Predicting Resale Value of Knives from a Texas Government Surplus Store

Using Machine Learning to Support an Ebay Store's Financial Success

Data Obtainment Notebook

This notebook displays the code used to collect and process data from eBay using two of eBay's public APIs and scraping from their proprietary webabb "Terapeak".

Author: Dylan Dey

Overview

Texas State Surplus Store (https://www.tfc.texas.gov/divisions/supportserv/prog/statesurplus/)

What happens to all those items that get confiscated by the TSA? Some end up in a Texas store. (https://www.wfaa.com/article/news/local/what-happens-to-all-those-items-that-get-confiscated-by-the-tsa-some-end-up-in-a-texas-store/287-ba80dac3-d91a-4b28-952a-0aaf4f69ff95)

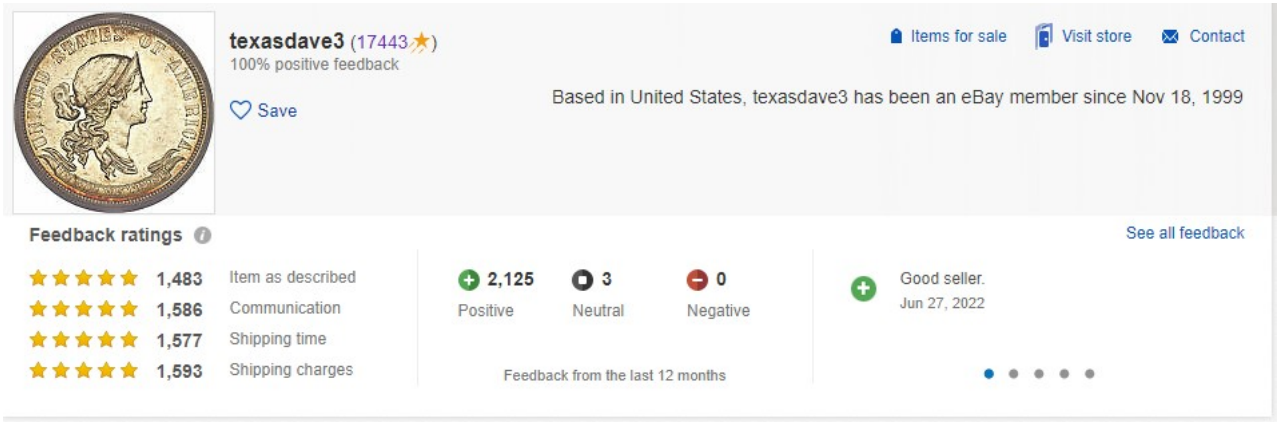
Texas Surplus Store PDF (https://www.tfc.texas.gov/divisions/supportserv/prog/statesurplus/State%20Surplus%20Brochure-one%20bar_rev%201-10-2022.pdf)

Everything that doesn't make it through Texas airports can be found at one Austin store (https://cbsaustin.com/news/local/everything-that-doesnt-make-it-through-texas-airports-can-be-found-at-one-austin-store)

The Texas Facilities Commission collects left behind possessions, salvage, and surplus from Texas state agencies such as DPS, TXDOT, TCEQ, and Texas Parks & Wildlife. Examples of commonly available items include vehicles, furniture, office equipment and supplies, small electronics, and heavy equipment. The goal of this project is to create a predictive model in order to determine the resale value of knivse from the Texas State Surplus Store on eBay.

Business Problem

Family Ebay Store Front (https://www.ebay.com/str/texasdave3?mkcid=16&mkevt=1&mkrid=711-127632-2357-0&ssspo=ZW3G27tGR_m&sssrc=3418065&ssuid=&widget_ver=artemis&media=COPY)



Texas Dave's Knives (https://www.ebay.com/str/texasdave3/Knives/_i.html?store_cat=3393246519)

While taking online courses to transition careers during a difficult time of my life, I was also helping my family during a turbulent time for everyone. I have been employed at their retail store in San Antonio for the past several months and have been contributing significantly to their online reselling business on eBay. I would help source newer, cheaper products from Austin to try and resell at the retail store in San Antonio or online to earn some money, support our family business. This is how I discovered the Texas State Surplus Store.

My family has been running a resale shop and selling on Ebay and other sites for years and lately the business has picked up. Consumer behavior is shifting: getting a deal on eBay, or Goodwill, or hitting up a vintage boutique shop to find a unique treasure is now brag worthy. Plus, people like the idea of sustainability - sending items to landfills is becoming very socially unacceptable – why not repurpose a used item? With the pandemic related disruption of “normal” business and supply chains and the economic uncertainty of these times there is definitely an upswing in interest in the resale market.

Online sales sites like Ebay offer a worldwide robust buyer base for just about every product regardless of condition. Ebay allows the reseller to find both bargain hunters for common items and enthusiasts searching for rare collectible items.

An Ebay business has some pain points, however. Selection of an item to sell is the main pain point. The item should be readily available in decent condition for the seller to purchase at a low price but not so widely available that the market is saturated with that item. Then there needs to be a demand for the item – it should be something collectible that with appeal to hobbyists that would pay premium prices for hard-to-get items. Alternatively, it would be something useful to a large number of people even in a used condition. The item should be small enough to be easily shipped. It should not be difficult to ship either—that is it should not have hazardous chemicals, batteries etc. that would add costs to the shipping. Additionally, Ebay has strict rules about authentication and certification in many item categories- so obvious “high value” items like jewelry or designer purses are so restricted that it is not feasible for the average Ebay seller to offer them .

This project recommends an item that would answer these concerns – pocket knives, These can be rare and collectible and also practical and useful. There are knife collector forums and subReddits, showing there is an interest among collectors. A look at eBay listings shows rare knives selling for thousands of dollars each. Knives are also a handy every day tool – and based on the number showing up in the Texas Surplus shop they are easy to lose and so need replacing often. This means there is a market for more common ones as well. The great thing about single blade, modern, factory manufactured pocketknives is that they all weigh roughly 0.5 lbs making them cheap to ship. For my modeling purposes, it is safe to assume a flat shipping rate of 4.95(US Dollars) including the cost of wholesale purchased padded envelopes. And there are no restrictions on mailing these items and they are not fragile so no special packaging is needed.

