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

Categorizing Nine Different Knives as Profitable or Not Profitable using a CNN for Knife Images and a RNN for Titles from Ebay Listings

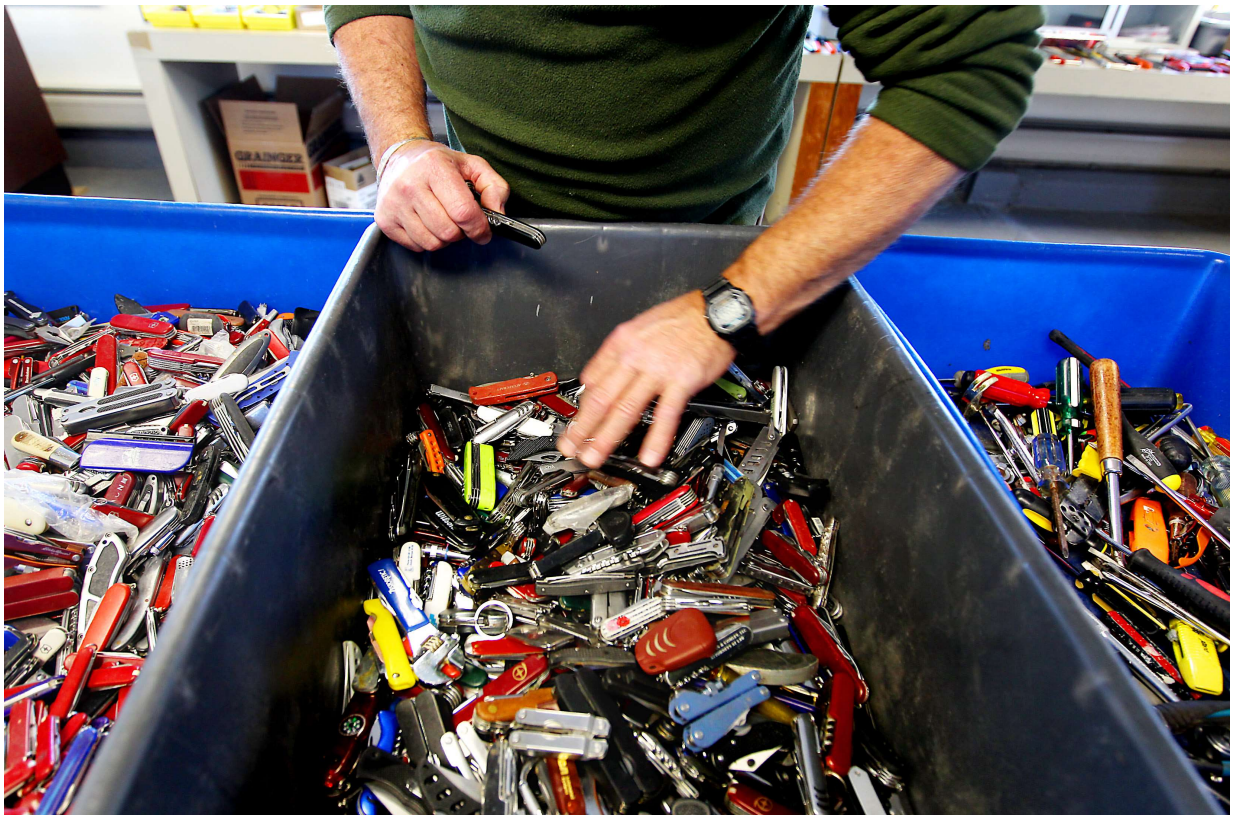
Author: Dylan Dey

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)

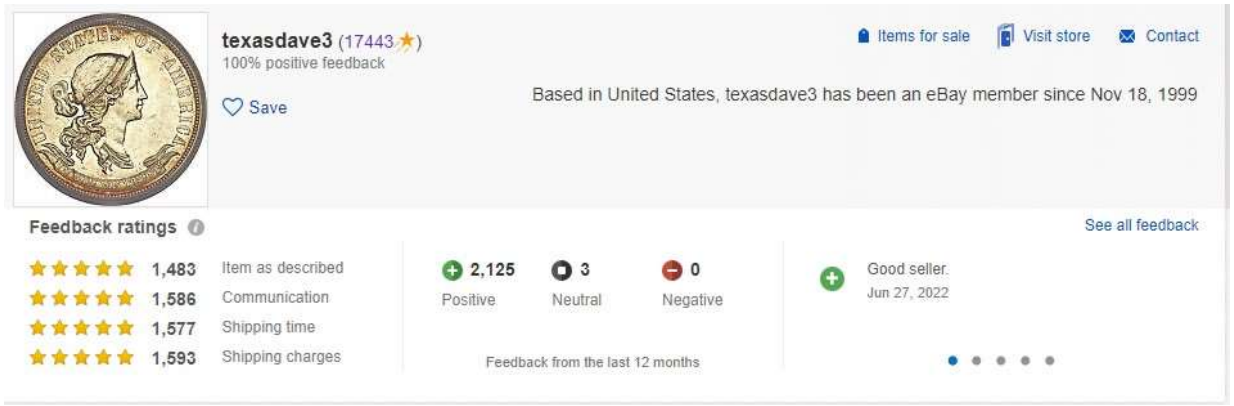


[Everything that doesn't make it through Texas airports can be found at one Austin store (https://cbsaustin.com/news/local/everything-that-doesnt-make-it-through-texas-airports-can-be-found-at-one-austin-store)](https://cbsaustin.com/news/local/everything-that-doesnt-make-it-through-texas-airports-can-be-found-at-one-austin-store)

> 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)](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 and keep us all afloat. 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 nine buckets of presorted brand knives that I was interested in, specifically. The other buckets are full of unbranded knives that usually are crowded with way too many people. These other bins, however, are behind glass, presorted, branded(and therefore have specific characteristics and logos for my model to identify), and priced higher.

** knife bucket image **

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 of[category_id=<48818>](https://www.ebay.com/sch/48818/i.html?_from=R40&_nkw=knife) knives.

> 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.

#

Questions to consider:

- What are the business's pain points related to this project?
- How did you pick the data analysis question(s) that you did?
- Why are these questions important from a business perspective?

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?

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)](https://developer.ebay.com/develop/ebay-sdks)

[Python SDK to simplify making calls (https://github.com/timotheus/ebaysdk-python/wiki/Trading-API-Class)](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 [1]:
```
# from ebaysdk.finding import Connection
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
```

In [2]:
```
from sklearn.model_selection import train_test_split
```

In [ ]:
```
# for filename in listdir('C:/Users/12108/Documents/GitHub/Neural_Network_Predict
#     if filename.endswith(".jpg"):
#         try:
#             im = Image.load(filename)
#             im.verify() #I perform also verify, don't know if he sees other typ
#             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_Predic
    if filename.endswith('.jpg'):
        try:
            img = Image.open('C:/Users/12108/Documents/GitHub/Neural_Network_Prec
            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_
```

In [ ]:
```
len(os.listdir('C:/Users/12108/Documents/GitHub/Neural_Network_Predicting_Reselle
```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```
root = 'C:/Users/12108/Documents/GitHub/Neural_Network_Predicting_Reseller_Succes
```

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(['FileN
img_meta_df[["Width", "Height"]] = pd.DataFrame(img_meta_df["Size"].tolist(), inc
img_meta_df["Aspect Ratio"] = round(img_meta_df["Width"] / img_meta_df["Height"],

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

In [ ]:
```
# 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.
ax.set_title("Image Resolution")
ax.set_xlabel("Width", size=14)
ax.set_ylabel("Height", size=14);
```

In [ ]:
```
# 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.
ax.set_title("Image Resolution")
ax.set_xlabel("Width", size=14)
ax.set_ylabel("Height", size=14);
```

In [ ]:
```
img_meta_df.describe()
```

In [ ]:
```
df = pd.read_csv('data/concat_df_bench.csv')
```

In [ ]:
```
df_ready = df.dropna(axis=1).copy()
```

In [ ]:
```
df_ready['PictureURL'].head(3).apply(print)
```

In [3]:
```
batch_size = 32
img_height = 100
img_width = 100
```

In [4]:
```
import os
import PIL

import tensorflow as tf
```

In [5]:
```
from tensorflow.keras.layers import Input, Conv2D, Dense, Flatten, Dropout, Globa
from tensorflow.keras.models import Model
```

In [6]:
```
X_TRAIN = []
Y_TRAIN = []
```

In [7]:
```
path = 'C:/Users/12108/Documents/GitHub/Neural_Network_Predicting_Reseller_Succes
```

In [ ]:
```
for filename in os.listdir(path):
    if filename.endswith(".jpg"):
        try:
            img_array = cv2.imread(str(path + filename))
            new_array = cv2.resize(img_array, (img_height, img_width))
            X_TRAIN.append(new_array)
        except Exception as e:
            print(f'error at {filename}')
```

In [ ]:
```
X_TRAIN
```

In [ ]:
```
for img in os.listdir(path):
            try:
                img_array = cv.imread(os.path.join(path, img))
                new_array = cv.resize(img_array, (img_width, img_height))
                X_TRAIN.append(new_array)
            except Exception as e:
                pass
```

In [ ]:
```

```

In [ ]:
```
img_array = cv2.imread(os.path.join(path, '3.jpg'))
new_array = cv2.resize(img_array, (img_height, img_width))
```

In [ ]:
```
np.shape(new_array)
```

In [ ]:
```
for filename in os.listdir(path):
    if filename.endswith(".jpg"):
        print(path + filename)
```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```
# # Interactive Plotting

# # Import libraries
# from matplotlib.widgets import LassoSelector
# from matplotlib.path import Path as mplPath

# # Lasso Selection of data points
# class SelectFromCollection:
#     def __init__(self, ax, collection, alpha_other=0.3):
#         self.canvas = ax.figure.canvas
#         self.collection = collection

#         self.xys = collection.get_offsets()
#         self.lasso = LassoSelector(ax, onselect=self.onselect)
#         self.ind = []

#     def onselect(self, verts):
#         path = mplPath(verts)
#         self.ind = np.nonzero(path.contains_points(self.xys))[0]
#         self.canvas.draw_idle()

#     def disconnect(self):
#         self.canvas.draw_idle()

# # Show the original image upon picking the point
# def on_pick(event):
#     ind = event.ind[0]
#     w, h = points.get_offsets().data[ind]
#     img_file = Path(img_meta_df.iloc[ind, 0])
#     if Path(root,img_file).is_file():
#         print(f"Showing: {img_file}")
#         img = Image.open(Path(root,img_file))
#         figs = plt.figure(figsize=(5, 5))
#         axs = figs.add_subplot(111)
#         axs.set_title(Path(img_file).name, size=14)
#         axs.set_xticks([])
#         axs.set_yticks([])
#         axs.set_xlabel(f'Dim: {round(w)} x {round(h)}', size=14)
#         axs.imshow(img)
#         figs.tight_layout()
#         figs.show()

# # Save selected image filenames
# def save_selected_imgs(df, fileName = Path("Images to discard.csv")):
#     if fileName.is_file():
#         orgData = pd.read_csv(fileName)
#         df = pd.concat([orgData, df])
#         df.set_axis(['FileName'], axis='columns').to_csv(fileName, index=False)

# # Store selected points upon pressing "enter"
# def accept(event):
#     if event.key == "enter":
#         selected_imgs = img_meta_df.iloc[selector.ind, 0].to_frame()
#         save_selected_imgs(selected_imgs)
#         print("Selected images:")
#         print(selected_imgs)
```

```
#         selector.disconnect()
#         fig.canvas.draw()

# # Plot the 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=
# ax.set_title("Press enter to after selecting the points.")
# ax.set_xlabel("Width", size=14)
# ax.set_ylabel("Height", size=14)

# # Add interaction
# selector = SelectFromCollection(ax, points)
# fig.canvas.mpl_connect("key_press_event", accept)
# fig.canvas.mpl_connect('pick_event', on_pick)
# plt.show();
```

