2: 18898.2	valid_0's l1: 54.317
[72]	valid_0's l2: 18893.2	valid_0's l1: 54.2916
[73]	valid_0's l2: 18911.3	valid_0's l1: 54.313
[74]	valid_0's l2: 18908.2	valid_0's l1: 54.2932
[75]	valid_0's l2: 18980.4	valid_0's l1: 54.3615
[76]	valid_0's l2: 18975.7	valid_0's l1: 54.3363
[77]	valid_0's l2: 18961.4	valid_0's l1: 54.2976
[78]	valid_0's l2: 18955	valid_0's l1: 54.2481
[79]	valid_0's l2: 18997.7	valid_0's l1: 54.3619
[80]	valid_0's l2: 18983.3	valid_0's l1: 54.3105
[81]	valid_0's l2: 18980.6	valid_0's l1: 54.2766
[82]	valid_0's l2: 18979.2	valid_0's l1: 54.2669
[83]	valid_0's l2: 18970.2	valid_0's l1: 54.2211
[84]	valid_0's l2: 18964.5	valid_0's l1: 54.1828
[85]	valid_0's l2: 18959.8	valid_0's l1: 54.1507
[86]	valid_0's l2: 18963.6	valid_0's l1: 54.1319
```

```
[87]	valid_0's l2: 18957.4	valid_0's l1: 54.1187
[88]	valid_0's l2: 18959.4	valid_0's l1: 54.0854
[89]	valid_0's l2: 19026.1	valid_0's l1: 54.1243
[90]	valid_0's l2: 19022.5	valid_0's l1: 54.1001
Did not meet early stopping. Best iteration is:
[66]	valid_0's l2: 18707.4	valid_0's l1: 54.3178
[1]	valid_0's l2: 41752.2	valid_0's l1: 85.6906
Training until validation scores don't improve for 50 rounds.
[2]	valid_0's l2: 40345.8	valid_0's l1: 83.1183
[3]	valid_0's l2: 39013.7	valid_0's l1: 80.5966
[4]	valid_0's l2: 37954.3	valid_0's l1: 78.9442
[5]	valid_0's l2: 37141.6	valid_0's l1: 77.7644
[6]	valid_0's l2: 36194.5	valid_0's l1: 76.5962
[7]	valid_0's l2: 35401	valid_0's l1: 74.5304
[8]	valid_0's l2: 34734.9	valid_0's l1: 73.5535
[9]	valid_0's l2: 34026.9	valid_0's l1: 72.5069
[10]	valid_0's l2: 33567.1	valid_0's l1: 71.4836
[11]	valid_0's l2: 33011.5	valid_0's l1: 69.9816
[12]	valid_0's l2: 32579.2	valid_0's l1: 68.7463
[13]	valid_0's l2: 32187.8	valid_0's l1: 67.7977
[14]	valid_0's l2: 31823.4	valid_0's l1: 66.7604
[15]	valid_0's l2: 31627.1	valid_0's l1: 66.5051
[16]	valid_0's l2: 31391	valid_0's l1: 65.8094
[17]	valid_0's l2: 31223.8	valid_0's l1: 65.4631
[18]	valid_0's l2: 30997.6	valid_0's l1: 64.7727
[19]	valid_0's l2: 30785.9	valid_0's l1: 64.0169
[20]	valid_0's l2: 30619.4	valid_0's l1: 63.5559
[21]	valid_0's l2: 30470.8	valid_0's l1: 62.9804
[22]	valid_0's l2: 30339.7	valid_0's l1: 62.7967
[23]	valid_0's l2: 30199.9	valid_0's l1: 62.2812
[24]	valid_0's l2: 30080	valid_0's l1: 62.0171
[25]	valid_0's l2: 29972.1	valid_0's l1: 61.8242
[26]	valid_0's l2: 29897.4	valid_0's l1: 61.4284
[27]	valid_0's l2: 29842.6	valid_0's l1: 61.242
[28]	valid_0's l2: 29786.5	valid_0's l1: 61.0057
[29]	valid_0's l2: 29730.2	valid_0's l1: 60.8515
[30]	valid_0's l2: 29689.7	valid_0's l1: 60.7213
[31]	valid_0's l2: 29643.4	valid_0's l1: 60.6032
[32]	valid_0's l2: 29582.6	valid_0's l1: 60.3093
[33]	valid_0's l2: 29482.9	valid_0's l1: 60.2474
[34]	valid_0's l2: 29457.9	valid_0's l1: 60.1152
[35]	valid_0's l2: 29418.1	valid_0's l1: 59.9809
[36]	valid_0's l2: 29365.4	valid_0's l1: 59.7827
[37]	valid_0's l2: 29321.6	valid_0's l1: 59.5878
[38]	valid_0's l2: 29269.8	valid_0's l1: 59.3585
[39]	valid_0's l2: 29251.4	valid_0's l1: 59.214
[40]	valid_0's l2: 29260.9	valid_0's l1: 59.1861
[41]	valid_0's l2: 29243.5	valid_0's l1: 59.0823
```