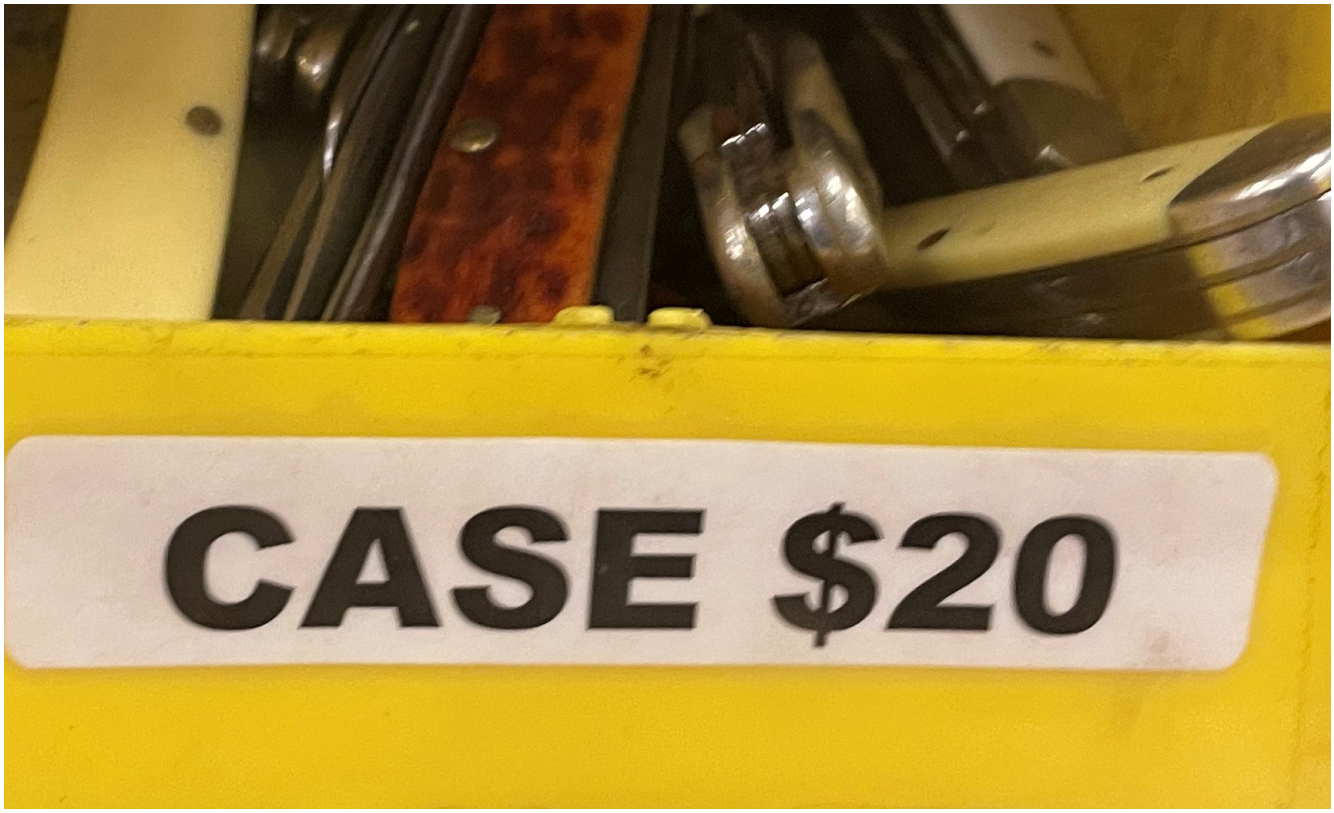
The second pain point is buying at a cost low enough to make a profit. It is not enough to just buy low and sell at a higher price as expenses need to be considered. Ebay collects insertion fees and final value fees on all sales. The fees vary with seller level (rating) and some portions are a percent of final sale. I have been selling knives from the lower priced bins and the mean seller fee for my sales so far is about 13.5% of the sold price. So that is a cost to consider right up front.

A third pain point is the cost of excess inventory. A seller can obtain quality items at a reasonable cost and then the inventory may sit with no sales, meaning the capital expended is sitting tied up in unwanted items. This inventory carry cost is a drain on profitability. This project is meant to help avoid purchasing the wrong items for resale.

As already mentioned, I have been experimenting with low cost used knives for resale but have not risked a large capital investment in the higher end items. The goal of this project is to attempt to address the pain points to determine if a larger investment would pay off. Can I identify which knives are worth investing in so that I can turn a decent profit and hopefully avoid excess inventory? A data driven approach would help avoid costly mistakes from the "system" resellers currently employ, which seems to be mainly a gambler's approach. By managing resources upfront through a model, I can effectively increase my return on investment with messy data such as pictures and titles. The magic of Neural Networks!

There are eight buckets of presorted brand knives that I was interested in, specifically. These bins are behind glass, presorted, branded(and therefore have specific characteristics and logos for my model to identify), and priced higher. However, the staff has a very large amount of confiscated items flowing into the facility to list for resale, and when that happens they will not have time to preset them and they end up in huge buckets of unsorted knives for people to dig through. The brands will be priced the same, they are just no longer sorted and harder to find. This particular scenario is where a NN could really shine to help add more inventory to our Ebay website without risking more money or spending extra time than simply digging through the presorted bins everytime. Expanding the bins to pull inventory from will increase the chance of finding inventory worth reselling.

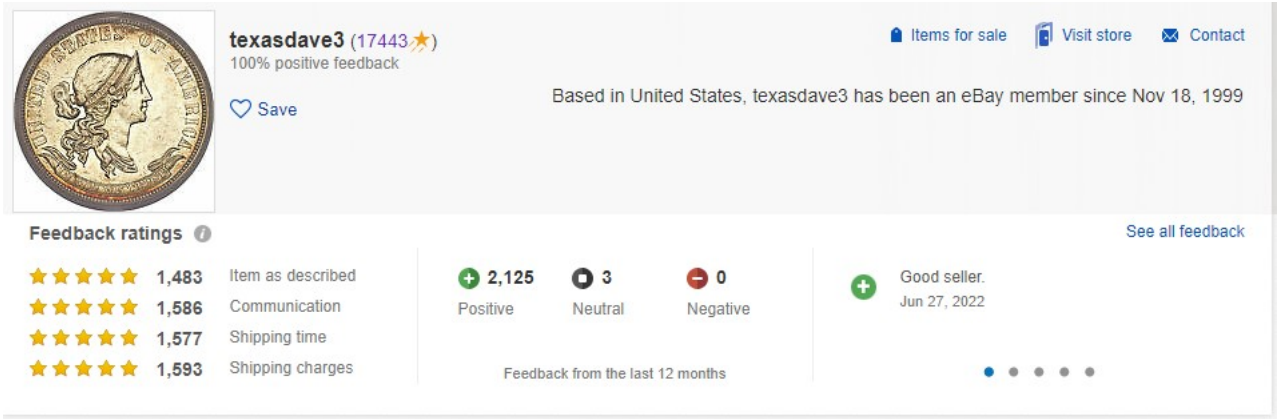
sorted bucket example



overflow example

Data Understanding

Family Ebay Store Front (https://www.ebay.com/str/texasdave3?mkcid=16&mkevt=1&mkrid=711-127632-2357-0&ssspo=ZW3G27tGR_m&sssrc=3418065&ssuid=&widget_ver=artemis&media=COPY)



Texas Dave's Knives (https://www.ebay.com/str/texasdave3/Knives/_i.html?store_cat=3393246519)

> There are eight buckets of presorted brand knives that I was interested in exploring from the Texas Surplus Store. These bins are behind glass, presorted, branded(and therefore have specific characteristics and logos for my model to identify), and priced higher. However, the staff has a very large amount of confiscated items flowing into the facility to list for resale, and when that happens they will not have time to preset them and they end up in huge buckets of unsorted knives for people to dig through. The brands will be priced the same, they are just no longer sorted and harder to find. This particular scenario is where a NN could really shine to help add more inventory to our Ebay website without risking more money or spending extra time than simply digging through the presorted bins everytime. Expanding the bins to pull inventory from will increase the chance of finding inventory worth reselling.

The Eight Pocketknife brands and their associated cost at the Texas Surplus Store:

- Benchmade: $45.00
- Buck: $20.00
- Case/Casexx: $20.00
- CRKT: $15.00
- Kershaw: $15.00
- SOG: $15.00
- Spyderco: $30.00
- Victorinox: $20.00

Ebay Developer Website (https://developer.ebay.com/)

Ebay has a website for developers to create an account and register an application keyset in order to make API call requests to their live website. By making a findItemsAdvanced call to the eBay Finding APIVersion 1.13.0, I was able to get a large dataset of category_id=<48818> (https://www.ebay.com/sch/48818/i.html?_from=R40&_nkw=knife) knives listed for sale. This data is limited to anything listed within the past 90 days from when the API call was made.