In [ ]:
```
for img in os.listdir(path):
            try:
                img_array = cv.imread(os.path.join(path, img))
                new_array = cv.resize(img_array, (img_width, img_height))
                X_TRAIN.append(new_array)
            except Exception as e:
                pass
```

In [ ]:
```
X_TRAIN
```

In [ ]:
```
df = pd.read_csv("data/merged_item_specs.csv")
```

In [ ]:
```
df.drop(columns="Unnamed: 0", axis=1, inplace=True)
```

In [ ]:
```
df_used = df[df['condition'] != 1000.0].copy()
df_new = df[df['condition'] == 1000.0].copy()
```

In [ ]:
```
df = pd.read_csv("data/concat_df.csv",sep=',', error_bad_lines=False, index_col=F
```

In [ ]:

In [ ]:

In [ ]:
```
ls data
```

In [ ]:
```
df = pd.read_csv('data/tera_df_prepared.csv')
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

```
```

In [ ]:

```
```

In [ ]:

```
df = pd.read_csv('data/tera_df_prepared.csv', sep=',', error_bad_lines=False, inc
```

In [14]:

```
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'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['profit'] <profit_upper_limit) &
             (df['ROI'] < ROI_upper_limit) &
             (df['profit'] < profit_upper_limit) &
             (df['ROI'] > ROI_lower_limit)]

    return new_df
```

In [ ]:
```
num_columns = df.columns[3:8].values

for column in num_columns:
    df[column] = df[column].apply(lambda x: float(x))

df['cost'] = df['cost'].apply(lambda x: float(x))

df.info()
```

In [ ]:
```
df = apply_iqr_filter(df)
```

In [ ]:
```
df.to_csv('tera_df_filtered.csv', index=False)
```

In [ ]:
```
df_best = df[(df['brand'] == 'benchmade') |
             (df['brand'] == 'buck') |
             (df['brand'] == 'case') |
             (df['brand'] == 'spyderco')]
```

In [ ]:
```
df_best.head()
```

In [ ]:
```
df_listed = pd.read_csv('data/full_dataset2.csv')
```

In [ ]:
```
df_best_listed = df_listed[(df_listed['benchmade'].isna() == False) |
                           (df_listed['buck'].isna() == False) |
                           (df_listed['case'].isna() == False) |
                           (df_listed['spyderco'].isna() == False)]
```

In [ ]:
```
item_ids = df_best_listed["itemId"].values
```

In [ ]:
```
item_ids = list(item_ids)
```

In [ ]:
```
import time
```

In [ ]:
```
from ebaysdk.shopping import Connection as Shopping
```

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


import seaborn as sns
```

In [ ]:
```
response_dict = response.dict()

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

In [ ]:
```
results_list_of_dicts = response_dict['Item']
```

In [ ]:
```
#create function for organizing API call
def prepare_data(data_list):
    """
    This function takes in a list of dictionaries and prepares it
    for analysis
    """

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

    for business_data in data_list:

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

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

        keys = ['ItemID', 'Title', 'GalleryURL',
                'ItemSpecifics', 'PictureURL']

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


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

    return results
```

In [ ]:
```
def process_list(my_list):

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



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



    #index dict to appropriate index
    results_list_of_dicts = response_dict['Item']

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

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

    return full_dataset
```

In [ ]:
```
itemIds = list(df_best_listed.itemId)
```

In [ ]:
```

```

In [ ]:
```
full_dataset = []
for i in range(0, len(itemIds), 20):
    process_list(itemIds[i:i+20])
    time.sleep(1)
```

In [ ]:
```
df = pd.DataFrame(full_dataset)

df.drop_duplicates(subset='ItemID', inplace=True)
```

In [ ]:
```
df['ItemID'] = df['ItemID'].apply(lambda x: float(x))
```

In [ ]:
```
df.info()
```

In [ ]:
```
df.to_csv('data/itemSpecs2.csv')
```

In [ ]:
```
df = df.merge(df_best_listed, left_on='ItemID', right_on='itemId')
```

In [ ]:
```
df.head()
```

In [ ]:
```
df.to_csv('data/merged_item_specs.csv')
```

In [ ]:
```
df.drop(columns=['crkt', 'sog', 'victorinox', 'kershaw', 'leatherman'], inplace=T
```

In [ ]:
```
df_exploded = df.explode('PictureURL')
```

In [ ]:
```
df_exploded.info()
```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```
# df_bench2 = pd.read_csv("data/tera_benchmade.csv")
# # df_buck2 = pd.read_csv("data/tera_buck.csv")
# df_case2 = pd.read_csv("data/tera_case.csv")
# df_crkt2 = pd.read_csv("data/tera_CRKT.csv")
# df_kershaw2 = pd.read_csv("data/tera_kershaw.csv")
# df_leatherman2 = pd.read_csv("data/tera_leatherman.csv")
# df_sog = pd.read_csv("data/tera_SOG.csv")
# df_spyderco2 = pd.read_csv("data/tera_spyderco.csv")
# df_victorinox2 = pd.read_csv("data/tera_victorinox.csv")
```

In [ ]:
```
# tera_buck = pd.read_csv('data/tera_buckeroo.csv')
```

In [ ]:
```
# df_bench2.rename({'Data_field':'title',
#                   'Data_field1':'avg_price',
#                   'Data_field2': 'avg_shipping',
#                   'Field2': 'date_sold'
#                  }, axis=1, inplace=True)
```

In [ ]:
```
# df_bench2.head()
```

In [ ]:
```
# df_bench2.drop(['Field', 'Price'], axis=1, inplace=True)
```

In [ ]:
```
# df_bench2.head()
```

In [ ]:
```
# df_bench2.to_csv('data/tera_benchmade_clean.csv', index=False)
```

In [ ]:
```
# df_case2.head()
```

In [ ]:
```
# df_case2.rename({'url':'title',
#                   'Field5':'title2',
#                   'Field4': 'date_sold',
#                  }, axis=1, inplace=True)
```

In [ ]:
```
# df_case2['title'].fillna(df_case2['title2'], inplace=True)
```

In [ ]:
```
# df_case2.drop(['Field', 'units_sold', 'title2', 'Price'], axis=1, inplace=True)
```

In [ ]:
```
# df_case2.head()
```

In [ ]:
```
# df_case2.to_csv('data/tera_case_clean.csv', index=False)
```

In [ ]:
```
# tera_buck['title'].fillna(tera_buck['title2'], inplace=True)
```

In [ ]:
```
# tera_buck.drop('title2', axis=1, inplace=True)
```

In [ ]:
```
# df_crkt2.rename({'Field2':'date_sold',
#                  'title1': 'title',
#                  'avg_cost':'avg_price'}, axis=1, inplace=True)
```

In [ ]:
```
# df_crkt2.head()
```

In [ ]:
```
# df_crkt2['title'].fillna(df_crkt2['title2'], inplace=True)
```

In [ ]:
```
# df_crkt2.head()
```

In [ ]:
```
# df_crkt2.drop(['Field', 'Price', 'title2'], axis=1, inplace=True)
```

In [ ]:
```
# df_crkt2.head()
```

In [ ]:
```
# df_crkt2.to_csv('data/tera_crkt_clean.csv', index=False)
```

In [ ]:
```

```

In [ ]:
```
# df_kershaw2.rename({'Field':'url', 'Field3':'date_sold', 'Field1': 'title'}, ax
```

In [ ]:
```
# df_kershaw2.head()
```

In [ ]:
```
# df_kershaw2.drop(['Field2', 'Price', 'url'], axis=1, inplace=True)
```

In [ ]:
```
# df_kershaw2.to_csv('data/tera_kershaw_clean.csv', index=False)
```

In [ ]:
```
# df_leatherman2.head()
```

In [ ]:
```
# df_leatherman2['title'].fillna(df_leatherman2['title1'], inplace=True)
```

In [ ]:
```
# df_leatherman2.head()
```

In [ ]:
```
# df_leatherman2.rename({'Field2':'date_sold'}, axis=1, inplace=True)
```

In [ ]:
```
# df_leatherman2.drop(['Field', 'Price', 'title1'], axis=1, inplace=True)
```

In [ ]:
```
# df_leatherman2.head()
```

In [ ]:
```
# df_leatherman2.to_csv('data/tera_leatherman_clean.csv', index=False)
```

In [ ]:
```
# df_sog.rename({'title1': 'title'}, axis=1, inplace=True)
```

In [ ]:
```
# df_sog['title'].fillna(df_sog['title2'], inplace=True)
```

In [ ]:
```
# df_sog.head()
```

In [ ]:
```
# df_sog.drop(['Field', 'Price', 'title2'], axis=1, inplace=True)
```

In [ ]:
```
# df_sog.head()
```

In [ ]:
```
# df_sog.to_csv('data/tera_sog_clean.csv',index=False)
```

In [ ]:
```
# df_spyderco2.head()
```

In [ ]:
```
# df_spyderco2.rename({'Field3': 'title', 'Field2': 'date_sold'}, axis=1, inplace
```

In [ ]:
```
# df_spyderco2['title'].fillna(df_spyderco2['name'], inplace=True)
```

In [ ]:
```
# df_spyderco2.head()
```

In [ ]:
```
# df_spyderco2.drop(['units_sold', 'Price', 'name'], axis=1, inplace=True)
```

In [ ]:
```
# df_spyderco2.head()
```

In [ ]:
```
# df_spyderco2.to_csv('data/tera_spyderco_clean.csv', index=False)
```

In [ ]:
```
# df_victorinox2.rename({'FfImage':'Image', 'Field3':'date_sold'}, axis=1, inplac
```

In [ ]:
```
# df_victorinox2.head()
```

In [ ]:
```
# df_victorinox2['title'].fillna(df_victorinox2['title2'], inplace=True)
```

In [ ]:
```
# df_victorinox2.head()
```

In [ ]:
```
# df_victorinox2.drop(['url','Field2', 'Price', 'title2'], axis=1, inplace=True)
```

In [ ]:
```
# df_victorinox2.head()
```

In [ ]:
```
# df_victorinox2.to_csv('data/tera_victorinox_clean.csv', index=False)
```

In [ ]:
```
bucket_dict = {'benchmade': 45.0,
               'buck': 20.0,
               'case': 20.0,
               'crkt': 15.0,
               'kershaw': 15.0,
               'leatherman': 30.0,
               'sog': 15.0,
               'spyderco': 30.0,
               'victorinox': 20.0
              }

#x = position of bucket_dictionary
def prepare_tera_df(df, x):
    df['avg_price'] = df['avg_price'].str.replace("$", "")
    df['avg_price'] = df['avg_price'].str.replace(",", "")
    df['avg_price'] = df['avg_price'].apply(float)

    df['avg_shipping'] = df['avg_shipping'].str.replace("$", "")
    df['avg_shipping'] = df['avg_shipping'].str.replace(",", "")
    df['avg_shipping'] = df['avg_shipping'].apply(float)

    df['converted_price'] = (df['avg_price'] + df['avg_shipping'])

    df['profit'] = (df['converted_price'] - list(bucket_dict.values())[x])
    df['ROI'] = (df['profit']/(list(bucket_dict.values())[x]))*100.0

    df['brand'] = list(bucket_dict.keys())[x]
    df['cost'] = list(bucket_dict.values())[x]


    return df
```