```
[42]	valid_0's l2: 29212.2	valid_0's l1: 58.9476
[43]	valid_0's l2: 29189.6	valid_0's l1: 58.7556
[44]	valid_0's l2: 29148.4	valid_0's l1: 58.6155
[45]	valid_0's l2: 29119.8	valid_0's l1: 58.5287
[46]	valid_0's l2: 29097	valid_0's l1: 58.4442
[47]	valid_0's l2: 29096.5	valid_0's l1: 58.347
[48]	valid_0's l2: 29071.2	valid_0's l1: 58.2065
[49]	valid_0's l2: 29061.2	valid_0's l1: 58.159
[50]	valid_0's l2: 29035.7	valid_0's l1: 58.0624
[51]	valid_0's l2: 29009.9	valid_0's l1: 58.0142
[52]	valid_0's l2: 29019.7	valid_0's l1: 57.9443
[53]	valid_0's l2: 29008.8	valid_0's l1: 57.9722
[54]	valid_0's l2: 29001.9	valid_0's l1: 57.906
[55]	valid_0's l2: 28983.5	valid_0's l1: 57.8046
[56]	valid_0's l2: 28954.8	valid_0's l1: 57.6935
[57]	valid_0's l2: 28940.1	valid_0's l1: 57.6274
[58]	valid_0's l2: 28908.7	valid_0's l1: 57.5442
[59]	valid_0's l2: 28884.5	valid_0's l1: 57.4833
[60]	valid_0's l2: 28883.2	valid_0's l1: 57.4409
[61]	valid_0's l2: 28877.7	valid_0's l1: 57.466
[62]	valid_0's l2: 28879.2	valid_0's l1: 57.4117
[63]	valid_0's l2: 28850.2	valid_0's l1: 57.3158
[64]	valid_0's l2: 28834.8	valid_0's l1: 57.2416
[65]	valid_0's l2: 28830.3	valid_0's l1: 57.1923
[66]	valid_0's l2: 28816	valid_0's l1: 57.131
[67]	valid_0's l2: 28836.2	valid_0's l1: 57.12
[68]	valid_0's l2: 28824.9	valid_0's l1: 57.0742
[69]	valid_0's l2: 28794.2	valid_0's l1: 57.0454
[70]	valid_0's l2: 28784.2	valid_0's l1: 57.0421
[71]	valid_0's l2: 28781.3	valid_0's l1: 57.0357
[72]	valid_0's l2: 28763.5	valid_0's l1: 57.0064
[73]	valid_0's l2: 28756.8	valid_0's l1: 56.9854
[74]	valid_0's l2: 28735.2	valid_0's l1: 56.9617
[75]	valid_0's l2: 28725.8	valid_0's l1: 56.9685
[76]	valid_0's l2: 28690.9	valid_0's l1: 56.9552
[77]	valid_0's l2: 28683.9	valid_0's l1: 56.9091
[78]	valid_0's l2: 28685.3	valid_0's l1: 56.9083
[79]	valid_0's l2: 28665.8	valid_0's l1: 56.8626
[80]	valid_0's l2: 28650.4	valid_0's l1: 56.8077
[81]	valid_0's l2: 28645.5	valid_0's l1: 56.7785
[82]	valid_0's l2: 28637.6	valid_0's l1: 56.7421
[83]	valid_0's l2: 28619.7	valid_0's l1: 56.7259
[84]	valid_0's l2: 28616.3	valid_0's l1: 56.7169
[85]	valid_0's l2: 28600.4	valid_0's l1: 56.6804
[86]	valid_0's l2: 28589.7	valid_0's l1: 56.6199
[87]	valid_0's l2: 28584.6	valid_0's l1: 56.6002
[88]	valid_0's l2: 28563.9	valid_0's l1: 56.5225
[89]	valid_0's l2: 28549	valid_0's l1: 56.4659
```

