# Constructing A Neural Network To Classify Knives from a Texas Government Surplus Store

## Categorizing Eight Different Brands as Profitable or Not Profitable using a CNN for Knife Images and a RNN for Titles from Ebay Listings

**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)

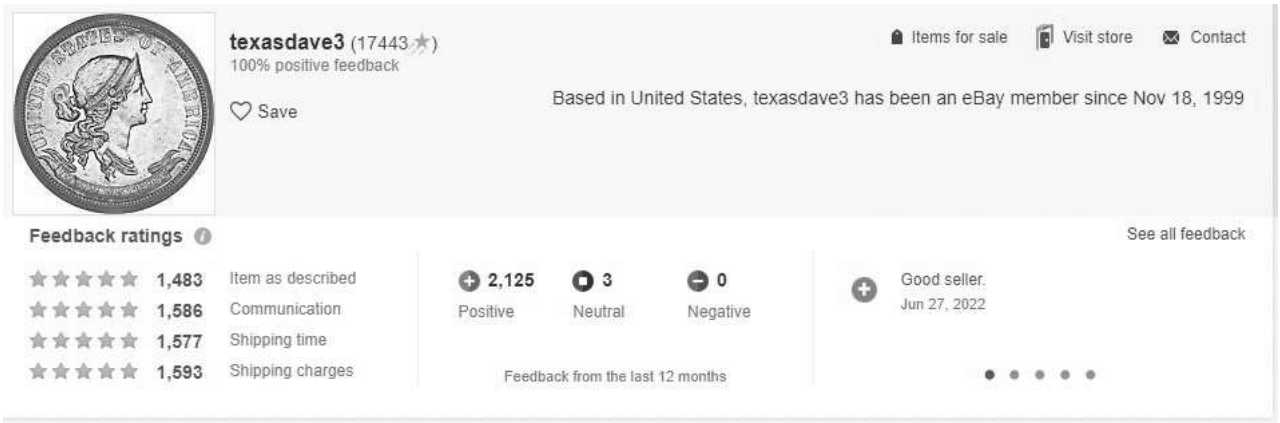


Everything that doesn't make it through Texas airports can be found at one Austin store (https://cbsaustin.com/news/local/everything-that-doesnt-make-it-through-texas-airports-can-be-found-at-one-austin-store)

> 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 Neural Network Classification Model in order to categorize knives available at this store as either profitable or not 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 Facillities Retail 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 frsgile 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.

** knife bucket image **

## Data Understanding

Describe the data being used for this project.

Questions to consider:

- Where did the data come from, and how do they relate to the data analysis questions?
- What do the data represent? Who is in the sample and what variables are included?
- What is the target variable?
- What are the properties of the variables you intend to use?

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

Ebay has a seperate website for develoers in order 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 ofcategory_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, blade range, year, and UPC/MPN.

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 7 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.

### Still need to talk about target and more details on titles and images

# 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.

Unfortunately, the API limits you to 100 pages and 100 entries MAX. So even if I tried to loop to page 101 to grab just one more entry, ebay will throw an error to the connection. However, 10,000 items seems like a reasonable amount of data for this project.

The test cell Below Sends a findItemsAdvancedRequest call to the ebay traditional (non-RESTful) finding api. If I were browsing on Ebay's website, I would be doing the following on the webpage:

- Typing the keywords 'knife' in the search bar
- filter for only used knives using the navigation box
- filter for only fixed price buy type (no auctions) using the navigation box
- filter for only a select few brands that I care about by placing check marks in appropriate boxes
- set the max items on my page to 10 and scroll all the way down and gawk at all the pretty knives

# SCRUB/EXPLORE

In [122]:
```
import pandas as pd
import  json
import requests
import numpy as np
import re
# import preprocess_ddey117 as pp
import matplotlib.pyplot as plt
%matplotlib inline
from PIL import Image


import seaborn as sns
from sklearn.model_selection import train_test_split
import os
from collections import Counter


from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Dense, Input, GlobalMaxPooling1D
from tensorflow.keras.layers import LSTM, Embedding, Flatten
from tensorflow.keras.layers import Conv1D, MaxPooling1D, GlobalMaxPooling2D
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dropout, BatchNormalization
from tensorflow.keras.models import Model
from keras import models
from keras import layers
import tensorflow as tf

from keras_preprocessing.image import ImageDataGenerator
```

In [2]:
```
def apply_iqr_filter(df):

    price_Q1 = df['converted_price'].quantile(0.25)
    price_Q3 = df['converted_price'].quantile(0.75)
    price_iqr = price_Q3 - price_Q1

    profit_Q1 = df['profit'].quantile(0.25)
    profit_Q3 = df['profit'].quantile(0.75)
    profit_iqr = profit_Q3 - profit_Q1

    ROI_Q1 = df['ROI'].quantile(0.25)
    ROI_Q3 = df['ROI'].quantile(0.75)
    ROI_iqr = ROI_Q3 - ROI_Q1

    price_upper_limit = price_Q3 + (1.5 * price_iqr)
    price_lower_limit = price_Q1 - (1.5 * price_iqr)

    profit_upper_limit = profit_Q3 + (1.5 * profit_iqr)
    profit_lower_limit = profit_Q1 - (1.5 * profit_iqr)

    ROI_upper_limit = ROI_Q3 + (1.5 * ROI_iqr)
    ROI_lower_limit = ROI_Q1 - (1.5 * ROI_iqr)

#     print(f'Brand: {df.brand[0]}')
#     print(f'price upper limit: ${np.round(price_upper_limit,2)}')
#     print(f'price lower limit: ${np.round(price_lower_limit,2)}')
#     print('-----------------------------------')
#     print(f'profit upper limit: ${np.round(profit_upper_limit,2)}')
#     print(f'profit lower limit: ${np.round(profit_lower_limit,2)}')
#     print('-----------------------------------')
#     print(f'ROI upper limit: {np.round(ROI_upper_limit,2)}%')
#     print(f'ROI lower limit: {np.round(ROI_lower_limit,2)}%')
#     print('-----------------------------------')


    new_df = df[(df['converted_price'] <= price_upper_limit) &
                (df['converted_price'] >= price_lower_limit) &
                (df['profit'] <= profit_upper_limit) &
                (df['ROI'] <= ROI_upper_limit) &
                (df['profit'] <= profit_upper_limit) &
                (df['ROI'] >= ROI_lower_limit)]

    return new_df

def download(row):
    filename = os.path.join(root_folder, str(row.name) + im_extension)

# create folder if it doesn't exist
    os.makedirs(os.path.dirname(filename), exist_ok=True)

    url = row.Image
#     print(f"Downloading {url} to {filename}")

    try:
        r = requests.get(url, allow_redirects=True)
        with open(filename, 'wb') as f:
            f.write(r.content)
    except:
        print(f'{filename} error')


def cardinality_threshold(column,threshold=0.75,return_categories_list=True):
    #calculate the threshold value using
    #the frequency of instances in column
    threshold_value=int(threshold*len(column))
    #initialize a new list for lower cardinality column
    categories_list=[]
    #initialize a variable to calculate sum of frequencies
    s=0
    #Create a dictionary (unique_category: frequency)
    counts=Counter(column)

    #Iterate through category names and corresponding frequencies after sorting the categories
    #by descending order of frequency
    for i,j in counts.most_common():
        #Add the frequency to the total sum
        s += dict(counts)[i]
        #append the category name to the categories list
        categories_list.append(i)
        #Check if the global sum has reached the threshold value, if so break the loop
        if s >= threshold_value:
            break
      #append the new 'Other' category to list
    categories_list.append('Other')

    #Take all instances not in categories below threshold
```

```
    #that were kept and lump them into the
    #new 'Other' category.
    new_column = column.apply(lambda x: x if x in categories_list else 'Other')

    #Return the transformed column and
    #unique categories if return_categories = True
    if(return_categories_list):
        return new_column,categories_list
    #Return only the transformed column if return_categories=False
    else:
        return new_column
```

In [3]:
```
import seaborn as sns
```

## Listed Data

This section explores data retrieved from making public eBay API calls. It contains more detailed information compared to the data retrieved from scraping the Teraform website.

In [4]:
```
listed_df = pd.read_csv('listed_data/listed_knives_df.csv',
                        dtype={'UPC': str,
                               'Year': str,
                               'MPN': str})
```

In [5]:
```
listed_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 33716 entries, 0 to 33715
Data columns (total 38 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   itemId                          33716 non-null  int64
 1   title                           33716 non-null  object
 2   galleryURL                      33689 non-null  object
 3   viewItemURL                     33716 non-null  object
 4   autoPay                         33716 non-null  bool
 5   postalCode                      32247 non-null  object
 6   returnsAccepted                 33716 non-null  bool
 7   condition                       33716 non-null  float64
 8   topRatedListing                 33716 non-null  bool
 9   pictureURLLarge                 30809 non-null  object
 10  pictureURLSuperSize             30574 non-null  object
 11  shipping_cost                   33716 non-null  float64
 12  price_in_US                     33716 non-null  float64
 13  converted_price                 33716 non-null  float64
 14  brand                           33716 non-null  object
 15  cost                            33716 non-null  float64
 16  profit                          33716 non-null  float64
 17  ROI                             33716 non-null  float64
 18  PictureURL                      33713 non-null  object
 19  Location                        33714 non-null  object
 20  Country                         33716 non-null  object
 21  Blade Material                  19397 non-null  object
 22  Model                           26328 non-null  object
 23  Opening Mechanism               20062 non-null  object
 24  Number of Blades                22108 non-null  object
 25  Handle Material                 21884 non-null  object
 26  Blade Type                      15644 non-null  object
 27  specBrand                       32821 non-null  object
 28  Color                           23650 non-null  object
 29  Type                            26105 non-null  object
 30  Country/Region of Manufacture   18608 non-null  object
 31  Blade Edge                      17991 non-null  object
 32  Lock Type                       15007 non-null  object
 33  Original/Reproduction           13689 non-null  object
 34  Blade Range                     10960 non-null  object
 35  Dexterity                       12712 non-null  object
 36  MPN                             7021 non-null   object
 37  Year                            2168 non-null   object
dtypes: bool(3), float64(7), int64(1), object(27)
memory usage: 9.1+ MB
```

In [6]:
```
df_listed = apply_iqr_filter(listed_df).copy()
```

In [7]:
```
df_listed['converted_price'].describe()
```

Out[7]:
```
count    30848.000000
mean        84.878289
std         56.861007
min          2.500000
25%         40.110000
50%         72.000000
75%        115.000000
max        270.000000
Name: converted_price, dtype: float64
```

In [8]:
```
df_listed.drop(['shipping_cost',
                'price_in_US','cost',
                'Original/Reproduction',
                'specBrand'], axis=1,
                inplace=True)
```

**itemId:** Unique identifier given to each listing on Ebay.

**title:** The title of the listing posted by a seller. Should correspond with the brand and model of each knife.

**galleryURL:** URL for the Gallery thumbnail image (usually around 100X100 or less)

**viewItemURL:** The url of the posting on Ebay.

**autoPay:** Boolean. This field indicates if the seller requests immediate payment for the item. If true, immediate payment is required before the checkout process can begin. If false, immediate payment is not requested.

**postalCode:** Postal code of seller

**returnsAccepted:** Boolean. Will accept returns or not.

**condition:** Ebay has codes for condition such as the following (https://developer.ebay.com/Devzone/finding/CallRef/Enums/conditionIdList.html):

1000 - New. A brand-new, unused, unopened, unworn, undamaged item.

3000 - Used. An item that has been used previously. The item may have some signs of cosmetic wear, but is fully operational and functions as intended.

These are the two codes that pertain to this project.

**topRatedListing:** Indicates whether the item is Top Rated Plus item (https://pages.ebay.com/topratedplus/index.html). A top rated plus item:
- is listed by experienced sellers with highest buyer ratings;
- Sellers commit to shipping your items in a business day with tracking provided and offer at least a 14-day, money-back return policy;

**pictureURLLarge:** URL for item's picture url with size 400x400

**pictureURLSuperSize:** URL for item's picture url with size 800x800

**shipping_cost:** Cost that seller chose to add to total price for shipping purposes.

**price_in_US:** Price of the listing without shipping in US currency.

**converted_price:** Price of listing with shipping in US currency.

**cost:** fixed cost of the knife at the surplus store

**profit:** This column corresponds to the potential profit of buying a certain knife at the surplus store and then selling it on ebay. Total money recieved for the knife - total cost of the knife (cost at surplus store plus shipping plus overhead cost).

**ROI:** Return on Investment in percent of american dollars.

**PictureURL:** List. This field shows the URL to a full-size version of one image associated with the eBay listing. A PictureURL field is returned for each image in the eBay listing. At least one PictureURL field is always returned since every eBay listing must have at least one picture. Max value is 12 for the relevant item category.

**Location:** City, State location of listing

**Country:** country of listing

### Everything below was information taking from the "itemSpecifics" call to the Shopping API which returned a list of item:value pairs that correspond to what a seller on Ebay is supposed to fill out when posting a listing to sell. For this project, this cooresponds to a list of aspects of each knife determined by each seller, including grammatical and spelling errors

**Model:** Brand related column. Each brand of knive has a line of models that they sell.

**Country/Region of Manufacture:** Country knife was manufactured.

**Blade Material:** Material of the knife blade

**Blade Type:** Shape of the blade. Most common blade typesL

**Blade Edge:** Is the edge serrated or something else?

**Dexterity:** Left handed or right handed or ambi.

**Type:** Item Specific value filled in by the seller about what type of knife they are selling, eg pocketknife or multitool

**Color:** Color of blade handle.

**Number of Blades**

**Opening Mechanism:** Assisted opening or something else?

**Handle Material**

**Lock Type:** Manual lock or something else?

**Product Line:** Brand specific column, similar to model.

**Blade Range:** The length of the blade acceptable range according to manufacturer

**Year:** Year knife was manufactured

**MPN:** Manufacturer part number. Most manufacturers list identifiers like MPN directly on a product's packaging, and often right on the product itself.

**UPC**

**Pattern:** Brand specific string column. Pattern in this case refers to the specific style that brands release for each knife. Ex: Case Trapper 9254 SS.

GalleryURL:

pictureURLLarge:


pictureURLSuperSize:


example of a list of pics from the pictureURL column



In [9]:
```
df_listed.columns
```

Out[9]:
```
Index(['itemId', 'title', 'galleryURL', 'viewItemURL', 'autoPay', 'postalCode',
       'returnsAccepted', 'condition', 'topRatedListing', 'pictureURLLarge',
       'pictureURLSuperSize', 'converted_price', 'brand', 'profit', 'ROI',
       'PictureURL', 'Location', 'Country', 'Blade Material', 'Model',
       'Opening Mechanism', 'Number of Blades', 'Handle Material',
       'Blade Type', 'Color', 'Type', 'Country/Region of Manufacture',
       'Blade Edge', 'Lock Type', 'Blade Range', 'Dexterity', 'MPN', 'Year'],
      dtype='object')
```

In [10]:
```
df_listed.drop('Type', axis=1, inplace=True)
```

In [11]:
```
#run this to view example of pictures avaliable in a random row

# df_listed[['galleryURL',
#             'pictureURLLarge',
#             'pictureURLSuperSize',
#             'PictureURL']].sample(1).apply([print])
```

In [12]:
```
str_columns = ['Location', 'Country', 'Model',
               'Country/Region of Manufacture',
               'Blade Material', 'Blade Type',
               'Blade Edge', 'Dexterity',
               'Color', 'Number of Blades',
               'Opening Mechanism', 'Handle Material',
               'Lock Type', 'Blade Range']
```

In [13]:
```
import re
import os
import sys

import pandas as pd
import numpy as np
```

In [14]:
```
for col in str_columns:
    df_listed[col] = df_listed[col].str.lower()
```

In [15]:
```
df_listed['Location'].get(5)
```

Out[15]:
```
'goodyear, arizona'
```

In [16]:
```
pattern = ".*,\s*([^\d,]+?)(?:\s*\d+)?$"
df_listed['State_or_Province'] = df_listed['Location'].str.extract(pattern)
```

In [17]:
```
df_listed.loc[df_listed['Model'].notnull(), 'brand'].value_counts()
```

Out[17]:
```
case          10240
spyderco       4007
kershaw        2930
victorinox     2694
buck           1538
crkt            939
benchmade       848
sog             743
Name: brand, dtype: int64
```

In [18]:
```
df_listed.head(10).transpose()
```

Out[18]:

| | 0 | 1 | |
|---|---|---|---|
| **itemId** | 165768472932 | 314297176821 | 154317 |
| **title** | benchmade 535 bugout® cpm-s30v blade grivory h... | manual benchmade 150802 crooked river wood kni... | benchmade mini bugout 533 knife white ha |
| **galleryURL** | https://i.ebayimg.com/thumbs/images/g/uTQAAOSw... | https://i.ebayimg.com/thumbs/images/g/DLcAAOSw... | https://i.ebayimg.com/thumbs/images/g/vsQA |
| **viewItemURL** | https://www.ebay.com/itm/BENCHMADE-535-Bugout-... | https://www.ebay.com/itm/Manual-BENCHMADE-1508... | https://www.ebay.com/itm/Benchmade-Mini-E |
| **autoPay** | True | True | |
| **postalCode** | 531** | 484** | |
| **returnsAccepted** | True | True | |
| **condition** | 3000 | 3000 | |
| **topRatedListing** | False | False | |
| **pictureURLLarge** | https://i.ebayimg.com/00/s/MTYwMFgxNjAw/z/uTQA... | https://i.ebayimg.com/00/s/ODAwWDgwMA==/z/DLcA... | https://i.ebayimg.com/00/s/MTA2N1gxNjAw/z |
| **pictureURLSuperSize** | https://i.ebayimg.com/00/s/MTYwMFgxNjAw/z/uTQA... | https://i.ebayimg.com/00/s/ODAwWDgwMA==/z/DLcA... | https://i.ebayimg.com/00/s/MTA2N1gxNjAw/z |
| **converted_price** | 119.9 | 199.99 | |
| **brand** | benchmade | benchmade | ben |
| **profit** | 56.313 | 125.991 | |
| **ROI** | 117.319 | 262.482 | |
| **PictureURL** | ['https://i.ebayimg.com/00/s/MTYwMFgxNjAw/z/uT... | ['https://i.ebayimg.com/00/s/ODAwWDgwMA==/z/DL... | ['https://i.ebayimg.com/00/s/MTA2N1gxNjA |
| **Location** | union grove, wisconsin | mount morris, michigan | chattanooga, ter |
| **Country** | us | us | |
| **Blade Material** | NaN | NaN | cp |
| **Model** | bugout 535 | crooked river | bug |
| **Opening Mechanism** | manual | manual | |
| **Number of Blades** | 1 | 1 | |
| **Handle Material** | NaN | wood | |
| **Blade Type** | drop point | drop point | dr |
| **Color** | green | black | |
| **Country/Region of Manufacture** | united states | united states | |
| **Blade Edge** | NaN | NaN | |
| **Lock Type** | NaN | NaN | |
| **Blade Range** | NaN | NaN | |
| **Dexterity** | NaN | NaN | |
| **MPN** | NaN | NaN | |
| **Year** | NaN | NaN | |
| **State_or_Province** | wisconsin | michigan | ter |

## Year column exploration

In [19]:
```
df_listed.loc[df_listed['Year'].notnull(), 'brand'].value_counts()
```

Out[19]:
```
case    2017
Name: brand, dtype: int64
```

In [20]:
```
# tot_case = pd.read_csv('listed_data/total_list_case.csv')
# tot_caseXX = pd.read_csv('listed_data/total_list_caseXX.csv')
# case = pd.concat([tot_case,tot_caseXX])
```

In [21]:
```
# case = apply_iqr_filter(case).copy()
```

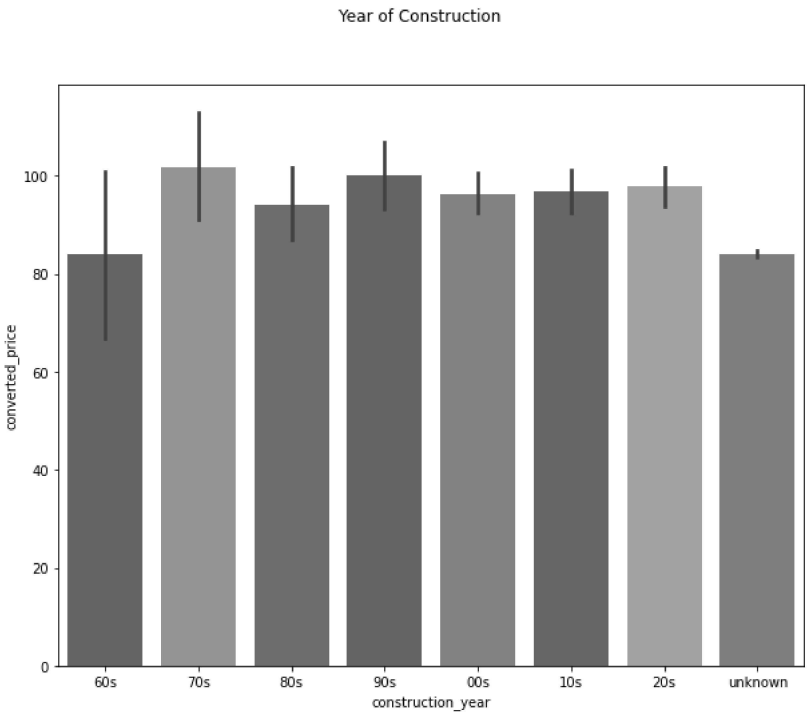
In [22]:
```
def year_wrangler(row):
    if row['Year'] >= 1960 and row['Year'] < 1970:
        return '60s'
    elif row['Year'] >= 1970 and row['Year'] < 1980:
        return '70s'
    elif row['Year'] >= 1980 and row['Year'] < 1990:
        return '80s'
    elif row['Year'] >= 1990 and row['Year'] < 2000:
        return '90s'
    elif row['Year'] >= 2000 and row['Year'] < 2010:
        return '00s'
    elif row['Year'] >= 2010 and row['Year'] < 2020:
        return '10s'
    elif row['Year'] >= 2020 and row['Year'] < 2030:
        return '20s'

    else:
        return 'unknown'

def bin_knife_year(knife_df):
    knife_df['construction_year'] = knife_df.apply(lambda row: year_wrangler(row), axis=1)
    return knife_df
```

In [23]:
```
pattern = re.compile("(\d{4}$)")
df_listed['Year'] = df_listed['Year'].str.extract(pattern)
```

In [24]:
```
df_listed['Year'] = df_listed['Year'].fillna(0)

df_listed['Year'] = df_listed['Year'].astype(int)
```

In [25]:
```
df_listed = bin_knife_year(df_listed)
```

In [26]:
```
fig = plt.figure(figsize=(10,8))
fig.suptitle('Year of Construction')
sns.barplot(x= 'construction_year',
            y='converted_price',
            data=df_listed,
            order = ['60s', '70s',
                     '80s', '90s',
                     '00s', '10s',
                     '20s', 'unknown'])
plt.show();
```

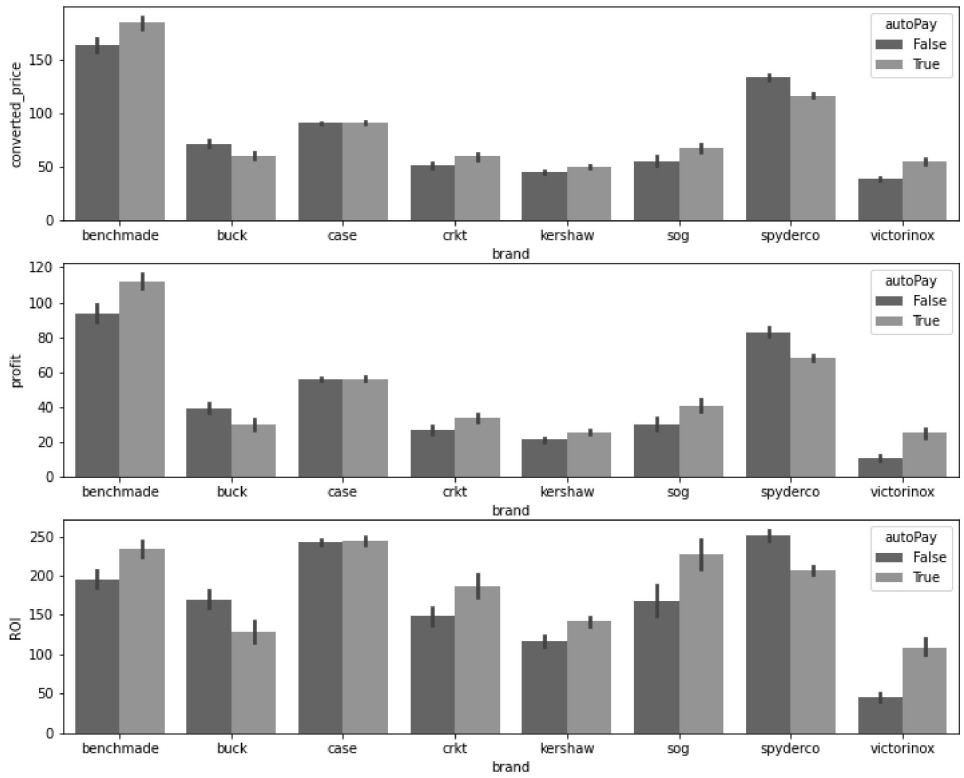


In [27]:
```
df_listed.columns
```

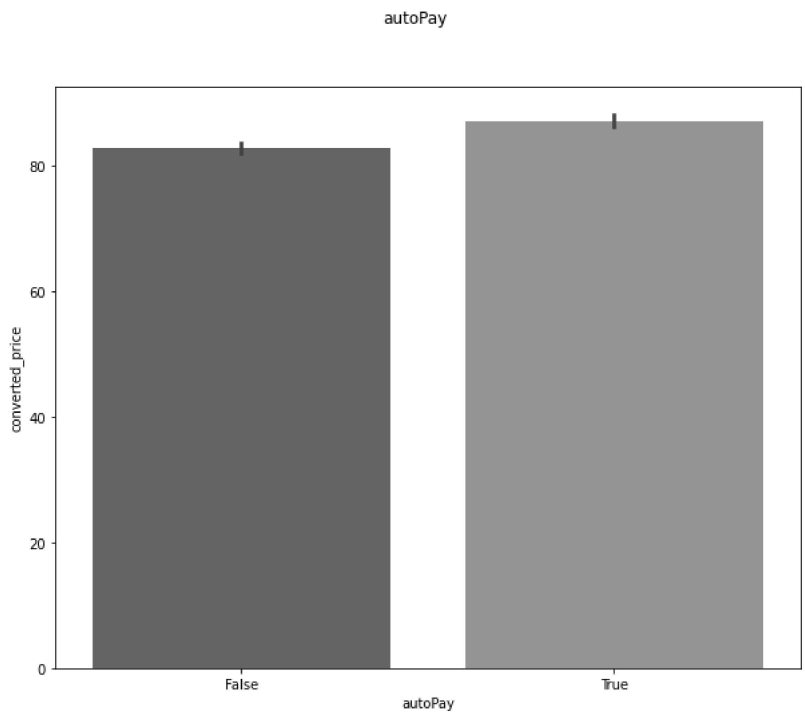
Out[27]:
```
Index(['itemId', 'title', 'galleryURL', 'viewItemURL', 'autoPay', 'postalCode',
       'returnsAccepted', 'condition', 'topRatedListing', 'pictureURLLarge',
       'pictureURLSuperSize', 'converted_price', 'brand', 'profit', 'ROI',
       'PictureURL', 'Location', 'Country', 'Blade Material', 'Model',
       'Opening Mechanism', 'Number of Blades', 'Handle Material',
       'Blade Type', 'Color', 'Country/Region of Manufacture', 'Blade Edge',
       'Lock Type', 'Blade Range', 'Dexterity', 'MPN', 'Year',
       'State_or_Province', 'construction_year'],
      dtype='object')
```