In [ ]:
```
tera_bench = prepare_tera_df(df_bench2,0)
tera_case = prepare_tera_df(df_case2,2)
tera_crkt = prepare_tera_df(df_crkt2, 3)
tera_kershaw = prepare_tera_df(df_kershaw2, 4)
tera_leatherman = prepare_tera_df(df_leatherman2, 5)
tera_sog = prepare_tera_df(df_sog, 6)
tera_spyderco = prepare_tera_df(df_spyderco2, 7)
tera_victorinox = prepare_tera_df(df_victorinox2, 8)
```

In [ ]:
```
tera_buck = prepare_tera_df(tera_buck, 1)
```

In [ ]:
```
df = pd.concat([tera_bench, tera_buck, tera_case, tera_crkt,
                tera_kershaw, tera_leatherman,
                tera_sog, tera_spyderco, tera_victorinox])
```

In [ ]:
```
df.columns
```

In [ ]:
```
df.drop(['Field2','Unnamed: 0'], axis=1, inplace=True)
```

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

In [ ]:
```
df.info()
```

In [ ]:
```
df['brand'].value_counts()
```

In [ ]:
```
tera_bench.to_csv('data/tera_bench_prepared.csv', index=False)
tera_case.to_csv('data/tera_case_prepared.csv', index=False)
tera_crkt.to_csv('data/tera_crkt_prepared.csv', index=False)
tera_kershaw.to_csv('data/tera_kershaw_prepared.csv', index=False)
tera_leatherman.to_csv('data/tera_leatherman_prepared.csv', index=False)
tera_sog.to_csv('data/tera_sog_prepared.csv', index=False)
tera_spyderco.to_csv('data/tera_spyderco_prepared.csv', index=False)
tera_victorinox.to_csv('data/tera_victorinox_prepared.csv', index=False)

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

In [17]:
```
tera_bench = pd.read_csv('data/tera_bench_prepared.csv')
tera_buck = pd.read_csv('data/tera_buckeroo.csv')
tera_case = pd.read_csv('data/tera_case_prepared.csv')
tera_crkt = pd.read_csv('data/tera_crkt_prepared.csv')
tera_kershaw = pd.read_csv('data/tera_kershaw_prepared.csv')
tera_leatherman = pd.read_csv('data/tera_leatherman_prepared.csv')
tera_sog = pd.read_csv('data/tera_sog_prepared.csv')
tera_spyderco = pd.read_csv('data/tera_spyderco_prepared.csv')
tera_victorinox = pd.read_csv('data/tera_victorinox_prepared.csv')
```

In [8]:
```
df = pd.read_csv('data/tera_df_prepared.csv', sep=',', error_bad_lines=False, inc
```

In [ ]:
```
df.head()
```

In [ ]:
```
df.info()
```

In [ ]:
```
df.brand.value_counts()
```

In [ ]:
```

```

In [ ]:
```
# df_bench2 = pd.read_csv("data/benchmade_scraped.csv")
# df_buck2 = pd.read_csv("data/buck_scraped.csv")
# df_case2 = pd.read_csv("data/case_scraped.csv")
# df_crkt2 = pd.read_csv("data/CRKT_scraped.csv")
# df_kershaw2 = pd.read_csv("data/kershaw_scraped.csv")
# df_spyderco2 = pd.read_csv("data/spyderco_scraped.csv")
# df_victorinox2 = pd.read_csv("data/victorinox_scraped.csv")
```

In [ ]:
```
# brands = ['victorinox', 'kershaw', 'buck',
#           'case xx', 'benchmade', 'spyderco',
#           'crkt', 'case']

# df_buck2.brand = df_buck2.brand.str.lower()

# df_buck2 = df_buck2[df_buck2.brand.isin(brands)]

# df_buck2.brand.value_counts()
```

In [ ]:
```
# df_buck2['buck'] = 20.0
```

In [ ]:
```
# brands = ['victorinox', 'kershaw', 'buck',
#           'case xx', 'benchmade', 'spyderco',
#           'crkt', 'case']

# def prepare_brands2(df, bucket_dict_position):

#     df.brand = df.brand.str.lower()

#     df = df[df.brand.isin(brands)].copy()

#     df[str(list(bucket_dict.keys())[bucket_dict_position])] = float(list(bucket

#     df['profit'] = (df['price'] - df[list(bucket_dict.keys())[bucket_dict_posit
#     df['ROI'] = (df['profit']/(df[list(bucket_dict.keys())[bucket_dict_positior

#     return df
```

In [ ]:
```
#No duplicates
df.title.duplicated().sum()
```

In [ ]:
```
display(pd.concat(g for _, g in df.groupby("title") if len(g) > 1).tail(20))
```

In [ ]:
```
display(pd.concat(g for _, g in df.groupby("title") if len(g) > 1).tail(20).heac
```

In [ ]:
```
# print('Benchmade Value Counts')
# display(df.benchmade.value_counts(normalize=False))
# print('-----------------------------------------')

# print('Buck Value Counts')
# display(df.buck.value_counts(normalize=False))
# print('-----------------------------------------')

# print('Case Value Counts')
# display(df.case.value_counts(normalize=False))
# print('-----------------------------------------')

# print('CRKT Value Counts')
# display(df.crkt.value_counts(normalize=False))
# print('-----------------------------------------')

# print('Kershaw Value Counts')
# display(df.kershaw.value_counts(normalize=False))
# print('-----------------------------------------')

# print('Leatherman Value Counts')
# display(df.leatherman.value_counts(normalize=False))
# print('-----------------------------------------')


# print('Spyderco Value Counts')
# display(df.spyderco.value_counts(normalize=False))
# print('-----------------------------------------')

# print('Victorinox Value Counts')
# display(df.victorinox.value_counts(normalize=False))
# print('-----------------------------------------')
# print('')

# display(df.info())
```

In [ ]:
```
# df_bench = pd.read_csv("data/df_bench.csv")
# df_buck = pd.read_csv("data/df_buck.csv")
# df_case = pd.read_csv("data/df_case.csv")
# df_crkt = pd.read_csv("data/df_crkt.csv")
# df_kershaw = pd.read_csv("data/df_kershaw.csv")
# df_leatherman = pd.read_csv("data/df_leatherman.csv")
# df_spyderco = pd.read_csv("data/df_spyderco.csv")
# df_victorinox = pd.read_csv("data/df_victorinox.csv")
```

In [ ]:
```
# print(f'df_bench length: {len(df_bench)}')
# print(f'df_buck length: {len(df_buck)}')
# print(f'df_case length: {len(df_case)}')
# print(f'df_crkt length: {len(df_crkt)}')
# print(f'df_kershaw length: {len(df_kershaw)}')
# print(f'df_spyderco length: {len(df_spyderco)}')
# print(f'df_victorinox length: {len(df_victorinox)}')
```

Domain Understading: Cost Breakdown

- padded envelopes: $0.50 per knife
- flatrate shipping: $4.45 per knife
- brand knife at surplus store: 15, 20, 30, or 45 dollars per knife
- overhead expenses (gas, cleaning suplies, sharpening supplies, etc): $7

In [ ]:
```
df.drop(columns=['shippingInfo', 'listingInfo', 'sellingStatus'], inplace=True)
df.head()
```

In [ ]:
```
# bucket_dict = {'benchmade': 45.0,
#                'buck': 20.0,
#                'case': 20.0,
#                'crkt': 15.0,
#                'kershaw': 15.0,
#                'leatherman': 30.0,
#                'spyderco': 30.0,
#                'victorinox': 20.0
#               }
```

In [ ]:
```
# #flip the bucket dictionary to divide dataframe by MY cost of knives
# #at the surplus store

# flipped = {}

# for key, value in bucket_dict.items():
#     if value not in flipped:
#         flipped[value] = [key]
#     else:
#         flipped[value].append(key)
```

In [ ]:
```
#there are only 4 bins to work with. Much easier.
# flipped
```

In [ ]:
```
# df_15 = df.loc[(df['crkt'] != 0) | (df['kershaw'] != 0)]
# df_20 = df.loc[(df['buck'] != 0) | (df['case'] != 0)| (df['victorinox'] != 0)]
# df_30 = df.loc[(df['leatherman'] != 0) | (df['spyderco'] != 0)]
# df_45 = df.loc[df['benchmade'] != 0]
```

In [9]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 66881 entries, 0 to 66880
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   Image            66881 non-null  object
 1   date_sold        52819 non-null  object
 2   title            59488 non-null  object
 3   avg_price        66881 non-null  object
 4   avg_shipping     66881 non-null  object
 5   converted_price  66881 non-null  object
 6   profit           66881 non-null  object
 7   ROI              66881 non-null  object
 8   brand            66881 non-null  object
 9   cost             66881 non-null  object
 10  url              2458 non-null   object
dtypes: object(11)
memory usage: 5.6+ MB
```

In [10]:
```
num_columns = df.columns[3:8].values
```

In [11]:
```
for column in num_columns:
    df[column] = df[column].apply(lambda x: float(x))
```

In [12]:
```
df['cost'] = df['cost'].apply(lambda x: float(x))
```

In [13]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 66881 entries, 0 to 66880
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Image            66881 non-null  object 
 1   date_sold        52819 non-null  object 
 2   title            59488 non-null  object 
 3   avg_price        66881 non-null  float64
 4   avg_shipping     66881 non-null  float64
 5   converted_price  66881 non-null  float64
 6   profit           66881 non-null  float64
 7   ROI              66881 non-null  float64
 8   brand            66881 non-null  object 
 9   cost             66881 non-null  float64
 10  url              2458 non-null   object 
dtypes: float64(6), object(5)
memory usage: 5.6+ MB
```

In [ ]:
```

```

In [18]:
```
df_45 = df.loc[df['brand'] == 'benchmade']
df_30 = df.loc[(df['brand'] == 'leatherman') | (df['brand'] == 'spyderco')]
df_20 = df.loc[(df['brand'] == 'buck') | (df['brand'] == 'case') | (df['brand'] =
df_15 = df.loc[(df['brand'] == 'crkt') | (df['brand'] == 'kershaw') | (df['brand'
```

In [19]:
```
display(df_45['brand'].value_counts())
display(df_30['brand'].value_counts())
display(df_20['brand'].value_counts())
display(df_15['brand'].value_counts())
```

```
benchmade    7139
Name: brand, dtype: int64

spyderco      7383
leatherman    2829
Name: brand, dtype: int64

buck          10000
case           9981
victorinox     9981
Name: brand, dtype: int64

kershaw    9981
crkt       5525
sog        4062
Name: brand, dtype: int64
```

In [ ]:
```
# create histogram for each column
df['converted_price'].hist(figsize = (18, 15), bins = 'auto');
```

In [ ]:
```
# # create histogram for each column
# df['profit'].hist(figsize = (18, 15), bins = 'auto');
```

In [ ]:
```
sns.boxplot(x=df['converted_price'])
```

In [ ]:
```
sns.boxplot(x=df['profit'])
```

In [ ]:
```
display(df_15.info())
display(df_20.info())
display(df_30.info())
display(df_45.info())
```

In [ ]:
```
# def apply_iqr_filter2(df):

#     price_Q1 = df['price'].quantile(0.25)
#     price_Q3 = df['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'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['price'] < price_upper_limit) &
#                 (df['profit'] <profit_upper_limit) &
#                 (df['ROI'] < ROI_upper_limit) &
#                 (df['profit'] < profit_upper_limit) &
#                 (df['ROI'] > ROI_lower_limit)]