```
[90]        valid_0's l2: 28540.8       valid_0's l1: 56.4127
Did not meet early stopping. Best iteration is:
[90]        valid_0's l2: 28540.8       valid_0's l1: 56.4127
[1]        valid_0's l2: 121648       valid_0's l1: 88.8191
Training until validation scores don't improve for 50 rounds.
[2]        valid_0's l2: 120039       valid_0's l1: 86.1705
[3]        valid_0's l2: 118553       valid_0's l1: 83.6374
[4]        valid_0's l2: 117149       valid_0's l1: 81.8754
[5]        valid_0's l2: 115988       valid_0's l1: 80.205
[6]        valid_0's l2: 115064       valid_0's l1: 78.8404
[7]        valid_0's l2: 114218       valid_0's l1: 76.7041
[8]        valid_0's l2: 113543       valid_0's l1: 75.7138
[9]        valid_0's l2: 112950       valid_0's l1: 74.5385
[10]        valid_0's l2: 112352       valid_0's l1: 73.4136
[11]        valid_0's l2: 111795       valid_0's l1: 71.7969
[12]        valid_0's l2: 111335       valid_0's l1: 70.6119
[13]        valid_0's l2: 110800       valid_0's l1: 69.5058
[14]        valid_0's l2: 110338       valid_0's l1: 68.4691
[15]        valid_0's l2: 110045       valid_0's l1: 67.9732
[16]        valid_0's l2: 109821       valid_0's l1: 67.288
[17]        valid_0's l2: 109581       valid_0's l1: 66.8752
[18]        valid_0's l2: 109281       valid_0's l1: 66.1579
[19]        valid_0's l2: 109097       valid_0's l1: 65.5323
[20]        valid_0's l2: 108840       valid_0's l1: 64.9738
[21]        valid_0's l2: 108697       valid_0's l1: 64.4101
[22]        valid_0's l2: 108466       valid_0's l1: 64.1669
[23]        valid_0's l2: 108348       valid_0's l1: 63.6795
[24]        valid_0's l2: 108131       valid_0's l1: 63.3258
[25]        valid_0's l2: 107972       valid_0's l1: 63.0708
[26]        valid_0's l2: 107914       valid_0's l1: 62.7983
[27]        valid_0's l2: 107721       valid_0's l1: 62.569
[28]        valid_0's l2: 107436       valid_0's l1: 62.2431
[29]        valid_0's l2: 107345       valid_0's l1: 62.0238
[30]        valid_0's l2: 107285       valid_0's l1: 61.892
[31]        valid_0's l2: 107218       valid_0's l1: 61.716
[32]        valid_0's l2: 107129       valid_0's l1: 61.4247
[33]        valid_0's l2: 106880       valid_0's l1: 61.313
[34]        valid_0's l2: 106801       valid_0's l1: 61.0407
[35]        valid_0's l2: 106724       valid_0's l1: 60.8532
[36]        valid_0's l2: 106689       valid_0's l1: 60.6233
[37]        valid_0's l2: 106647       valid_0's l1: 60.4282
[38]        valid_0's l2: 106595       valid_0's l1: 60.2063
[39]        valid_0's l2: 106559       valid_0's l1: 60.0964
[40]        valid_0's l2: 106522       valid_0's l1: 59.9813
[41]        valid_0's l2: 106484       valid_0's l1: 59.8037
[42]        valid_0's l2: 106444       valid_0's l1: 59.7034
[43]        valid_0's l2: 106426       valid_0's l1: 59.609
[44]        valid_0's l2: 106403       valid_0's l1: 59.4786
```