When you log into Ebay as a buyer and search knife in the search bar, the response that loads outputs Knives, Swords & Blades. Nested one category furtheris Collectible Folding Knives with an id of 182981. Nested one further is Modern Folding Knives(43333), and then finally, the category_id of most interest, 48818, Factory Manufactured Modern Collectible Folding Knives.

The eBay Finding APIVersion 1.13.0 findItemsAdvanced (https://developer.ebay.com/devzone/finding/callref/finditemsadvanced.html) call returns a lot of usefull information about listings, including itemId(a unique identifier for ebay listings),price, shipping price, area code, the title of the listing, the url for the listing, whether the seller set autoPay for the listing or whether the seller is a top rated seller or not, the condition of the item being sold, whether the seller accepts returns, and various links to images of the item being sold at different resolutions. If you look at a typical eBay listing, however, there is usually more minute information available that is required to be filled out by the seller upon posting the listing. To get this information, another API must be used that accepts the itemId of listings to return more details.

The eBay Shopping APIVersion 1247 GetMultipleItems (https://developer.ebay.com/Devzone/shopping/docs/CallRef/GetMultipleItems.html) call accepts itemIds and returns seller authored details on the item for sale in their listing. This was used to get information such as of the model or product line for the knife being listed, blade material, blade type, blade edge type, color, the number of blades, opening mechanism, handle material, lock type, and blade range.

All of the data gathered from eBay's public API is limited to listed data posted in the past 90 days and doesn't include a "sold" price. Sold data is locked behind eBay's proprietary webapp, known as Terapeak. Data on this pay to play webapp has an option for sold data that goes back 2 years! Therefore, gaining access to this webapp and scraping all relevant pages proved to be very valuable and bypasses the limits of the free API. I used my reletavely new eBay seller's account to sign up for a free trial of terapeak and scraped useful data for sold, used knives of the 8 relevant brands. Information scraped includes Images, titles, price sold, shipping cost.

A majority of the data was scraped from eBays proprietary Terapeak webapp, as this data goes back 2 years as compared to the API listed data that only goes back 90 days. It is assumed a large enough amount of listed data should approximate sold data well enough to prove useful for this project.

The target feature for the model to predict is the total price (shipping included) that a knife should be listed on eBay. One model will be using titles and images in order to find potential listings that are undervalued and could be worth investing in. Another model will accept only images as input, as this is an input that can easily be obtained in person at the store. This model will use past sold data of knives on eBay in order to determine within an acceptable amount of error the price it will resale for on eBay (shipping included) using only an image.

Domain Understading: Cost Breakdown

- padded envelopes: $0.50 per knife
- flatrate shipping: $4.45 per knife
- brand knife at surplus store: 15, 20, 30, or 45 dollars per knife
- overhead expenses (gas, cleaning suplies, sharpening supplies, etc): $3.00
- Ebay's comission, with 13% being a reasonable approximation

Data Obtainment

'Ebay FindingService', '1.12.0', 'findItemsAdvanced', 'eBaySDK/2.2.0 Python/3.8.5 Windows/10'

Ebay suggested SDKs on ebay developer website (https://developer.ebay.com/develop/ebay-sdks)

Python SDK to simplify making calls (https://github.com/timotheus/ebaysdk-python/wiki/Trading-API-Class)

eBay Finding APIVersion 1.13.0 call index (https://developer.ebay.com/devzone/finding/CallRef/index.html)

finditemsAdvanced Call Reference (https://developer.ebay.com/devzone/finding/CallRef/findItemsAdvanced.html)

The Ebay developer website suggests using an SDK in order to make a call to their APIs. I decided to git clone the Python SDK to simplify making calls (https://github.com/timotheus/ebaysdk-python/wiki/Trading-API-Class) and used the .yaml file from the github repository in order to store all of my necessary developer/security keys. Please feel free to read through the documentation in the github and the documentation in the API reference to see what all is available using this SDK and API.

the API limits you to the first 100 pages of whatever response you recieve from a request.

Ebay offers people with a basic seller subscription and above to access their research tools. Terapeak product research allows "targeted insights about markets you're interested in, simply search by keyword or product, and apply filters such as Listing type, Start price, Buyer country, and Time of day." Terapeak (https://www.ebay.com/help/selling/selling-tools/terapeak-research?id=4853)

The Terapeak product research website allows access to the sale price of products that go back 2 years. The Terapeak product research website was filtered for USED Factory Manufactured Modern Collectible Folding Knives (category_id 48818) sold prices for all 8 knife models of interest in the last 2 years.

In [1]:

```
from ebaysdk.finding import Connection
import requests
from ebaysdk.shopping import Connection as Shopping
import pandas as pd
import  json
import numpy as np
import re
# import preprocess_ddey117 as pp
import matplotlib.pyplot as plt
%matplotlib inline
from PIL import Image
import ast

import seaborn as sns

pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)
```