In [28]:
```
display(df_listed['autoPay'].value_counts())

fig, axes  = plt.subplots(figsize=(12,10),nrows=3)
sns.barplot(x= 'brand', y='converted_price', ax=axes[0],hue='autoPay',data=df_listed)
sns.barplot(x= 'brand', y='profit', ax=axes[1],hue='autoPay',data=df_listed)
sns.barplot(x= 'brand', y='ROI', ax=axes[2],hue='autoPay',data=df_listed)
plt.show();
```

```
False    16434
True     14414
Name: autoPay, dtype: int64
```

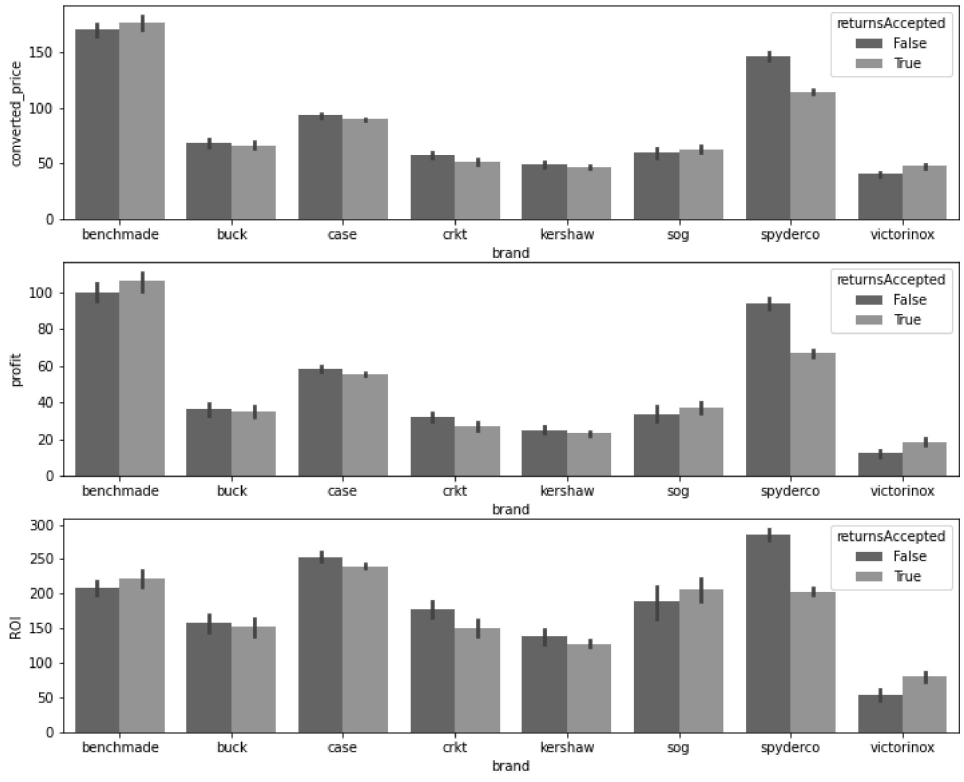


In [29]:
```
fig = plt.figure(figsize=(10,8))
fig.suptitle('autoPay')
sns.barplot(x= 'autoPay', y='converted_price', data=df_listed)
plt.show();
```

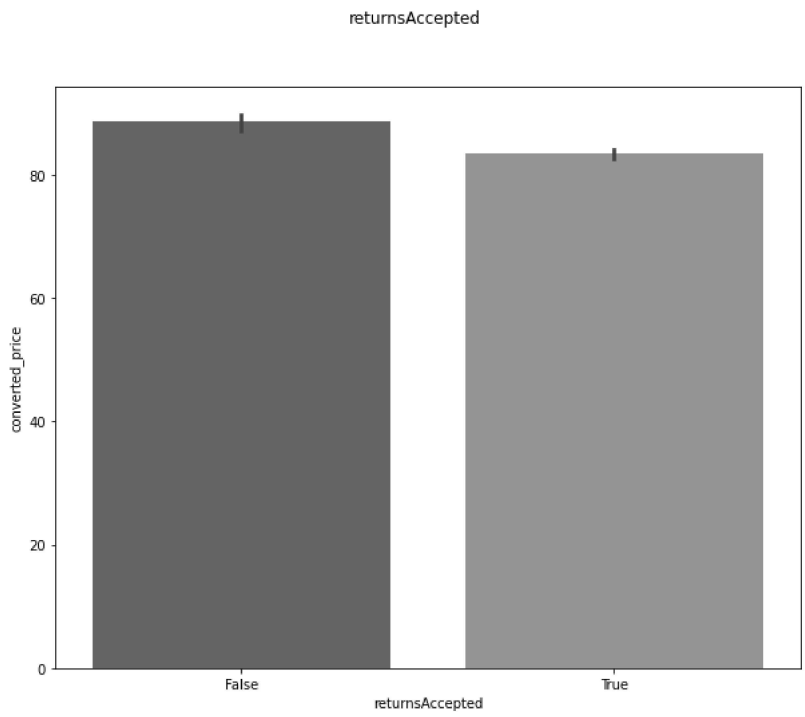



In [30]:
```
display(df_listed['returnsAccepted'].value_counts())

fig, axes  = plt.subplots(figsize=(12,10),nrows=3)
sns.barplot(x= 'brand', y='converted_price', ax=axes[0],hue='returnsAccepted',data=df_listed)
sns.barplot(x= 'brand', y='profit', ax=axes[1],hue='returnsAccepted',data=df_listed)
sns.barplot(x= 'brand', y='ROI', ax=axes[2],hue='returnsAccepted',data=df_listed)
plt.show();
```

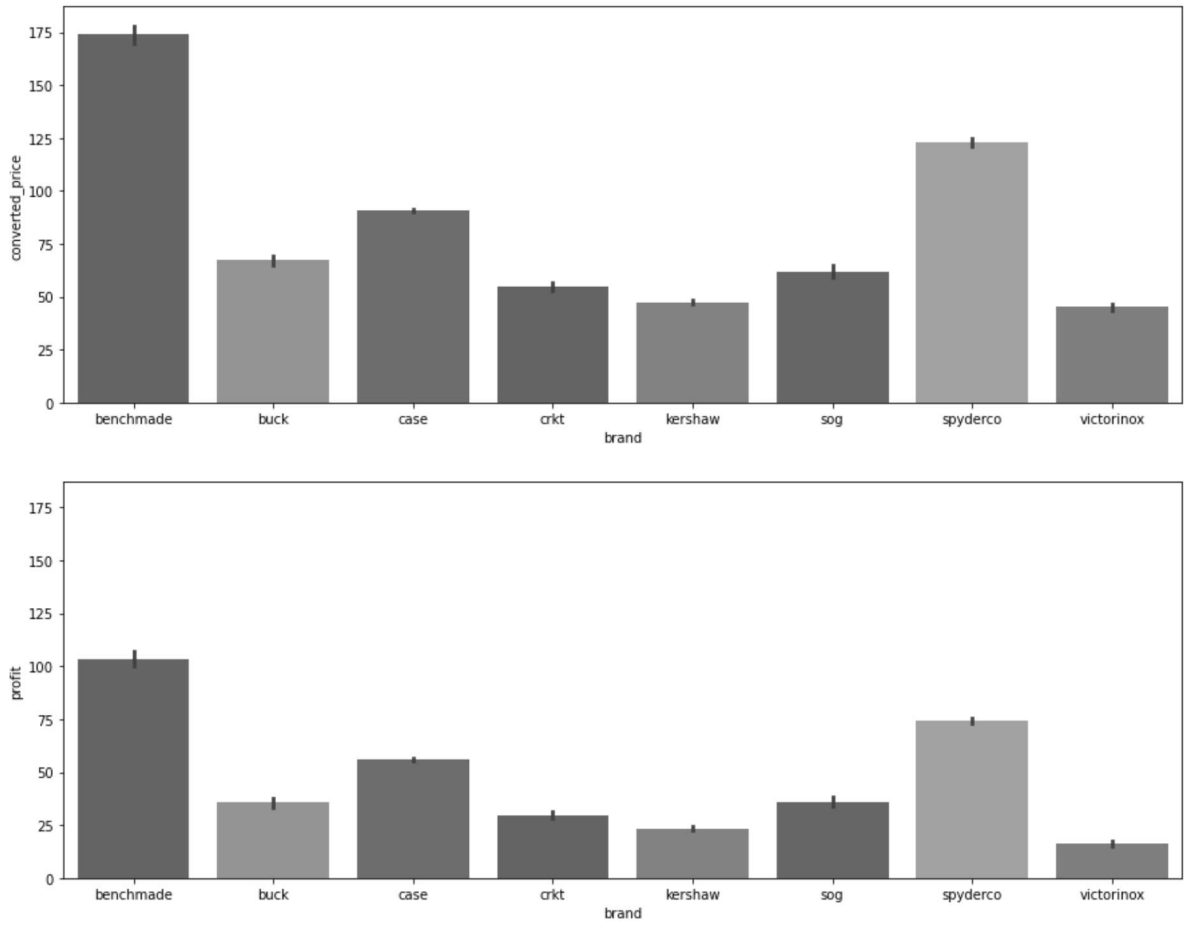
```
True     21764
False     9084
Name: returnsAccepted, dtype: int64
```



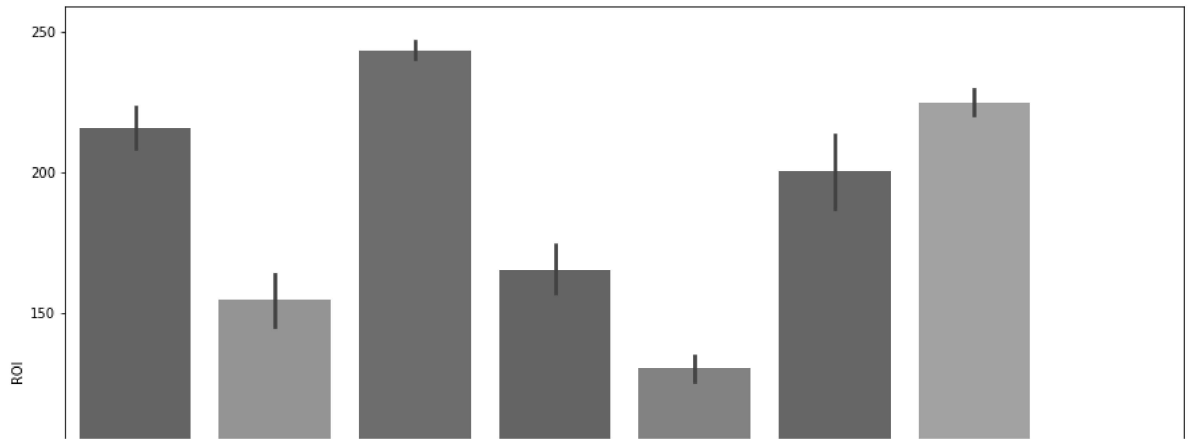
In [31]:
```
fig = plt.figure(figsize=(10,8))
fig.suptitle('returnsAccepted')
sns.barplot(x= 'returnsAccepted', y='converted_price', data=df_listed)
plt.show();
```



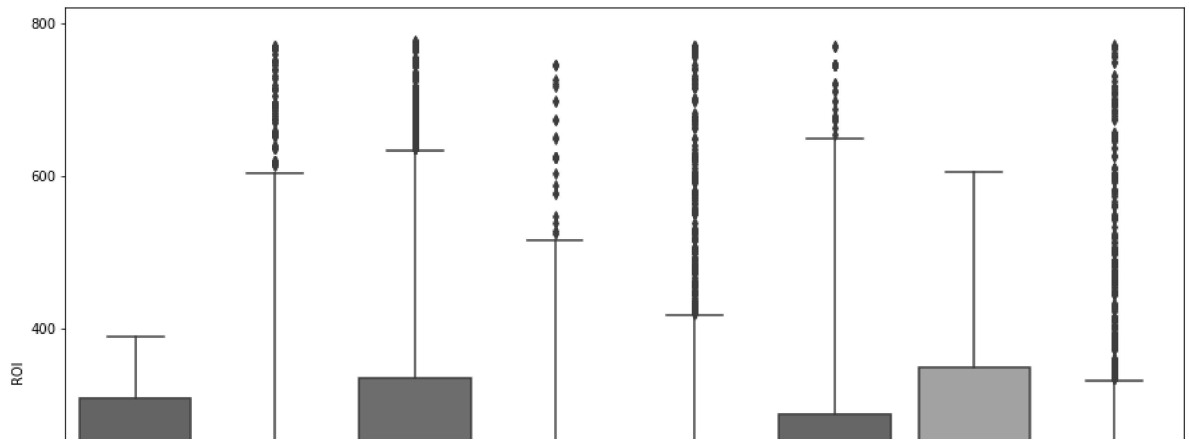

In [32]:
```
fig, axes  = plt.subplots(figsize=(15,12),nrows=2, sharey=True)
sns.barplot(x= 'brand', y='converted_price', ax=axes[0],data=df_listed)
sns.barplot(x= 'brand', y='profit', ax=axes[1],data=df_listed)
plt.show();
```



In [33]:
```
fig, axes  = plt.subplots(figsize=(15,10),nrows=1)
sns.barplot(x= 'brand', y='ROI',data=df_listed)
plt.show();
```




In [34]:
```
fig, axes  = plt.subplots(figsize=(15,10),nrows=1)
sns.boxplot(x= 'brand', y='ROI',data=df_listed)
plt.show();
```



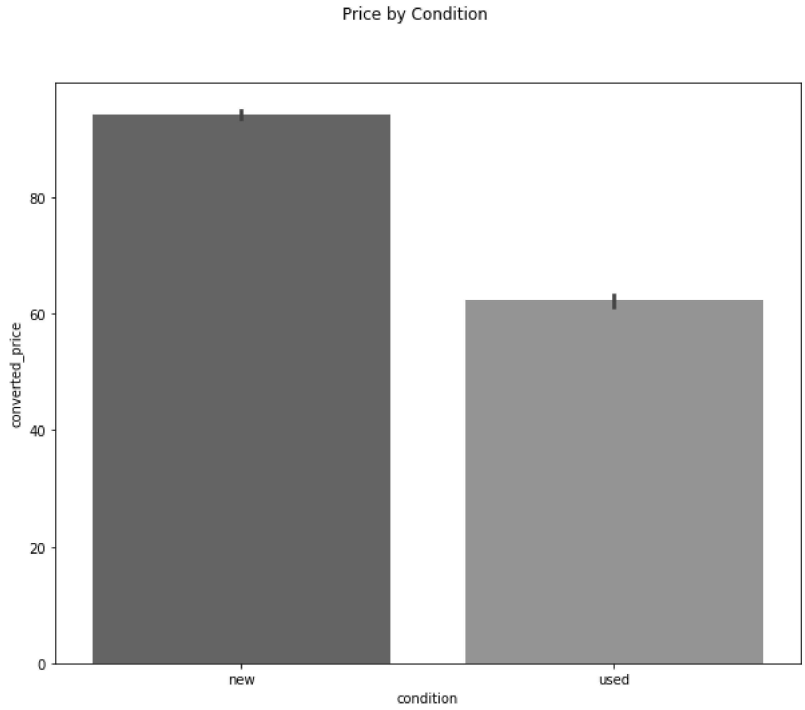

In [35]:
```
df_listed.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 30848 entries, 0 to 33715
Data columns (total 34 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   itemId                        30848 non-null  int64
 1   title                         30848 non-null  object
 2   galleryURL                    30821 non-null  object
 3   viewItemURL                   30848 non-null  object
 4   autoPay                       30848 non-null  bool
 5   postalCode                    29647 non-null  object
 6   returnsAccepted               30848 non-null  bool
 7   condition                     30848 non-null  float64
 8   topRatedListing               30848 non-null  bool
 9   pictureURLLarge               28142 non-null  object
 10  pictureURLSuperSize           27930 non-null  object
 11  converted_price               30848 non-null  float64
 12  brand                         30848 non-null  object
 13  profit                        30848 non-null  float64
```

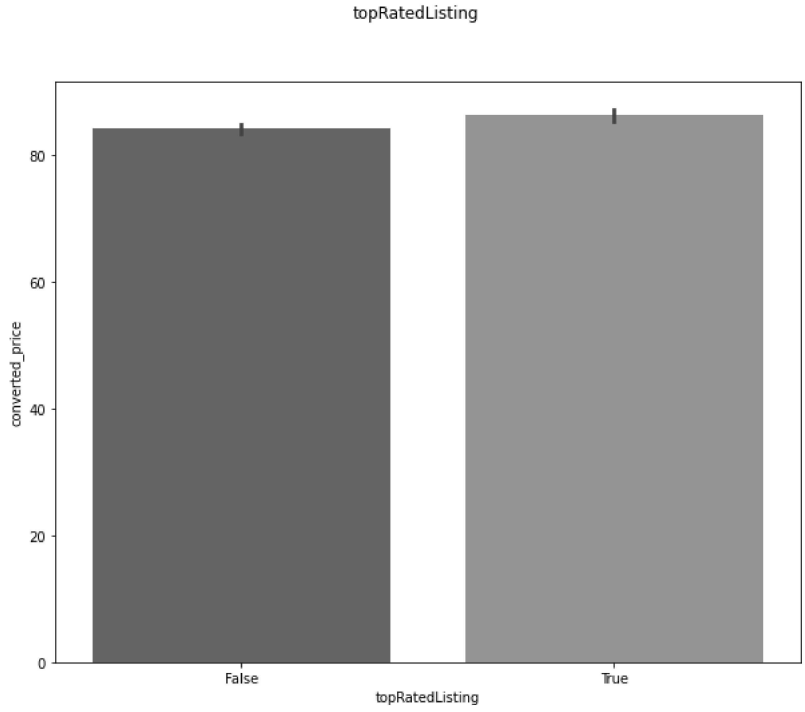
In [36]:
```
df_listed['condition'].unique()
```

Out[36]:
```
array([3000., 1000.])
```

In [37]:
```
fig = plt.figure(figsize=(10,8))
fig.suptitle('Price by Condition')
sns.barplot(x= 'condition',
            y='converted_price',
            data=df_listed)
plt.xticks([0,1], ['new', 'used'])

plt.show();
```

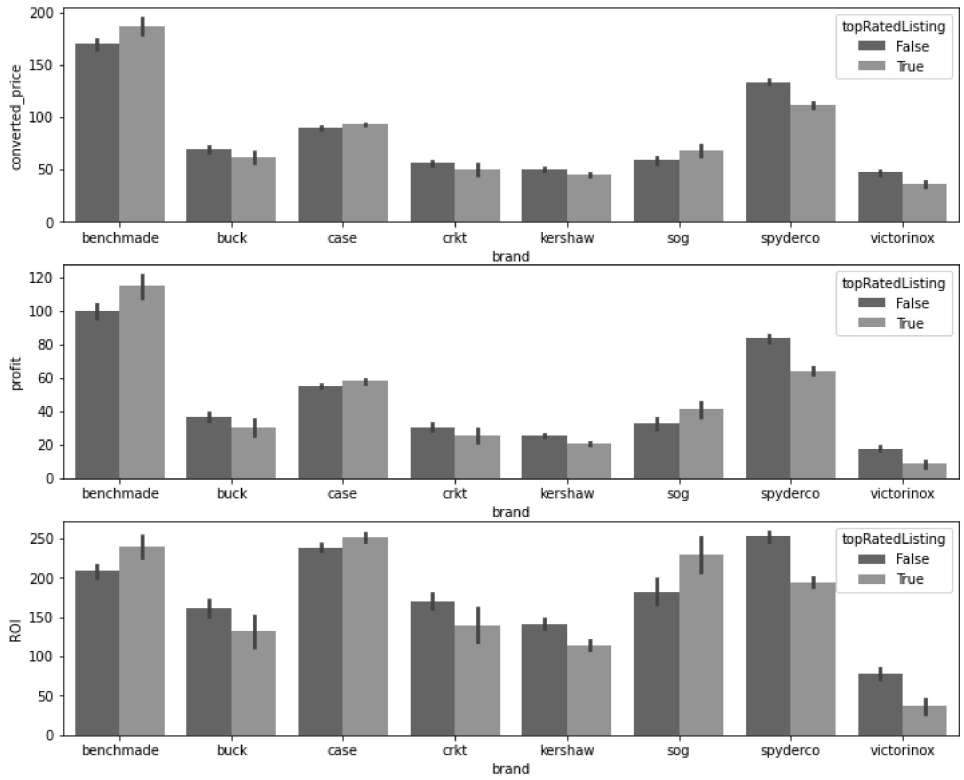



In [38]:
```
fig = plt.figure(figsize=(10,8))
fig.suptitle('topRatedListing')
sns.barplot(x= 'topRatedListing', y='converted_price', data=df_listed)
plt.show();
```



In [39]:
```
display(df_listed['topRatedListing'].value_counts())

fig, axes  = plt.subplots(figsize=(12,10),nrows=3)
sns.barplot(x= 'brand', y='converted_price', ax=axes[0],hue='topRatedListing',data=df_listed)
sns.barplot(x= 'brand', y='profit', ax=axes[1],hue='topRatedListing',data=df_listed)
sns.barplot(x= 'brand', y='ROI', ax=axes[2],hue='topRatedListing',data=df_listed)
plt.show();
```

```
False    19782
True     11066
Name: topRatedListing, dtype: int64
```




In [40]:
```
df_listed['Location'].value_counts()
```

Out[40]:
```
knoxville, tennessee        2697
prestonsburg, kentucky       951
crestwood, kentucky          873
coeburn, virginia            793
berea, kentucky              673
                            ...
fairview, tennessee            1
prescott valley, arizona       1
reedsville, pennsylvania       1
broad run, virginia            1
dayton, kentucky               1
Name: Location, Length: 3092, dtype: int64
```

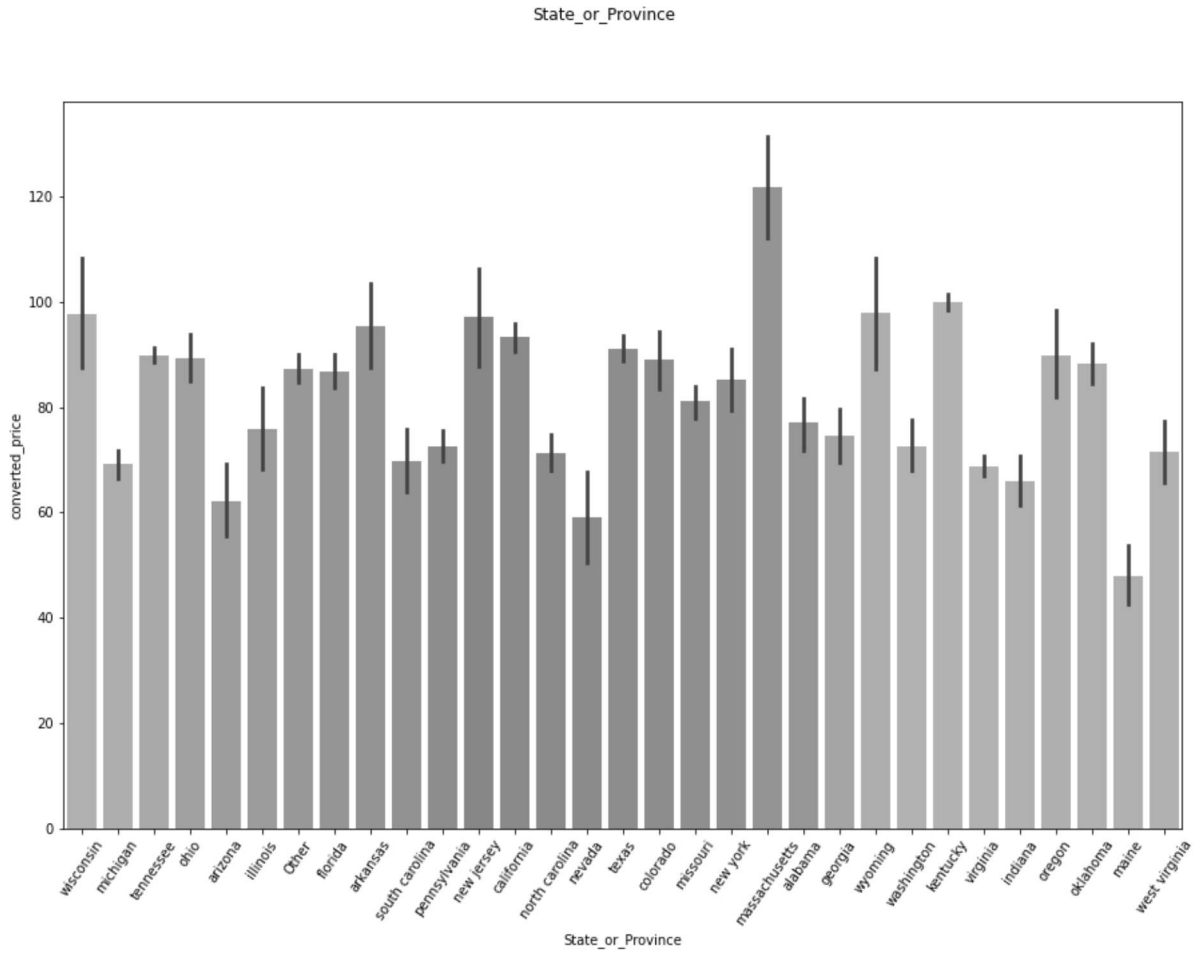
In [41]:
```
transformed_loc = cardinality_threshold(df_listed['Location'],
                                        threshold=0.7,
                                        return_categories_list=False)
transformed_loc.value_counts()
```

Out[41]:
```
Other                   9246
knoxville, tennessee    2697
prestonsburg, kentucky   951
crestwood, kentucky      873
coeburn, virginia        793
                        ...
kanagawa                  27
ashland, kentucky         27
kosciusko, mississippi    27
plano, texas              27
auckland                  27
Name: Location, Length: 191, dtype: int64
```

In [42]:
```
transformed_state = cardinality_threshold(df_listed['State_or_Province'],
                                          threshold=0.93,
                                          return_categories_list=False)
transformed_state.value_counts()
```

Out[42]:
```
kentucky          3657
tennessee         3653
california        2306
virginia          2269
texas             2178
Other             2155
florida           1305
michigan          1253
pennsylvania      1217
missouri          1043
north carolina    1002
ohio               684
colorado           590
washington         573
georgia            562
indiana            546
new york           486
oklahoma           479
alabama            407
south carolina     364
illinois           307
wyoming            296
maine              291
arizona            286
west virginia      234
new jersey         231
arkansas           228
oregon             225
massachusetts      223
wisconsin          219
nevada             219
Name: State_or_Province, dtype: int64
```