```
[45]	valid_0's l2: 106377	valid_0's l1: 59.3857
[46]	valid_0's l2: 106355	valid_0's l1: 59.2843
[47]	valid_0's l2: 106334	valid_0's l1: 59.2039
[48]	valid_0's l2: 106325	valid_0's l1: 59.1402
[49]	valid_0's l2: 106306	valid_0's l1: 59.0641
[50]	valid_0's l2: 106287	valid_0's l1: 59.0063
[51]	valid_0's l2: 106219	valid_0's l1: 58.9772
[52]	valid_0's l2: 106204	valid_0's l1: 58.9381
[53]	valid_0's l2: 106055	valid_0's l1: 58.8929
[54]	valid_0's l2: 106047	valid_0's l1: 58.8287
[55]	valid_0's l2: 106035	valid_0's l1: 58.8088
[56]	valid_0's l2: 106020	valid_0's l1: 58.7031
[57]	valid_0's l2: 106010	valid_0's l1: 58.6497
[58]	valid_0's l2: 105989	valid_0's l1: 58.5447
[59]	valid_0's l2: 105976	valid_0's l1: 58.5213
[60]	valid_0's l2: 105959	valid_0's l1: 58.472
[61]	valid_0's l2: 105946	valid_0's l1: 58.4487
[62]	valid_0's l2: 105936	valid_0's l1: 58.4153
[63]	valid_0's l2: 105903	valid_0's l1: 58.3075
[64]	valid_0's l2: 105894	valid_0's l1: 58.2923
[65]	valid_0's l2: 105862	valid_0's l1: 58.1982
[66]	valid_0's l2: 105809	valid_0's l1: 58.1576
[67]	valid_0's l2: 105804	valid_0's l1: 58.1249
[68]	valid_0's l2: 105787	valid_0's l1: 58.0687
[69]	valid_0's l2: 105772	valid_0's l1: 58.0297
[70]	valid_0's l2: 105755	valid_0's l1: 58.0248
[71]	valid_0's l2: 105743	valid_0's l1: 57.9718
[72]	valid_0's l2: 105735	valid_0's l1: 57.9357
[73]	valid_0's l2: 105718	valid_0's l1: 57.8431
[74]	valid_0's l2: 105711	valid_0's l1: 57.7893
[75]	valid_0's l2: 105703	valid_0's l1: 57.7716
[76]	valid_0's l2: 105700	valid_0's l1: 57.7289
[77]	valid_0's l2: 105698	valid_0's l1: 57.6906
[78]	valid_0's l2: 105689	valid_0's l1: 57.6534
[79]	valid_0's l2: 105674	valid_0's l1: 57.6026
[80]	valid_0's l2: 105580	valid_0's l1: 57.595
[81]	valid_0's l2: 105571	valid_0's l1: 57.5783
[82]	valid_0's l2: 105576	valid_0's l1: 57.5816
[83]	valid_0's l2: 105571	valid_0's l1: 57.5457
[84]	valid_0's l2: 105565	valid_0's l1: 57.5299
[85]	valid_0's l2: 105550	valid_0's l1: 57.4917
[86]	valid_0's l2: 105544	valid_0's l1: 57.4686
[87]	valid_0's l2: 105505	valid_0's l1: 57.4586
[88]	valid_0's l2: 105457	valid_0's l1: 57.426
[89]	valid_0's l2: 105412	valid_0's l1: 57.4114
[90]	valid_0's l2: 105405	valid_0's l1: 57.3677
Did not meet early stopping. Best iteration is:
[90]	valid_0's l2: 105405	valid_0's l1: 57.3677
```

```
/anaconda3/lib/python3.7/site-packages/lightgbm/engine.py:118: UserWarning: Found `num_iteratio
  warnings.warn("Found `{}` in params. Will use it instead of argument".format(alias))
```

```
In [86]: print("Train MSE:",model_eval[0])
         print("Train RMSE:",model_eval[1])
         print("Train R2:",model_eval[2])
         print()
         print("Validation MSE:",model_eval[3])
         print("Validation RMSE:",model_eval[4])
         print("Validation R2:",model_eval[5])
```

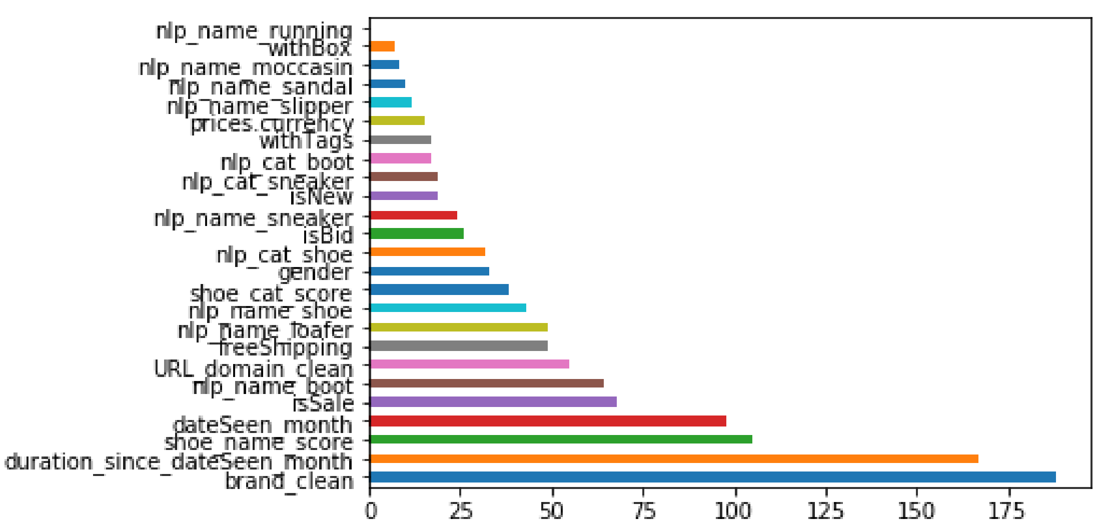
```
Train MSE: 54152.192870355546
Train RMSE: 231.83767626733402
Train R2: 0.24969104636289474