Define Necessary Functions

In [2]:

```
#This function is a helper function created for the "knife_request" below.
#It unpacks some of the nested data from eBay API calls
#It also creates the new feature "converted_price"
#"converted_price" is the price of the item for sale plus shipping cost.
def prepare_df(df):
    price_list = []
    ship_price_list = []
    condition_list = []
    condition = None
    for row in full_dataset:
        listed_price = float(row['sellingStatus']['convertedCurrentPrice']['value'])
        price_list.append(listed_price)

        try:
            listed_ship_price = float(row['shippingInfo']['shippingServiceCost']['value'])
            ship_price_list.append(listed_ship_price)
        except:
            listed_ship_price = 0
            ship_price_list.append(listed_ship_price)

        try:
            condition = float(row['condition']['conditionId'])
            condition_list.append(condition)
        except:
            conditon = 0
            condition_list.append(condition)

    df['shipping_cost'] = ship_price_list
    df['price_in_US'] = price_list
    df['condition'] = condition_list

    #create new feature 'converted price'
    df['converted_price'] = df['shipping_cost'] + df['price_in_US']
    df.drop_duplicates(subset=['itemId'],  keep='first', inplace=True)
    df.reset_index(drop=True, inplace=True)

    return df

#dictionary for preparing brands
bucket_dict = {'benchmade': 45.0,
               'buck': 20.0,
               'case': 20.0,
               'crkt': 15.0,
               'kershaw': 15.0,
               'sog': 15.0,
               'spyderco': 30.0,
               'victorinox': 20.0
              }

#a helper function used with knife_request
#it is used to create new columns of interest
#the brand of knife from the API call
#the cost of the knife from the Surplus Store
#profit for reselling a used surplus knife on eBay
#Return on Investment for reselling the knife
#All columns in US dollars
def prepare_brands(df, bucket_dict_position, overhead_cost=3):

    df.title = df.title.apply(str.lower)

    #remove special characters
#     df.title.apply(pp.remove_special_chars)
    df['brand'] = str(list(bucket_dict.keys())[bucket_dict_position])
    df['cost'] = float(list(bucket_dict.values())[bucket_dict_position]+4.95+overhead_cost)
    df['profit'] = ((df['converted_price']*.87) -  df['cost'])
    df['ROI'] = (df['profit']/( df['cost']))*100.0

    return df
# Help organize paginated data from API calls
def prepare_data(data_list):
    """
    This function takes in a list of dictionaries and prepares it
    for analysis
    """

    # Make a new list to hold results
    results = []

    for business_data in data_list:

        # Make a new dictionary to hold prepared data for this business
        prepared_data = {}

        # Extract name, review_count, rating, and price key-value pairs
        # from business_data and add to prepared_data
        # If a key is not present in business_data, add it to prepared_data
```

```
        # with an associated value of None

        keys = ['itemId', 'title', 'galleryURL',
                'viewItemURL', 'autoPay', 'postalCode',
                'sellingStatus', 'shippingInfo', 'listingInfo',
                'returnsAccepted', 'condition', 'topRatedListing',
                'galleryPlusPictureURL','pictureURLLarge',
                'pictureURLSuperSize']

        for key in keys:
            prepared_data[key] = business_data.get(key, None)
            results.append(prepared_data)


        # Add to list if all values are present
#         if all(prepared_data.values()):
#             results.append(prepared_data)


    return results
#main function for making findingAPI calls to eBay
def knife_request(Brand, dict_pos):
    api = Connection(config_file='ebay.yaml', debug=False, siteid="EBAY-US")
    #first request gets number of pages from paginationOutput of first page
    request = {
                'categoryId': 48818,
                'itemFilter': [
                                {'name': 'ListingType', 'value': 'FixedPrice'}
                              ],
                'aspectFilter': [
                                  {'aspectName': 'Brand', 'aspectValueName': Brand}],

                'outputSelector': ['PictureURLLarge', 'PictureURLSuperSize'],


                'paginationInput': {
                                    'entriesPerPage': 100,
                                    'pageNumber': 1

                                    },

                }

    #     request['paginationInput']['pageNumber'] = page

    response = api.execute('findItemsAdvanced', request)


    response_pages = response.dict()

    full_dataset = []

    total_pages = int(response_pages['paginationOutput']['totalPages'])

    if total_pages > 100:
        pages_to_request = 100

    else:
        pages_to_request = total_pages - 1
        #subtract number of pages by one to avoid errors

    #Loop through available pages
    for page in range(1, pages_to_request):
        # Add or update the "offset" key-value pair in url_params

        # Make the query and get the response

        api = Connection(config_file='ebay.yaml', debug=False, siteid="EBAY-US")

        request = {
                'categoryId': 48818,
                'itemFilter': [
                                {'name': 'ListingType', 'value': 'FixedPrice'}
                              ],
                'aspectFilter': [
                                  {'aspectName': 'Brand', 'aspectValueName': Brand}],

                'outputSelector': ['PictureURLLarge', 'PictureURLSuperSize'],


                'paginationInput': {
                                    'entriesPerPage': 100,
                                    'pageNumber': page

                                    },

```

```
                }


        response = api.execute('findItemsAdvanced', request)

        #save the response as a json dict
        response_dict = response.dict()


        #index dict to appropriate index
        results_list_of_dicts = response_dict['searchResult']['item']

        # Call the prepare_data function to get a list of processed data
        prepared_knives = prepare_data(results_list_of_dicts)

        # Extend full_dataset with this list (don't append, or you'll get
        # a list of lists instead of a flat list)
        full_dataset.extend(prepared_knives)

    # Check the length of the full dataset. It will be up to `total`,
    # potentially less if there were missing values
    display(len(full_dataset))

    df = pd.DataFrame(full_dataset)

    df = prepare_df(df)

    df = prepare_brands(df, dict_pos)

    return df
#Used to prepare data from eBays shopping API
#Shopping API used to collect more detailed info
#about individual knives
def prepare_dataIds(data_list):
    """
    This function takes in a list of dictionaries and prepares it
    for analysis
    """

    # Make a new list to hold results
    results = []

    for business_data in data_list:

        # Make a new dictionary to hold prepared data for this business
        prepared_data = {}

        # Extract name, review_count, rating, and price key-value pairs
        # from business_data and add to prepared_data
        # If a key is not present in business_data, add it to prepared_data
        # with an associated value of None

        keys = ['ItemID','GalleryURL','PictureURL',
                'Location','ConvertedCurrentPrice',
                'Title','ItemSpecifics',
                'Country','ConditionID']

        for key in keys:
            prepared_data[key] = business_data.get(key, None)
            results.append(prepared_data)


        # Add to list if all values are present
#         if all(prepared_data.values()):
#             results.append(prepared_data)


    return results
#Shopping api accepts a max of 20 itemIDs
#this function was created to automate
#making API calls in 20 unique itemId chuncks
def process_list(my_list):

    api = Shopping(config_file='ebay.yaml', debug=False, siteid="EBAY-US")
    request = {
                'itemID': my_list,
                'IncludeSelector': 'ItemSpecifics'
               }
    response = api.execute('GetMultipleItems', request)



    #save the response as a json dict
    response_dict = response.dict()
```

```
259
260     #index dict to appropriate index
261     results_list_of_dicts = response_dict['Item']
262
263     # Call the prepare_data function to get a list of processed data
264     prepared_knives = prepare_dataIds(results_list_of_dicts)
265
266     # Extend full_dataset with this list (don't append, or you'll get
267     # a list of lists instead of a flat list)
268     full_dataset.extend(prepared_knives)
269
270     return full_dataset
271
272 bucket_dict = {'benchmade': 45.0,
273                'buck': 20.0,
274                'case': 20.0,
275                'crkt': 15.0,
276                'kershaw': 15.0,
277                'sog': 15.0,
278                'spyderco': 30.0,
279                'victorinox': 20.0
280               }
281 #special function for reformatting terapeak scraped data
282 #x = position of bucket_dictionary
283 def prepare_tera_df(df, x, overhead_cost=3):
284     df['price_in_US'] = df['price_in_US'].str.replace("$", "")
285     df['price_in_US'] = df['price_in_US'].str.replace(",", "")
286     df['price_in_US'] = df['price_in_US'].apply(float)
287
288     df['shipping_cost'] = df['shipping_cost'].str.replace("$", "")
289     df['shipping_cost'] = df['shipping_cost'].str.replace(",", "")
290     df['shipping_cost'] = df['shipping_cost'].apply(float)
291
292     df['brand'] = list(bucket_dict.keys())[x]
293     df['converted_price'] = (df['price_in_US'] + df['shipping_cost'])
294     df['cost'] = list(bucket_dict.values())[x] + overhead_cost + 4.95
295     df['profit'] = ((df['converted_price']*.87) -  df['cost'])
296     df['ROI'] = (df['profit']/ df['cost'])*100.0
297
298     return df
299
300 # helper function with "transform_item_specifics"
301 def fix(col):
302     dd = dict()
303     for d in col:
304         values = list(d.values())
305         if len(values) == 2:
306             dd[values[0]] = values[1]
307     return dd
308
309 #function for extracted item Specifics from Shopping API data
310 def transform_item_specifics(df, perc=65.0):
311
312     df.dropna(subset=['ItemSpecifics'], inplace=True)
313     df['ItemSpecifics'] = df['ItemSpecifics'].apply(lambda x: ast.literal_eval(x))
314     df['item_list'] = df['ItemSpecifics'].apply(lambda x: x['NameValueList'])
315
316     df['ItemSpecifics'] = df['ItemSpecifics'].apply(lambda x: [x['NameValueList']] if isinstance(x['NameValueList']
317
318     df['ItemSpecifics'] = df['ItemSpecifics'].apply(fix)
319
320     df = pd.json_normalize(df['ItemSpecifics'])
321
322     min_count =  int(((100-perc)/100)*df.shape[0] + 1)
323     mod_df = df.dropna(axis=1,
324                        thresh=min_count)
325
326     return mod_df
327
328 # This function removes noisy data
329 #lots/sets/groups of knives can
330 #confuse the model from predicting
331 #the appropriate value of individual knives
332 def data_cleaner(df):
333     lot = re.compile('(?<!-\S)lot(?![^\s.,:?!])')
334     group = re.compile('(group)')
335     is_set = re.compile('(?<!-\S)set(?![^\s.,?!])')
336     df['title'] = df['title'].str.lower()
337     trim_list = [lot,group,is_set]
338     for item in trim_list:
339         df.loc[df['title'].apply(lambda x: re.search(item, x)).notnull(), 'trim'] = 1
340     to_drop = df.loc[df['trim'] == 1].index
341     df.drop(to_drop, inplace=True)
342     df.drop('trim', axis=1, inplace=True)
343
344     return df
```