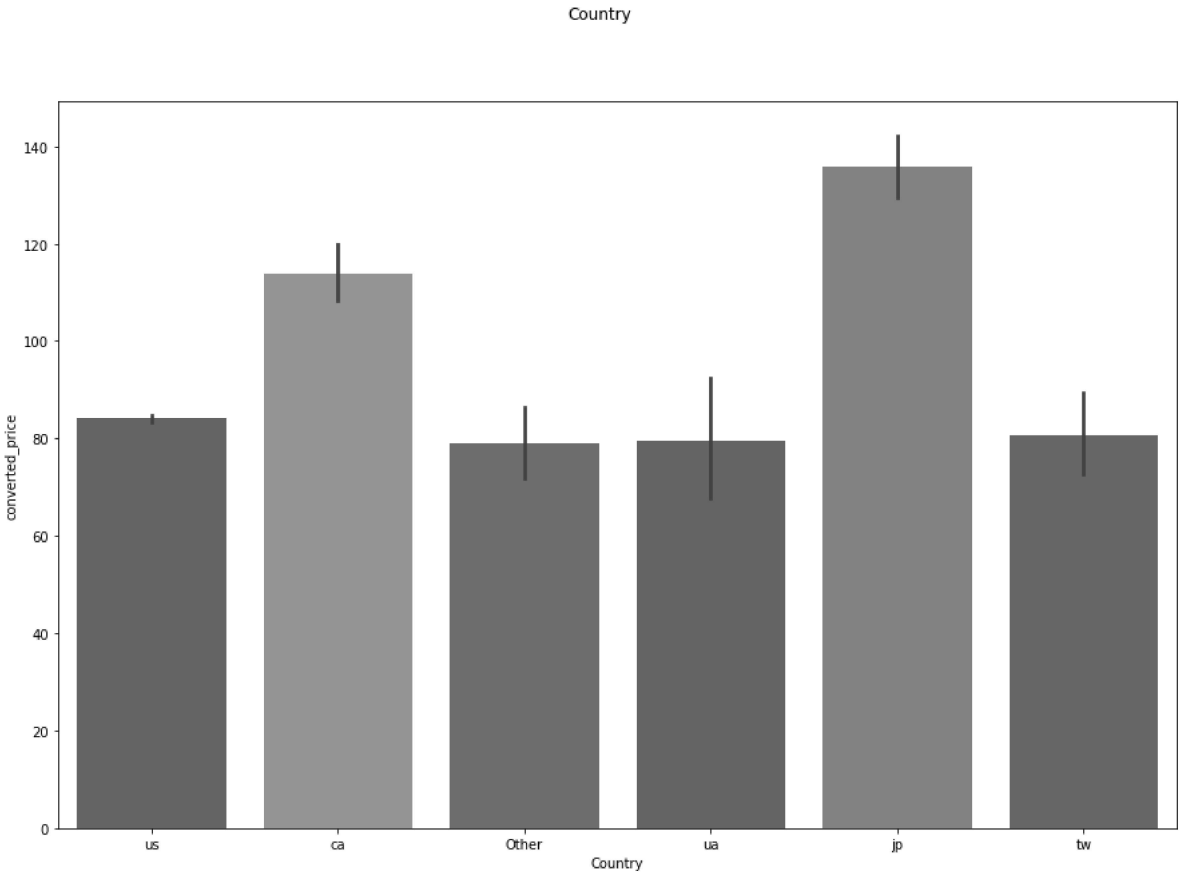
In [43]:
```
df_transformed = df_listed.copy()
df_transformed['State_or_Province'] = transformed_state
# df_sorted = df_transformed.sort_values('converted_price')

fig = plt.figure(figsize=(15,10))
fig.suptitle('State_or_Province')
sns.barplot(x= 'State_or_Province', y='converted_price', data=df_transformed)
plt.xticks(rotation=55)
plt.show();
```




In [44]:
```
transformed_country = cardinality_threshold(df_listed['Country'],
                                            threshold=0.992,
                                            return_categories_list=False)
transformed_country.value_counts()
```

Out[44]:
```
us       29741
ca         422
jp         276
Other      182
ua         133
tw          94
Name: Country, dtype: int64
```

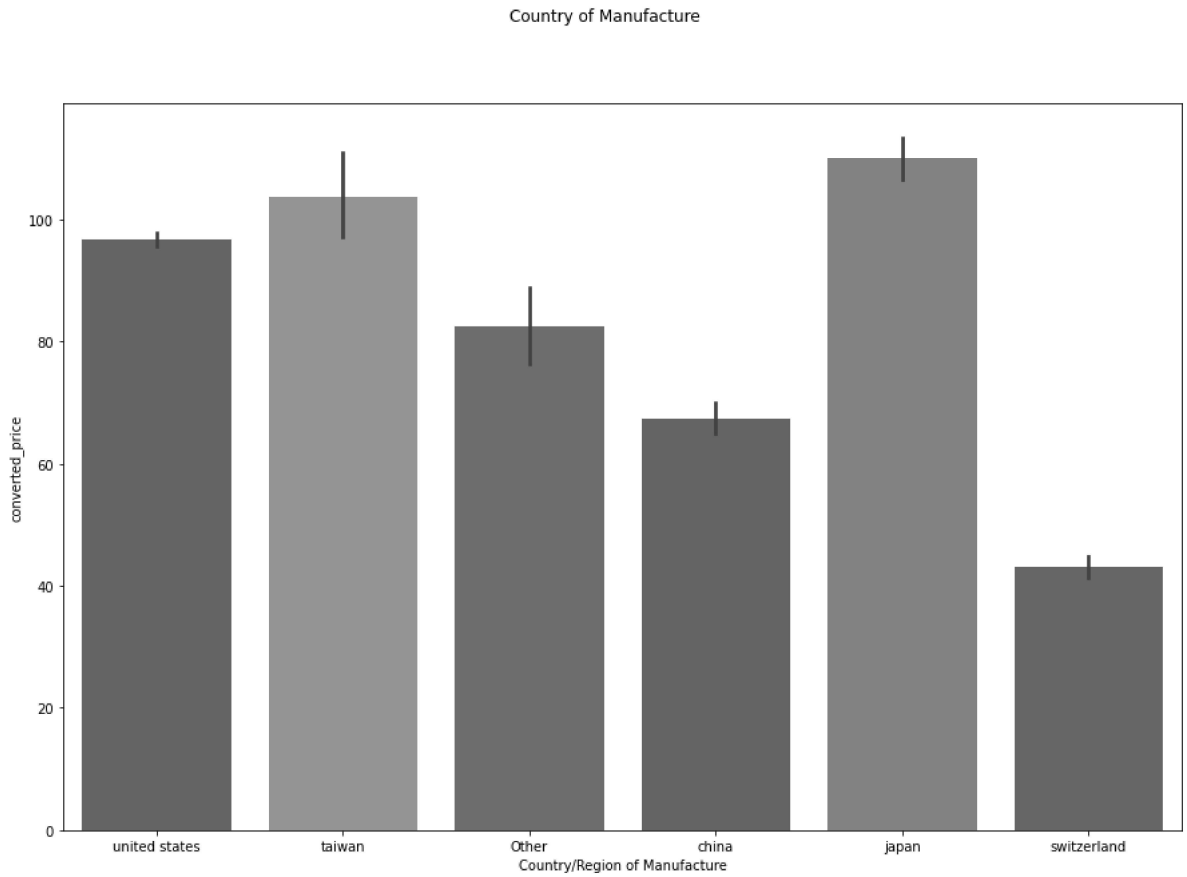
In [45]:
```
df_transformed = df_listed.copy()
df_transformed['Country'] = transformed_country
# df_sorted = df_transformed.sort_values('converted_price')

fig = plt.figure(figsize=(15,10))
fig.suptitle('Country')
sns.barplot(x= 'Country', y='converted_price', data=df_transformed)
plt.show();
```




In [46]:
```
transformed_rom = cardinality_threshold(df_listed['Country/Region of Manufacture'],
                                        threshold=0.98,
                                        return_categories_list=False)
transformed_rom.value_counts()
```

Out[46]:
```
united states    10889
switzerland       2058
china             1852
japan             1434
taiwan             351
Other              324
Name: Country/Region of Manufacture, dtype: int64
```

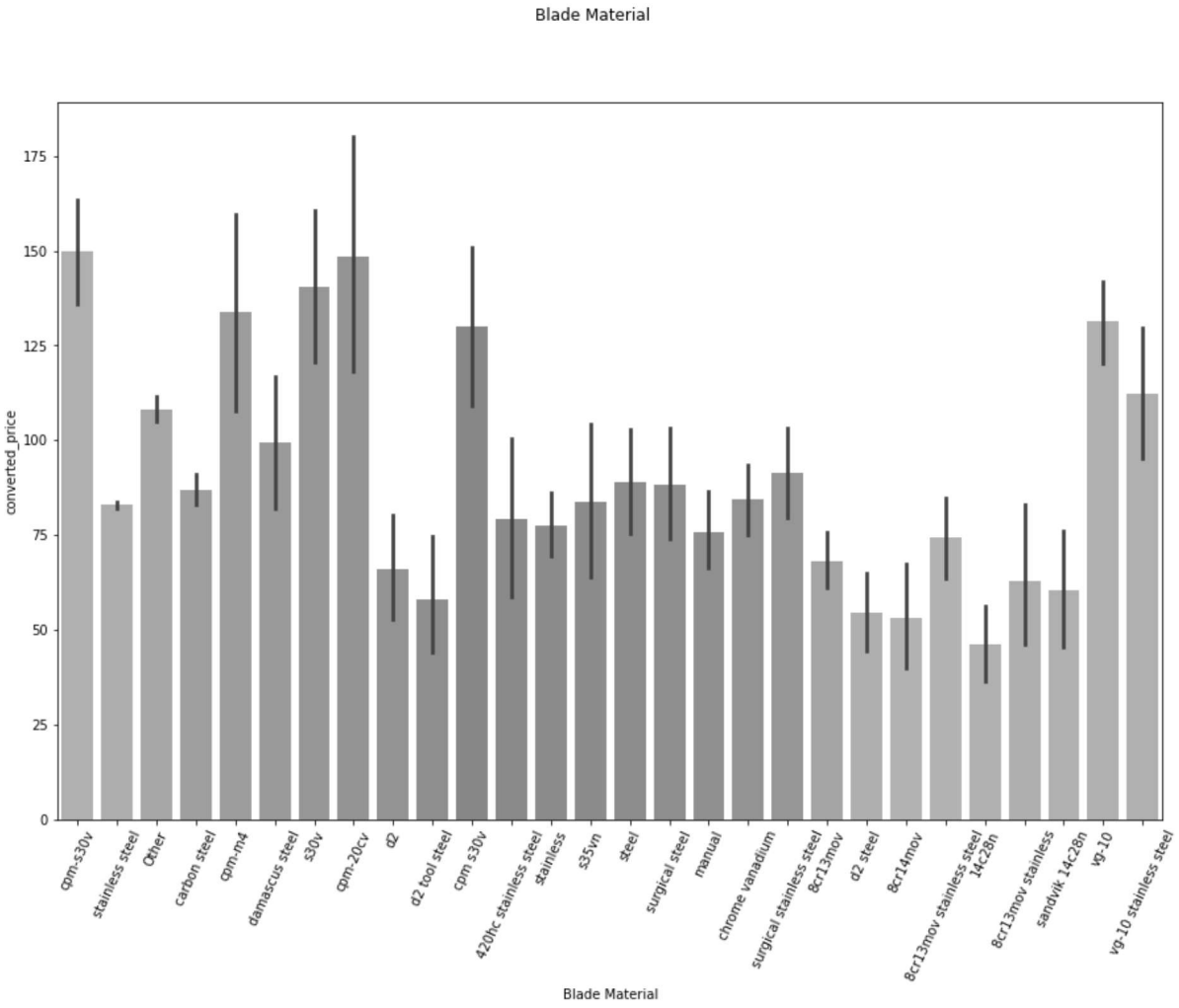
In [47]:
```
df_transformed = df_listed.copy()
df_transformed['Country/Region of Manufacture'] = transformed_rom
# df_sorted = df_transformed.sort_values('converted_price')

fig = plt.figure(figsize=(15,10))
fig.suptitle('Country of Manufacture')
sns.barplot(x= 'Country/Region of Manufacture', y='converted_price', data=df_transformed)
plt.show();
```




In [48]:
```
transformed_blm = cardinality_threshold(df_listed['Blade Material'],
                                        threshold=0.94,
                                        return_categories_list=False)
transformed_blm.value_counts()
```

Out[48]:
```
stainless steel             13695
Other                        1844
carbon steel                  635
8cr13mov                      238
vg-10                         143
8cr13mov stainless steel      120
stainless                     116
manual                         79
steel                          76
chrome vanadium                75
cpm-s30v                       71
surgical stainless steel       64
s30v                           57
damascus steel                 56
d2 steel                       42
d2                             41
vg-10 stainless steel          40
420hc stainless steel          38
8cr13mov stainless             36
cpm s30v                       36
d2 tool steel                  31
cpm-m4                         30
14c28n                         30
surgical steel                 29
sandvik 14c28n                 22
s35vn                          22
cpm-20cv                       22
8cr14mov                       22
Name: Blade Material, dtype: int64
```

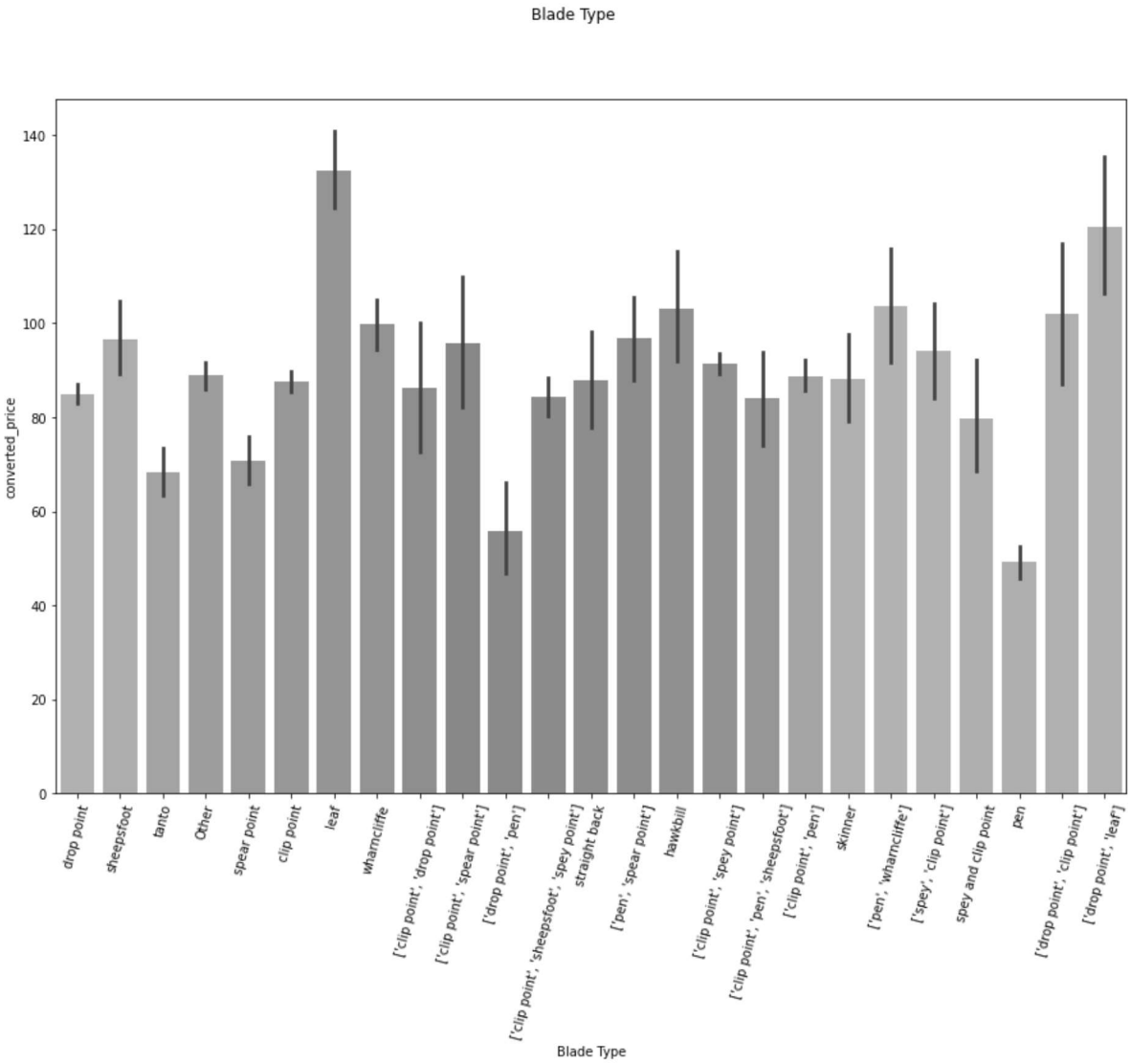
In [49]:
```
df_transformed = df_listed.copy()
df_transformed['Blade Material'] = transformed_blm
# df_sorted = df_transformed.sort_values('converted_price')

fig = plt.figure(figsize=(15,10))
fig.suptitle('Blade Material')
sns.barplot(x= 'Blade Material', y='converted_price', data=df_transformed)
plt.xticks(rotation=65)
plt.show();
```



In [50]:
```
transformed_blt = cardinality_threshold(df_listed['Blade Type'],
                                        threshold=0.95,
                                        return_categories_list=False)
transformed_blt.value_counts()
```

Out[50]:
```
drop point                                  3960
clip point                                  2419
['clip point', 'spey point']                1537
Other                                       1521
pen                                          662
['clip point', 'pen']                        646
spear point                                  513
tanto                                        479
wharncliffe                                  473
['clip point', 'sheepsfoot', 'spey point']   456
sheepsfoot                                   257
leaf                                         257
['pen', 'spear point']                       122
straight back                                112
['drop point', 'pen']                        107
hawkbill                                      90
['drop point', 'leaf']                        80
['drop point', 'clip point']                  78
skinner                                       75
['clip point', 'pen', 'sheepsfoot']           75
['spey', 'clip point']                        67
['pen', 'wharncliffe']                        66
['clip point', 'drop point']                  58
['clip point', 'spear point']                 52
spey and clip point                           42
Name: Blade Type, dtype: int64
```

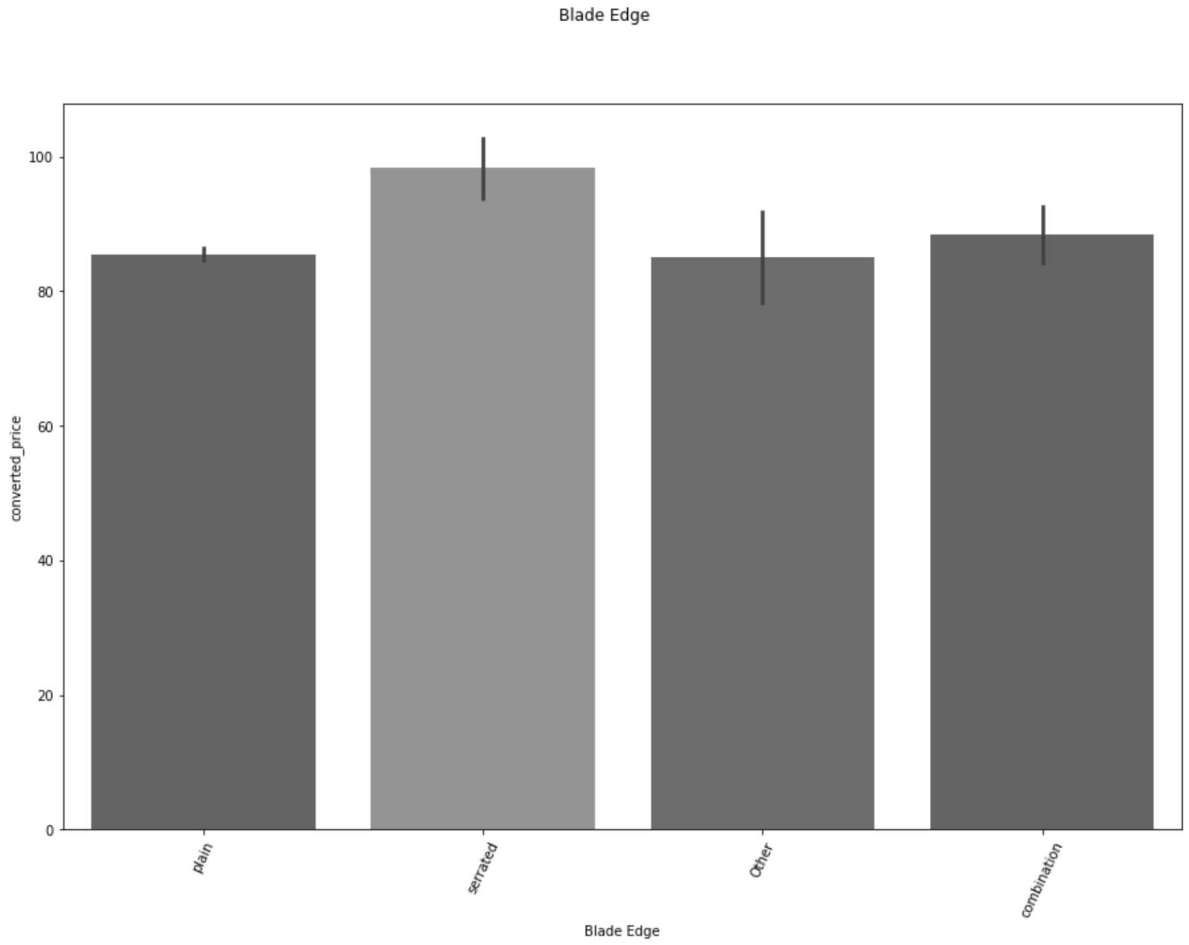
In [51]:
```
df_transformed = df_listed.copy()
df_transformed['Blade Type'] = transformed_blt
# df_sorted = df_transformed.sort_values('converted_price')

fig = plt.figure(figsize=(15,10))
fig.suptitle('Blade Type')
sns.barplot(x= 'Blade Type', y='converted_price', data=df_transformed)
plt.xticks(rotation=75)
plt.show();
```




In [52]:
```
transformed_ble = cardinality_threshold(df_listed['Blade Edge'],
                                        threshold=0.99,
                                        return_categories_list=False)
transformed_ble.value_counts()
```

Out[52]:
```
plain          14459
combination      974
serrated         673
Other            255
Name: Blade Edge, dtype: int64
```

In [53]:
```
df_transformed = df_listed.copy()
df_transformed['Blade Edge'] = transformed_ble
# df_sorted = df_transformed.sort_values('converted_price')

fig = plt.figure(figsize=(15,10))
fig.suptitle('Blade Edge')
sns.barplot(x= 'Blade Edge', y='converted_price', data=df_transformed)
plt.xticks(rotation=65)
plt.show();
```



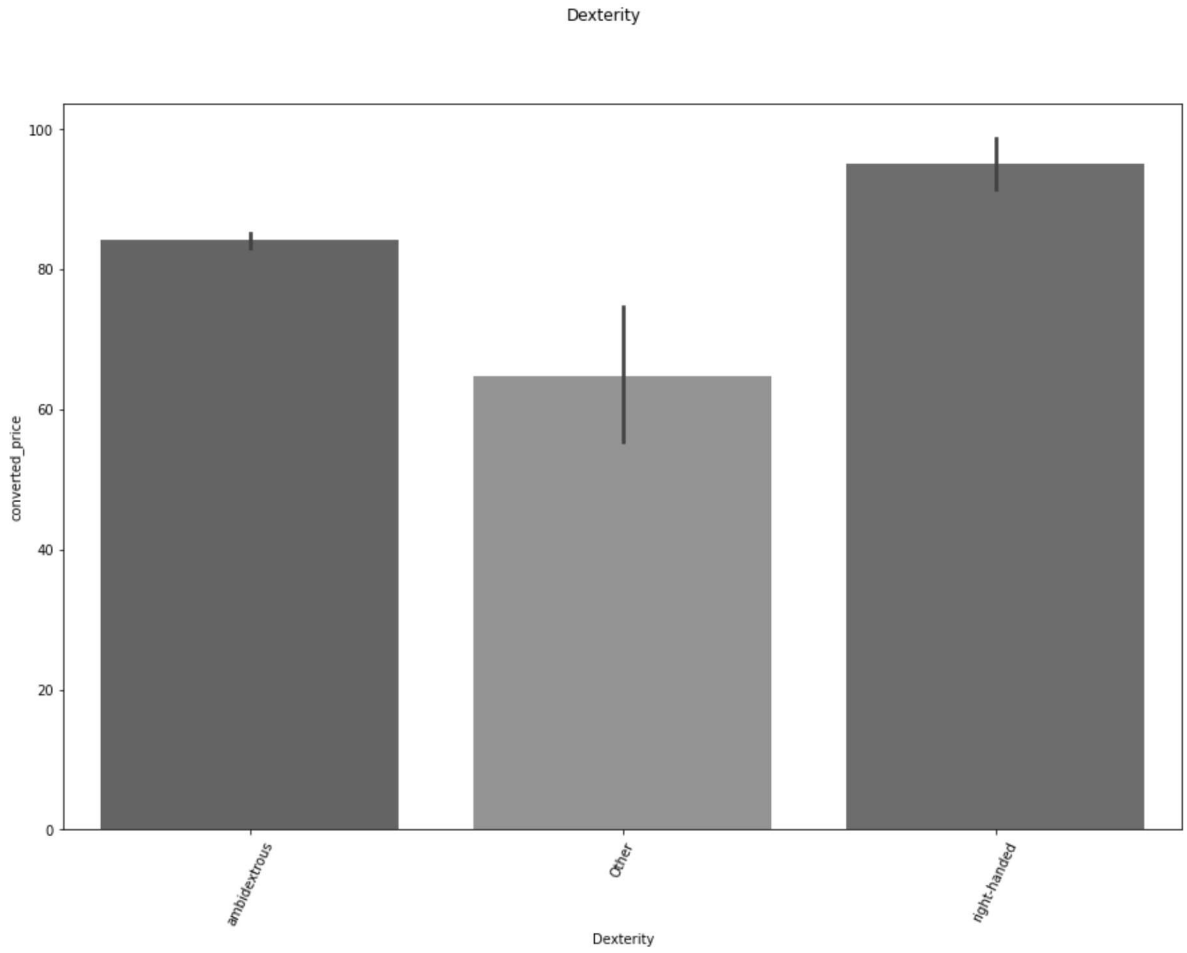

In [54]:
```
df_listed.columns
```

Out[54]:
```
Index(['itemId', 'title', 'galleryURL', 'viewItemURL', 'autoPay', 'postalCode',
       'returnsAccepted', 'condition', 'topRatedListing', 'pictureURLLarge',
       'pictureURLSuperSize', 'converted_price', 'brand', 'profit', 'ROI',
       'PictureURL', 'Location', 'Country', 'Blade Material', 'Model',
       'Opening Mechanism', 'Number of Blades', 'Handle Material',
       'Blade Type', 'Color', 'Country/Region of Manufacture', 'Blade Edge',
       'Lock Type', 'Blade Range', 'Dexterity', 'MPN', 'Year',
       'State_or_Province', 'construction_year'],
      dtype='object')
```

In [55]:
```
transformed_dex = cardinality_threshold(df_listed['Dexterity'],
                                        threshold=0.99,
                                        return_categories_list=False)
transformed_dex.value_counts()
```

Out[55]:
```
ambidextrous    10462
right-handed     1113
Other              88
Name: Dexterity, dtype: int64
```