#     return new_df
```

In [20]:
```
tera_bench = apply_iqr_filter(tera_bench)
tera_buck = apply_iqr_filter(tera_bench)
tera_case =  apply_iqr_filter(tera_case)
tera_crkt = apply_iqr_filter(tera_crkt)
tera_kershaw = apply_iqr_filter(tera_kershaw)
tera_leatherman = apply_iqr_filter(tera_leatherman)
tera_sog = apply_iqr_filter(tera_sog)
tera_spyderco = apply_iqr_filter(tera_spyderco)
tera_victorinox = apply_iqr_filter(tera_victorinox)
```

```
price upper limit: $287.11
price lower limit: $-28.27
-----------------------------------
profit upper limit: $242.11
profit lower limit: $-73.27
-----------------------------------
ROI upper limit: 538.03%
ROI lower limit: -162.82%
-----------------------------------
price upper limit: $267.49
price lower limit: $-19.15
-----------------------------------
profit upper limit: $222.49
profit lower limit: $-64.15
-----------------------------------
ROI upper limit: 494.42%
ROI lower limit: -142.56%
-----------------------------------
price upper limit: $165.38
price lower limit: $-35.23
-----------------------------------
profit upper limit: $145.38
profit lower limit: $-55.23
-----------------------------------
ROI upper limit: 726.88%
ROI lower limit: -276.12%
-----------------------------------
price upper limit: $72.75
price lower limit: $-9.25
-----------------------------------
profit upper limit: $57.75
profit lower limit: $-24.25
-----------------------------------
ROI upper limit: 385.0%
ROI lower limit: -161.67%
-----------------------------------
price upper limit: $73.64
price lower limit: $-13.4
-----------------------------------
profit upper limit: $58.64
profit lower limit: $-28.4
-----------------------------------
ROI upper limit: 390.93%
ROI lower limit: -189.33%
-----------------------------------
price upper limit: $92.02
price lower limit: $-20.03
```

```
----------------------------------
profit upper limit: $62.02
profit lower limit: $-50.02
----------------------------------
ROI upper limit: 206.72%
ROI lower limit: -166.75%
----------------------------------
price upper limit: $95.4
price lower limit: $-23.0
----------------------------------
profit upper limit: $80.4
profit lower limit: $-38.0
----------------------------------
ROI upper limit: 536.0%
ROI lower limit: -253.33%
----------------------------------
price upper limit: $287.62
price lower limit: $-91.98
----------------------------------
profit upper limit: $257.62
profit lower limit: $-121.98
----------------------------------
ROI upper limit: 858.75%
ROI lower limit: -406.58%
----------------------------------
price upper limit: $64.0
price lower limit: $-16.0
----------------------------------
profit upper limit: $44.0
profit lower limit: $-36.0
----------------------------------
ROI upper limit: 220.0%
ROI lower limit: -180.0%
----------------------------------
```

In [21]:
```
df_15 = apply_iqr_filter(df_15)
df_20 = apply_iqr_filter(df_20)
df_30 = apply_iqr_filter(df_30)
df_45 = apply_iqr_filter(df_45)
```

```
price upper limit: $80.82
price lower limit: $-16.49
----------------------------------
profit upper limit: $65.82
profit lower limit: $-31.5
----------------------------------
ROI upper limit: 438.83%
ROI lower limit: -209.97%
----------------------------------
price upper limit: $122.14
price lower limit: $-42.9
----------------------------------
profit upper limit: $102.14
profit lower limit: $-62.9
----------------------------------
ROI upper limit: 510.7%
ROI lower limit: -314.5%
----------------------------------
price upper limit: $252.49
price lower limit: $-90.69
----------------------------------
profit upper limit: $222.49
profit lower limit: $-120.69
----------------------------------
ROI upper limit: 741.62%
ROI lower limit: -402.31%
----------------------------------
price upper limit: $287.11
price lower limit: $-28.27
----------------------------------
profit upper limit: $242.11
profit lower limit: $-73.27
----------------------------------
ROI upper limit: 538.03%
ROI lower limit: -162.82%
----------------------------------
```

In [37]:
```
df_filtered = apply_iqr_filter(df.copy())
```

```
price upper limit: $166.52
price lower limit: $-64.33
----------------------------------
profit upper limit: $126.43
profit lower limit: $-69.06
----------------------------------
ROI upper limit: 535.42%
ROI lower limit: -285.25%
----------------------------------
```

In [38]:
```
df_filtered.reset_index(drop=True, inplace=True)
```

In [ ]:
```
df2 = pd.read_csv("data/merged_item_specs.csv")
```

In [ ]:
```
df_spyder2 = df2.loc[df2['spyderco'].isna()==False].copy()
```

In [ ]:
```
df_spyder2.drop('Unnamed: 0', axis=1, inplace=True)
```

In [ ]:
```
df_spyder2.reset_index(drop=True, inplace=True)
```

In [ ]:
```
from ast import literal_eval
```

In [ ]:
```
df_spyder2.dropna(subset=['PictureURL'], inplace=True)
```

In [ ]:
```
df_spyder2['PictureURL'] = df_spyder2['PictureURL'].apply(literal_eval) #convert
```

In [ ]:
```
df_spyder = df_spyder2.explode('PictureURL').copy()
```

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

In [ ]:
```

```

In [ ]:
```
df_spyder2 = apply_iqr_filter(df_spyder2)
```

In [ ]:
```
df_bench2 = df2.loc[df2['benchmade'].isna()==False].copy()
df_bench2.drop('Unnamed: 0', axis=1, inplace=True)
df_bench2.reset_index(drop=True, inplace=True)
```

In [ ]:
```
df_bench2 = apply_iqr_filter(df_bench2)
```

In [ ]:
```
sns.histplot(df_bench2['converted_price']);
```

In [ ]:
```
sns.histplot(df_bench2['profit'])
```

In [ ]:
```
sns.histplot(df_bench2['ROI'])
```

In [ ]:
```
df_bench2['PictureURL'] = df_bench2['PictureURL'].apply(literal_eval)
```

In [ ]:
```
df_bench = df_bench2.explode('PictureURL').copy()
```

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

In [ ]:
```
# df_15 = df_scrub.loc[(df_scrub['crkt'] != 0) | (df_scrub['kershaw'] != 0)]
# df_20 = df_scrub.loc[(df_scrub['buck'] != 0) | (df_scrub['case'] != 0)| (df_scr
# df_30 = df_scrub.loc[(df_scrub['leatherman'] != 0) | (df_scrub['spyderco'] != 0
# df_45 = df_scrub.loc[df_scrub['benchmade'] != 0]
```

In [ ]:
```
df_bench.info()
```

In [ ]:
```
fig, axes  = plt.subplots(figsize=(15,15), ncols=2, sharey=True, sharex=True)
sns.histplot(df_new['converted_price'], ax=axes[0], color='gold')
sns.histplot(df_used['converted_price'], ax=axes[1], color='firebrick')


for n in range(2):
    col = n%2  # just the remainder of n divided by 2
    axes[col].set_xlabel('Resale Value(US Dollars)', fontsize=20)
    axes[col].set_ylabel('number of knives listed (ebay)', fontsize=15)
    axes[col].tick_params(axis='both', labelsize=13)