In [3]:
```
1 bucket_dict
```

Out[3]:
```
{'benchmade': 45.0,
 'buck': 20.0,
 'case': 20.0,
 'crkt': 15.0,
 'kershaw': 15.0,
 'sog': 15.0,
 'spyderco': 30.0,
 'victorinox': 20.0}
```

Beginning of API calls for listed data. To be merged with item specific data using ebay itemIds.

Listed Data

Running functions to call the Finding API and return datasets for cat () knives for sale listed on ebay in the last 90 days. (explain how ebay rules work)

```
bench_df = knife_request('Benchmade', 0)
buck_df = knife_request('Buck', 1)
case_df = knife_request('Case', 2)
df_caseXX = knife_request('Case XX', 2)
df_crkt = knife_request("CRKT", 3)
df_sog = knife_request('SOG', 5)
df_spyderco = knife_request('Spyderco', 6)
```

```
bench_df.to_csv('listed_data/df_bench1.csv', index=False)
buck_df.to_csv('listed_data/df_buck.csv', index=False)
case_df.to_csv('listed_data/df_case.csv', index=False)
df_caseXX.to_csv('listed_data/df_CaseXX.csv', index=False)
df_crkt.to_csv('listed_data/df_crkt.csv', index=False)
df_sog.to_csv('listed_data/df_sog.csv', index=False)
df_spyderco.to_csv('listed_data/df_spyderco.csv', index=False)
```

Kershaw and victorinox data was requested using the FindingAPI below after tweaking some pagination through trial and error to maximize data.

```
full_dataset = []
for page in range(1, 57):
#         # Add or update the "offset" key-value pair in url_params

#         # Make the query and get the response

    api = Connection(config_file='ebay.yaml', debug=False, siteid="EBAY-US")

    request = {
                'categoryId': 48818,
                'itemFilter': [
                                {'name': 'ListingType', 'value': 'FixedPrice'}
                              ],
                'aspectFilter': [
                                  {'aspectName': 'Brand', 'aspectValueName': 'Kershaw'}],

                'outputSelector': ['PictureURLLarge', 'PictureURLSuperSize'],


                'paginationInput': {
                                    'entriesPerPage': 100,
                                    'pageNumber': page

                                    },

                }

        #     request['paginationInput']['pageNumber'] = page

    response = api.execute('findItemsAdvanced', request)

    #save the response as a json dict
    response_dict = response.dict()

    #index dict to appropriate index
    results_list_of_dicts = response_dict['searchResult']['item']

    # Call the prepare_data function to get a list of processed data
    prepared_knives = prepare_data(results_list_of_dicts)

    # Extend full_dataset with this list (don't append, or you'll get
    # a list of lists instead of a flat list)
    full_dataset.extend(prepared_knives)

    # Check the length of the full dataset. It will be up to `total`,
    # potentially less if there were missing values

    df = pd.DataFrame(full_dataset)

df_kershaw = prepare_df(df)
df_kershaw = prepare_brands(df_kershaw, 4)
df_kershaw.to_csv('listed_data/df_kershaw.csv', index=False)
```

```
full_dataset = []
for page in range(1, 86):

    api = Connection(config_file='ebay.yaml', debug=False, siteid="EBAY-US")

    request = {
                'categoryId': 48818,
                'itemFilter': [
                                {'name': 'ListingType', 'value': 'FixedPrice'}
                              ],
                'aspectFilter': [
                                  {'aspectName': 'Brand', 'aspectValueName': 'Victorinox'}],

                'outputSelector': ['PictureURLLarge', 'PictureURLSuperSize'],


                'paginationInput': {
                                    'entriesPerPage': 100,
                                    'pageNumber': page

                                    },

                }

    response = api.execute('findItemsAdvanced', request)

    response_dict = response.dict()

    results_list_of_dicts = response_dict['searchResult']['item']

    prepared_knives = prepare_data(results_list_of_dicts)

    full_dataset.extend(prepared_knives)

df_victorinox = pd.DataFrame(full_dataset)
df_victorinox = prepare_df(df_victorinox)
df_victorinox = prepare_brands(df_victorinox, 7)
df_victorinox.to_csv('listed_data/df_victorinox.csv', index=False)
```

start of API call section using IDs from preview listed datasets to get Item Specific data from ebay. This will return more descriptive information about the knives, pulling from a container on the website that sellers must complete to post a listing.

In [4]:

```
df_bench = pd.read_csv("listed_data/df_bench.csv")
df_buck = pd.read_csv("listed_data/df_buck.csv")
df_case = pd.read_csv("listed_data/df_case.csv")
df_caseXX = pd.read_csv("listed_data/df_CaseXX.csv")
df_crkt = pd.read_csv("listed_data/df_crkt.csv")
df_kersh = pd.read_csv("listed_data/df_kershaw.csv")
df_sog = pd.read_csv("listed_data/df_sog.csv")
df_spyd = pd.read_csv("listed_data/df_spyderco.csv")
df_vict = pd.read_csv("listed_data/df_victorinox.csv")


df_list = [df_bench,df_buck,
           df_case,df_caseXX,
           df_crkt,df_kersh,
           df_sog,df_spyd,
           df_vict]

for dataframe in df_list:
    dataframe.drop('galleryPlusPictureURL', axis=1, inplace=True)



benchIds = df_bench.itemId.values.tolist()
buckIds = df_buck.itemId.values.tolist()
caseIds = df_case.itemId.values.tolist()
caseXXIds = df_caseXX.itemId.values.tolist()
crktIds = df_crkt.itemId.values.tolist()
kershawIds = df_kersh.itemId.values.tolist()
sogIds = df_sog.itemId.values.tolist()
spydIds = df_spyd.itemId.values.tolist()
victIds = df_vict.itemId.values.tolist()
```

ShoppingAPI call to return benchmade item specific data.

```
full_dataset = []
for i in range(0, len(benchIds), 20):
    process_list(benchIds[i:i+20])

bench = pd.DataFrame(full_dataset)
bench.drop_duplicates(subset=['ItemID'], inplace=True)
bench.info()
```

ShoppingAPI call to return buck item specific data.

```
full_dataset = []
for i in range(0, len(buckIds), 20):
    process_list(buckIds[i:i+20])

buck = pd.DataFrame(full_dataset)
buck.drop_duplicates(subset=['ItemID'], inplace=True)
buck.info()
```