Validation MSE: 56057.96978601177
Validation RMSE: 221.9822457375664
Validation R2: 0.28469013569772733
```

```
In [87]: print('Feature importance barplot:')
         (pd.Series(result_model.feature_importances_, index=X_lgbm.columns)
            .nlargest(40)
            .plot(kind='barh'))
```

```
Feature importance barplot:
```

```
Out[87]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1fd0d898>
```



```
In [88]: y_pred = result_model.predict(X_test, num_iteration=result_model.best_iteration_)
         print("Test MSE:",metrics.mean_squared_error(y_test, y_pred))
         print("Test RMSE:",math.sqrt(metrics.mean_squared_error(y_test, y_pred)))
         print("Test R2:",metrics.r2_score(y_test, y_pred))
```

```
Test MSE: 88788.67273620391
Test RMSE: 297.9742820046789
Test R2: 0.15410361478433254
```

7.0.1 Model Implementation Summary

In summary, the performance of the LightGBM model as implemented **did not yield the low error (MSE, RMSE)/high explanation of variance (R2 score) result** that an applicant like myself might have hoped for. It did demonstrate a *small improvement in performance over the baseline model (linear regression model)* and the *worse-case scenario analysis (measure of central tendency comparison)* in the test fitting.

Train, validation and test set MSE, RMSE and R2 are (54152, 231 USD, 25%), (56058, 222 USD, 29%) and (88788, 298, 15%)

Listed below are some of the pros and cons of the LightGBM mode as currently implemented:

Pros

- The test set predictions vs true test values' R2 score explained a meager **7%** more of the variance in the data than the test set using the linear model (and that's before considering R2). That's also **15%** more than the worse case scenario (R2 of ~0).
- Similarly, the RMSE of the LightGBM model's performance was reduced by **11.50 USD** from the RMSE of the linear model's performance (noting that RMSE is in USD, the units of the target variable). Given that the average value of the target variable (y_test) is **~113 USD**, that's a favorable dollar value decrease in error with respect to the average price of test set products - however, it is nowhere near the dollar value of RMSE error.
- (Similar results apply for MSE, however the units are in USD squared - arguably less intuitive if there's a need for explainability to a stakeholder or client.)
- As discussed in 'Cons', I was not able to programatically optimize the hyperparameter tuning of the model (say, using Bayesian Optimization). This can also be a pro as it means the performance of the model can still improve with additional hyperparameter tweaking.
- Although LightGBM typically performs better on larger datasets than the one at hand (say, 100K+ rows of data points), the model still demonstrated a minute increase with respect to the baseline model (both the linear and worse case).

Cons

- The model is clearly overfitting the training data despite my attempts to tweak the regularization parameters in LightGBM (L1, L2 costs). This is arguably the biggest problem with the model at the moment - due in part to using such a complex supervised learning model that can be prone to overfitting. Perhaps comparing the current evaluation metrics to those of other techniques (ensemble methods, for example) would be worth a shot. On the training and validation sets, the model can perform as well as explaining 50% of the variance in the data (be careful, needs to be sanity-checked with adjusted_R2) and brings the RMSE down to the order of 10^1 USD. For those values, the model did not generalize to similar evaluation metrics on the test set, however.

- Additional hyperparameter tuning is needed to fully leverage the power of the current modelling technique + improve its performance (discussed in 'Future Improvements' as well).

- The performance is still not ideal given that the order of magnitude of test set RMSE is on par with that of the average target variable value. Although the **R2 score already doesn't look immediately appealing, the RMSE score confirms that the dollar-value error is too large for this model to confidently predict men's shoe prices to a reasonable confidence level.**

- Furthermore, given more time, I would also evaluate **adjusted_R2**; which is a **function of the number of independent variables in the model**. It's possible that the percentage of variance in the data explained by the model will decrease once the number of independent features are accounted for. That being said, the current model does not suffer from worryingly high dimensionality, so it would likely not decrease dramatically in explained variance.

7.0.2 Conclusion

Ultimately, I would not advise using this model for men's shoe prices' predictions in production in its current implementation.