fig.suptitle("Resale Value of Surplus Store Knives by Price", fontsize=24, x=0.44
fig.legend(labels=["new", "used"], loc=(.77, .74), fontsize='x-large')
plt.show();
```

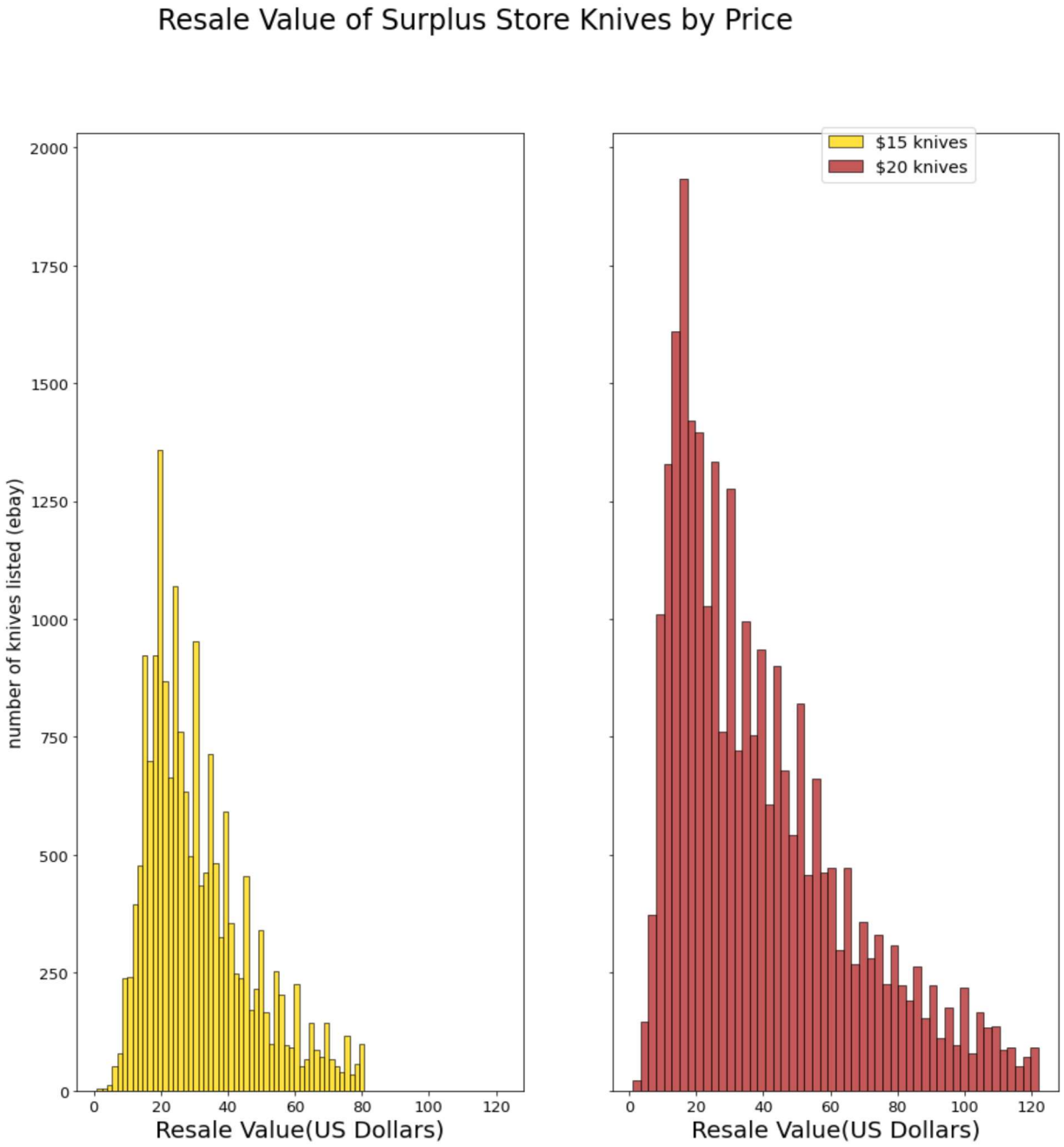
In [ ]:
```
def describe_num_cats(df, brand):
    display(df[df[brand].isna() == False][['converted_price', 'profit', 'ROI']].d
```

In [ ]:
```
df_used[df_used['benchmade'] != 0][['converted_price', 'profit', 'ROI']].describe
```

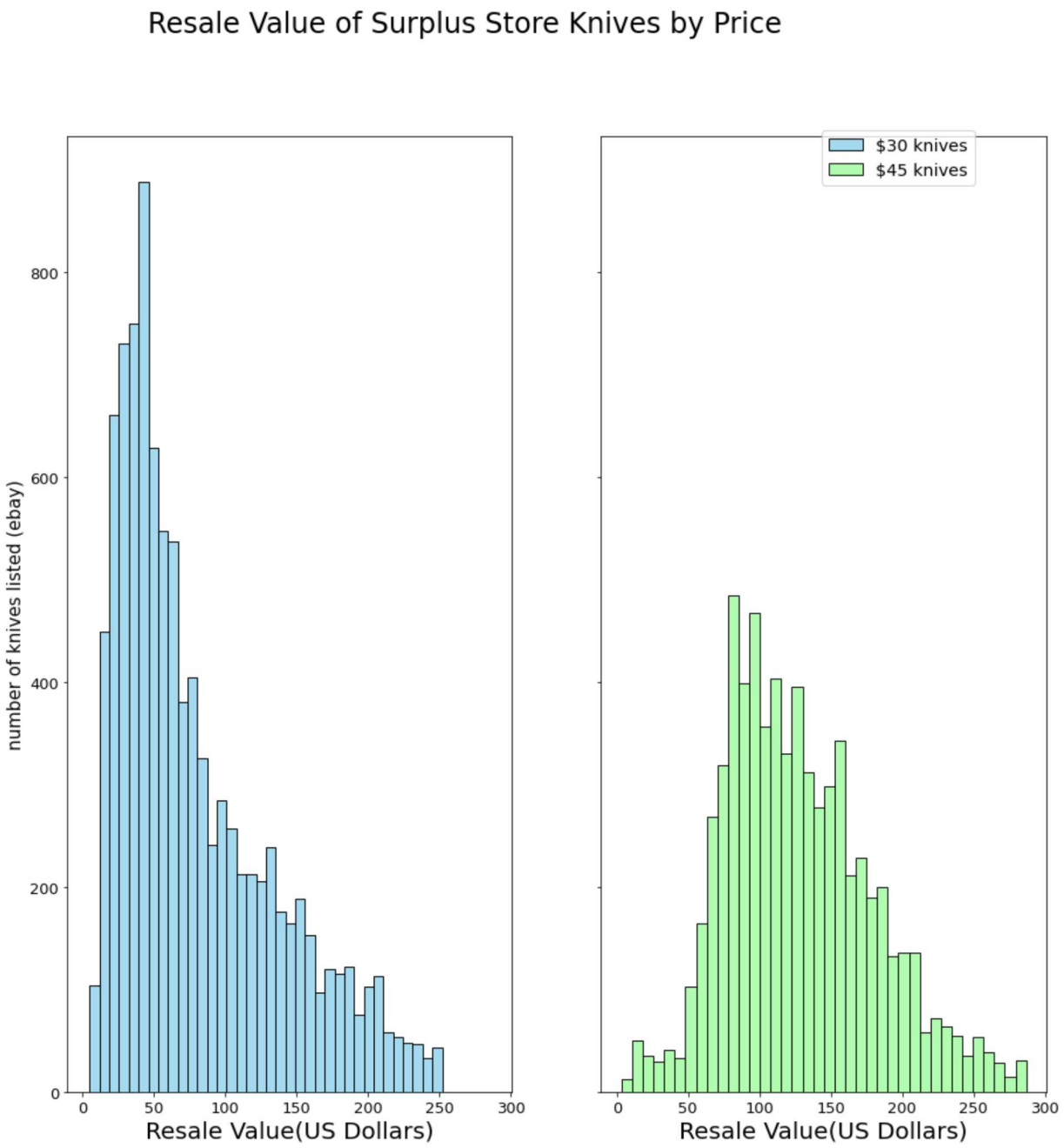
In [ ]:
```
df.columns
```

In [ ]:
```
bucket_dict = {'benchmade': 45.0,
               'buck': 20.0,
               'case': 20.0,
               'crkt': 15.0,
               'kershaw': 15.0,
               'leatherman': 30.0,
               'sog': 15.0,
               'spyderco': 30.0,
               'victorinox': 20.0
              }
```

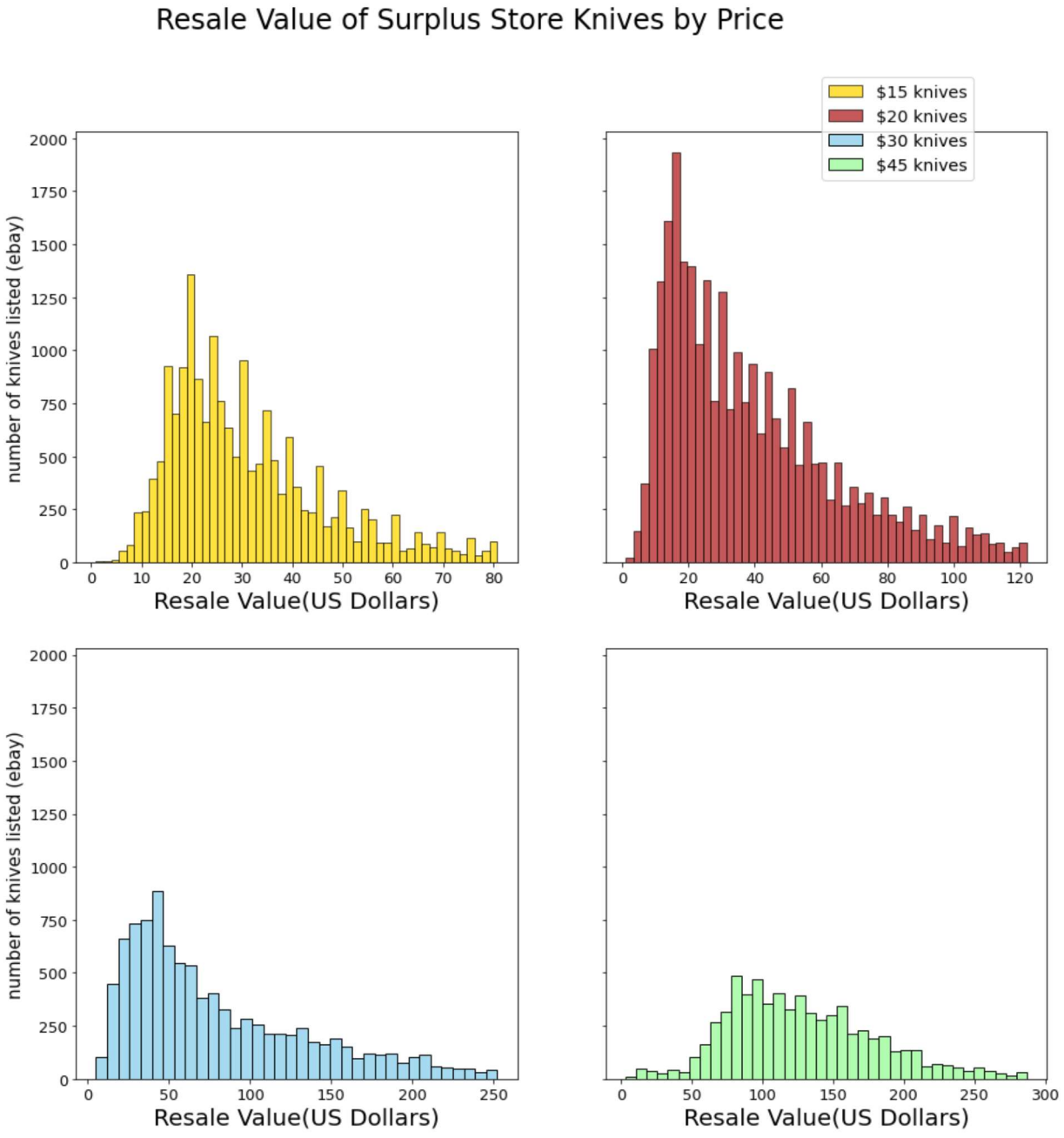
In [22]:
```
fig, axes  = plt.subplots(figsize=(15,15), ncols=2, sharey=True, sharex=True)
sns.histplot(df_15['converted_price'], ax=axes[0], color='gold')
sns.histplot(df_20['converted_price'], ax=axes[1], color='firebrick')


for n in range(2):
    col = n%2  # just the remainder of n divided by 2
    axes[col].set_xlabel('Resale Value(US Dollars)', fontsize=20)
    axes[col].set_ylabel('number of knives listed (ebay)', fontsize=15)
    axes[col].tick_params(axis='both', labelsize=13)

fig.suptitle("Resale Value of Surplus Store Knives by Price", fontsize=24, x=0.44
fig.legend(labels=["$15 knives", "$20 knives", "$30 knives", "$45 knives"], loc=(
plt.show();
```



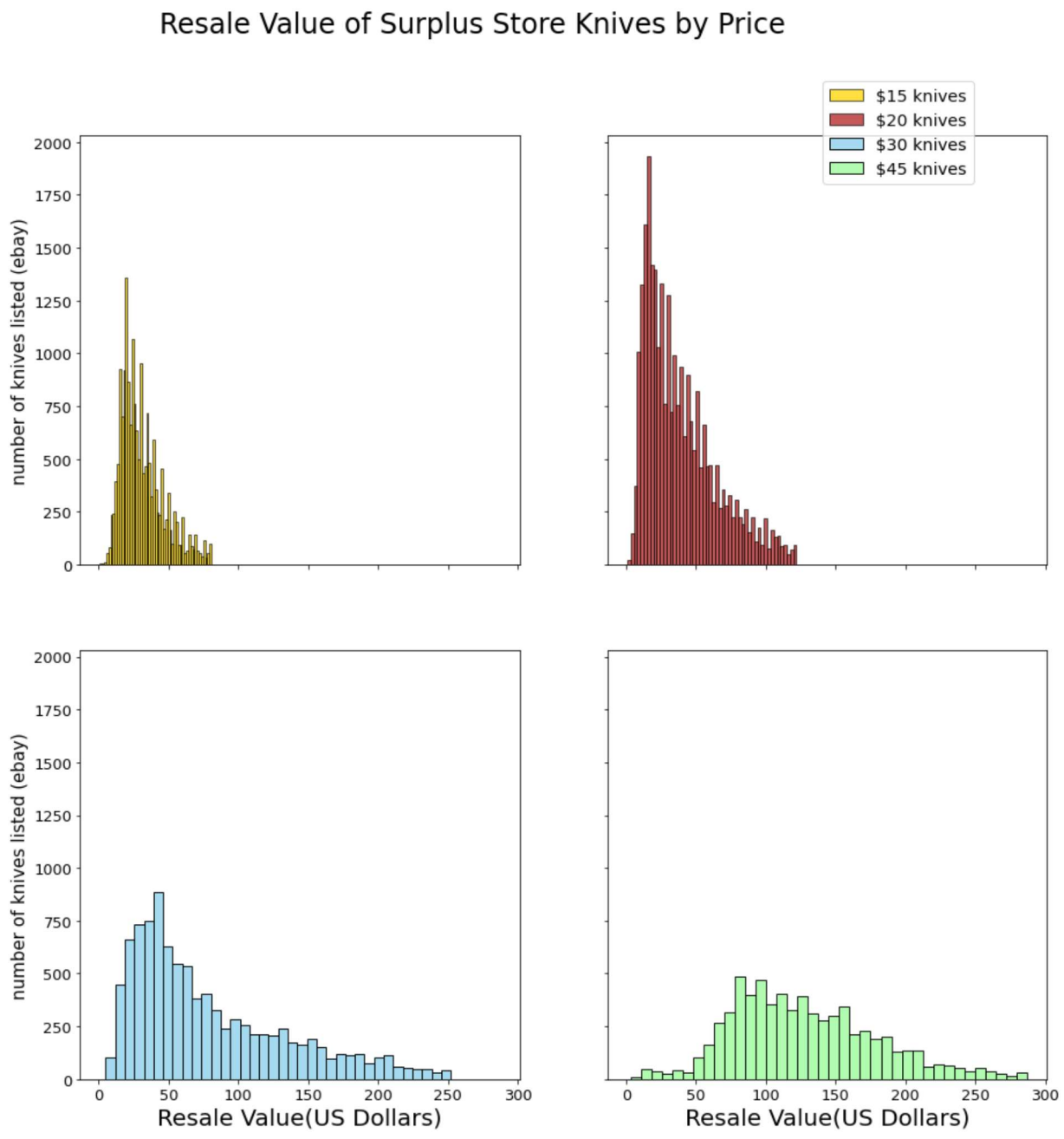
In [23]:
```
fig, axes  = plt.subplots(figsize=(15,15), ncols=2, sharey=True, sharex=True)
sns.histplot(df_30['converted_price'], ax=axes[0],  color='skyblue')
sns.histplot(df_45['converted_price'], ax=axes[1], color='palegreen')