ShoppingAPI call to return case brand item specific data.

```
full_dataset = []
for i in range(0, len(caseIds), 20):
    process_list(caseIds[i:i+20])

df_case = pd.DataFrame(full_dataset)
df_case.drop_duplicates(subset=['ItemID'], inplace=True)
df_case.info()
```

ShoppingAPI call to return caseXX brand item specific data.

```
full_dataset = []
for i in range(0, len(caseXXIds), 20):
    process_list(caseXXIds[i:i+20])

df_caseXX = pd.DataFrame(full_dataset)
df_caseXX.drop_duplicates(subset=['ItemID'], inplace=True)
df_caseXX.info()
```

ShoppingAPI call to return crkt item specific data.

```
full_dataset = []
for i in range(0, len(crktIds), 20):
    process_list(crktIds[i:i+20])

crkt = pd.DataFrame(full_dataset)
crkt.drop_duplicates(subset=['ItemID'], inplace=True)
crkt.info()
```

ShoppingAPI call to return kershaw item specific data.

```
full_dataset = []
for i in range(0, len(kershawIds), 20):
    process_list(kershawIds[i:i+20])

kershaw = pd.DataFrame(full_dataset)
kershaw.drop_duplicates(subset=['ItemID'], inplace=True)
kershaw.info()
```

ShoppingAPI call to return SOG item specific data.

```
full_dataset = []
for i in range(0, len(sogIds), 20):
    process_list(sogIds[i:i+20])

sog = pd.DataFrame(full_dataset)
sog.drop_duplicates(subset=['ItemID'], inplace=True)
sog.info()
```

#ShoppingAPI call to return spyderco item specific data.

```
full_dataset = []
for i in range(0, len(spydIds), 20):
    process_list(spydIds[i:i+20])
spyd = pd.DataFrame(full_dataset)
spyd.drop_duplicates(subset=['ItemID'], inplace=True)
spyd.info()
```

ShoppingAPI call to return victorinox item specific data.

```
full_dataset = []
for i in range(0, len(victIds), 20):
    process_list(victIds[i:i+20])

vict = pd.DataFrame(full_dataset)
vict.drop_duplicates(subset=['ItemID'], inplace=True)
vict.info()
```

```
bench.to_csv("listed_data/benchIds.csv", index=False)
buck.to_csv("listed_data/buckIds.csv", index=False)
df_case.to_csv("listed_data/caseIds.csv", index=False)
df_caseXX.to_csv("listed_data/caseXXIds.csv", index=False)
crkt.to_csv("listed_data/crktIds.csv", index=False)
kershaw.to_csv("listed_data/kershawIds.csv", index=False)
leath.to_csv("listed_data/leathIds.csv", index=False)
sog.to_csv("listed_data/sogIds.csv", index=False)
spyd.to_csv("listed_data/spydIds.csv", index=False)
vict.to_csv("listed_data/victIds.csv", index=False)
```

Beginning of prep to merge original listed data with item specific data requested using a seperate API for more complete details about all listings gathered.

In [5]:

```
df_bench = pd.read_csv("listed_data/df_bench.csv")
df_buck = pd.read_csv("listed_data/df_buck.csv")
df_case = pd.read_csv("listed_data/df_case.csv")
df_caseXX = pd.read_csv("listed_data/df_CaseXX.csv")
df_crkt = pd.read_csv("listed_data/df_crkt.csv")
df_kersh = pd.read_csv("listed_data/df_kershaw.csv")
df_sog = pd.read_csv("listed_data/df_sog.csv")
df_spyd = pd.read_csv("listed_data/df_spyderco.csv")
df_vict = pd.read_csv("listed_data/df_victorinox.csv")


bench = pd.read_csv("listed_data/benchIds.csv")
buck = pd.read_csv("listed_data/buckIds.csv")
case = pd.read_csv("listed_data/caseIds.csv")
caseXX = pd.read_csv("listed_data/caseXXIds.csv")
crkt = pd.read_csv("listed_data/crktIds.csv")
kershaw = pd.read_csv("listed_data/kershawIds.csv")
sog = pd.read_csv("listed_data/sogIds.csv")
spyd = pd.read_csv("listed_data/spydIds.csv")
vict = pd.read_csv("listed_data/victIds.csv")



listed_df = pd.concat([df_bench,df_buck,
                       df_case,df_caseXX,
                       df_crkt,df_kersh,
                       df_sog,df_spyd,
                       df_vict])

listed_df.drop('galleryPlusPictureURL', axis=1, inplace=True)

Ids_df = pd.concat([bench,buck,
                    case,caseXX,
                    crkt,kershaw,
                    sog,spyd,vict])



Ids_df.rename({'Title': 'title',
               'ItemID': 'itemId'},
              axis=1,inplace=True)

Ids_df.drop(['ConditionID','ConvertedCurrentPrice'],
            axis=1, inplace=True)




Ids_df['title'] = Ids_df['title'].str.lower()


df_merged = listed_df.merge(Ids_df)

df_spec = transform_item_specifics(df_merged, perc=65.0)

df_spec.drop('Brand', axis=1, inplace=True)

aspect_df = df_merged.join(df_spec)

aspect_df = data_cleaner(aspect_df).copy()
aspect_df.drop(['sellingStatus', 'shippingInfo',
                'GalleryURL', 'ItemSpecifics',
                'item_list', 'listingInfo'],
                axis=1, inplace=True)

listed_used_knives = listed_df.loc[listed_df['condition'] != 1000.0]
listed_used_knives.reset_index(drop=True, inplace=True)
```

In [6]:
```
aspect_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 40114 entries, 0 to 41950
Data columns (total 35 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   itemId                          40114 non-null  int64  
 1   title                           40114 non-null  object 
 2   galleryURL                      40083 non-null  object 
 3   viewItemURL                     40114 non-null  object 
 4   autoPay                         40114 non-null  bool   
 5   postalCode                      38442 non-null  object 
 6   returnsAccepted                 40114 non-null  bool   
 7   condition                       40113 non-null  float64
 8   topRatedListing                 40114 non-null  bool   
 9   pictureURLLarge                 36701 non-null  object 
 10  pictureURLSuperSize             36435 non-null  object 
 11  shipping_cost                   40114 non-null  float64
 12  price_in_US                     40114 non-null  float64
 13  converted_price                 40114 non-null  float64
 14  brand                           40114 non-null  object 
 15  cost                            40114 non-null  float64
 16  profit                          40114 non-null  float64
 17  ROI                             40114 non-null  float64
 18  PictureURL                      40110 non-null  object 
 19  Location                        40112 non-null  object 
 20  Country                         40114 non-null  object 
 21  Blade Material                  23513 non-null  object 
 22  Model                           31790 non-null  object 
 23  Opening Mechanism               24171 non-null  object 
 24  Number of Blades                26806 non-null  object 
 25  Handle Material                 26493 non-null  object 
 26  Blade Type                      18963 non-null  object 
 27  Color                           28346 non-null  object 
 28  Type                            31278 non-null  object 
 29  Country/Region of Manufacture   22696 non-null  object 
 30  Lock Type                       19288 non-null  object 
 31  Blade Edge                      21703 non-null  object 
 32  Dexterity                       16285 non-null  object 
 33  Original/Reproduction           17150 non-null  object 
 34  Blade Range                     15675 non-null  object 
dtypes: bool(3), float64(7), int64(1), object(24)
memory usage: 10.2+ MB
```