8 Future Improvements

1. Additional **data cleaning** for missing and erroneous values
2. Free shipping and free returns currently lumped together
3. **Predicting any currency price** as opposed to **predicting USD-converted mean price**
4. Improve (currently rudimentary) **NLP modelling**
5. **Increase dataset size** + supplement with 3rd party data; additional (clean) data points should theoretically improve LightGBM's evaluation metric performance
6. More explicit **hyperparameter tuning (Bayesian Optimization)**
7. Better leverage of the **prices.offer column** for discount data.

1. I know for a fact that there are still non-men's shoe products contained within the dataset that may or may not be skewing the distribution of prices + introducing biases. Additional cleaning is required, in particular in the name and categories columns - to be explicit, removing rows/data points (products in particular) whose category is not part of men's shoes. (Recall the apriori assumption that we would include all men's footwear in this analysis; mocassins, slippers, etc). This might in turn improve the NLP performance of the nlp_cat_score and nlp_name_score columns.

2. I would hypothesize that this improvement would have little impact on the model's performance overall, but free shipping and free returns are currently being encoded together in the same categorical column. The suggested improvement would be to make separate encodings for each field: increases dimensionality and sparseness (possibly for little payoff), but worth revisiting. Low priority improvement.

3. Currently, the model is training and testing on prices converted entirely to USD. It could be worth seeing how the model performs keeping the mean prices in their original listing currencies (granted, it would introduce additional complications - like assessing RMSE of a target variable that is theoretically in different units). Would likely involve having to reformulate the problem itself.

4. The NLP modelling of the name and category columns completed so far has been very basic given the time constraint of the project, but already some of the feature-engineered columns demonstrate high importance in the model's performance. With a more thorough NLP assessment of those two columns (and other high cardinality ordinal columns like brand, among others), I believe NLP-based "binning" of the products will greatly improve the model's ability to discern product segmentation with respect to price.

5. LightGBM is a complex supervised learning model that usually works best when supplemented with large amounts of data (due to the bias-variance tradeoff, there is an inherent danger of overfitting the training set). Acquiring more data points for the model **may** help with performance - emphasis is on help since, as I previously mentioned, the model would benefit most from either re-engineering the input features, or reframing the target of the problem. Furthermore, we can try supplementing the existing data with external 3rd party data. For example, we can join Amazon Product Advertising API data on the ASIN column (granted, there is a lot of missing data in the column currently), which could add additional features like SalesRank for measuring the relative popularity of a product on Amazon's platform to other products in its 'category'. My recommendation would still be to improve the model performance first (by improving the feature engineering process) before trying to throw more data at it.

6. More explicit hyperparameter tuning (which would benefit in optimization performance from the increase in dataset row-size from 5). A common method utilized in tandem with LightGBM for parameter optimization is Bayesian Optimization, a technique I did not have time to implement given the time constraint. It may be worth trying in the future to see if it improves performance at all and solves the issue of overfitting the training set.

7. I hypothesize that better leveraging the *'prices.offer'* column of the dataset could lead to a significant improvement in performance via additional feature engineering that I was unable to tap into given the assignment time constraint. As far as columns of Strings go in the dataset, it is arguably one of the best formatted and easiest to extract valuable information from using simple string manipulation. I should have prioritized feature engineering from this column over some of the other categorical encoding variables that I spent time on (like withTags which ended up having a low feature importance, etc). Given more time, I would attempt to extract columns for a USD dollar amount of discount and percentage discount respectively using string manipulations (maybe NLP if necessary, but don't think it would be at first sight).