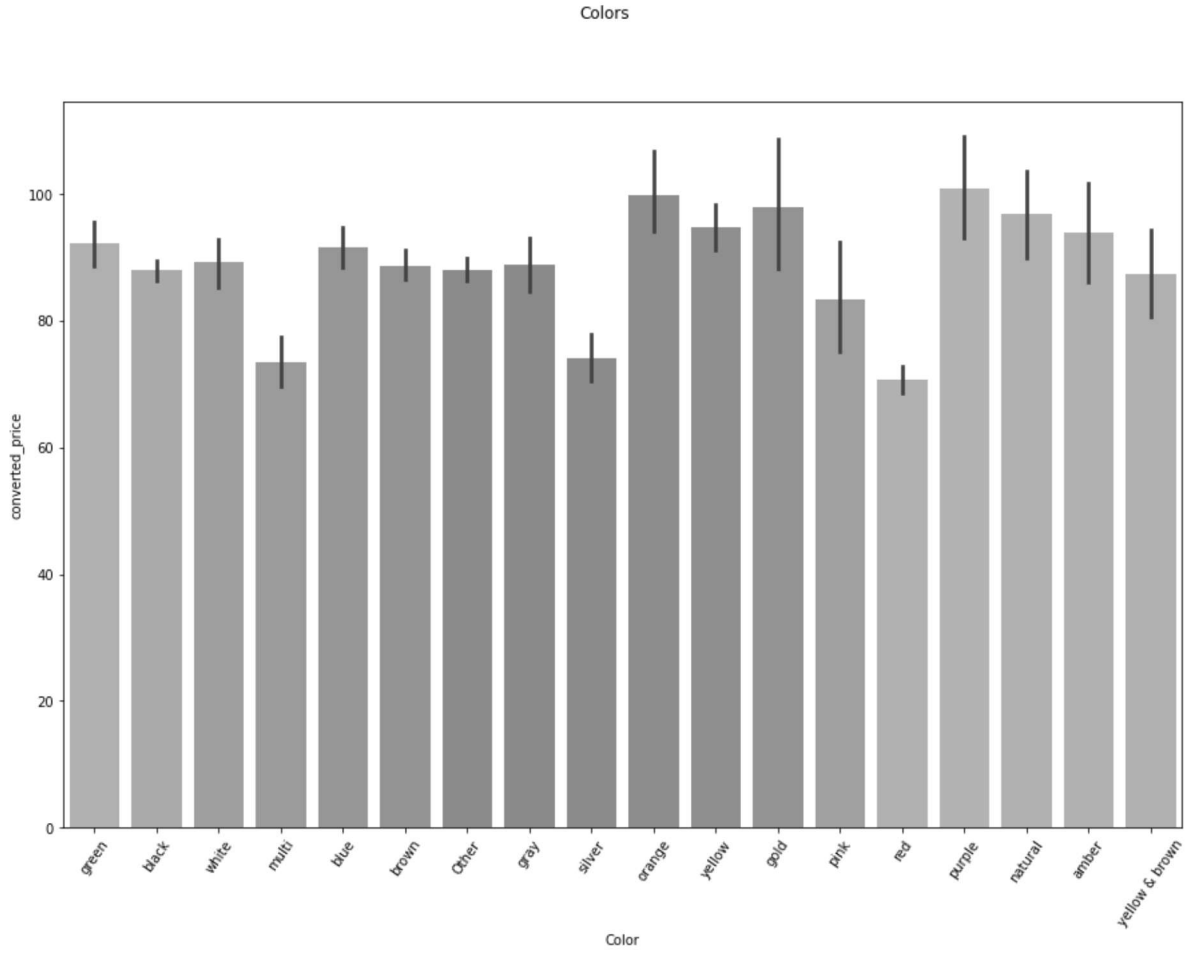
In [56]:
```
df_transformed = df_listed.copy()
df_transformed['Dexterity'] = transformed_dex
# df_sorted = df_transformed.sort_values('converted_price')

fig = plt.figure(figsize=(15,10))
fig.suptitle('Dexterity')
sns.barplot(x= 'Dexterity', y='converted_price', data=df_transformed)
plt.xticks(rotation=65)
plt.show();
```



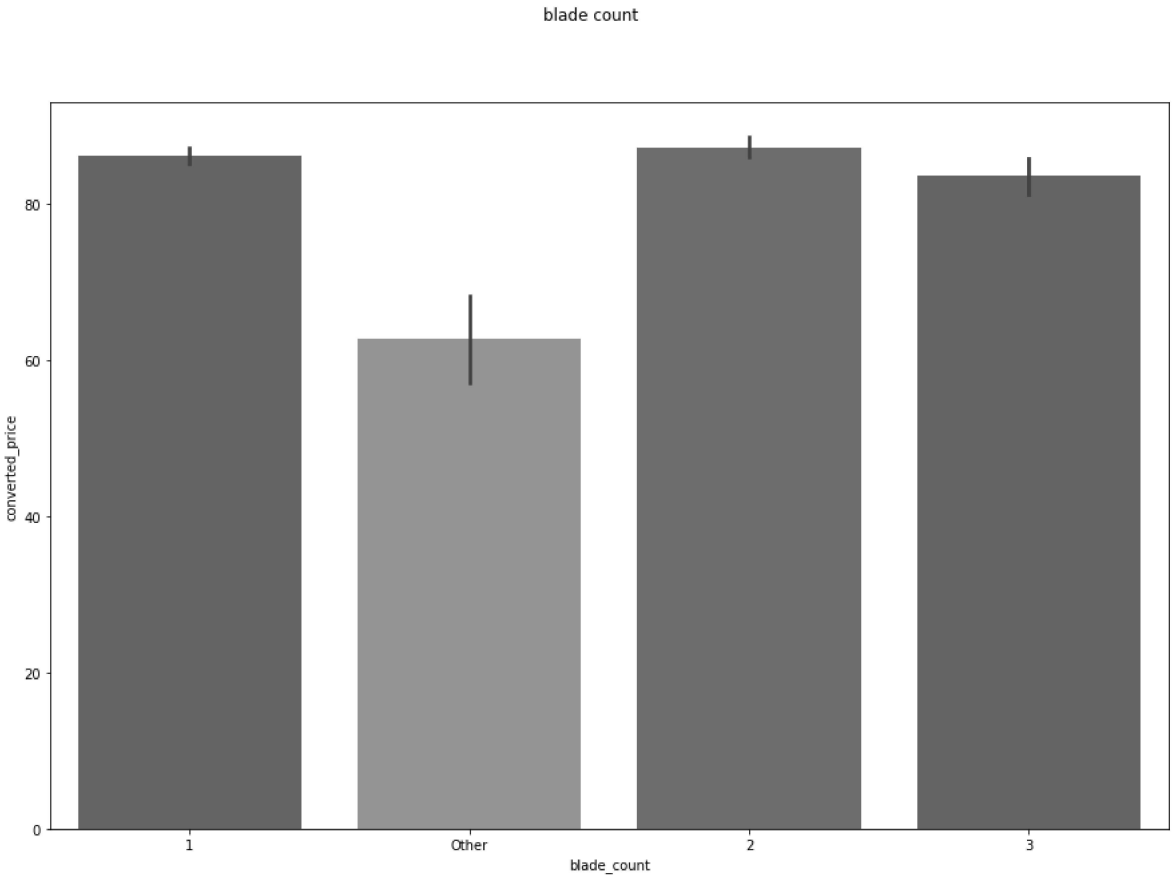

In [57]:
```
transformed_color = cardinality_threshold(df_listed['Color'],
                                          threshold=0.88,
                                          return_categories_list=False)
transformed_color.value_counts()
```

Out[57]:
```
black             5937
Other             3601
red               2375
brown             1804
blue              1448
green             1078
silver             921
gray               889
multi              745
yellow             732
white              655
orange             380
purple             208
natural            192
pink               184
gold               146
yellow & brown     130
amber              130
Name: Color, dtype: int64
```

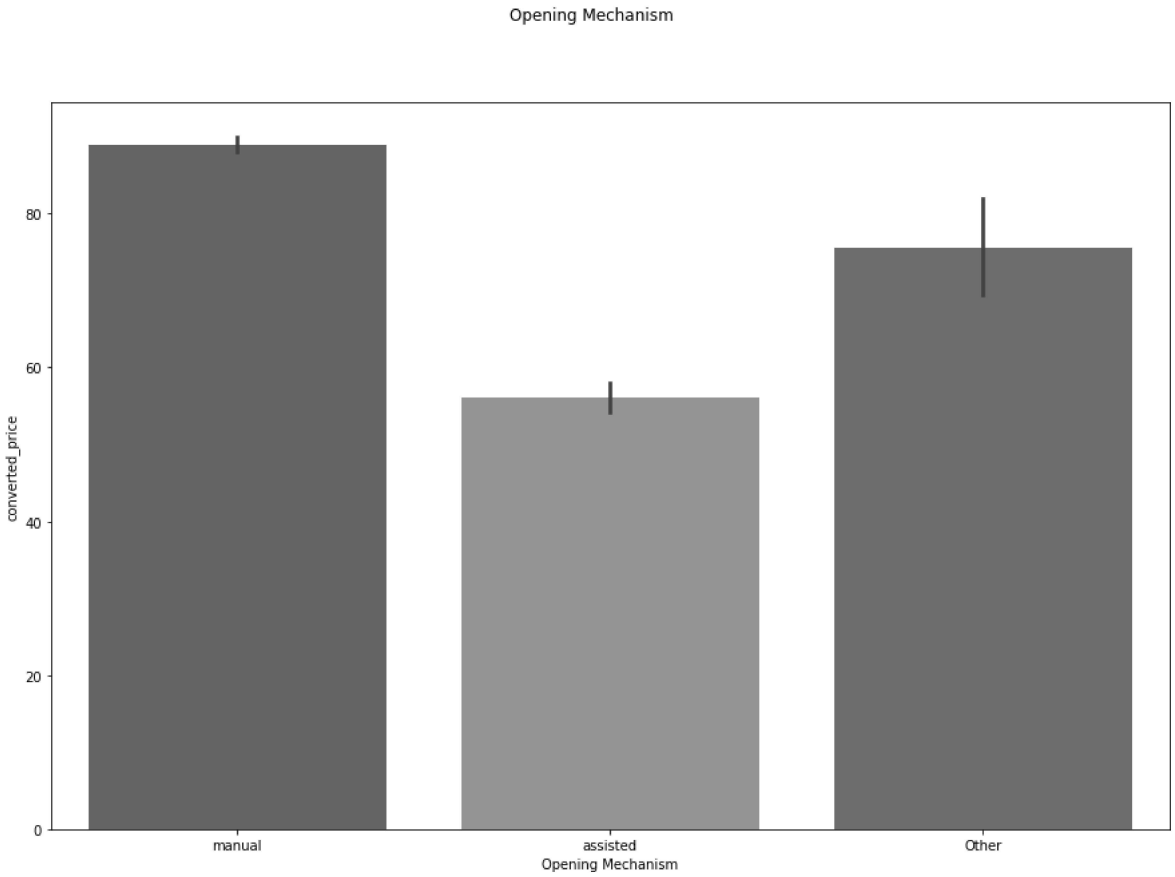
In [58]:
```
df_transformed = df_listed.copy()
df_transformed['Color'] = transformed_color
# df_sorted = df_transformed.sort_values('converted_price')

fig = plt.figure(figsize=(15,10))
fig.suptitle('Colors')
sns.barplot(x= 'Color', y='converted_price', data=df_transformed)
plt.xticks(rotation=55)
plt.show();
```



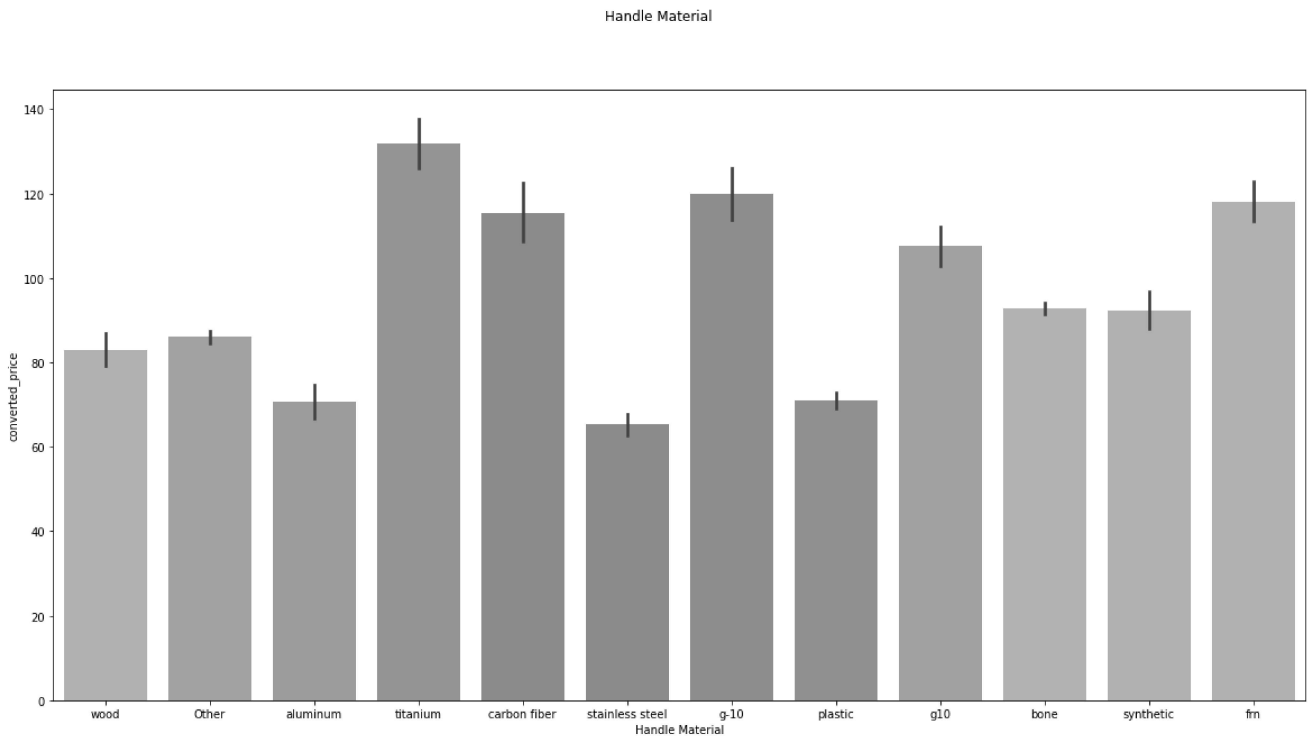
In [59]:
```
transformed_blade_count = cardinality_threshold(df_listed['Number of Blades'],
                                                threshold=0.97,
                                                return_categories_list=False)
transformed_blade_count.value_counts()
df_transformed = df_listed.copy()
df_transformed['blade_count'] = transformed_blade_count


fig = plt.figure(figsize=(15,10))
fig.suptitle('blade count')
sns.barplot(x= 'blade_count', y='converted_price', data=df_transformed)
plt.show();
```



In [60]:
```
transformed_OM = cardinality_threshold(df_listed['Opening Mechanism'],
                                       threshold=0.95,
                                       return_categories_list=False)

df_transformed = df_listed.copy()
df_transformed['Opening Mechanism'] = transformed_OM


fig = plt.figure(figsize=(15,10))
fig.suptitle('Opening Mechanism')
sns.barplot(x= 'Opening Mechanism', y='converted_price', data=df_transformed)
plt.show();
```



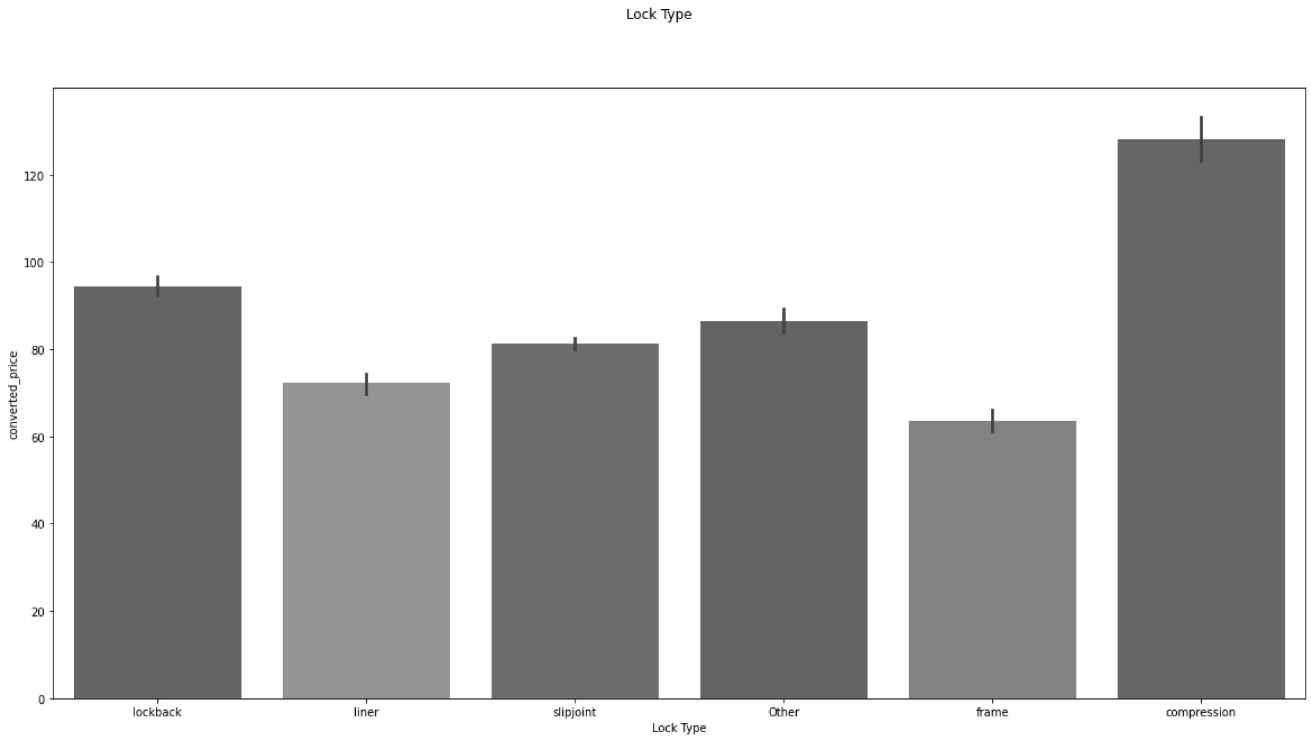
In [61]:
```
transformed_HM = cardinality_threshold(df_listed['Handle Material'],
                                       threshold=0.8,
                                       return_categories_list=False)

df_transformed = df_listed.copy()
df_transformed['Handle Material'] = transformed_HM


fig = plt.figure(figsize=(20,10))
fig.suptitle('Handle Material')
sns.barplot(x= 'Handle Material', y='converted_price', data=df_transformed)
plt.show();
```



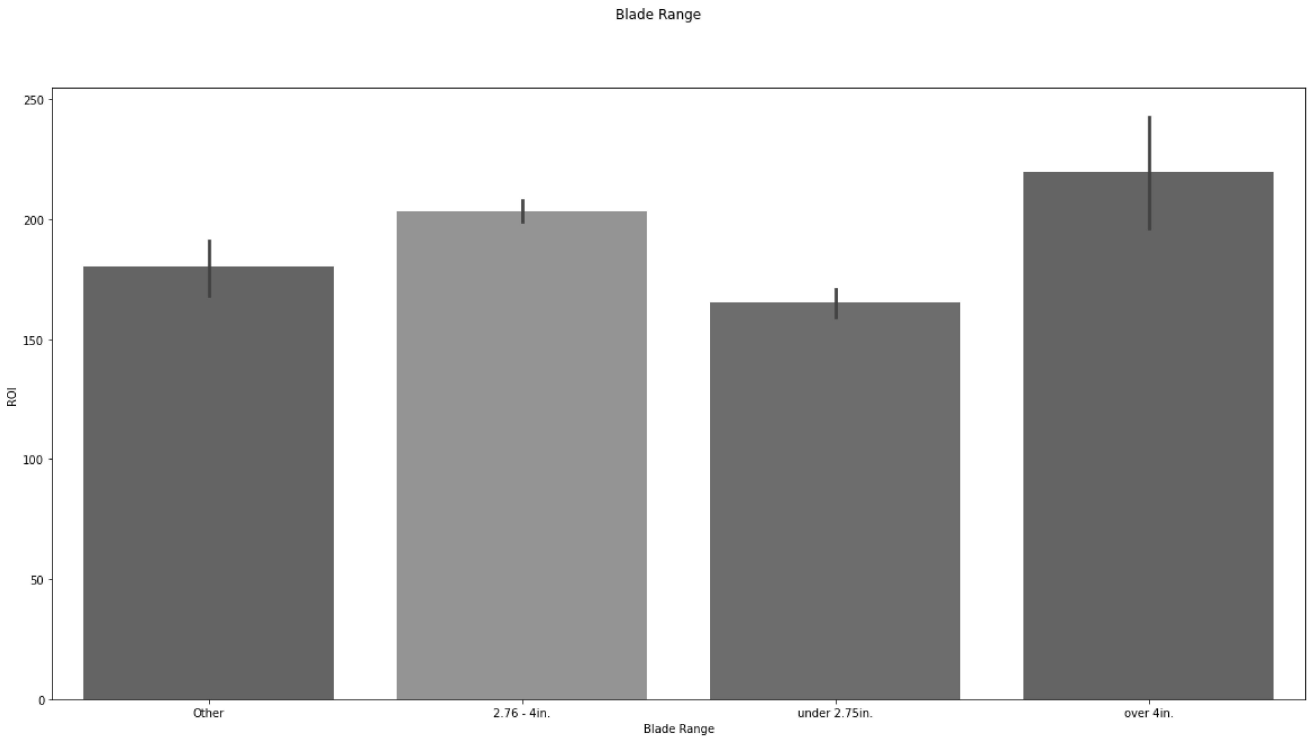

In [62]:
```
df_listed.columns
```

Out[62]:
```
Index(['itemId', 'title', 'galleryURL', 'viewItemURL', 'autoPay', 'postalCode',
       'returnsAccepted', 'condition', 'topRatedListing', 'pictureURLLarge',
       'pictureURLSuperSize', 'converted_price', 'brand', 'profit', 'ROI',
       'PictureURL', 'Location', 'Country', 'Blade Material', 'Model',
       'Opening Mechanism', 'Number of Blades', 'Handle Material',
       'Blade Type', 'Color', 'Country/Region of Manufacture', 'Blade Edge',
       'Lock Type', 'Blade Range', 'Dexterity', 'MPN', 'Year',
       'State_or_Province', 'construction_year'],
      dtype='object')
```

In [63]:
```
transformed_LT = cardinality_threshold(df_listed['Lock Type'],
                                       threshold=0.93,
                                       return_categories_list=False)


df_transformed = df_listed.copy()
df_transformed['Lock Type'] = transformed_LT


fig = plt.figure(figsize=(20,10))
fig.suptitle('Lock Type')
sns.barplot(x= 'Lock Type', y='converted_price', data=df_transformed)
plt.show();
```



In [64]:
```
transformed_BR = cardinality_threshold(df_listed['Blade Range'],
                                       threshold=0.97,
                                       return_categories_list=False)


df_transformed = df_listed.copy()
df_transformed['Blade Range'] = transformed_BR


fig = plt.figure(figsize=(20,10))
fig.suptitle('Blade Range')
sns.barplot(x= 'Blade Range', y='ROI', data=df_transformed)
plt.show();
```




### Sold Data: Scraped from the Terapeak website

This is a larger dataset containing listings that go back 2 years instead of 90 days. The data was also filtered to only include sold data of used knives. That means all prices in the dataset reflect the true final sale price of each item.