In [7]:
```
listed_used_knives.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14310 entries, 0 to 14309
Data columns (total 21 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   itemId               14310 non-null  int64  
 1   title                14310 non-null  object 
 2   galleryURL           14309 non-null  object 
 3   viewItemURL          14310 non-null  object 
 4   autoPay              14310 non-null  bool   
 5   postalCode           13942 non-null  object 
 6   sellingStatus        14310 non-null  object 
 7   shippingInfo         14310 non-null  object 
 8   listingInfo          14310 non-null  object 
 9   returnsAccepted      14310 non-null  bool   
 10  condition            14309 non-null  float64
 11  topRatedListing      14310 non-null  bool   
 12  pictureURLLarge      13579 non-null  object 
 13  pictureURLSuperSize  13524 non-null  object 
 14  shipping_cost        14310 non-null  float64
 15  price_in_US          14310 non-null  float64
 16  converted_price      14310 non-null  float64
 17  brand                14310 non-null  object 
 18  cost                 14310 non-null  float64
 19  profit               14310 non-null  float64
 20  ROI                  14310 non-null  float64
dtypes: bool(3), float64(7), int64(1), object(10)
memory usage: 2.0+ MB
```

In [8]:
```
aspect_df.drop(['Original/Reproduction'],
                    axis=1, inplace=True)
```

In [9]:
```
aspect_df.to_csv("listed_data/listed_aspect_df.csv", index=False)
```

In [10]:
```
listed_used_knives.to_csv("listed_data/listed_used_knives.csv", index=False)
```

End of section for obtaining listed data from eBay APIs. Below is the start of processing scraped data from eBay's seller exclusive website. This data goes back 2 years and is filtered to include only used knives with final sale values. The listed data above only goes back 90 days and only shows listings currently up for sale.

In [11]:
```
sold_bench = pd.read_csv("terapeak_data/bench_scraped2.csv")
sold_buck1 = pd.read_csv("terapeak_data/buck_scraped2.csv")
sold_buck2 = pd.read_csv("terapeak_data/buck_scraped2_reversed.csv")
sold_case = pd.read_csv("terapeak_data/case_scraped2.csv")
sold_caseXX1 = pd.read_csv("terapeak_data/caseXX_scraped2.csv")
sold_caseXX2 = pd.read_csv("terapeak_data/caseXX2_reversed.csv")
sold_crkt = pd.read_csv("terapeak_data/crkt_scraped.csv")
sold_kershaw1 = pd.read_csv("terapeak_data/kershaw_scraped2.csv")
sold_kershaw2 = pd.read_csv("terapeak_data/kershaw_scraped2_reversed.csv")
sold_sog = pd.read_csv("terapeak_data/SOG_scraped2.csv")
sold_spyd = pd.read_csv("terapeak_data/spyd_scraped2.csv")
sold_vict1 = pd.read_csv("terapeak_data/vict_scraped.csv")
sold_vict2 = pd.read_csv("terapeak_data/vict_reversed.csv")

sold_list = [sold_bench,sold_buck1,
             sold_buck2,sold_case,
             sold_caseXX1,sold_caseXX2,
             sold_crkt,sold_kershaw1,
             sold_kershaw2,sold_sog,
             sold_spyd, sold_vict1,
             sold_vict2]

df_dict = {'benchmade': sold_bench,
           'buck1': sold_buck1,
           'buck2': sold_buck2,
           'case':sold_case,
           'caseXX1':sold_caseXX1,
           'caseXX2':sold_caseXX2,
           'crkt':sold_crkt,
           'kershaw1':sold_kershaw1,
           'kershaw2':sold_kershaw2,
           'sog':sold_sog,
           'spyderco':sold_spyd,
           'vict1':sold_vict1,
           'vict2':sold_vict2}


for key,val in df_dict.items():
    print(key)
    display(val.info())
```

```
benchmade
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8791 entries, 0 to 8790
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   Image        8791 non-null   object
 1   url          1843 non-null   object
 2   date_sold    8791 non-null   object
 3   price_in_US  8791 non-null   object
 4   shipping_    8791 non-null   object
 5   Text         8791 non-null   object
dtypes: object(6)
memory usage: 412.2+ KB

None

buck1
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9999 entries, 0 to 9998
```

In [12]:
```
for val in df_dict.values():
    val.rename({'Text': 'title',
                'shipping_': 'shipping_cost'},
               axis=1, inplace=True)

    val['date_sold'] = pd.to_datetime(val['date_sold'])

sold_buck = pd.concat([sold_buck1,sold_buck2])
sold_caseXX = pd.concat([sold_caseXX1,sold_caseXX2])
sold_kershaw = pd.concat([sold_kershaw1,sold_kershaw2])
sold_vict = pd.concat([sold_vict1,sold_vict2])


for key,val in df_dict.items():
    print(key)
    display(val.columns)
```

```
benchmade
```

```
Index(['Image', 'url', 'date_sold', 'price_in_US', 'shipping_cost', 'title'], dtype='object')
```

```
buck1
```

```
Index(['Image', 'url', 'date_sold', 'price_in_US', 'shipping_cost', 'title'], dtype='object')
```

```
buck2
```

```
Index(['Image', 'url', 'date_sold', 'price_in_US', 'shipping_cost', 'title'], dtype='object')
```

```
case
```

```
Index(['Image', 'url', 'date_sold', 'price_in_US', 'shipping_cost', 'title'], dtype='object')
```

```
caseXX1
```

```
Index(['Image', 'url', 'date_sold', 'price_in_US', 'shipping_cost', 'title'], dtype='object')
```

```
caseXX2
```

```
Index(['Image', 'url', 'date_sold', 'price_in_US', 'shipping_cost', 'title'], dtype='object')
```

```
crkt
```

```
Index(['Image', 'url', 'date_sold', 'price_in_US', 'shipping_cost', 'title'], dtype='object')
```

```
kershaw1
```

```
Index(['Image', 'url', 'date_sold', 'price_in_US', 'shipping_cost', 'title'], dtype='object')
```

```
kershaw2
```

```
Index(['Image', 'url', 'date_sold', 'price_in_US', 'shipping_cost', 'title'], dtype='object')
```

```
sog
```

```
Index(['Image', 'url', 'date_sold', 'price_in_US', 'shipping_cost', 'title'], dtype='object')
```

```
spyderco
```

```
Index(['Image', 'url', 'date_sold', 'price_in_US', 'shipping_cost', 'title'], dtype='object')
```

```
vict1
```

```
Index(['Image', 'url', 'date_sold', 'price_in_US', 'shipping_cost', 'title'], dtype='object')
```

```
vict2
```

```
Index(['Image', 'url', 'date_sold', 'price_in_US', 'shipping_cost', 'title'], dtype='object')
```

In [13]:
```
sold_bench = prepare_tera_df(sold_bench, 0)
sold_buck = prepare_tera_df(sold_buck, 1)
sold_case = prepare_tera_df(sold_case, 2)
sold_caseXX = prepare_tera_df(sold_caseXX, 2)
sold_crkt = prepare_tera_df(sold_crkt, 3)
sold_kershaw = prepare_tera_df(sold_kershaw, 4)
sold_sog = prepare_tera_df(sold_sog, 5)
sold_spyd = prepare_tera_df(sold_spyd, 6)
sold_vict = prepare_tera_df(sold_vict, 7)
```