for n in range(2):
    col = n%2  # just the remainder of n divided by 2
    axes[col].set_xlabel('Resale Value(US Dollars)', fontsize=20)
    axes[col].set_ylabel('number of knives listed (ebay)', fontsize=15)
    axes[col].tick_params(axis='both', labelsize=13)

fig.suptitle("Resale Value of Surplus Store Knives by Price", fontsize=24, x=0.44
fig.legend(labels=["$30 knives", "$45 knives"], loc=(.77, .84), fontsize='x-large
plt.show();
```



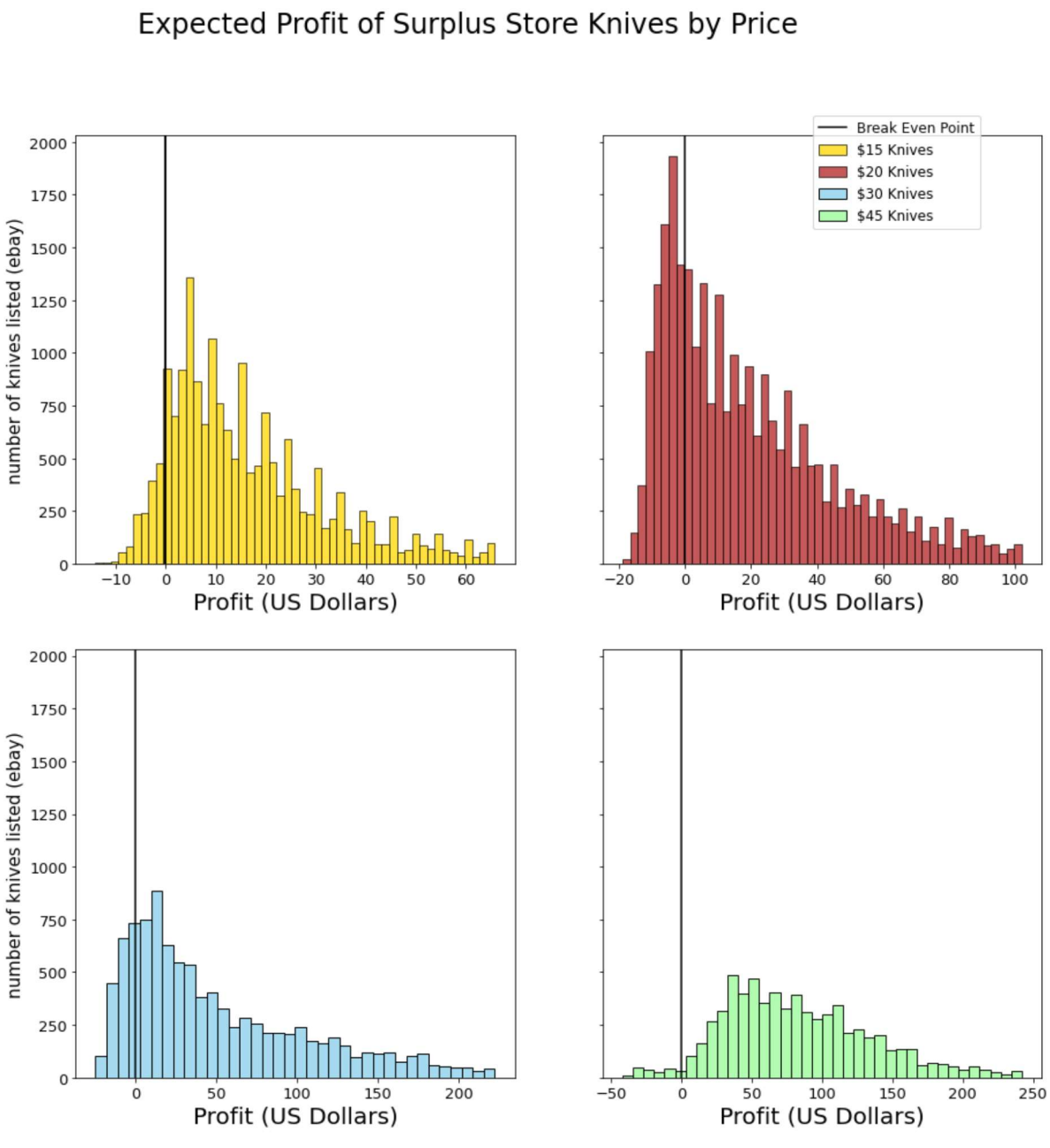
In [25]:
```
fig, axes  = plt.subplots(figsize=(15,15), ncols=2, nrows=2, sharey=True)
sns.histplot(df_15['converted_price'], ax=axes[0][0], color='gold')
sns.histplot(df_20['converted_price'], ax=axes[0][1], color='firebrick')
sns.histplot(df_30['converted_price'], ax=axes[1][0],  color='skyblue')
sns.histplot(df_45['converted_price'], ax=axes[1][1], color='palegreen')

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('Resale Value(US Dollars)', fontsize=20)
    axes[row][col].set_ylabel('number of knives listed (ebay)', fontsize=15)
    axes[row][col].tick_params(axis='both', labelsize=13)

fig.suptitle("Resale Value of Surplus Store Knives by Price", fontsize=24, x=0.44
fig.legend(labels=["$15 knives", "$20 knives", "$30 knives", "$45 knives"], loc=(
plt.show();
```



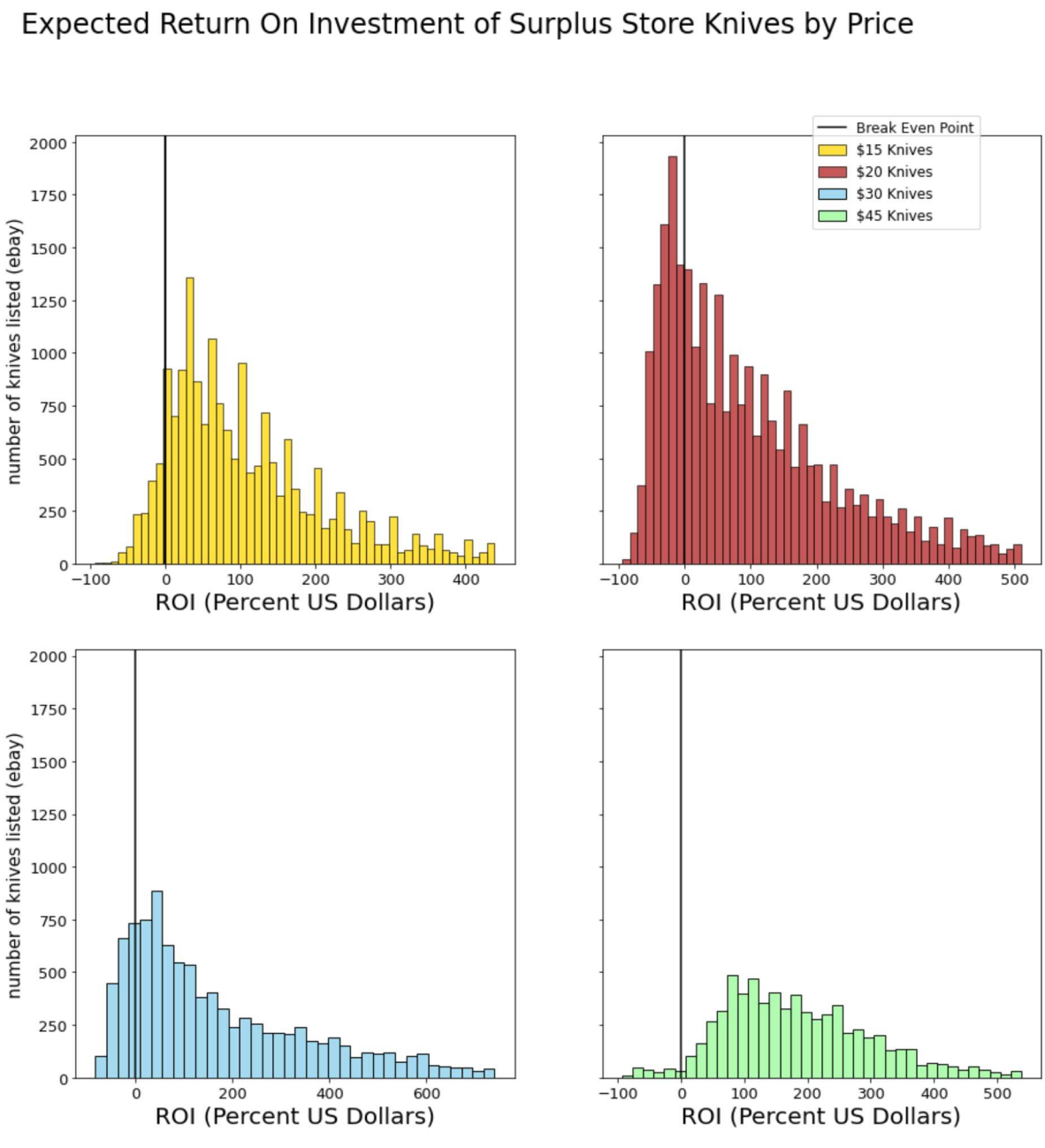
In [26]:
```
fig, axes  = plt.subplots(figsize=(15,15), ncols=2, nrows=2, sharey=True, sharex=
sns.histplot(df_15['converted_price'], ax=axes[0][0], color='gold')
sns.histplot(df_20['converted_price'], ax=axes[0][1], color='firebrick')
sns.histplot(df_30['converted_price'], ax=axes[1][0],  color='skyblue')
sns.histplot(df_45['converted_price'], ax=axes[1][1], color='palegreen')