In [65]:
```
sold_df = pd.read_csv('terapeak_data/sold_df.csv')
```

In [66]:
```
sold_df.brand.value_counts()
```

Out[66]:
```
case          17337
buck          11384
kershaw       10766
victorinox     9486
spyderco       6165
benchmade      5792
crkt           4292
sog            3018
Name: brand, dtype: int64
```

In [67]:
```
sold_df.drop(['price_in_US',
              'shipping_cost'],
             axis=1, inplace=True)
```

In [68]:
```
df_sold = apply_iqr_filter(sold_df).copy()
```

In [69]:
```
used_listed = listed_df.loc[listed_df['condition'] != 1000]
```

In [70]:
```
used_listed.reset_index(drop=True,inplace=True)
```

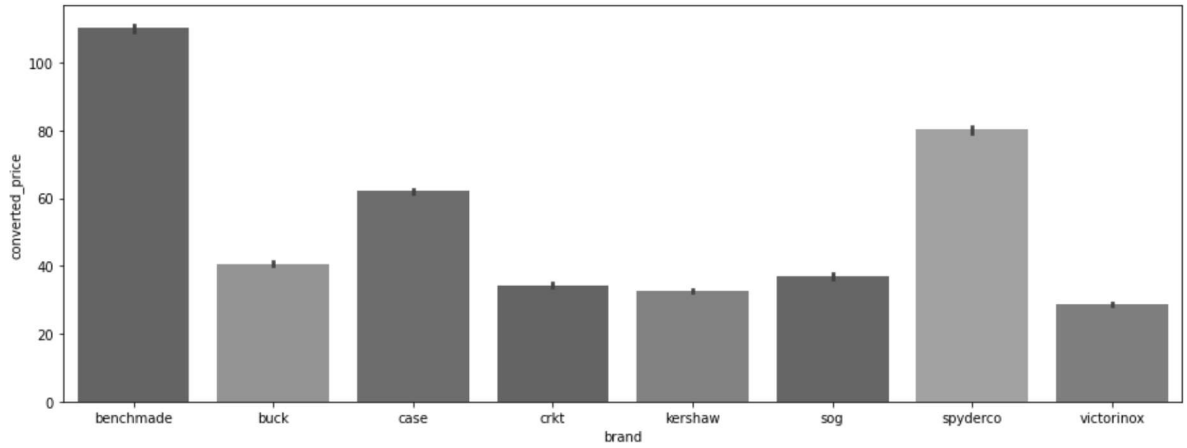
In [71]:
```
used_listed.columns
```

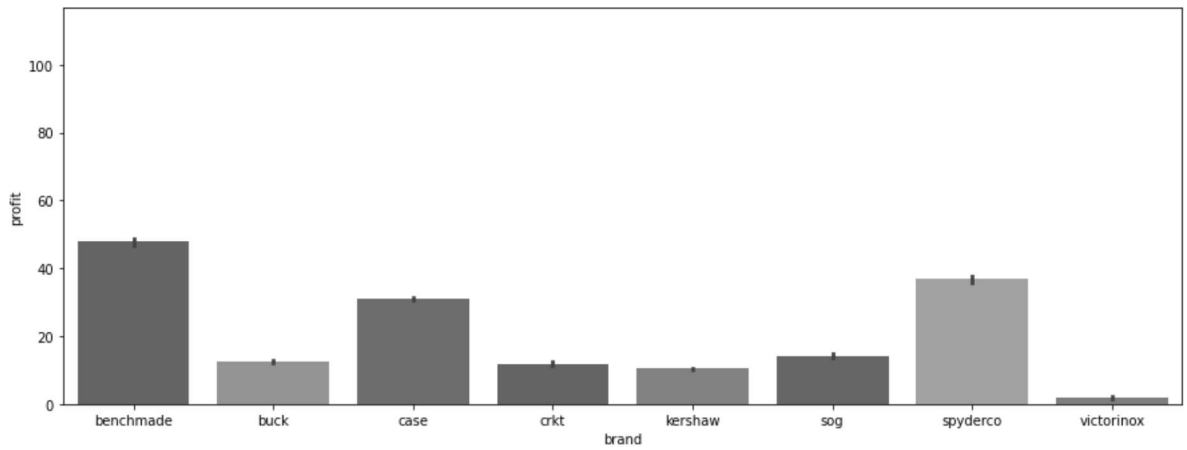
Out[71]:
```
Index(['itemId', 'title', 'galleryURL', 'viewItemURL', 'autoPay', 'postalCode',
       'returnsAccepted', 'condition', 'topRatedListing', 'pictureURLLarge',
       'pictureURLSuperSize', 'shipping_cost', 'price_in_US',
       'converted_price', 'brand', 'cost', 'profit', 'ROI', 'PictureURL',
       'Location', 'Country', 'Blade Material', 'Model', 'Opening Mechanism',
       'Number of Blades', 'Handle Material', 'Blade Type', 'specBrand',
       'Color', 'Type', 'Country/Region of Manufacture', 'Blade Edge',
       'Lock Type', 'Original/Reproduction', 'Blade Range', 'Dexterity', 'MPN',
       'Year'],
      dtype='object')
```

In [72]:
```
cols = ['title','pictureURLLarge','converted_price','brand','profit','ROI']
used_listed2 = used_listed[cols].copy()
df1 = pd.concat([sold_df, used_listed2]).copy()
df1['Image'].fillna(df1['pictureURLLarge'], inplace=True)
```

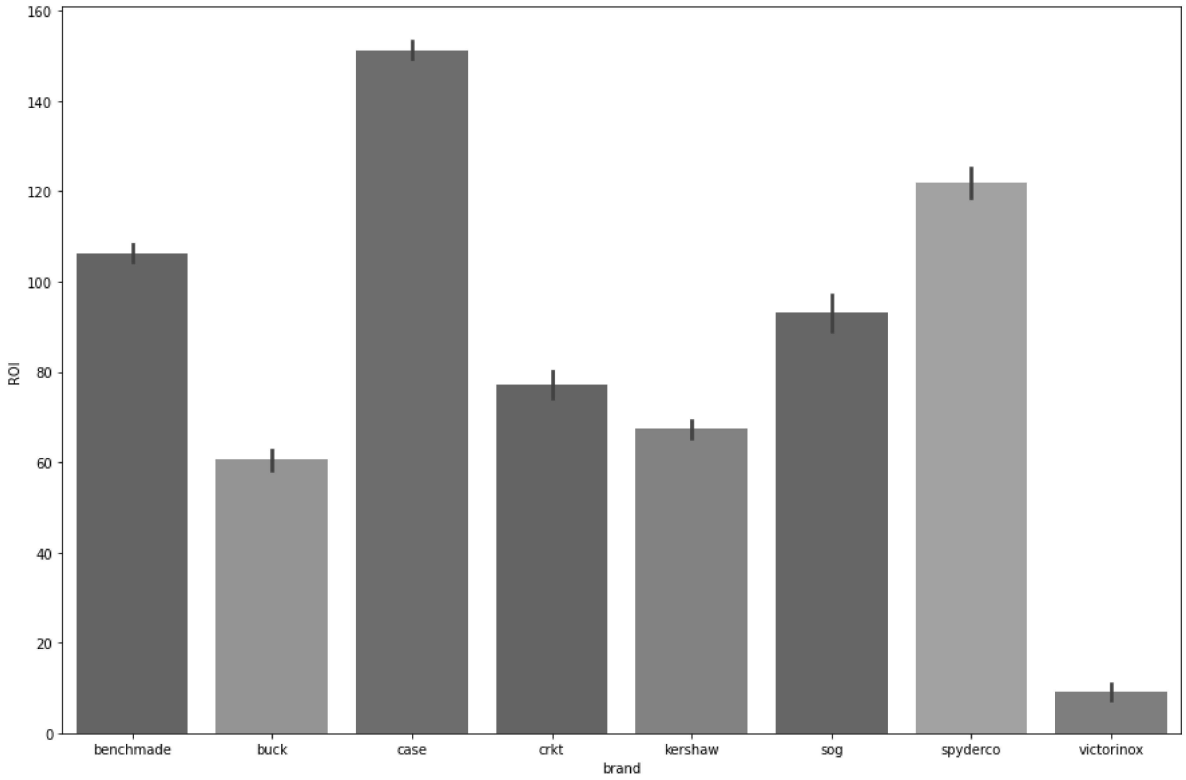
In [73]:
```
df = apply_iqr_filter(df1).copy()
```

In [74]:
```
fig, axes  = plt.subplots(figsize=(15,12),nrows=2, sharey=True)
sns.barplot(x= 'brand', y='converted_price', ax=axes[0],data=df)
sns.barplot(x= 'brand', y='profit', ax=axes[1],data=df)
plt.show();
```

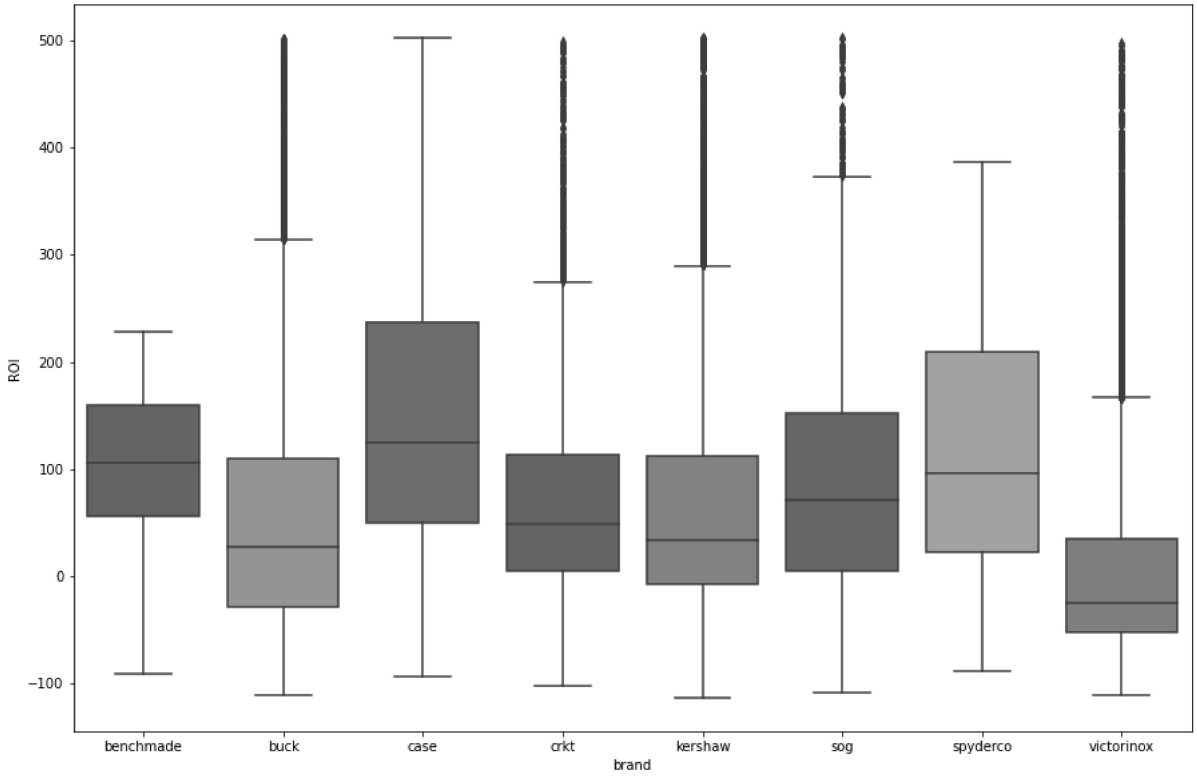


In [75]:
```
fig, axes  = plt.subplots(figsize=(15,10),nrows=1)
sns.barplot(x= 'brand', y='ROI',data=df)
plt.show();
```



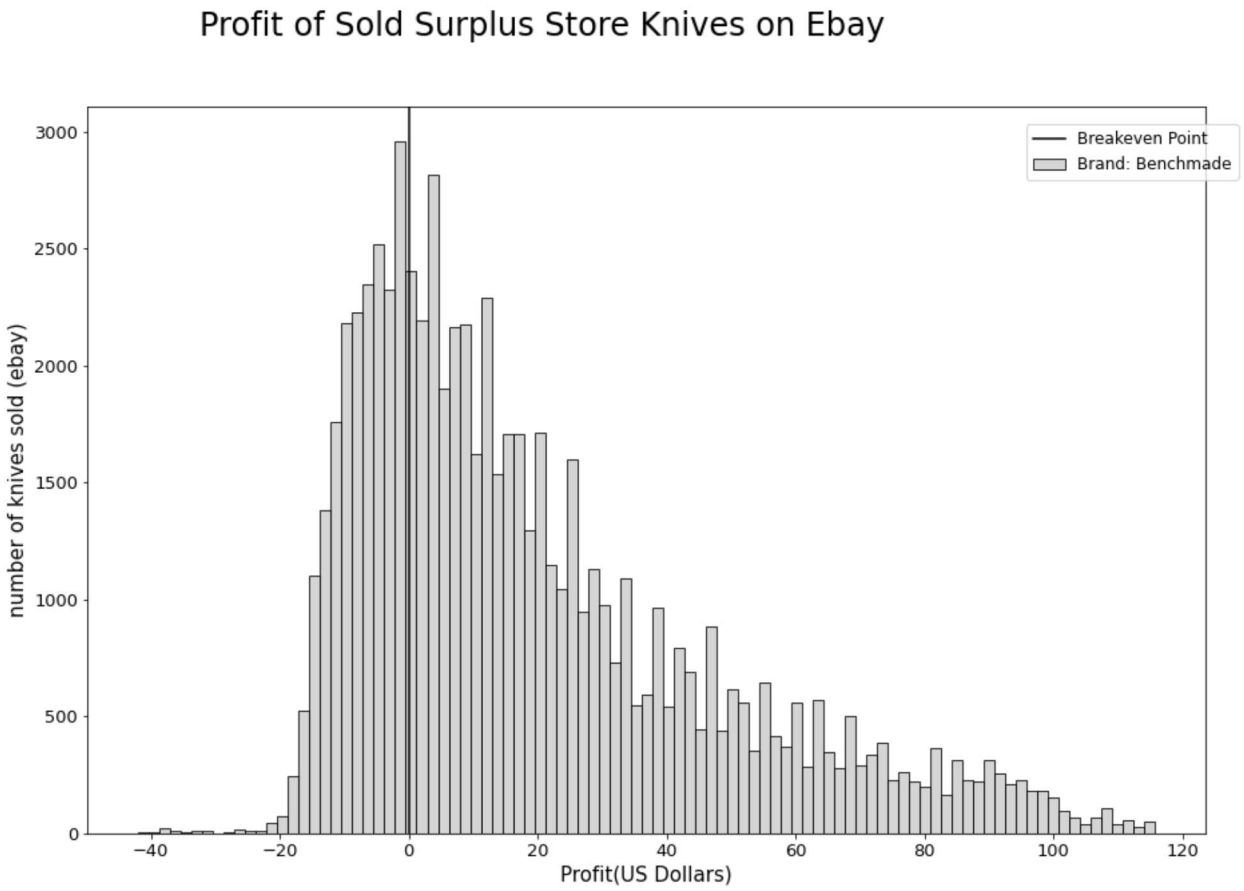

In [76]:
```
fig, axes  = plt.subplots(figsize=(15,10),nrows=1)
sns.boxplot(x= 'brand', y='ROI',data=df)
plt.show();
```



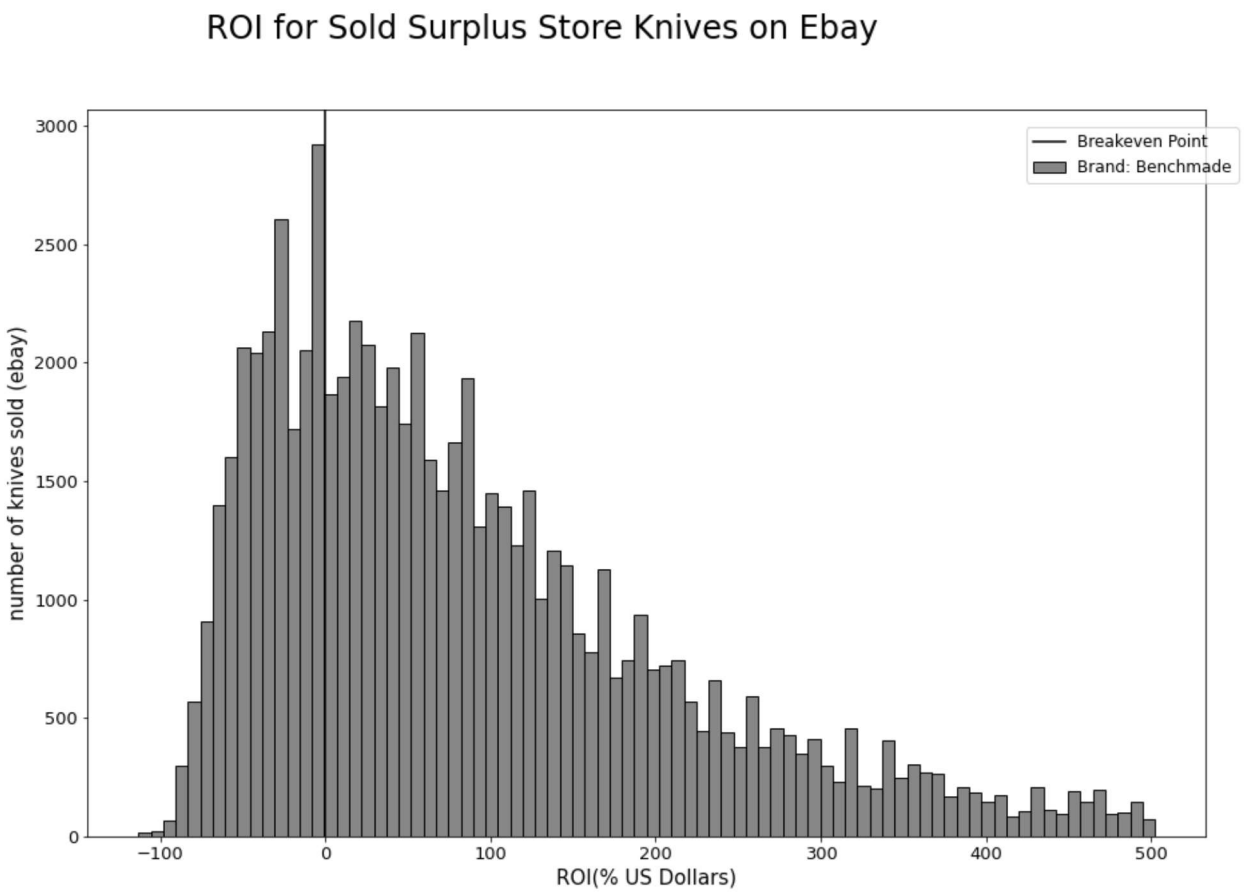
In [77]:
```
bench_sold = df.loc[df['brand'] == 'benchmade'].copy()
buck_sold = df.loc[df['brand'] == 'buck'].copy()
case_sold = df.loc[df['brand'] == 'case'].copy()
crkt_sold = df.loc[df['brand'] == 'crkt'].copy()
kershaw_sold = df.loc[df['brand'] == 'kershaw'].copy()
sog_sold = df.loc[df['brand'] == 'sog'].copy()
spyd_sold = df.loc[df['brand'] == 'spyderco'].copy()
vict_sold = df.loc[df['brand'] == 'victorinox'].copy()
```

In [78]:
```
fig, axes  = plt.subplots(figsize=(15,10), ncols=1)

sns.histplot(df['converted_price'], color='skyblue', label='Brand: Benchmade');
axes.set_xlabel('Price Sold(US Dollars)', fontsize=15)
axes.set_ylabel('number of knives sold (ebay)', fontsize=15)
axes.tick_params(axis='both', labelsize=13)


fig.suptitle("Price of Sold Surplus Store Knives on Ebay", fontsize=24, x=0.44)
fig.legend(loc=(.8, .80), fontsize='large')
plt.show();
```

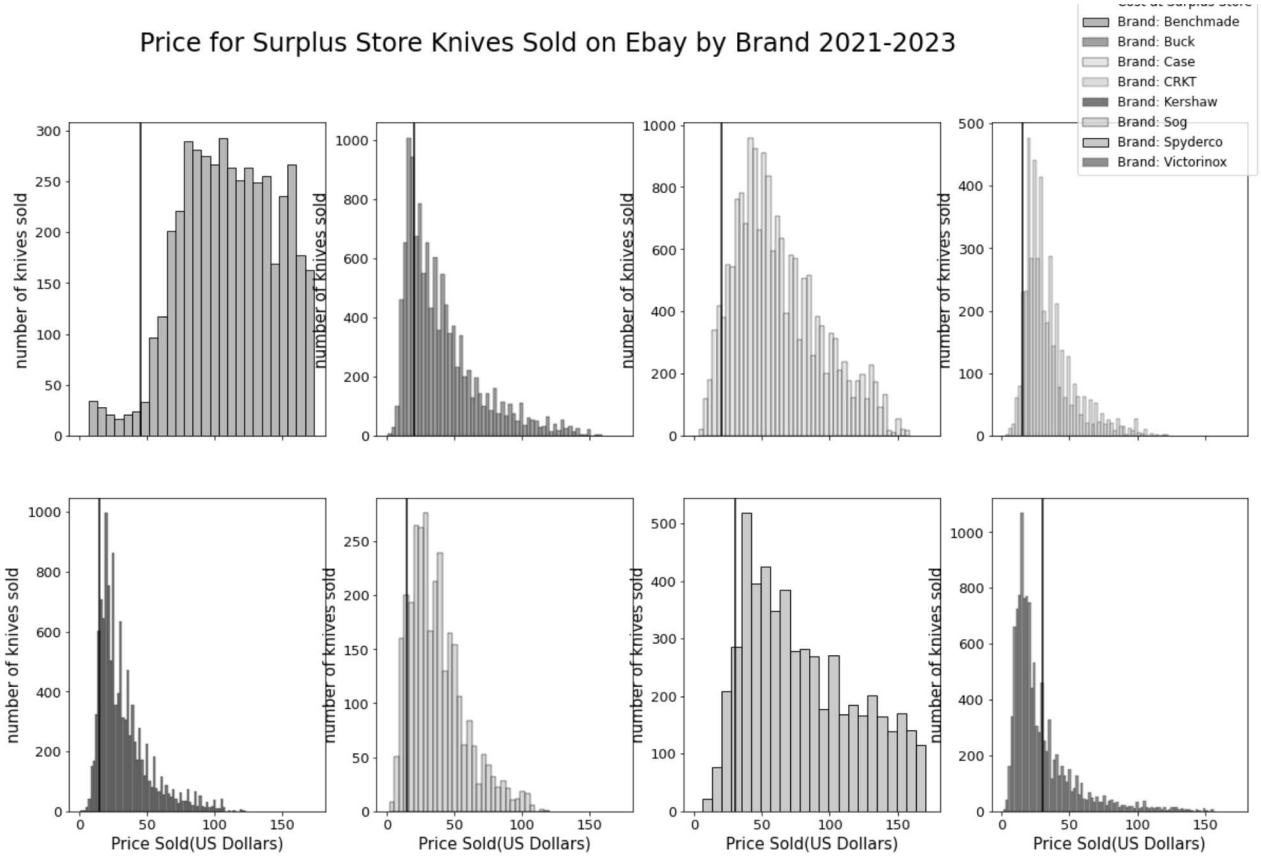
In [79]:
```
fig, axes  = plt.subplots(figsize=(15,10), ncols=1)

sns.histplot(df['profit'], color='lightgreen', label='Brand: Benchmade');
axes.set_xlabel('Profit(US Dollars)', fontsize=15)
axes.set_ylabel('number of knives sold (ebay)', fontsize=15)
axes.tick_params(axis='both', labelsize=13)

axes.axvline(x = 0, color = 'black', label= 'Breakeven Point')
fig.suptitle("Profit of Sold Surplus Store Knives on Ebay", fontsize=24, x=0.44)
fig.legend(loc=(.8, .80), fontsize='large')
plt.show();
```



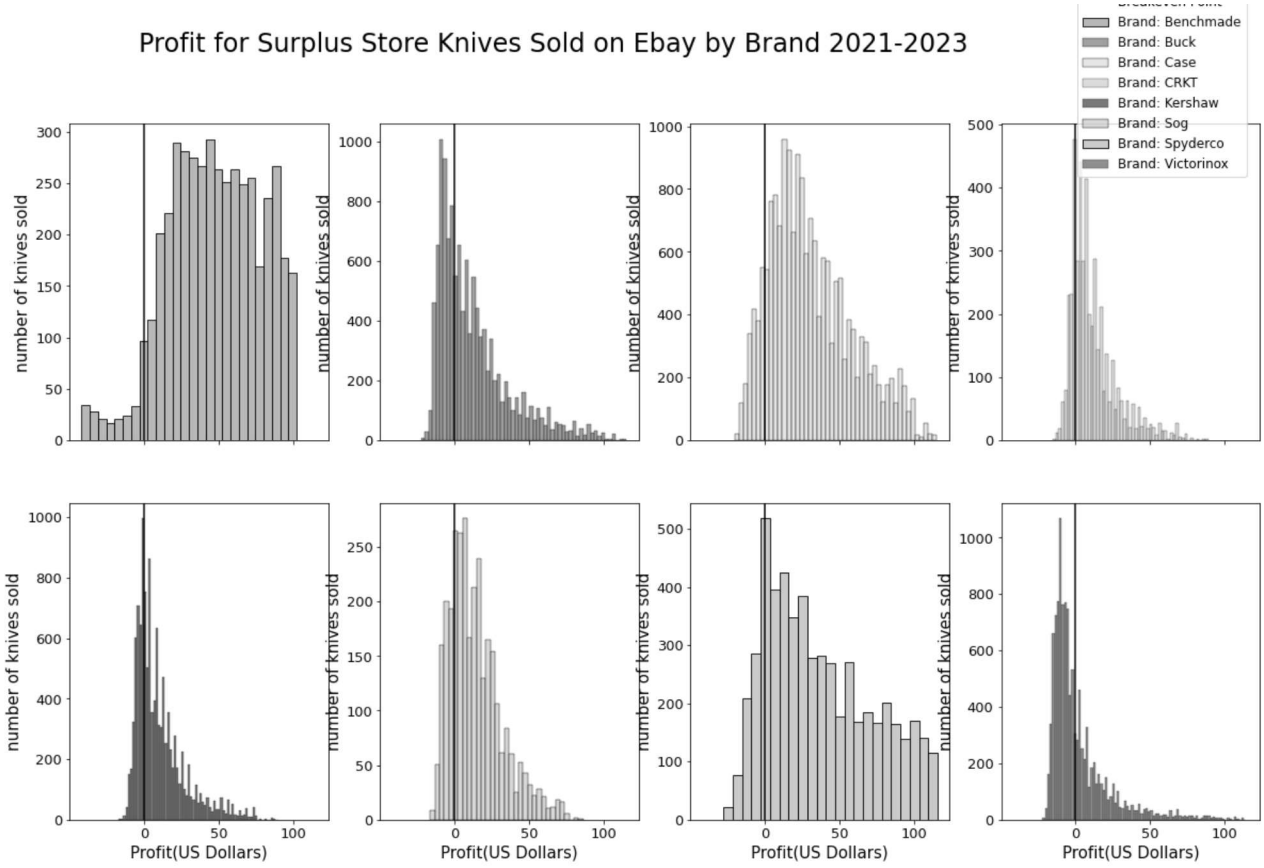
In [80]:
```
fig, axes  = plt.subplots(figsize=(15,10), ncols=1)

sns.histplot(df['ROI'], color='forestgreen', label='Brand: Benchmade');
axes.set_xlabel('ROI(% US Dollars)', fontsize=15)
axes.set_ylabel('number of knives sold (ebay)', fontsize=15)
axes.tick_params(axis='both', labelsize=13)

axes.axvline(x = 0, color = 'black', label= 'Breakeven Point')
fig.suptitle("ROI for Sold Surplus Store Knives on Ebay", fontsize=24, x=0.44)
fig.legend(loc=(.8, .80), fontsize='large')
plt.show();
```



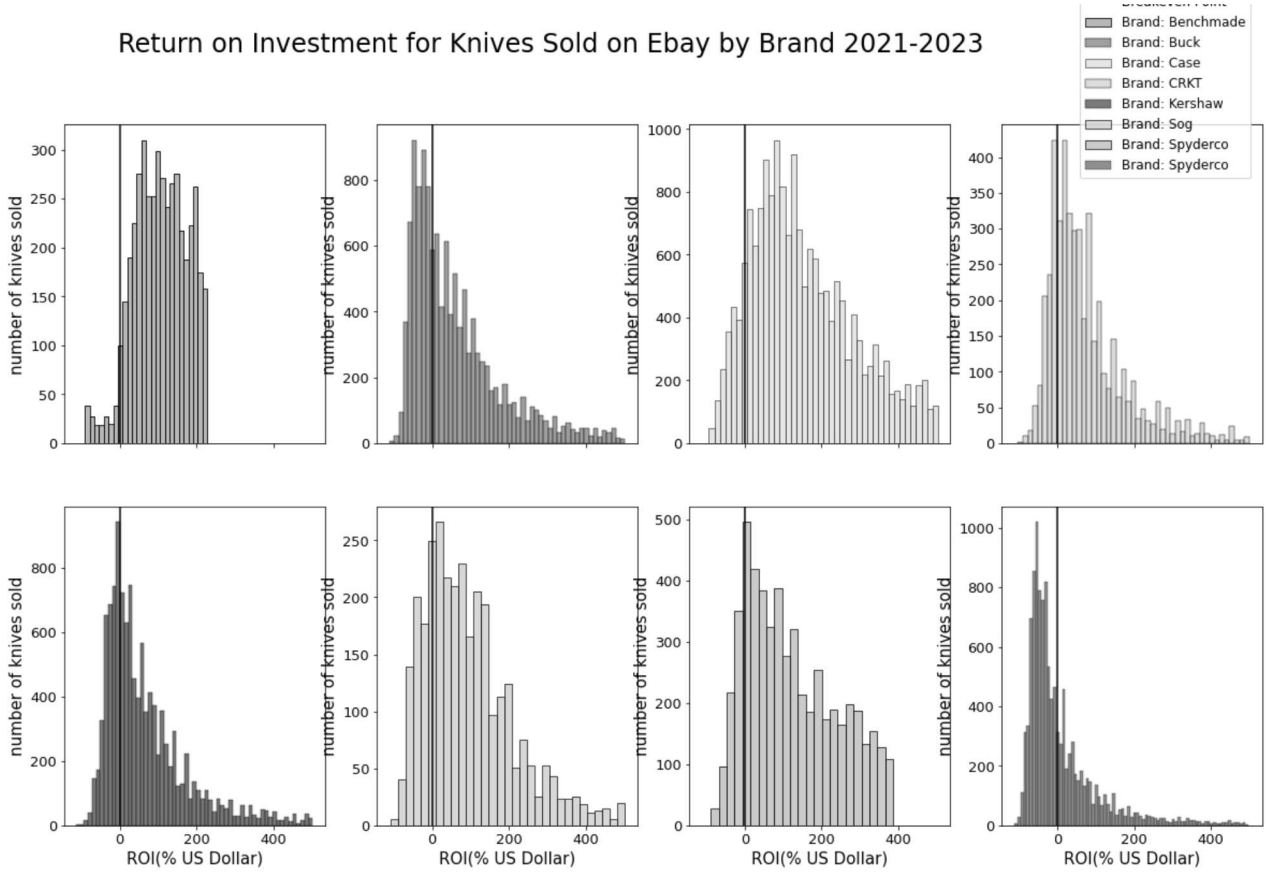
In [81]:
```
fig, axes  = plt.subplots(figsize=(20,12), ncols=4, nrows=2, sharex=True)
sns.histplot(bench_sold['converted_price'], ax=axes[0][0], color='lightcoral', label='Brand: Benchmade')
sns.histplot(buck_sold['converted_price'], ax=axes[0][1], color='chocolate', label='Brand: Buck')
sns.histplot(case_sold['converted_price'], ax=axes[0][2],  color='khaki', label='Brand: Case')
sns.histplot(crkt_sold['converted_price'], ax=axes[0][3], color='palegreen', label='Brand: CRKT')
sns.histplot(kershaw_sold['converted_price'], ax=axes[1][0], color='firebrick', label='Brand: Kershaw')
sns.histplot(sog_sold['converted_price'], ax=axes[1][1],  color='lightblue', label='Brand: Sog')
sns.histplot(spyd_sold['converted_price'], ax=axes[1][2],  color='plum', label='Brand: Spyderco')
sns.histplot(vict_sold['converted_price'], ax=axes[1][3],  color='royalblue', label='Brand: Victorinox')


for n in range(8):
    row = n//4
    col = n%4
    axes[row][col].set_xlabel('Price Sold(US Dollars)', fontsize=15)
    axes[row][col].set_ylabel('number of knives sold', fontsize=15)
    axes[row][col].tick_params(axis='both', labelsize=13)

axes[0][0].axvline(x = 45, color = 'black', label= 'Cost at Surplus Store')
axes[0][1].axvline(x = 20, color = 'black')
axes[0][2].axvline(x = 20, color = 'black')
axes[0][3].axvline(x = 15, color = 'black')
axes[1][0].axvline(x = 15, color = 'black')
axes[1][1].axvline(x = 15, color = 'black')
axes[1][2].axvline(x = 30, color = 'black')
axes[1][3].axvline(x = 30, color = 'black')
fig.suptitle("Price for Surplus Store Knives Sold on Ebay by Brand 2021-2023", fontsize=24, x=0.44)
fig.legend(loc=(.82, .8), fontsize='large')
plt.show();
```