In [14]:
```
for dataframe in df_dict.values():
    dataframe['title'] = dataframe['title'].str.lower()
    dataframe['title'] = dataframe['title'].str.strip()
```

In [15]:
```
sold_df = pd.concat([sold_bench, sold_buck,
                     sold_case, sold_caseXX,
                     sold_crkt, sold_kershaw,
                     sold_sog, sold_spyd,
                     sold_vict])

sold_df['brand'].value_counts()
```

Out[15]:
```
case          28492
kershaw       19447
buck          18917
victorinox    14867
spyderco       9206
benchmade      8791
crkt           6742
sog            4858
Name: brand, dtype: int64
```

In [16]:
```
sold_df.to_csv("terapeak_data/terapeak_df.csv", index=False)
```

In [17]:
```
sold_knives = data_cleaner(sold_df).copy()
sold_knives.reset_index(drop=True, inplace=True)
```

In [18]:
```
sold_knives.brand.value_counts()
```

Out[18]:
```
case          18918
kershaw       12957
buck          12534
victorinox     9437
spyderco       6046
benchmade      5712
crkt           4276
sog            3006
Name: brand, dtype: int64
```

In [19]:
```
sold_knives.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 72886 entries, 0 to 72885
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Image            72886 non-null  object
 1   url              15070 non-null  object
 2   date_sold        72886 non-null  datetime64[ns]
 3   price_in_US      72886 non-null  float64
 4   shipping_cost    72886 non-null  float64
 5   title            72886 non-null  object
 6   brand            72886 non-null  object
 7   converted_price  72886 non-null  float64
 8   cost             72886 non-null  float64
 9   profit           72886 non-null  float64
 10  ROI              72886 non-null  float64
dtypes: datetime64[ns](1), float64(6), object(4)
memory usage: 6.1+ MB
```

In [20]:
```
sold_knives['cost']
```

Out[20]:
```
0        52.95
1        52.95
2        52.95
3        52.95
4        52.95
         ...
72881    27.95
72882    27.95
72883    27.95
72884    27.95
72885    27.95
Name: cost, Length: 72886, dtype: float64
```

In [21]:
```
sold_bench.to_csv("terapeak_data/tera_bench_prepared.csv", index=False)
sold_buck.to_csv("terapeak_data/tera_buck_prepared.csv", index=False)
sold_case.to_csv("terapeak_data/tera_case_prepared.csv", index=False)
sold_caseXX.to_csv("terapeak_data/tera_caseXX_prepared.csv", index=False)
sold_crkt.to_csv("terapeak_data/tera_crkt_prepared.csv", index=False)
sold_kershaw.to_csv("terapeak_data/tera_kershaw_prepared.csv", index=False)
sold_sog.to_csv("terapeak_data/tera_sog_prepared.csv", index=False)
sold_spyd.to_csv("terapeak_data/tera_spyd_prepared.csv", index=False)
sold_knives.to_csv("terapeak_data/sold_df.csv", index=False)
```

The below block of code merged all available teraform ebay itemIds with the appropriate data. This was done in order to call the ebay Shopping API that will only accept itemIds as input. However, much of the data is older than 90 days and can no longer be accessed using the ebay Shopping API. Therefore, the teraform data will unfortunatly lack additional item specific data.

```
teradf_benchIDs = pd.read_csv("teraform_data/tera_benchmade_itemID.csv")
teradf_buckIDs = pd.read_csv("teraform_data/tera_buck_ItemIDs.csv")
teradf_caseIDs = pd.read_csv("teraform_data/tera_case_itemIDs.csv")
teradf_kershawIDs = pd.read_csv("teraform_data/tera_kershaw_ItemIDs.csv")
teradf_sogIDs = pd.read_csv("teraform_data/tera_sog_ItemIDs.csv")
teradf_spydIDs = pd.read_csv("teraform_data/tera_spyderco_ItemIDs.csv")

dfID_list = [teradf_benchIDs,teradf_buckIDs,
             teradf_caseIDs, teradf_kershawIDs,
             teradf_sogIDs, teradf_spydIDs]

for dataframe in dfID_list:
    dataframe.rename({'Field4': 'date_sold',
                      'Data_field': 'itemID',
                      'Title': 'title'},
                       axis=1, inplace=True)

teradf_kershawIDs.rename({'item': 'title'},
                         axis=1, inplace=True)

for dataframe in dfID_list:
    dataframe.dropna(inplace=True)


for dataframe in dfID_list:
    dataframe.rename({'Field4': 'date_sold',
                      'Data_field': 'itemID',
                      'Title': 'title'},
                       axis=1, inplace=True)
    dataframe.dropna(inplace=True)
    dataframe['itemID'] = dataframe['itemID'].apply(int)

teradf_kershawIDs.rename({'item': 'title'},
                         axis=1, inplace=True)

tera_benchIds = teradf_benchIDs.itemID.values.tolist()
tera_buckIds = teradf_buckIDs.itemID.values.tolist()
tera_caseIds = teradf_caseIDs.itemID.values.tolist()
tera_kershawIds = teradf_kershawIDs.itemID.values.tolist()
tera_sogIds = teradf_sogIDs.itemID.values.tolist()
tera_spydIds = teradf_spydIDs.itemID.values.tolist()

idMerge_bench = teradf_bench.merge(teradf_benchIDs, on='Image')
idMerge_buck = teradf_buck.merge(teradf_buckIDs)
idMerge_case = teradf_case.merge(teradf_caseIDs)
idMerge_kershaw = teradf_kershaw.merge(teradf_kershawIDs)
idMerge_spyd = teradf_spyd.merge(teradf_spydIDs)
idMerge_sog = teradf_sog.merge(teradf_sogIDs)

# idMerge_bench.to_csv('teraform_data/tera_bench_idMerge.csv', index=False)
# idMerge_buck.to_csv('teraform_data/tera_buck_idMerge.csv', index=False)
# idMerge_case.to_csv('teraform_data/tera_case_idMerge.csv', index=False)
# idMerge_kershaw.to_csv('teraform_data/tera_kershaw_idMerge.csv', index=False)
# idMerge_spyd.to_csv('teraform_data/tera_spyd_idMerge.csv', index=False)
# idMerge_sog.to_csv('teraform_data/tera_sog_idMerge.csv', index=False)



#Create row for converted Price of Knives in US dollars
price_list = []
for row in full_dataset:
    listed_price = np.float(row['sellingStatus']['convertedCurrentPrice']['value'])
    price_list.append(listed_price)

df['price_in_US'] = price_list
```

```
#atttempt to pull shipping cost from json dict
shipping_cost_list = []
for row in full_dataset:
    shipping_cost = np.float(row['shippingInfo']['shippingServiceCost']['value'])
    shipping_cost_list.append(shipping_cost)

df['shipping_price'] = shipping_cost_list



#pull shipping cost from json dict with regex
df['shipping_cost'] = df['shippingInfo'].apply(lambda x: re.findall("(\d+\S+\d)", json.dumps(x)))
df['shipping_cost'] = df['shipping_cost'].apply(lambda x: ''.join(x))
df.drop(df[df['shipping_cost'] == ''].index, inplace=True)
df['shipping_cost'] = df['shipping_cost'].apply(lambda x: np.float(x))

#create new feature 'converted price'
df['converted_price'] = df['shipping_cost'] + df['price_in_US']
df = df.drop_duplicates(subset=['title', 'galleryURL'], keep='first')
display(df.head())
display(df.info())

df.to_csv('data/full_dataset.csv', index=False)
```