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('Resale Value(US Dollars)', fontsize=20)
    axes[row][col].set_ylabel('number of knives listed (ebay)', fontsize=15)
    axes[row][col].tick_params(axis='both', labelsize=13)

fig.suptitle("Resale Value of Surplus Store Knives by Price", fontsize=24, x=0.44
fig.legend(labels=["$15 knives", "$20 knives", "$30 knives", "$45 knives"], loc=(
plt.show();
```



In [27]:
```
fig, axes  = plt.subplots(figsize=(15,15), ncols=2, nrows=2, sharey=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 Knive
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 listed (ebay)', fontsize=15)
    axes[row][col].tick_params(axis='both', labelsize=13)
    axes[row][col].axvline(x = 0, color = 'black')

axes[0][0].axvline(x = 0, color = 'black', label= 'Break Even Point')
fig.suptitle("Expected Profit of Surplus Store Knives by Price", fontsize=24, x=0
fig.legend(loc=(.77, .80), fontsize='large')

plt.show();
```


Expected Profit of Surplus Store Knives by Price
Break Even Point
$15 Knives
$20 Knives
$30 Knives
$45 Knives
number of knives listed (ebay)
Profit (US Dollars)
number of knives listed (ebay)
Profit (US Dollars)
number of knives listed (ebay)
Profit (US Dollars)
Profit (US Dollars)

In [28]:
```
fig, axes  = plt.subplots(figsize=(15,15), ncols=2, nrows=2, sharey=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 (Percent US Dollars)', fontsize=20)
    axes[row][col].set_ylabel('number of knives listed (ebay)', fontsize=15)
    axes[row][col].tick_params(axis='both', labelsize=13)
    axes[row][col].axvline(x = 0, color = 'black')

axes[0][0].axvline(x = 0, color = 'black', label= 'Break Even Point')
fig.suptitle("Expected Return On Investment of Surplus Store Knives by Price", fo
fig.legend(loc=(.77, .80), fontsize='large')

plt.show();
```



In [ ]:
```
df_20_new = pd.concat([tera_buck, tera_case])
df_30_new = tera_spyderco.copy()
```

In [ ]:
```
fig, axes  = plt.subplots(figsize=(15,15), ncols=2, nrows=2, sharey=True)
sns.histplot(df_15['converted_price'], ax=axes[0][0], color='gold')
sns.histplot(df_20_new['converted_price'], ax=axes[0][1], color='firebrick')
sns.histplot(df_30_new['converted_price'], ax=axes[1][0],  color='skyblue')
sns.histplot(df_45['converted_price'], ax=axes[1][1], color='palegreen')

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('Resale Value(US Dollars)', fontsize=20)
    axes[row][col].set_ylabel('number of knives listed (ebay)', fontsize=15)
    axes[row][col].tick_params(axis='both', labelsize=13)

fig.suptitle("Resale Value of Surplus Store Knives by Price", fontsize=24, x=0.44
fig.legend(labels=["$15 knives", "$20 knives", "$30 knives", "$45 knives"], loc=(
plt.show();
```

In [ ]:
```
fig, axes  = plt.subplots(figsize=(15,15), ncols=2, nrows=2, sharey=True)
sns.histplot(df_15['profit'], ax=axes[0][0], color='gold', label="$15 Knives")
sns.histplot(df_20_new['profit'], ax=axes[0][1], color='firebrick',  label="$20 K
sns.histplot(df_30_new['profit'], ax=axes[1][0],  color='skyblue',  label="$30 Kn
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 listed (ebay)', fontsize=15)
    axes[row][col].tick_params(axis='both', labelsize=13)
    axes[row][col].axvline(x = 0, color = 'black')

axes[0][0].axvline(x = 0, color = 'black', label= 'Break Even Point')
fig.suptitle("Expected Profit of Surplus Store Knives by Price", fontsize=24, x=0
fig.legend(loc=(.77, .80), fontsize='large')

plt.show();
```

In [ ]:

In [ ]:

In [ ]:

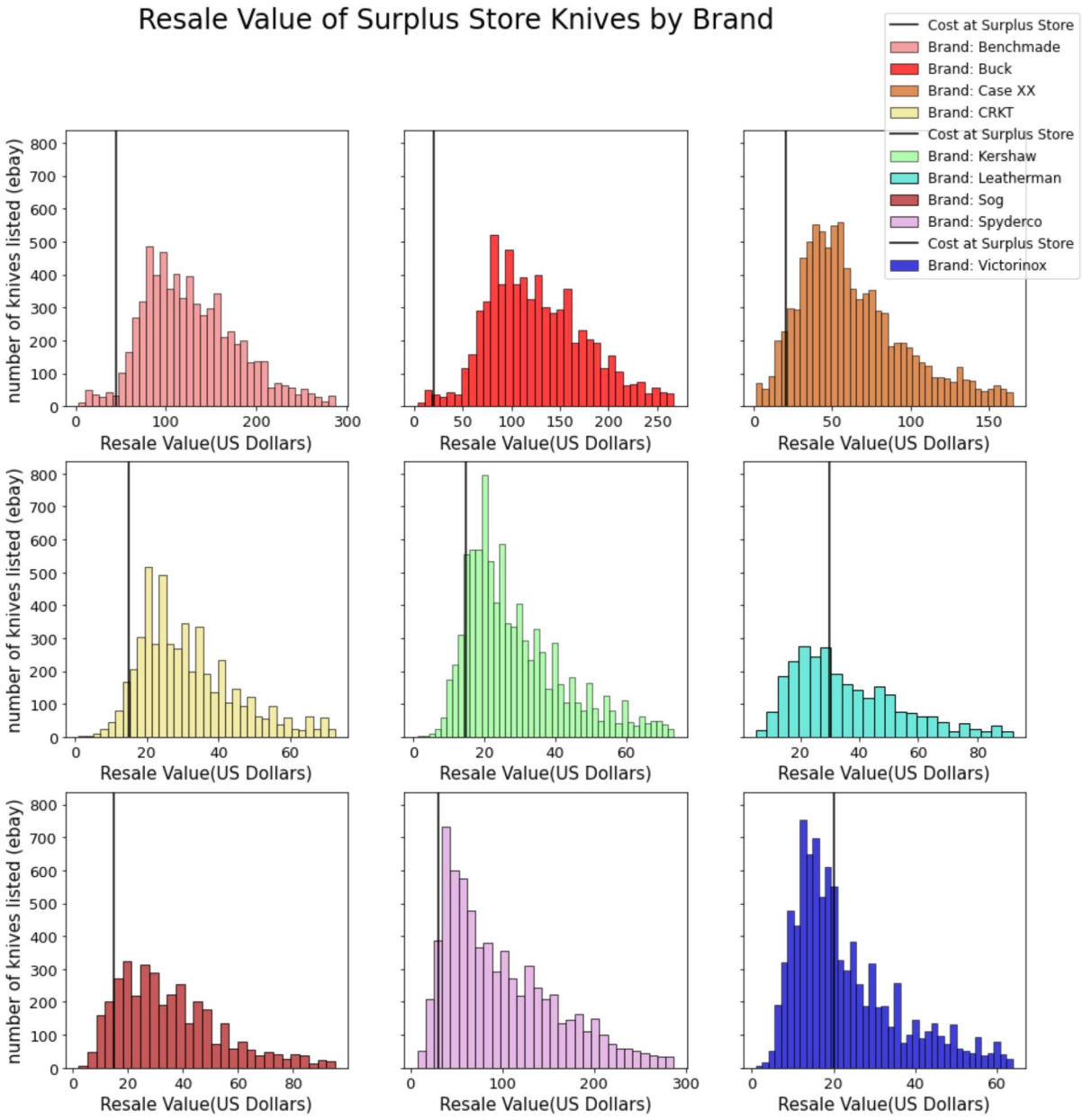
In [ ]:

In [ ]:
```
# tera_bench = df.loc[df['brand'] == 'benchmade']
# tera_buck = df.loc[df['brand'] == 'buck']
# tera_case = df.loc[df['brand'] == 'case']
# tera_crkt = df.loc[df['brand'] == 'crkt']
# tera_kershaw = df.loc[df['brand'] == 'kershaw']
# tera_leatherman = df.loc[df['brand'] == 'leatherman']
# tera_sog = df.loc[df['brand'] == 'sog']
# tera_spyderco = df.loc[df['brand'] == 'spyderco']
# tera_victorinox = df.loc[df['brand'] == 'victorinox']
```

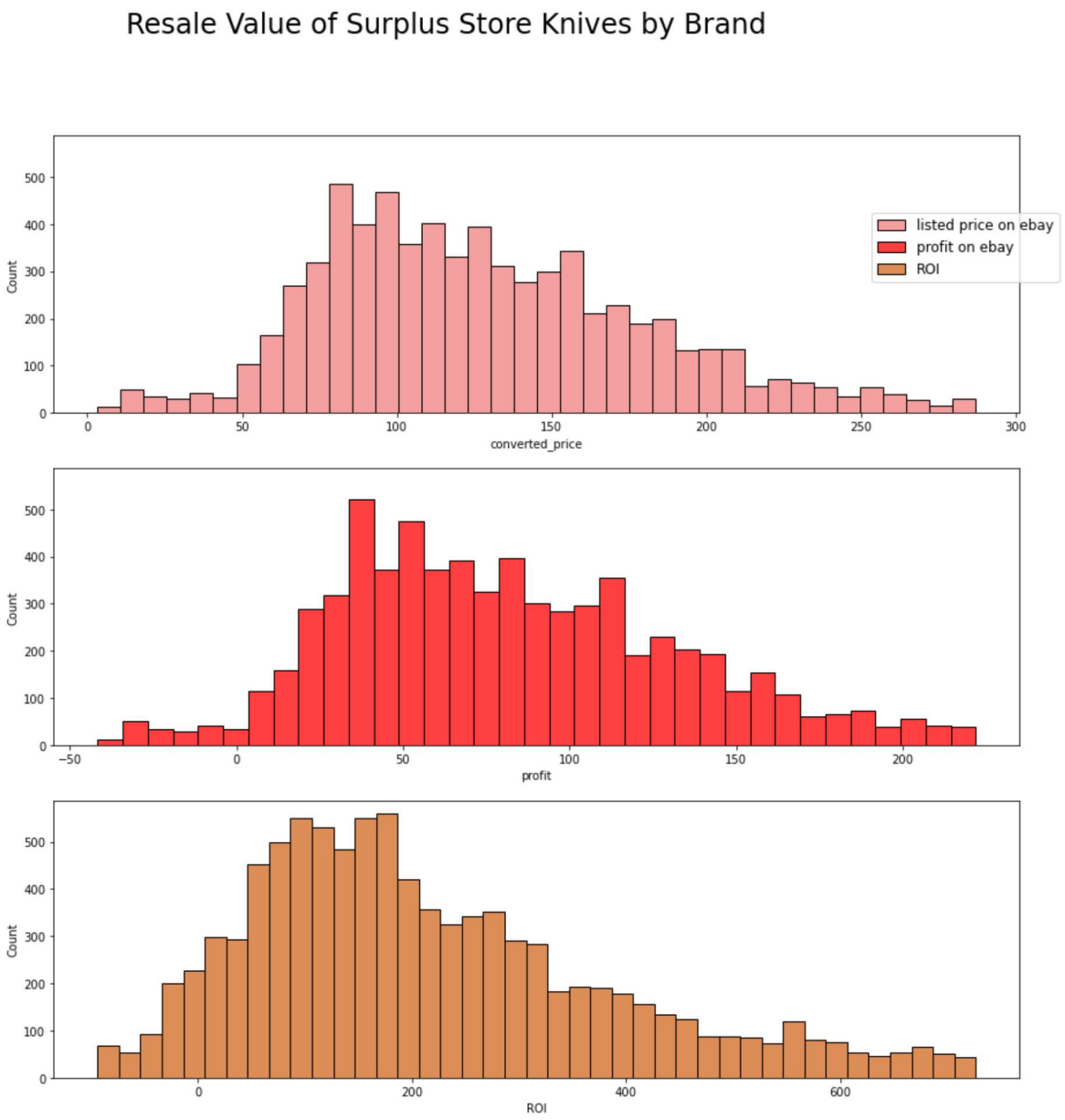
In [ ]:
```
# df_bench2 = df2.loc[df['benchmade'] != 0]
# df_buck2 = df2.loc[df['buck'] != 0]
# df_case2 = df2.loc[df['case'] != 0]
# df_crkt2 = df2.loc[df['crkt'] != 0]
# df_kershaw2 = df2.loc[df['kershaw'] != 0]
# df_spyderco2 = df2.loc[df['spyderco'] != 0]
# df_victorinox2 = df2.loc[df['victorinox'] != 0]
```

In [ ]:
```
bucket_dict
```

In [29]:
```
fig, axes  = plt.subplots(figsize=(15,15), ncols=3, nrows=3, sharey=True)
sns.histplot(tera_bench['converted_price'], ax=axes[0][0], color='lightcoral', la
sns.histplot(tera_buck['converted_price'], ax=axes[0][1], color='red', label='Bra
sns.histplot(tera_case['converted_price'], ax=axes[0][2], color='chocolate', labe
sns.histplot(tera_crkt['converted_price'], ax=axes[1][0],  color='khaki', label='
sns.histplot(tera_kershaw['converted_price'], ax=axes[1][1], color='palegreen', l
sns.histplot(tera_leatherman['converted_price'], ax=axes[1][2], color='turquoise'
sns.histplot(tera_sog['converted_price'], ax=axes[2][0], color='firebrick', label
sns.histplot(tera_spyderco['converted_price'], ax=axes[2][1],  color='plum', labe
sns.histplot(tera_victorinox['converted_price'], ax=axes[2][2], color='mediumblue