In [82]:
```
fig, axes  = plt.subplots(figsize=(20,12), ncols=4, nrows=2, sharex=True)
sns.histplot(bench_sold['profit'], ax=axes[0][0], color='lightcoral', label='Brand: Benchmade')
sns.histplot(buck_sold['profit'], ax=axes[0][1], color='chocolate', label='Brand: Buck')
sns.histplot(case_sold['profit'], ax=axes[0][2],  color='khaki', label='Brand: Case')
sns.histplot(crkt_sold['profit'], ax=axes[0][3], color='palegreen', label='Brand: CRKT')
sns.histplot(kershaw_sold['profit'], ax=axes[1][0], color='firebrick', label='Brand: Kershaw')
sns.histplot(sog_sold['profit'], ax=axes[1][1],  color='lightblue', label='Brand: Sog')
sns.histplot(spyd_sold['profit'], ax=axes[1][2],  color='plum', label='Brand: Spyderco')
sns.histplot(vict_sold['profit'], ax=axes[1][3],  color='royalblue', label='Brand: Victorinox')

for n in range(8):
    row = n//4
    col = n%4
    axes[row][col].set_xlabel('Profit(US Dollars)', fontsize=15)
    axes[row][col].set_ylabel('number of knives sold', fontsize=15)
    axes[row][col].tick_params(axis='both', labelsize=13)

axes[0][0].axvline(x = 0, color = 'black', label= 'Breakeven Point')
axes[0][1].axvline(x = 0, color = 'black')
axes[0][2].axvline(x = 0, color = 'black')
axes[0][3].axvline(x = 0, color = 'black')
axes[1][0].axvline(x = 0, color = 'black')
axes[1][1].axvline(x = 0, color = 'black')
axes[1][2].axvline(x = 0, color = 'black')
axes[1][3].axvline(x = 0, color = 'black')
fig.suptitle("Profit for Surplus Store Knives Sold on Ebay by Brand 2021-2023", fontsize=24, x=0.44)
fig.legend(loc=(.82, .8), fontsize='large')
plt.show();
```



In [83]:
```
fig, axes  = plt.subplots(figsize=(20,12), ncols=4, nrows=2, sharex=True)
sns.histplot(bench_sold['ROI'], ax=axes[0][0], color='lightcoral', label='Brand: Benchmade')
sns.histplot(buck_sold['ROI'], ax=axes[0][1], color='chocolate', label='Brand: Buck')
sns.histplot(case_sold['ROI'], ax=axes[0][2],  color='khaki', label='Brand: Case')
sns.histplot(crkt_sold['ROI'], ax=axes[0][3], color='palegreen', label='Brand: CRKT')
sns.histplot(kershaw_sold['ROI'], ax=axes[1][0], color='firebrick', label='Brand: Kershaw')
sns.histplot(sog_sold['ROI'], ax=axes[1][1],  color='lightblue', label='Brand: Sog')
sns.histplot(spyd_sold['ROI'], ax=axes[1][2],  color='plum', label='Brand: Spyderco')
sns.histplot(vict_sold['ROI'], ax=axes[1][3],  color='royalblue', label='Brand: Spyderco')


for n in range(8):
    row = n//4
    col = n%4
    axes[row][col].set_xlabel('ROI(% US Dollar)', fontsize=15)
    axes[row][col].set_ylabel('number of knives sold', fontsize=15)
    axes[row][col].tick_params(axis='both', labelsize=13)

axes[0][0].axvline(x = 0, color = 'black', label= 'Breakeven Point')
axes[0][1].axvline(x = 0, color = 'black')
axes[0][2].axvline(x = 0, color = 'black')
axes[0][3].axvline(x = 0, color = 'black')
axes[1][0].axvline(x = 0, color = 'black')
axes[1][1].axvline(x = 0, color = 'black')
axes[1][2].axvline(x = 0, color = 'black')
axes[1][3].axvline(x = 0, color = 'black')
fig.suptitle("Return on Investment for Knives Sold on Ebay by Brand 2021-2023", fontsize=24, x=0.44)
fig.legend(loc=(.82, .8), fontsize='large')
plt.show();
```



In [84]:
```
display(bench_sold['brand'].values[0])
display(bench_sold[['converted_price',
                    'profit', 'ROI']].describe())

display(buck_sold['brand'].values[0])
display(buck_sold[['converted_price',
                    'profit', 'ROI']].describe())

display(case_sold['brand'].values[0])
display(case_sold[['converted_price',
                    'profit', 'ROI']].describe())


display(crkt_sold['brand'].values[0])
display(crkt_sold[['converted_price',
                    'profit', 'ROI']].describe())


display(kershaw_sold['brand'].values[0])
display(kershaw_sold[['converted_price',
                    'profit', 'ROI']].describe())


display(sog_sold['brand'].values[0])
display(sog_sold[['converted_price',
                    'profit', 'ROI']].describe())


display(spyd_sold['brand'].values[0])
display(spyd_sold[['converted_price',
                    'profit', 'ROI']].describe())


display(vict_sold['brand'].values[0])
display(vict_sold[['converted_price',
                    'profit', 'ROI']].describe())
```

'benchmade'

| | converted_price | profit | ROI |
|---|---|---|---|
| count | 4512.000000 | 4512.000000 | 4512.000000 |
| mean | 110.186279 | 47.862063 | 106.231893 |
| std | 35.188949 | 30.614386 | 67.879611 |
| min | 6.990000 | -41.918700 | -90.233333 |
| 25% | 84.500000 | 25.515000 | 56.700000 |
| 50% | 110.000000 | 47.700000 | 106.000000 |
| 75% | 138.000000 | 72.060000 | 159.993167 |
| max | 173.000000 | 102.510000 | 227.800000 |

'buck'

| | converted_price | profit | ROI |
|---|---|---|---|
| count | 12110.000000 | 12110.000000 | 12110.000000 |
| mean | 40.734419 | 12.438944 | 60.587833 |
| std | 27.823154 | 24.206144 | 117.633410 |
| min | 0.990000 | -22.138700 | -110.693500 |
| 25% | 20.000000 | -5.600000 | -28.000000 |
| 50% | 32.950000 | 5.666500 | 28.006250 |
| 75% | 51.990000 | 22.231300 | 109.242500 |
| max | 159.000000 | 115.330000 | 501.434783 |

'case'

| | converted_price | profit | ROI |
|---|---|---|---|
| count | 17708.000000 | 17708.000000 | 17708.000000 |
| mean | 62.032045 | 30.967880 | 151.279894 |
| std | 31.824116 | 27.686981 | 133.952429 |
| min | 3.500000 | -19.955000 | -93.293500 |
| 25% | 38.100000 | 10.147000 | 50.300000 |
| 50% | 55.450000 | 25.241500 | 125.070859 |
| 75% | 81.862500 | 48.220375 | 237.132500 |
| max | 159.000000 | 115.330000 | 502.482500 |

'crkt'

| | converted_price | profit | ROI |
|---|---|---|---|
| count | 4659.000000 | 4659.000000 | 4659.000000 |
| mean | 34.379468 | 11.910137 | 77.192860 |
| std | 18.322582 | 15.940647 | 102.854776 |
| min | 3.050000 | -15.346500 | -102.310000 |
| 25% | 21.505000 | 0.709350 | 4.729000 |
| 50% | 29.490000 | 7.656300 | 48.200000 |
| 75% | 40.500000 | 17.235000 | 113.276000 |
| max | 123.000000 | 89.010000 | 499.150000 |

'kershaw'

| | converted_price | profit | ROI |
|---|---|---|---|
| count | 11702.000000 | 11702.000000 | 11702.000000 |
| mean | 32.605704 | 10.366963 | 67.301568 |
| std | 19.183257 | 16.689434 | 108.325761 |
| min | 1.140000 | -17.008200 | -113.388000 |
| 25% | 19.500000 | -1.035000 | -6.900000 |
| 50% | 26.500000 | 5.055000 | 33.410000 |
| 75% | 40.000000 | 16.800000 | 111.884000 |
| max | 124.000000 | 89.880000 | 502.050000 |

'sog'

| | converted_price | profit | ROI |
|---|---|---|---|
| count | 3086.000000 | 3086.000000 | 3086.000000 |
| mean | 37.068927 | 14.249967 | 93.082113 |
| std | 20.497066 | 17.832448 | 116.367597 |
| min | 1.950000 | -16.303500 | -108.690000 |
| 25% | 21.500000 | 0.705000 | 4.700000 |
| 50% | 33.250000 | 10.927500 | 71.400000 |
| 75% | 47.835000 | 23.616450 | 152.527500 |
| max | 119.990000 | 86.391300 | 501.760000 |

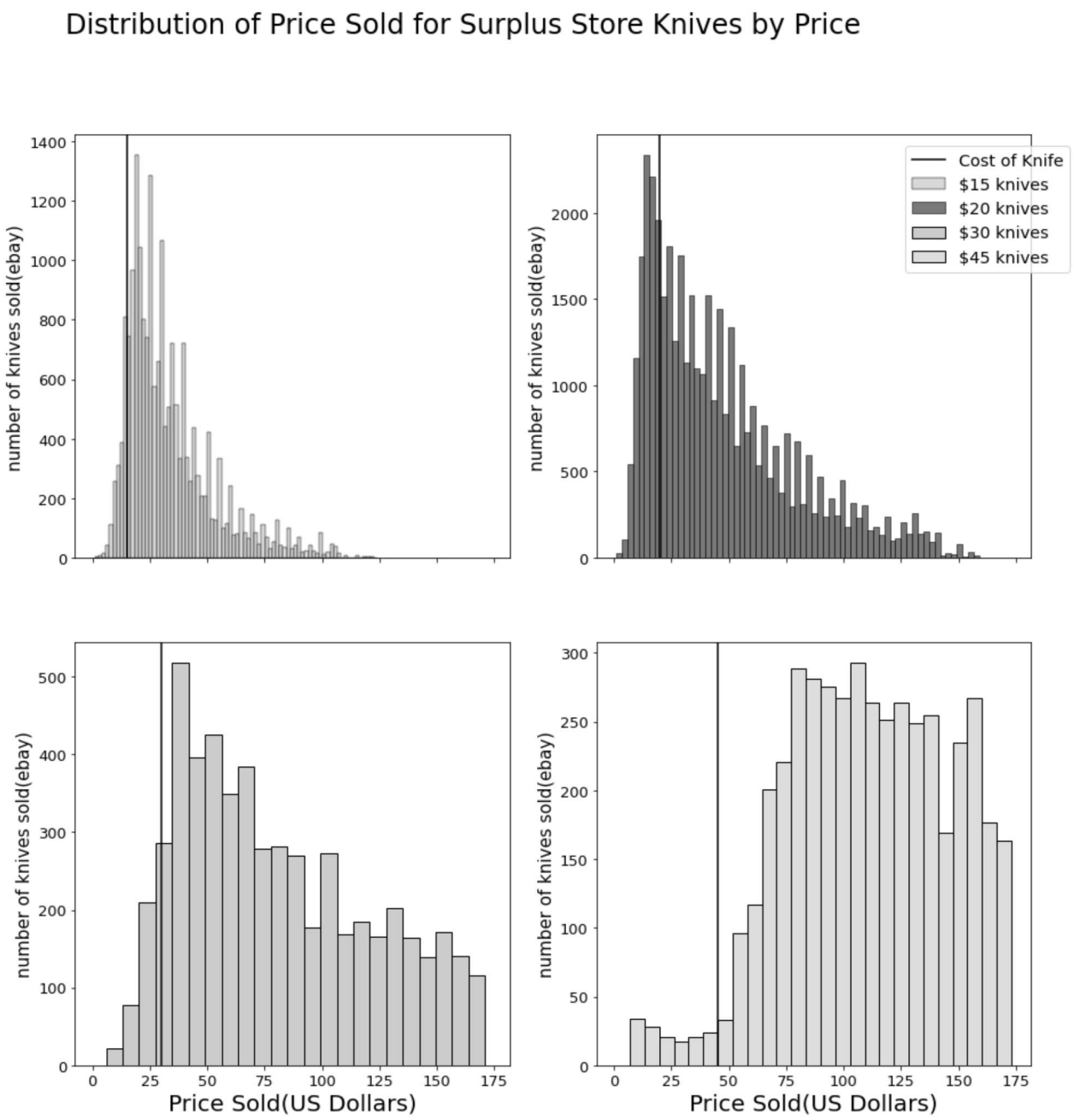
'spyderco'

| | converted_price | profit | ROI |
|---|---|---|---|
| count | 5398.000000 | 5398.000000 | 5398.000000 |
| mean | 80.200013 | 36.774011 | 121.847945 |
| std | 40.842921 | 35.533341 | 117.676520 |
| min | 5.990000 | -27.788700 | -88.250000 |
| 25% | 45.627500 | 6.695925 | 22.298000 |
| 50% | 71.400000 | 29.118000 | 96.625000 |
| 75% | 110.487500 | 63.124125 | 209.217500 |
| max | 171.000000 | 115.770000 | 385.900000 |

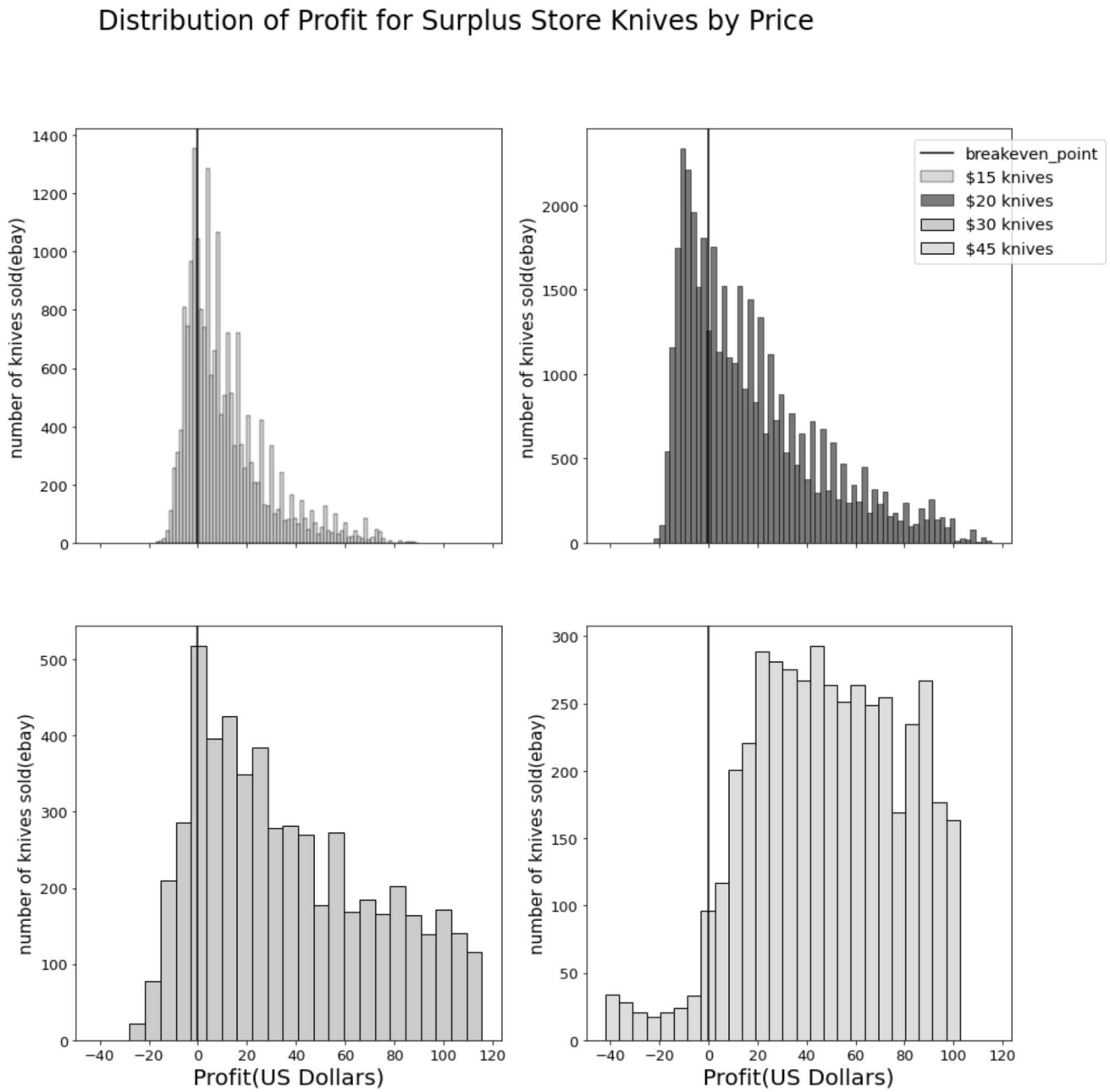
'victorinox'

| | converted_price | profit | ROI |
|---|---|---|---|
| **count** | 11497.000000 | 11497.000000 | 11497.000000 |
| **mean** | 28.653973 | 1.928957 | 9.148298 |
| **std** | 23.177921 | 20.164791 | 97.228776 |
| **min** | 1.040000 | -22.095200 | -110.476000 |
| **25%** | 14.000000 | -10.820000 | -51.925000 |
| **50%** | 20.280000 | -5.356400 | -25.129000 |
| **75%** | 34.970000 | 7.423900 | 35.640500 |
| **max** | 158.000000 | 114.460000 | 497.652174 |

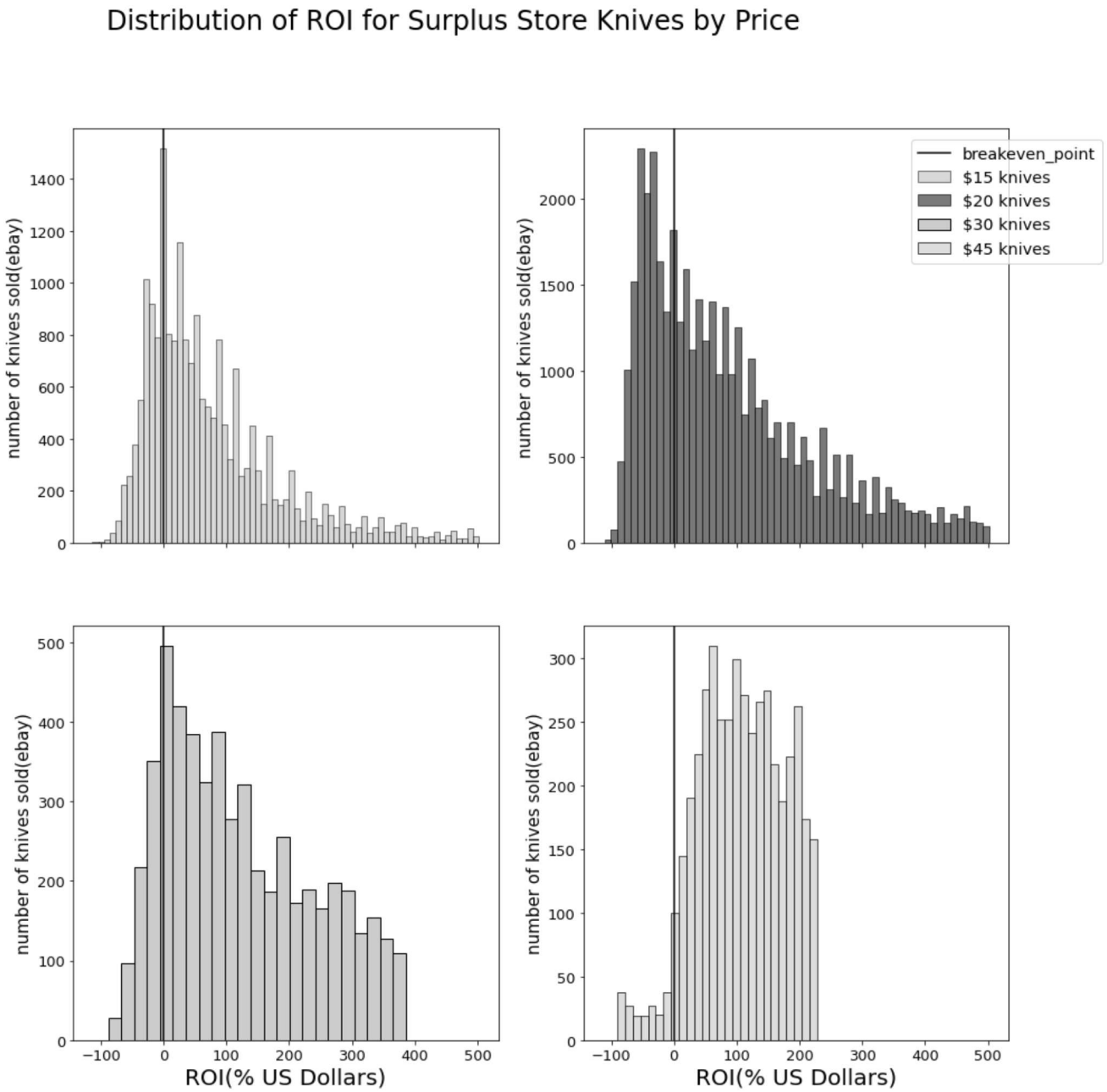
In [85]:
```
df_45 = df.loc[df['brand'] == 'benchmade']
df_30 = df.loc[df['brand'] == 'spyderco']
df_20 = df.loc[(df['brand'] == 'buck') | (df['brand'] == 'case') | (df['brand'] == 'victorinox')]
df_15 = df.loc[(df['brand'] == 'crkt') | (df['brand'] == 'kershaw') | (df['brand'] == 'sog')]


fig, axes  = plt.subplots(figsize=(15,15), ncols=2, nrows=2, sharex=True)
sns.histplot(df_15['converted_price'], ax=axes[0][0], color='gold', label='$15 knives')
sns.histplot(df_20['converted_price'], ax=axes[0][1], color='firebrick', label='$20 knives')
sns.histplot(df_30['converted_price'], ax=axes[1][0],  color='skyblue', label='$30 knives')
sns.histplot(df_45['converted_price'], ax=axes[1][1], color='palegreen', label='$45 knives')

for n in range(4):
    row = n//2 # n divided by 2 without the remainder
    col = n%2  # just the remainder of n divided by 2
    axes[row][col].set_xlabel('Price Sold(US Dollars)', fontsize=20)
    axes[row][col].set_ylabel('number of knives sold(ebay)', fontsize=15)
    axes[row][col].tick_params(axis='both', labelsize=13)


axes[0][0].axvline(x = 15, color = 'black', label= 'Cost of Knife')
axes[0][1].axvline(x = 20, color = 'black')
axes[1][0].axvline(x = 30, color = 'black')
axes[1][1].axvline(x = 45, color = 'black')
fig.suptitle("Distribution of Price Sold for Surplus Store Knives by Price", fontsize=24, x=0.44)
fig.legend(loc=(.82, .76), fontsize='x-large')
plt.show();
```


Distribution of Price Sold for Surplus Store Knives by Price
number of knives sold(ebay)
Cost of Knife
$15 knives
$20 knives
$30 knives
$45 knives
Price Sold(US Dollars)

In [86]:
```
fig, axes  = plt.subplots(figsize=(15,15), ncols=2, nrows=2, sharex=True)
sns.histplot(df_15['profit'], ax=axes[0][0], color='gold', label='$15 knives')
sns.histplot(df_20['profit'], ax=axes[0][1], color='firebrick', label='$20 knives')
sns.histplot(df_30['profit'], ax=axes[1][0],  color='skyblue', label='$30 knives')
sns.histplot(df_45['profit'], ax=axes[1][1], color='palegreen', label='$45 knives')

for n in range(4):
    row = n//2 # n divided by 2 without the remainder
    col = n%2  # just the remainder of n divided by 2
    axes[row][col].set_xlabel('Profit(US Dollars)', fontsize=20)
    axes[row][col].set_ylabel('number of knives sold(ebay)', fontsize=15)
    axes[row][col].tick_params(axis='both', labelsize=13)


axes[0][0].axvline(x = 0, color = 'black', label= 'breakeven_point')
axes[0][1].axvline(x = 0, color = 'black')
axes[1][0].axvline(x = 0, color = 'black')
axes[1][1].axvline(x = 0, color = 'black')
fig.suptitle("Distribution of Profit for Surplus Store Knives by Price", fontsize=24, x=0.44)
fig.legend(loc=(.82, .76), fontsize='x-large')
plt.show();
```



In [87]:
```
fig, axes  = plt.subplots(figsize=(15,15), ncols=2, nrows=2, sharex=True)
sns.histplot(df_15['ROI'], ax=axes[0][0], color='gold', label='$15 knives')
sns.histplot(df_20['ROI'], ax=axes[0][1], color='firebrick', label='$20 knives')
sns.histplot(df_30['ROI'], ax=axes[1][0],  color='skyblue', label='$30 knives')
sns.histplot(df_45['ROI'], ax=axes[1][1], color='palegreen', label='$45 knives')

for n in range(4):
    row = n//2 # n divided by 2 without the remainder
    col = n%2  # just the remainder of n divided by 2
    axes[row][col].set_xlabel('ROI(% US Dollars)', fontsize=20)
    axes[row][col].set_ylabel('number of knives sold(ebay)', fontsize=15)
    axes[row][col].tick_params(axis='both', labelsize=13)


axes[0][0].axvline(x = 0, color = 'black', label= 'breakeven_point')
axes[0][1].axvline(x = 0, color = 'black')
axes[1][0].axvline(x = 0, color = 'black')
axes[1][1].axvline(x = 0, color = 'black')
fig.suptitle("Distribution of ROI for Surplus Store Knives by Price", fontsize=24, x=0.44)
fig.legend(loc=(.82, .76), fontsize='x-large')
plt.show();
```



In [88]:
```
df_title = df.drop(['Image', 'url',
                    'date_sold', 'profit',
                    'ROI', 'brand', 'cost'],
                     axis=1).copy()
```

In [89]:
```
df_title.rename({'title': 'data',
                 'converted_price': 'labels'},
                axis=1, inplace=True)
```

In [90]:
```
mean_price = df_title['labels'].mean()
mean_price
```

Out[90]:
```
49.62917081729681
```

In [91]:
```
df_title['labels'] = (df_title['labels']/mean_price)
```

In [92]:
```
Y = df_title['labels'].values
```

In [ ]:
```
df_train, df_test, Ytrain, Ytest = train_test_split(df_title['data'], Y, test_size=0.2)
```

In [ ]:
```
# Convert sentences to sequences
MAX_VOCAB_SIZE = 40000
tokenizer = Tokenizer(num_words=MAX_VOCAB_SIZE)
tokenizer.fit_on_texts(df_train)
sequences_train = tokenizer.texts_to_sequences(df_train)
sequences_test = tokenizer.texts_to_sequences(df_test)
```

In [ ]:
```
# get word -> integer mapping
word2idx = tokenizer.word_index
V = len(word2idx)
print('Found %s unique tokens.' % V)
```

In [ ]:
```
# pad sequences so that we get a N x T matrix
data_train = pad_sequences(sequences_train)
print('Shape of data train tensor:', data_train.shape)

# get sequence length
T = data_train.shape[1]
```

In [ ]:
```
data_test = pad_sequences(sequences_test, maxlen=T)
print('Shape of data test tensor:', data_test.shape)
```

In [ ]:
```
# Create the RNN model

# We get to choose embedding dimensionality
D = 20

# Hidden state dimensionality
M = 15


i = Input(shape=(T,))
x = Embedding(V + 1, D)(i)
x = LSTM(M, return_sequences=True)(x)
x = GlobalMaxPooling1D()(x)
x = Dense(1, activation='linear')(x)

model = Model(i, x)
```

In [ ]:
```
# Compile and fit
model.compile(
  loss='MSE',
  optimizer='adam',
  metrics=['mae']
)


print('Training model...')
r = model.fit(
  data_train,
  Ytrain,
  epochs=5,
  validation_data=(data_test, Ytest)
)
```

In [ ]:
```
fig = plt.subplots(figsize=(12,8))
plt.plot(r.history['loss'], label='loss')
plt.plot(r.history['val_loss'], label='val_loss')
plt.title("Loss vs val Loss for RNN model on titles (MSE)", fontsize=15)
plt.xlabel("epochs", fontsize=15)
plt.ylabel("loss (mean squared error)", fontsize=15)
plt.legend();
plt.savefig('images/RNN_titles_MSE1.png')
```

In [ ]:
```
fig = plt.subplots(figsize=(12,8))
plt.plot(r.history['mae'], label='mae')
plt.plot(r.history['val_mae'], label='val_mae')
plt.title("Loss vs val Loss for RNN model on titles (MAE)", fontsize=15)
plt.xlabel("epochs", fontsize=15)
plt.ylabel("loss (mean absolute error)", fontsize=15)
plt.legend();
plt.savefig('images/RNN_titles_MAE1.png')
```

In [ ]:
```
0.276 * mean_price
```

In [ ]:
```
# Create the CNN model

# We get to choose embedding dimensionality
D = 20



i = Input(shape=(T,))
x = Embedding(V + 1, D)(i)
x = Conv1D(32, 3, activation='relu')(x)
x = MaxPooling1D(3)(x)
x = Conv1D(64, 3, activation='relu')(x)
x = MaxPooling1D(3)(x)
x = Conv1D(128, 3, activation='relu')(x)
x = GlobalMaxPooling1D()(x)
x = Dense(1, activation='linear')(x)

model = Model(i, x)
```

In [ ]:
```
# Compile and fit
model.compile(
  loss='MSE',
  optimizer='adam',
  metrics=['MSE']
)


print('Training model...')
r = model.fit(
  data_train,
  Ytrain,
  epochs=5,
  validation_data=(data_test, Ytest)
)
```

In [ ]:
```
# Plot loss per iteration
import matplotlib.pyplot as plt
plt.plot(r.history['loss'], label='loss')
plt.plot(r.history['val_loss'], label='val_loss')
plt.legend();
```

In [ ]:
```
# Plot accuracy per iteration
plt.plot(r.history['MSE'], label='MSE')
plt.plot(r.history['val_MSE'], label='val_MSE')
plt.legend();
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [93]:
```
df_imgs = df.drop(['title', 'url',
                   'date_sold', 'profit',
                   'ROI', 'brand', 'cost',
                   'pictureURLLarge'],
                     axis=1).copy()
```

In [94]:
```
df_imgs.dropna(subset=['Image'], inplace=True)
```

In [95]:
```
df_imgs.reset_index(drop=True, inplace=True)
```

In [96]:
```
df_imgs['file_index'] = df_imgs.index.values
df_imgs['file_index'] = df_imgs['file_index'].astype(str)
```

In [97]:
```
df_imgs['filename'] = df_imgs['file_index'] + '.jpg'
```

In [ ]:
```
# Identify Image Resolutions

# # Import Packages
# import pandas as pd
# import matplotlib.pyplot  as plt
# from PIL import Image
# from pathlib import Path
# import imagesize
# import numpy as np

# # Get the Image Resolutions
# imgs = [img.name for img in Path(root).iterdir() if img.suffix == ".jpg"]
# img_meta = {}
# for f in imgs: img_meta[str(f)] = imagesize.get(root+f)

# # Convert it to Dataframe and compute aspect ratio
# img_meta_df = pd.DataFrame.from_dict([img_meta]).T.reset_index().set_axis(['FileName', 'Size'], axis='columns', inplace=False)
# img_meta_df[["Width", "Height"]] = pd.DataFrame(img_meta_df["Size"].tolist(), index=img_meta_df.index)
# img_meta_df["Aspect Ratio"] = round(img_meta_df["Width"] / img_meta_df["Height"], 2)

# print(f'Total Nr of Images in the dataset: {len(img_meta_df)}')
# img_meta_df.head()



# # Visualize Image Resolutions

# fig = plt.figure(figsize=(8, 8))
# ax = fig.add_subplot(111)
# points = ax.scatter(img_meta_df.Width, img_meta_df.Height, color='blue', alpha=0.5, s=img_meta_df["Aspect Ratio"]*100, picker=1
# ax.set_title("Image Resolution")
# ax.set_xlabel("Width", size=14)
# ax.set_ylabel("Height", size=14);
```

In [ ]:
```
def download(row):
    filename = row.filepath

# create folder if it doesn't exist
#     os.makedirs(os.path.dirname(filename), exist_ok=True)

    url = row.Image
#     print(f"Downloading {url} to {filename}")

    try:
        r = requests.get(url, allow_redirects=True)
        with open(filename, 'wb') as f:
            f.write(r.content)
    except:
        print(f'{filename} error')
```

In [98]:
```
root_folder = 'C:/Users/12108/Documents/GitHub/Neural_Network_Predicting_Reseller_Success_Ebay/nn_images/'
df_imgs['filepath'] = root_folder + df_imgs['filename']
```

In [99]:
```
df_imgs['filepath'].sample(2).apply(print)
```

```
C:/Users/12108/Documents/GitHub/Neural_Network_Predicting_Reseller_Success_Ebay/nn_images/64142.jpg
C:/Users/12108/Documents/GitHub/Neural_Network_Predicting_Reseller_Success_Ebay/nn_images/741.jpg
```

Out[99]:
```
64142    None
741      None
Name: filepath, dtype: object
```

In [ ]:
```
df_imgs.apply(download, axis=1)
```

In [100]:
```
removed_files = []
pathway = 'C:/Users/12108/Documents/GitHub/Neural_Network_Predicting_Reseller_Success_Ebay/nn_images/'
for filename in os.listdir(pathway):
    if filename.endswith('.jpg'):
        try:
            img = Image.open(pathway + filename)  # open the image file
            img.verify()  # verify that it is, in fact an image
        except (IOError, SyntaxError) as e:
            print(filename)
            removed_files.append(filename)
            os.remove(pathway + filename)
```

In [101]:
```
to_drop = df_imgs.loc[df_imgs['filename'].isin(removed_files)].index.to_list()
```

In [102]:
```
df_imgs.drop(to_drop, inplace=True)
```

In [103]:
```
 img_list = os.listdir('C:/Users/12108/Documents/GitHub/Neural_Network_Predicting_Reseller_Success_Ebay/nn_images/')
```

In [104]:
```
img_df = df_imgs.loc[df_imgs['filename'].isin(img_list)].copy()
```

In [105]:
```
img_df.reset_index(drop=True, inplace=True)
```

In [106]:
```
img_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 70124 entries, 0 to 70123
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   Image            70124 non-null  object
 1   converted_price  70124 non-null  float64
 2   file_index       70124 non-null  object
 3   filename         70124 non-null  object
 4   filepath         70124 non-null  object
dtypes: float64(1), object(4)
memory usage: 2.7+ MB
```

In [107]:
```
img_df.rename({'Image': 'data',
               'converted_price': 'labels'},
                axis=1, inplace=True)
```

In [108]:
```
mean_price = img_df['labels'].mean()
mean_price
```

Out[108]:
```
49.620169699389656
```

In [109]:
```
img_df['labels'] = (img_df['labels']/mean_price)
```

In [110]:
```
Y = img_df['labels'].values
```

In [111]:
```
df_train, df_test, Ytrain, Ytest = train_test_split(img_df, Y, test_size=0.20)
```

In [ ]:
```

```

In [127]:
```
def to_grayscale(image):
    image = tf.image.rgb_to_grayscale(image)
    return image
```

In [157]:
```
#grayscale higher resolution
datagen=ImageDataGenerator(rescale=(1./255.),
                           validation_split=0.20)
```

In [158]:
```
train_generator=datagen.flow_from_dataframe(
dataframe=df_train,
directory= None,
x_col="filepath",
y_col="labels",
subset="training",
color_mode="grayscale",
target_size=(500, 500),
batch_size=100,
seed=55,
shuffle=True,
class_mode="raw")

valid_generator=datagen.flow_from_dataframe(
dataframe=df_train,
directory=None,
x_col="filepath",
y_col="labels",
subset="validation",
batch_size=100,
seed=55,
shuffle=True,
color_mode="grayscale",
target_size=(500, 500),
class_mode="raw")

test_datagen=ImageDataGenerator(rescale=1./255.)
test_generator=test_datagen.flow_from_dataframe(
dataframe=df_test,
directory=None,
x_col="filepath",
y_col="labels",
batch_size=100,
target_size=(500, 500),
color_mode="grayscale",
seed=55,
shuffle=False,
class_mode="raw")
```

```
Found 44880 validated image filenames.
Found 11219 validated image filenames.
Found 14025 validated image filenames.
```

In [159]:
```
train_generator.image_shape
```

Out[159]:
```
(500, 500, 1)
```

In [154]:
```
batch_size = 100
img_height = 500
img_width = 500
```

In [160]:
```
model = models.Sequential()

model.add(layers.Conv2D(16, (3, 3), padding='same', activation='relu',
                        input_shape=(500, 500, 1)))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(16, (3, 3), activation='relu', padding='same'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(32, (3, 3), padding='same', activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Flatten())

model.add(Dense(512, activation='relu'))
model.add(Dropout(0.1))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.1))
model.add(Dense(128, activation='relu'))
model.add(Dense(1, activation='linear'))

model.compile(loss='MSE',
              optimizer='Adam',
               metrics=['mae', 'mse'])
```

In [161]:
```
summary = model.fit(train_generator, epochs=3, validation_data=valid_generator)
```

```
Epoch 1/3
449/449 [==============================] - 11708s 26s/step - loss: 164.1137 - mae: 2.8969 - mse: 164.1137 - val_loss: 0.6431 -
val_mae: 0.6120 - val_mse: 0.6431
Epoch 2/3
449/449 [==============================] - 11937s 27s/step - loss: 0.9587 - mae: 0.7344 - mse: 0.9587 - val_loss: 0.5553 - val_
mae: 0.5941 - val_mse: 0.5553
Epoch 3/3
449/449 [==============================] - 11709s 26s/step - loss: 0.6594 - mae: 0.6202 - mse: 0.6594 - val_loss: 0.5231 - val_
mae: 0.5555 - val_mse: 0.5231
```

In [162]:
```
model.save('grayscale500res_CNN1.h5')
```

In [163]:
```
model.evaluate(valid_generator)
```

```
113/113 [==============================] - 466s 4s/step - loss: 0.5231 - mae: 0.5555 - mse: 0.5231
```

Out[163]:
```
[0.5230722427368164, 0.5554844737052917, 0.5230722427368164]
```

In [164]:
```
test_generator.reset()
pred=model.predict(test_generator,verbose=1)
```

```
141/141 [==============================] - 564s 4s/step
```

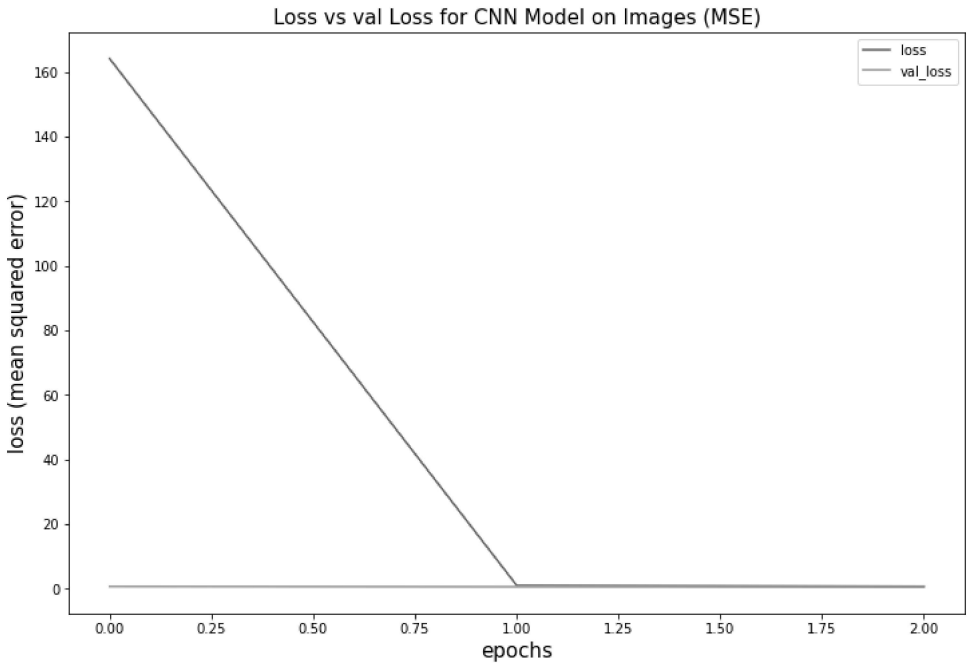
In [165]:
```
test_results = model.evaluate(test_generator)
```

```
141/141 [==============================] - 573s 4s/step - loss: 0.5115 - mae: 0.5490 - mse: 0.5115
```

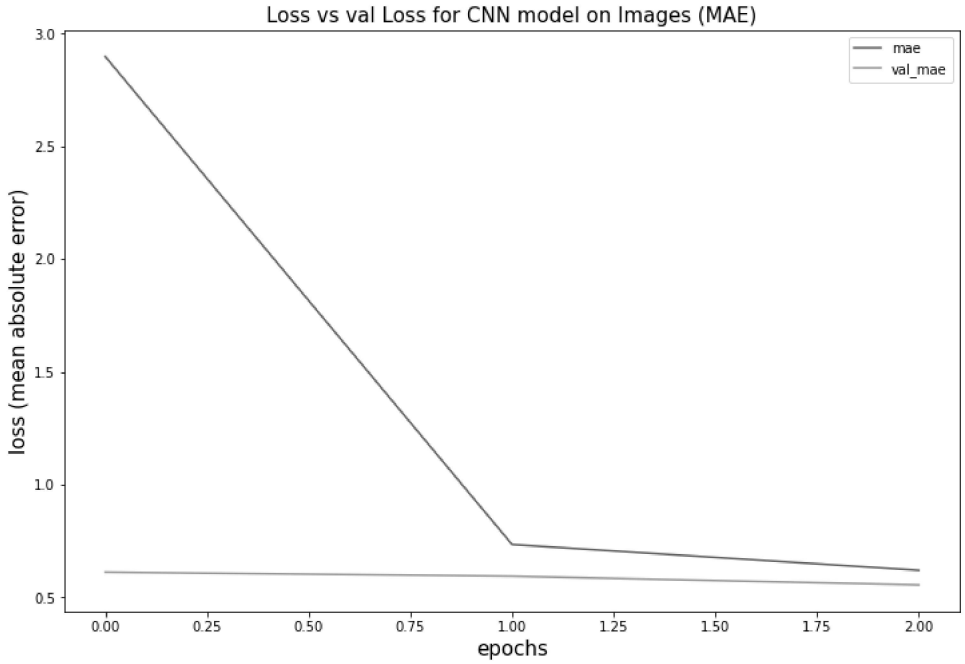
In [168]:
```
test_results
```

Out[168]:
```
[0.5114592909812927, 0.549023449420929, 0.5114592909812927]
```

In [173]:
```
fig = plt.subplots(figsize=(12,8))
plt.plot(summary.history['loss'], label='loss')
plt.plot(summary.history['val_loss'], label='val_loss')
plt.title("Loss vs val Loss for CNN Model on Images (MSE)", fontsize=15)
plt.xlabel("epochs", fontsize=15)
plt.ylabel("loss (mean squared error)", fontsize=15)
plt.legend();
plt.savefig('images/RNN_images_MSE1.png')
```




In [174]:
```
fig = plt.subplots(figsize=(12,8))
plt.plot(summary.history['mae'], label='mae')
plt.plot(summary.history['val_mae'], label='val_mae')
plt.title("Loss vs val Loss for CNN model on Images (MAE)", fontsize=15)
plt.xlabel("epochs", fontsize=15)
plt.ylabel("loss (mean absolute error)", fontsize=15)
plt.legend();
plt.savefig('images/CNN_images_MAE1.png')
```




In [176]:
```
from sklearn.metrics import mean_absolute_error
```

In [177]:
```
y_test = test_generator.labels
```

In [179]:
```
mean_absolute_error(y_test, pred)
```

Out[179]:
```
0.5490234818983017
```

In [*]:
```
fig = plt.figure()

residuals = (y_test - pred)


sns.displot(residuals, bins = 20)

fig.suptitle('Error Terms', fontsize = 20)

plt.xlabel('Errors', fontsize = 18);
```

In [*]:
```
import scipy.stats as stats
preds = y_hat_test
residuals = (y_test - preds)
sm.graphics.qqplot(residuals, dist=stats.norm, line='45', fit=True)
```

In [ ]:
```

```

In [ ]:
```
# # define two sets of inputs
# inputA = Input(shape=(32,))
# inputB = Input(shape=(128,))
# # the first branch operates on the first input
# x = Dense(8, activation="relu")(inputA)
# x = Dense(4, activation="relu")(x)
# x = Model(inputs=inputA, outputs=x)
# # the second branch opreates on the second input
# y = Dense(64, activation="relu")(inputB)
# y = Dense(32, activation="relu")(y)
# y = Dense(4, activation="relu")(y)
# y = Model(inputs=inputB, outputs=y)
# # combine the output of the two branches
# combined = concatenate([x.output, y.output])
# # apply a FC layer and then a regression prediction on the
# # combined outputs
# z = Dense(2, activation="relu")(combined)
# z = Dense(1, activation="linear")(z)
# # our model will accept the inputs of the two branches and
# # then output a single value
# model = Model(inputs=[x.input, y.input], outputs=z)
```

In [ ]:
```
0.6643 * median_price
```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```
# model = models.Sequential()

# model.add(layers.Conv2D(8, (3, 3), padding='same', activation='relu',
#                         input_shape=(100,100,3)))
# model.add(layers.BatchNormalization())

# model.add(layers.Conv2D(8, (3, 3), activation='relu', padding='same'))
# model.add(layers.BatchNormalization())
# model.add(layers.MaxPooling2D((2, 2)))


# model.add(layers.Conv2D(16, (3, 3), activation='relu', padding='same'))
# model.add(layers.BatchNormalization())
# model.add(layers.Conv2D(16, (3, 3), activation='relu', padding='same'))
# model.add(layers.BatchNormalization())
# model.add(layers.MaxPooling2D((2, 2)))




# model.add(layers.Conv2D(32, (3, 3), activation='relu', padding='same'))
# model.add(layers.BatchNormalization())
# model.add(layers.Conv2D(32, (3, 3), activation='relu', padding='same'))
# model.add(layers.BatchNormalization())
# model.add(layers.MaxPooling2D((2, 2)))


# model.add(layers.Conv2D(32, (3, 3), activation='relu', padding='same'))
# model.add(layers.BatchNormalization())
# model.add(layers.Conv2D(32, (3, 3), activation='relu', padding='same'))
# model.add(layers.BatchNormalization())
# model.add(layers.MaxPooling2D((2, 2)))


# model.add(layers.Conv2D(100, (3, 3), activation='relu', padding='same'))
# model.add(layers.BatchNormalization())
# model.add(layers.Conv2D(100, (3, 3), activation='relu', padding='same'))
# model.add(layers.BatchNormalization())
# model.add(layers.MaxPooling2D((2, 2)))


# model.add(layers.Flatten())

# model.add(Dense(512, activation='relu'))

# model.add(Dropout(0.1))

# model.add(Dense(1, activation='sigmoid'))
# opt = SGD(lr=0.001, momentum=0.9)
# model.compile(optimizer=opt,
#               loss='binary_crossentropy',
#                metrics=['accuracy'])
# #switching from regression to classifier.. no longer mse loss
# history = model.fit(X_train,
#                     y_train,
#                     epochs=5,
#                     batch_size=100,
#                     validation_data=(X_val, y_val))
```

In [ ]:
```
# model = tf.keras.Sequential([
#     tf.keras.layers.Rescaling(1./255),
#     tf.keras.layers.Conv2D(32, (3,3), activation='relu',  padding='same'),
#     tf.keras.layers.BatchNormalization(),
#     tf.keras.layers.Conv2D(32, (3,3), activation='relu',  padding='same'),
#     tf.keras.layers.BatchNormalization(),
#     tf.keras.layers.MaxPooling2D((2, 2)),

#     tf.keras.layers.Conv2D(64, (3,3), activation='relu',  padding='same'),
#     tf.keras.layers.BatchNormalization(),
#     tf.keras.layers.Conv2D(64, (3,3), activation='relu',  padding='same'),
#     tf.keras.layers.BatchNormalization(),
#     tf.keras.layers.MaxPooling2D((2, 2)),

#     tf.keras.layers.Conv2D(128, (3,3), activation='relu',  padding='same'),
#     tf.keras.layers.BatchNormalization(),
#     tf.keras.layers.Conv2D(128, (3,3), activation='relu',  padding='same'),
#     tf.keras.layers.BatchNormalization(),
#     tf.keras.layers.MaxPooling2D((2, 2)),

#     tf.keras.layers.Conv2D(250, (3,3), activation='relu',  padding='same'),
#     tf.keras.layers.BatchNormalization(),
#     tf.keras.layers.Conv2D(250, (3,3), activation='relu',  padding='same'),
#     tf.keras.layers.BatchNormalization(),
#     tf.keras.layers.MaxPooling2D((2, 2)),

#     tf.keras.layers.Flatten(),
#     tf.keras.layers.Dense(128, activation='relu'),
#     tf.keras.layers.Dropout(0.1),
#     tf.keras.layers.Dense(1, activation='sigmoid')
# ])
```

In [ ]:
```
# model.compile(
#     optimizer='adam',
#     loss=tf.keras.losses.BinaryCrossentropy(from_logits=False),
#     metrics=['accuracy'])
# history = model.fit(
#             train_ds,
#             validation_data=val_ds,
#             epochs=3)
#model.save('mymodel250250.h5')
```

In [ ]:
```
# for filename in listdir('C:/Users/12108/Documents/GitHub/Neural_Network_Predicting_Reseller_Success_Ebay/extra_bench'):
#     if filename.endswith(".jpg"):
#         try:
#             im = Image.load(filename)
#             im.verify() #I perform also verify, don't know if he sees other types o defects
#             im.close() #reload is necessary in my case
#             im = Image.load(filename)
#             im.transpose(PIL.Image.FLIP_LEFT_RIGHT)
#             im.close()
#         except:
#         print(f'{filename} is corrupted.')
```

In [ ]:
```
# for filename in os.listdir('C:/Users/12108/Documents/GitHub/Neural_Network_Predicting_Reseller_Success_Ebay/nn_images3'):
#     if filename.endswith('.jpg'):
#         try:
#             img = Image.open('C:/Users/12108/Documents/GitHub/Neural_Network_Predicting_Reseller_Success_Ebay/nn_images3/'+file
#             img.verify()  # verify that it is, in fact an image
#         except (IOError, SyntaxError) as e:
#             print(filename)
#             os.remove('C:/Users/12108/Documents/GitHub/Neural_Network_Predicting_Reseller_Success_Ebay/nn_images3/'+filename)
```

In [ ]:
```
# len(os.listdir('C:/Users/12108/Documents/GitHub/Neural_Network_Predicting_Reseller_Success_Ebay/nn_images3'))
```

In [ ]:
```
# removed_files = []
# for filename in os.listdir('C:/Users/12108/Documents/GitHub/Neural_Network_Predicting_Reseller_Success_Ebay/nn_images6'):
#     if filename.endswith('.jpg'):
#         try:
#             img = Image.open('C:/Users/12108/Documents/GitHub/Neural_Network_Predicting_Reseller_Success_Ebay/nn_images6/'+file
#             img.verify()  # verify that it is, in fact an image
#         except (IOError, SyntaxError) as e:
#             print(filename)
#             removed_files.append(filename)
#             os.remove('C:/Users/12108/Documents/GitHub/Neural_Network_Predicting_Reseller_Success_Ebay/nn_images6/'+filename)
```

In [ ]:
```
# root = 'C:/Users/12108/Documents/GitHub/Neural_Network_Predicting_Reseller_Success_Ebay/nn_images6/buy/'
```

In [ ]:
```
# Identify Image Resolutions

# # Import Packages
# import pandas as pd
# import matplotlib.pyplot  as plt
# from PIL import Image
# from pathlib import Path
# import imagesize
# import numpy as np

# # Get the Image Resolutions
# imgs = [img.name for img in Path(root).iterdir() if img.suffix == ".jpg"]
# img_meta = {}
# for f in imgs: img_meta[str(f)] = imagesize.get(root+f)

# # Convert it to Dataframe and compute aspect ratio
# img_meta_df = pd.DataFrame.from_dict([img_meta]).T.reset_index().set_axis(['FileName', 'Size'], axis='columns', inplace=False)
# img_meta_df[["Width", "Height"]] = pd.DataFrame(img_meta_df["Size"].tolist(), index=img_meta_df.index)
# img_meta_df["Aspect Ratio"] = round(img_meta_df["Width"] / img_meta_df["Height"], 2)

# print(f'Total Nr of Images in the dataset: {len(img_meta_df)}')
# img_meta_df.head()



# # Visualize Image Resolutions

# fig = plt.figure(figsize=(8, 8))
# ax = fig.add_subplot(111)
# points = ax.scatter(img_meta_df.Width, img_meta_df.Height, color='blue', alpha=0.5, s=img_meta_df["Aspect Ratio"]*100, picker=T
# ax.set_title("Image Resolution")
# ax.set_xlabel("Width", size=14)
# ax.set_ylabel("Height", size=14);
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:
```
from keras.models import Sequential
from keras_preprocessing.image import ImageDataGenerator
from keras.layers import Dense, Activation, Flatten, Dropout, BatchNormalization
from keras.layers import Conv2D, MaxPooling2D
from keras import regularizers, optimizers
import pandas as pd
import numpy as np
```

In [ ]:
```
df_train2 = df_train[['filepath', 'ROI']].copy()
```

In [ ]:
```
df_train2['ROI'] = (df_train2['ROI']/df_train2['ROI'].mean())
```

In [ ]:
```
df_test2 = df_test[['filepath', 'ROI']].copy()
df_test2['ROI'] = (df_test2['ROI']/df_test2['ROI'].mean())
```

In [ ]:

In [ ]:
```
from keras import models
from keras import layers
import tensorflow as tf
from tensorflow.keras.layers import Input, Dropout, Conv2D, Dense, Flatten, GlobalMaxPooling2D, MaxPooling2D, BatchNormalization
```

In [ ]:

In [ ]:
```
datagen=ImageDataGenerator(rescale=1./255.,
                           validation_split=0.25,
                           rotation_range=90,
                           horizontal_flip=True,
                           width_shift_range=.2,
                           height_shift_range=.2,
                           fill_mode='nearest')
```

In [ ]:
```
train_generator=datagen.flow_from_dataframe(
dataframe=df_train2,
directory= None,
x_col="filepath",
y_col="ROI",
subset="training",
batch_size=32,
seed=55,
shuffle=True,
class_mode="raw")

valid_generator=datagen.flow_from_dataframe(
dataframe=df_train2,
directory=None,
x_col="filepath",
y_col="ROI",
subset="validation",
batch_size=32,
seed=55,
shuffle=True,
class_mode="raw")

test_datagen=ImageDataGenerator(rescale=1./255.)
test_generator=test_datagen.flow_from_dataframe(
dataframe=df_test2,
directory=None,
x_col="filepath",
y_col='ROI',
batch_size=32,
seed=55,
shuffle=False,
class_mode='raw')
```

In [ ]:
```
model = models.Sequential()

model.add(layers.Conv2D(32, (3, 3), padding='same', activation='relu',
                        input_shape=(256 ,256,  3)))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Flatten())

model.add(Dense(256, activation='relu'))
model.add(Dropout(0.20))

model.add(Dense(1, activation='linear'))

model.compile(loss='mean_absolute_error',
              optimizer='Adam',
               metrics=['mae'])

summary = model.fit(train_generator, epochs=4, validation_data=valid_generator)
```

In [ ]:
```
# history = model.fit(X_train,
#                     y_train,
#                     epochs=32,
#                     batch_size=300,
#                     validation_data=(X_val, y_val))




# results_train = model.evaluate(X_test, y_test)

#model.summary()


# model.save('my_model_batch500.h5')
```

In [ ]:
```
# STEP_SIZE_TRAIN=train_generator.n//train_generator.batch_size
# STEP_SIZE_VALID=valid_generator.n//valid_generator.batch_size
# STEP_SIZE_TEST=test_generator.n//test_generator.batch_size
# model.fit_generator(generator=train_generator,
#                     steps_per_epoch=STEP_SIZE_TRAIN,
#                     validation_data=valid_generator,
#                     validation_steps=STEP_SIZE_VALID,
#                     epochs=10
#                    )
```

In [ ]:
```
model.fit(train_generator, epochs=10, validation_data=valid_generator)
```

In [ ]:
```
removed_files = []
for filename in os.listdir('C:/Users/12108/Documents/GitHub/Neural_Network_Predicting_Reseller_Success_Ebay/nn_images4'):
    if filename.endswith('.jpg'):
        try:
            img = Image.open('C:/Users/12108/Documents/GitHub/Neural_Network_Predicting_Reseller_Success_Ebay/nn_images4/'+filena
            img.verify()  # verify that it is, in fact an image
        except (IOError, SyntaxError) as e:
            print(filename)
            removed_files.append(filename)
            os.remove('C:/Users/12108/Documents/GitHub/Neural_Network_Predicting_Reseller_Success_Ebay/nn_images4/'+filename)
```

In [ ]:
```
to_drop = df.loc[df['filename'].isin(removed_files)].index
```

In [ ]:
```
df.drop(to_drop, inplace=True)
```

In [ ]:
```
df_train = df.sample(frac=0.8, random_state=111)
df_test=df.drop(df_train.index)
df_train2 = df_train[['filepath', 'ROI']].copy()
df_train2['ROI'] = (df_train2['ROI']/df_train2['ROI'].mean())
df_test2 = df_test[['filepath', 'ROI']].copy()
df_test2['ROI'] = (df_test2['ROI']/df_test2['ROI'].mean())
```

In [ ]:
```
datagen=ImageDataGenerator(rescale=1./255.,validation_split=0.25)
```

In [ ]:
```
train_generator=datagen.flow_from_dataframe(
dataframe=df_train2,
directory= None,
x_col="filepath",
y_col="ROI",
subset="training",
batch_size=32,
seed=55,
shuffle=True,
class_mode="raw")

valid_generator=datagen.flow_from_dataframe(
dataframe=df_train2,
directory=None,
x_col="filepath",
y_col="ROI",
subset="validation",
batch_size=32,
seed=55,
shuffle=True,
class_mode="raw")

test_datagen=ImageDataGenerator(rescale=1./255.)
test_generator=test_datagen.flow_from_dataframe(
dataframe=df_test2,
directory=None,
x_col="filepath",
y_col="ROI",
batch_size=32,
seed=55,
shuffle=False,
class_mode="raw")
```

In [ ]:
```
test_datagen=ImageDataGenerator(rescale=1./255.)
test_generator=test_datagen.flow_from_dataframe(
dataframe=df_test2,
directory=None,
x_col="filepath",
y_col="ROI",
batch_size=32,
seed=55,
shuffle=False,
class_mode="raw")
```

In [ ]:
```
model = models.Sequential()

model.add(layers.Conv2D(32, (3, 3), padding='same', activation='relu',
                        input_shape=(256 ,256,  3)))
# model.add(layers.BatchNormalization())
# model.add(layers.Conv2D(32, (3, 3), activation='relu', padding='same'))
# model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
# model.add(layers.BatchNormalization())
# model.add(layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
# model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(128, (3, 3), activation='relu', padding='same'))
# model.add(layers.BatchNormalization())
# model.add(layers.Conv2D(128, (3, 3), activation='relu', padding='same'))
# model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Flatten())

model.add(Dense(256, activation='relu'))
model.add(Dropout(0.1))

model.add(Dense(1, activation='linear'))

model.compile(loss='mean_absolute_error',
              optimizer='Adam',
               metrics=['mae'])
```

In [ ]:
```
summary = model.fit(train_generator, epochs=10, validation_data=valid_generator)
```

In [ ]:
```
model.evaluate(valid_generator)
```

In [ ]:
```
test_generator.reset()
pred=model.predict(test_generator,verbose=1)
```

In [ ]:
```
test_results = model.evaluate(test_generator)
```

In [ ]:
```
df_test['ROI'].mean() * 0.7096
```

In [ ]:
```
fig = plt.figure(figsize=(12,8))
plt.plot(summary.history['loss'])
plt.plot(summary.history['val_loss'])
plt.plot
plt.title('model loss')
plt.ylabel('loss(mean absolute error)')
plt.xlabel('epoch')
plt.legend(['train_loss', 'val_loss'], loc='upper right')
plt.show();
```

In [ ]:
```
len(pred)
```

In [ ]:
```
df_test3 = df_test2.reset_index(drop=True).copy()
```

In [ ]:
```
df_test3.sample(10)
```

In [ ]:
```
df_test2[27:30]
```

In [ ]:
```
pred[27:30].mean()
```

In [ ]:
```
df_test2[27:28]['ROI'].values * df_test['ROI'].mean()
```

In [ ]:
```
pred[27:30].mean() * df_test['ROI'].mean()
```

In [ ]:
```
model.save('no_batch_norm.h5')
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

# Data Science Processes

## Introduction

As discussed, this section is all about synthesizing your skills in order to work through a full Data Science workflow. In this lesson, you'll take a look at some general outlines for how Data Scientists organize their workflow and conceptualize their process.

## Objectives

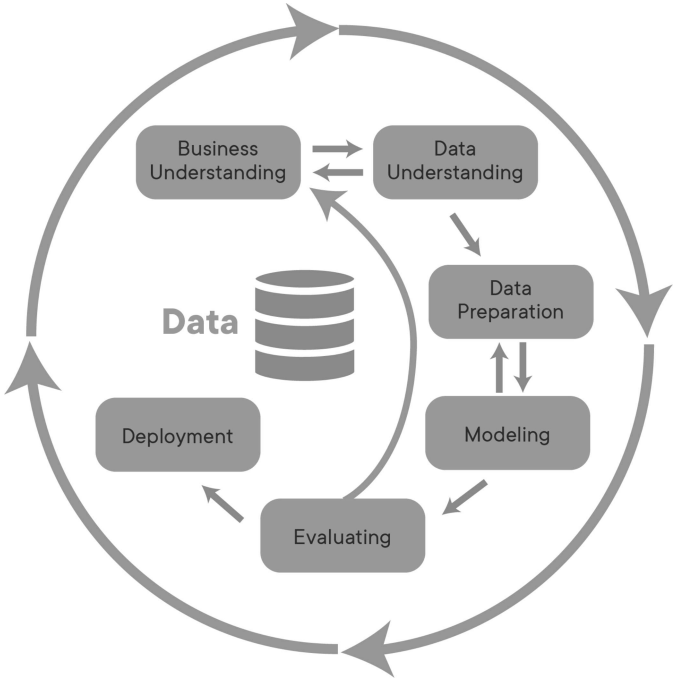
You will be able to:

- List the different data science process frameworks
- Compare and contrast popular data science process frameworks such as CRISP-DM, KDD, OSEMN

## What is a Data Science Process?

Data Science projects are often complex, with many stakeholders, data sources, and goals. Due to this, the Data Science community has created several methodologies for helping organize and structure Data Science Projects. In this lesson, you'll explore three of the most popular methodologies -- ***CRISP-DM***, ***KDD***, and ***OSEMN***, and explore how you can make use of them to keep your projects well-structured and organized.

## CRoss-Industry Standard Process for Data Mining (CRISP-DM)




***CRISP-DM*** is probably the most popular Data Science process in the Data Science world right now. Take a look at the visualization above to get a feel for CRISP-DM. Notice that CRISP-DM is an iterative process!

Let's take a look at the individual steps involved in CRISP-DM.

***Business Understanding:*** This stage is all about gathering facts and requirements. Who will be using the model you build? How will they be using it? How will this help the goals of the business or organization overall? Data Science projects are complex, with many moving parts and stakeholders. They're also time intensive to complete or modify. Because of this, it is very important that the Data Science team working on the project has a deep understanding of what the problem is, and how the solution will be used. Consider the fact that many stakeholders involved in the project may not have technical backgrounds, and may not even be from the same organization. Stakeholders from one part of the organization may have wildly different expectations about the project than stakeholders from a different part of the organization -- for instance, the sales team may be under the impression that a recommendation system project is meant to increase sales by recommending upsells to current customers, while the marketing team may be under the impression that the project is meant to help generate new leads by personalizing product recommendations in a marketing email. These are two very different interpretations of a recommendation system project, and it's understandable that both departments would immediately assume that the primary goal of the project is one that helps their organization. As a Data Scientist, it's up to you to clarify the requirements and make sure that everyone involved understands what the project is and isn't.

During this stage, the goal is to get everyone on the same page and to provide clarity on the scope of the project for everyone involved, not just the Data Science team. Generate and answer as many contextual questions as you can about the project.

Good questions for this stage include:

- Who are the stakeholders in this project? Who will be directly affected by the creation of this project?
- What business problem(s) will this Data Science project solve for the organization?
- What problems are inside the scope of this project?
- What problems are outside the scope of this project?
- What data sources are available to us?
- What is the expected timeline for this project? Are there hard deadlines (e.g. "must be live before holiday season shopping") or is this an ongoing project?
- Do stakeholders from different parts of the company or organization all have the exact same understanding about what this project is and isn't?

***Data Understanding:***

Once we have a solid understanding of the business implications for this project, we move on to understanding our data. During this stage, we'll aim to get a solid understanding of the data needed to complete the project. This step includes both understanding where our data is coming from, as well as the information contained within the data.

Consider the following questions when working through this stage:

- What data is available to us? Where does it live? Do we have the data, or can we scrape/buy/source the data from somewhere else?
- Who controls the data sources, and what steps are needed to get access to the data?
- What is our target?
- What predictors are available to us?
- What data types are the predictors we'll be working with?
- What is the distribution of our data?
- How many observations does our dataset contain? Do we have a lot of data? Only a little?
- Do we have enough data to build a model? Will we need to use resampling methods?
- How do we know the data is correct? How is the data collected? Is there a chance the data could be wrong?

***Data Preparation:***

Once we have a strong understanding of our data, we can move onto preparing the data for our modeling steps.

During this stage, we'll want to handle the following issues:

- Detecting and dealing with missing values
- Data type conversions (e.g. numeric data mistakenly encoded as strings)
- Checking for and removing multicollinearity (correlated predictors)
- Normalizing our numeric data
- Converting categorical data to numeric format through one-hot encoding

***Modeling:***

Once we have clean data, we can begin modeling! Remember, modeling, as with any of these other steps, is an iterative process. During this stage, we'll try to build and tune models to get the highest performance possible on our task.

Consider the following questions during the modeling step:

- Is this a classification task? A regression task? Something else?
- What models will we try?
- How do we deal with overfitting?
- Do we need to use regularization or not?
- What sort of validation strategy will we be using to check that our model works well on unseen data?
- What loss functions will we use?
- What threshold of performance do we consider as successful?

***Evaluation:***

During this step, we'll evaluate the results of our modeling efforts. Does our model solve the problems that we outlined all the way back during step 1? Why or why not? Often times, evaluating the results of our modeling step will raise new questions, or will cause us to consider changing our approach to the problem. Notice from the CRISP-DM diagram above, that the "Evaluation" step is unique in that it points to both *Business Understanding* and *Deployment*. As we mentioned before, Data Science is an iterative process -- that means that given the new information our model has provided, we'll often want to start over with another iteration, armed with our newfound knowledge! Perhaps the results of our model showed us something important that we had originally failed to consider the goal of the project or the scope. Perhaps we learned that the model can't be successful without more data, or different data. Perhaps our evaluation shows us that we should reconsider our approach to cleaning and structuring the data, or how we frame the project as a whole (e.g. realizing we should treat the problem as a classification rather than a regression task). In any of these cases, it is totally encouraged to revisit the earlier steps.

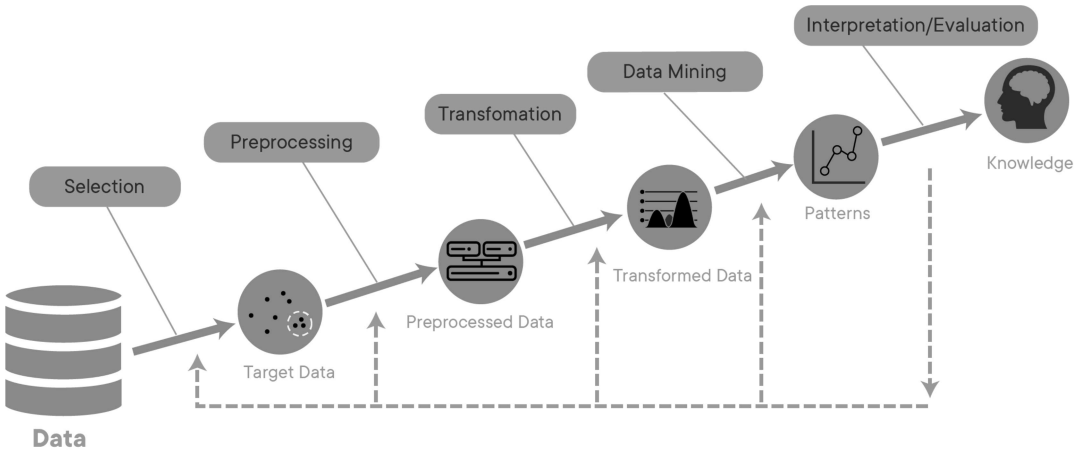
Of course, if the results are satisfactory, then we instead move onto deployment!

***Deployment:***

During this stage, we'll focus on moving our model into production and automating as much as possible. Everything before this serves as a proof-of-concept or an investigation. If the project has proved successful, then you'll work with stakeholders to determine the best way to implement models and insights. For example, you might set up an automated ETL (Extract-Transform-Load) pipelines of raw data in order to feed into a database and reformat it so that it is ready for modeling. During the deployment step, you'll actively work to determine the best course of action for getting the results of your project into the wild, and you'll often be involved with building everything needed to put the software into production.

This is one of the most rewarding steps of the entire Data Science process -- getting to see your work go live!

## Knowledge Discovery in Databases




***Knowledge Discovery in Databases***, or ***KDD*** is considered the oldest Data Science process. The creation of this process is credited to Gregory Piatetsky-Shapiro, who also runs the ever-popular Data Science blog, kdnuggets (https://www.kdnuggets.com/). If you're interested, read the original white paper on KDD, which can be found here (https://www.kdnuggets.com/gpspubs/aimag-kdd-overview-1992.pdf)!

The KDD process is quite similar to the CRISP-DM process. The diagram above illustrates every step of the KDD process, as well as the expected output at each stage.

***Selection***:

During this stage, you'll focus on selecting your problem, and the data that will help you answer it. This stage works much like the first stage of CRISP-DM -- you begin by focusing on developing an understanding of the domain the problem resides in (e.g. marketing, finance, increasing customer sales, etc), the previous work done in this domain, and the goals of the stakeholders involved with the process.

Once you've developed a strong understanding of the goals and the domain, you'll work to establish where your data is coming from, and which data will be useful to you. Organizations and companies usually have a ton of data, and only some of it will be relevant to the problem you're trying to solve. During this stage, you'll focus on examining the data sources available to you and gathering the data that you deem useful for the project.

The output of this stage is the dataset you'll be using for the Data Science project.

***Preprocessing***:

The preprocessing stage is pretty straightforward -- the goal of this stage is to "clean" the data by preprocessing it. For text data, this may include things like tokenization. You'll also identify and deal with issues like outliers and/or missing data in this stage.

In practice, this stage often blurs with the *Transformation* stage.

The output of this stage is preprocessed data that is more "clean" than it was at the start of this stage -- although the dataset is not quite ready for modeling yet.

***Transformation***:

During this stage, you'll take your preprocessed data and transform it in a way that makes it more ideal for modeling. This may include steps like feature engineering and dimensionality reduction. At this stage, you'll also deal with things like checking for and removing multicollinearity from the dataset. Categorical data should also be converted to numeric format through one-hot encoding during this step.

The output of this stage is a dataset that is now ready for modeling. All null values and outliers are removed, categorical data has been converted to a format that a model can work with, and the dataset is generally ready for experimentation with modeling.

***Data Mining***:

The Data Mining stage refers to using different modeling techniques to try and build a model that solves the problem we're after -- often, this is a classification or regression task. During this stage, you'll also define your parameters for given models, as well as your overall criteria for measuring the performance of a model.

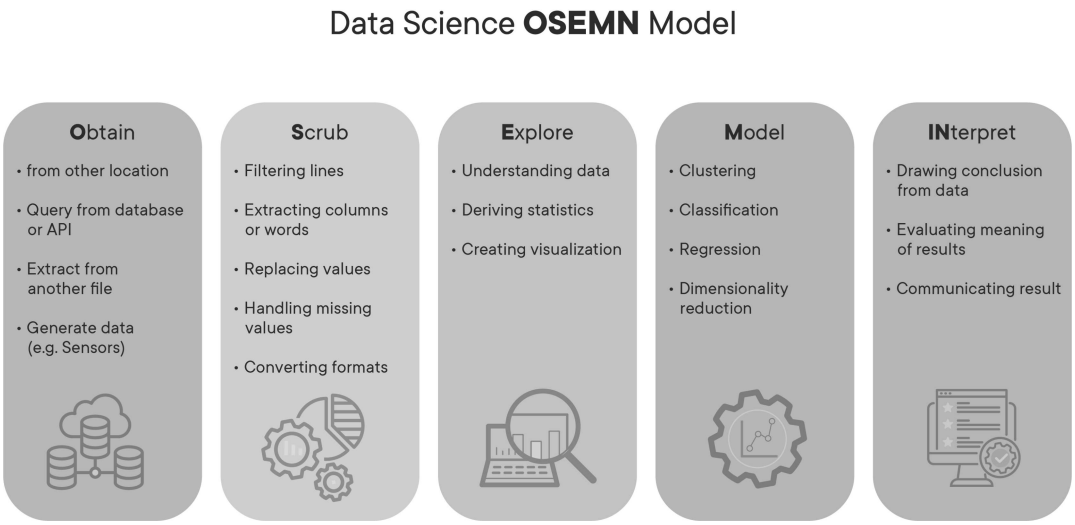
You may be wondering what Data Mining is, and how it relates to Data Science. In practice, it's just an older term that essentially means the same thing as Data Science. Dr. Piatetsky-Shapiro defines Data Mining as "the non-trivial extraction of implicit, previously unknown and potentially useful information from data." Making of things such as Machine Learning algorithms to find insights in large datasets that aren't immediately obvious without these algorithms is at the heart of the concept of Data Mining, just as it is in Data Science. In a pragmatic sense, this is why the terms Data Mining and Data Science are typically used interchangeably, although the term Data Mining is considered an older term that isn't used as often nowadays.

The output of this stage results from a fit to the data for the problem we're trying to solve.

***Interpretation/Evaluation***:

During this final stage of KDD, we focus on interpreting the "patterns" discovered in the previous step to help us make generalizations or predictions that help us answer our original question. During this stage, you'll consolidate everything you've learned to present it to stakeholders for guiding future actions. Your output may be a presentation that you use to communicate to non-technical managers or executives (never discount the importance of knowing PowerPoint as a Data Scientist!). Your conclusions for a project may range from "this approach didn't work" or "we need more data about {X}" to "this is ready for production, let's build it!".

## OSEMN




Adapted from: KDNuggets (https://www.kdnuggets.com/2018/02/data-science-command-line-book-exploring-data.html)

This brings us to the Data Science process we'll be using during this section -- OSEMN (sometimes referred as OSEMiN, and pronounced "OH-sum", rhymes with "possum"). This is the most straightforward of the Data Science processes discussed so far. Note that during this process, just like the others, the stages often blur together. It is completely acceptable (and often a best practice!) to float back and forth between stages as you learn new things about your problem,

dataset, requirements, etc. It's quite common to get to the modeling step and realize that you need to scrub your data a bit more or engineer a different feature and jump back to the "Scrub" stage, or go all the way back to the "Obtain" stage when you realize your current data isn't sufficient to solve this problem. As with any of these frameworks, OSEMN is meant to be treated more like a set of guidelines for structuring your project than set-in-stone steps that cannot be violated.

***Obtain***:

As with CRISP-DM and KDD, this step involves understanding stakeholder requirements, gathering information on the problem, and finally, sourcing data that we think will be necessary for solving this problem.

***Scrub***:

During this stage, we'll focus on preprocessing our data. Important steps such as identifying and removing null values, dealing with outliers, normalizing data, and feature engineering/feature selection are handled around this stage. The line with this stage really blurs with the *Explore* stage, as it is common to only realize that certain columns require cleaning or preprocessing as a result of the visualizations and explorations done during Step 3.

Note that although technically, categorical data should be one-hot encoded during this step, in practice, it's usually done after data exploration. This is because it is much less time-consuming to visualize and explore a few columns containing categorical data than it is to explore many different dummy columns that have been one-hot encoded.

***Explore***:

This step focuses on getting to know the dataset you're working with. As mentioned above, this step tends to blend with the *Scrub* step mentioned above. During this step, you'll create visualizations to really get a feel for your dataset. You'll focus on things such as understanding the distribution of different columns, checking for multicollinearity, and other tasks like that. If your project is a classification task, you may check the balance of the different classes in your dataset. If your problem is a regression task, you may check that the dataset meets the assumptions necessary for a regression task.

At the end of this step, you should have a dataset ready for modeling that you've thoroughly explored and are extremely familiar with.

***Model***:

This step, as with the last two frameworks, is also pretty self-explanatory. It consists of building and tuning models using all the tools you have in your data science toolbox. In practice, this often means defining a threshold for success, selecting machine learning algorithms to test on the project, and tuning the ones that show promise to try and increase your results. As with the other stages, it is both common and accepted to realize something, jump back to a previous stage like *Scrub* or *Explore*, and make some changes to see how it affects the model.

***Interpret***:

During this step, you'll interpret the results of your model(s), and communicate results to stakeholders. As with the other frameworks, communication is incredibly important! During this stage, you may come to realize that further investigation is needed, or more data. That's totally fine -- figure out what's needed, go get it, and start the process over! If your results are satisfactory to all stakeholders involved, you may also go from this stage right into putting your model into production and automating processes necessary to support it.

## A Note On Communicating Results

Regardless of the quality of your results, it's very important that you be aware of the business requirements and stakeholder expectations at all times! Generally, no matter which of the above processes you use, you'll communicate your results in a two-pronged manner:

- A short, high-level presentation covering your question, process, and results meant for non-technical audiences
- A detailed Jupyter Notebook demonstrating your entire process meant for technical audiences

In general, you can see why Data Scientists love Jupyter Notebooks! It is very easy to format results in a reproducible, easy-to-understand way. Although a detailed Jupyter Notebook may seem like the more involved of the two deliverables listed above, the high-level presentation is often the hardest! Just remember -- even if the project took you/your team over a year and utilized the most cutting-edge machine learning techniques available, you still need to be able to communicate your results in about 5 slides (using graphics, not words, whenever possible!), in a 5 minute presentation in a way that someone that can't write code can still understand and be convinced by!

## Conclusion

In this lesson, you learned about the different data science process frameworks including CRISP-DM, KDD, and OSEMN. You also learned that the data science process is iterative and that a typical data science project involves many different stakeholders who may not have a technical background. As such, it's important to recognize that data scientists must be able to communicate their findings in a non-technical way.