for n in range(9):
    row = n//3
    col = n%3
    axes[row][col].set_xlabel('Resale Value(US Dollars)', fontsize=15)
    axes[row][col].set_ylabel('number of knives listed (ebay)', 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[1][0].axvline(x = 15, color = 'black')
axes[1][1].axvline(x = 15, color = 'black', label= 'Cost at Surplus Store')
axes[1][2].axvline(x = 30, color = 'black')
axes[2][0].axvline(x = 15, color = 'black')
axes[2][1].axvline(x = 30, color = 'black')
axes[2][2].axvline(x = 20, color = 'black', label= 'Cost at Surplus Store')

fig.suptitle("Resale Value of Surplus Store Knives by Brand", fontsize=24, x=0.44
fig.legend(loc=(.8, .75), fontsize='large')
plt.show();
```


Resale Value of Surplus Store Knives by Brand
Cost at Surplus Store
Brand: Benchmade
Brand: Buck
Brand: Case XX
Brand: CRKT
Cost at Surplus Store
Brand: Kershaw
Brand: Leatherman
Brand: Sog
Brand: Spyderco
Cost at Surplus Store
Brand: Victorinox
number of knives listed (ebay)
Resale Value(US Dollars)

In [30]:
```
fig, axes  = plt.subplots(figsize=(15,15), nrows=3, sharey=True)
sns.histplot(tera_bench['converted_price'], ax=axes[0], color='lightcoral', label
sns.histplot(tera_buck['profit'], ax=axes[1], color='red', label='profit on ebay'
sns.histplot(tera_case['ROI'], ax=axes[2], color='chocolate', label='ROI')


# for n in range(9):
#     row = n//3
#     col = n%3
#     axes[row][col].set_xlabel('Resale Value(US Dollars)', fontsize=15)
#     axes[row][col].set_ylabel('number of knives listed (ebay)', 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[1][0].axvline(x = 15, color = 'black')
# axes[1][1].axvline(x = 15, color = 'black', label= 'Cost at Surplus Store')
# axes[1][2].axvline(x = 30, color = 'black')
# axes[2][0].axvline(x = 15, color = 'black')
# axes[2][1].axvline(x = 30, color = 'black')
# axes[2][2].axvline(x = 20, color = 'black', label= 'Cost at Surplus Store')

fig.suptitle("Resale Value of Surplus Store Knives by Brand", fontsize=24, x=0.44
fig.legend(loc=(.8, .75), fontsize='large')
plt.show();
```


Resale Value of Surplus Store Knives by Brand
listed price on ebay
profit on ebay
ROI
Count
converted_price
profit
ROI

In [39]:
```
df_filtered.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 60280 entries, 0 to 60279
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Image            60280 non-null  object 
 1   date_sold        47143 non-null  object 
 2   title            53303 non-null  object 
 3   avg_price        60280 non-null  float64
 4   avg_shipping     60280 non-null  float64
 5   converted_price  60280 non-null  float64
 6   profit           60280 non-null  float64
 7   ROI              60280 non-null  float64
 8   brand            60280 non-null  object 
 9   cost             60280 non-null  float64
 10  url              2320 non-null   object 
dtypes: float64(6), object(5)
memory usage: 5.1+ MB
```

In [40]:
```
df_filtered[['converted_price','profit','ROI']].describe()
```

Out[40]:

	converted_price	profit	ROI
count	60280.000000	60280.000000	60280.000000
mean	46.756484	24.658857	109.075325
std	34.203054	29.776469	123.672788
min	0.990000	-41.690000	-95.050000
25%	20.750000	2.950000	14.950000
50%	35.500000	16.000000	80.833333
75%	62.000000	39.000000	175.270833
max	166.500000	126.380000	535.333333

In [41]:
```
len(df_filtered.loc[df_filtered['ROI'] >= 80])/len(df_filtered)
```

Out[41]:
```
0.5051426675514267
```

In [42]:
```
df_filtered['binary_target'] = (df_filtered['ROI'] >= 80)
```

In [45]:
```
df_filtered['binary_target'].replace({True:1,False:0}, inplace=True)
```

In [ ]:
```
# fig, axes  = plt.subplots(figsize=(15,15), ncols=4, nrows=2, sharey=True)
# sns.histplot(df_bench['converted_price'], ax=axes[0][0], color='lightcoral', la
# sns.histplot(df_buck['converted_price'], ax=axes[0][1], color='chocolate', labe
# sns.histplot(df_case['converted_price'], ax=axes[0][2],  color='khaki', label=
# sns.histplot(df_crkt['converted_price'], ax=axes[0][3], color='palegreen', labe
# sns.histplot(df_kershaw['converted_price'], ax=axes[1][0], color='turquoise', l
# sns.histplot(df_leatherman['converted_price'], ax=axes[1][1], color='firebrick
# sns.histplot(df_spyderco['converted_price'], ax=axes[1][2],  color='plum', labe
# sns.histplot(df_victorinox['converted_price'], ax=axes[1][3], color='mediumblue
# sns.

# for n in range(8):
#     row = n//4
#     col = n%4
#     axes[row][col].set_xlabel('Resale Value(US Dollars)', fontsize=15)
#     axes[row][col].set_ylabel('number of knives listed (ebay)', 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')
# fig.suptitle("Resale Value of Surplus Store Knives by Brand", fontsize=24, x=0.
# fig.legend(loc=(.8, .80), fontsize='large')
# plt.show();
```

In [ ]:
```
# fig, axes  = plt.subplots(figsize=(15,15), ncols=4, nrows=2, sharey=True)
# sns.histplot(df_bench2['price'], ax=axes[0][0], color='lightcoral', label='Brar
# sns.histplot(df_buck2['price'], ax=axes[0][1], color='chocolate', label='Brand:
# sns.histplot(df_case2['price'], ax=axes[0][2],  color='khaki', label='Brand: Ca
# sns.histplot(df_crkt2['price'], ax=axes[0][3], color='palegreen', label='Brand:
# sns.histplot(df_kershaw2['price'], ax=axes[1][0], color='turquoise', label='Bra
# sns.histplot(df_spyderco2['price'], ax=axes[1][2],  color='plum', label='Brand:
# sns.histplot(df_victorinox2['price'], ax=axes[1][3], color='mediumblue', label=


# for n in range(8):
#     row = n//4
#     col = n%4
#     axes[row][col].set_xlabel('Resale Value(US Dollars)', fontsize=15)
#     axes[row][col].set_ylabel('number of knives listed (ebay)', 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')
# fig.suptitle("Resale Value of Surplus Store Knives by Brand", fontsize=24, x=0.
# fig.legend(loc=(.8, .80), fontsize='large')
# plt.show();
```

Best distributions look like Case XX and Spyderco. Should look more closely. The higher counts will make the listed prices closer to true value and will also help with

training the model. A larger sample size also seems to help normalize the price data.

In [ ]:
```
fig, axes  = plt.subplots(figsize=(15,15), ncols=3, nrows=3, sharey=True)
sns.histplot(tera_bench['profit'], ax=axes[0][0], color='lightcoral', label='Bran
sns.histplot(tera_buck['profit'], ax=axes[0][1], color='blue', label='Brand: Buck
sns.histplot(tera_case['profit'], ax=axes[0][2], color='chocolate', label='Brand
sns.histplot(tera_crkt['profit'], ax=axes[1][0],  color='khaki', label='Brand: CR
sns.histplot(tera_kershaw['profit'], ax=axes[1][1], color='palegreen', label='Bra
sns.histplot(tera_leatherman['profit'], ax=axes[1][2], color='turquoise', label='
sns.histplot(tera_sog['profit'], ax=axes[2][0], color='firebrick', label='Brand:
sns.histplot(tera_spyderco['profit'], ax=axes[2][1],  color='plum', label='Brand
sns.histplot(tera_victorinox['profit'], ax=axes[2][2], color='mediumblue', label=


for n in range(9):
    row = n//3
    col = n%3
    axes[row][col].set_xlabel('Expected Profit (US Dollars)', fontsize=15)
    axes[row][col].set_ylabel('number of knives listed (ebay)', fontsize=15)
    axes[row][col].tick_params(axis='both', labelsize=13)
    axes[row][col].axvline(x = 0, color = 'black')

axes[0][0].axvline(x = 0, color = 'black', label= 'Break Even Point')
fig.suptitle("Expected Profit of Surplus Store Knives by Brand", fontsize=24, x=0
fig.legend(loc=(.82, .80), fontsize='large')
plt.show();
```

In [ ]:
```
fig, axes  = plt.subplots(figsize=(15,15), ncols=3, nrows=3, sharey=True, sharex=
sns.histplot(tera_bench['profit'], ax=axes[0][0], color='lightcoral', label='Bran
sns.histplot(tera_buck['profit'], ax=axes[0][1], color='blue', label='Brand: Buck
sns.histplot(tera_case['profit'], ax=axes[0][2], color='chocolate', label='Brand
sns.histplot(tera_crkt['profit'], ax=axes[1][0],  color='khaki', label='Brand: CR
sns.histplot(tera_kershaw['profit'], ax=axes[1][1], color='palegreen', label='Bra
sns.histplot(tera_leatherman['profit'], ax=axes[1][2], color='turquoise', label='
sns.histplot(tera_sog['profit'], ax=axes[2][0], color='firebrick', label='Brand:
sns.histplot(tera_spyderco['profit'], ax=axes[2][1],  color='plum', label='Brand
sns.histplot(tera_victorinox['profit'], ax=axes[2][2], color='mediumblue', label=


for n in range(9):
    row = n//3
    col = n%3
    axes[row][col].set_xlabel('Expected Profit (US Dollars)', fontsize=15)
    axes[row][col].set_ylabel('number of knives listed (ebay)', fontsize=15)
    axes[row][col].tick_params(axis='both', labelsize=13)
    axes[row][col].axvline(x = 0, color = 'black')

axes[0][0].axvline(x = 0, color = 'black', label= 'Break Even Point')
fig.suptitle("Expected Profit of Surplus Store Knives by Brand", fontsize=24, x=0
fig.legend(loc=(.82, .80), fontsize='large')
plt.show();
```

In [ ]:
```
df_new = pd.concat([tera_bench, tera_buck, tera_case, tera_sog, tera_spyderco])
```

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

In [ ]:
```
df_new[['converted_price', 'profit', 'ROI']].describe()
```

In [ ]:
```
fig, axes  = plt.subplots(figsize=(15,15), nrows=3, sharey=True)
sns.histplot(df_new['converted_price'], ax=axes[0], color='lightcoral', label='Br
sns.histplot(df_new['profit'], ax=axes[1], color='blue', label='Brand: Buck')
sns.histplot(df_new['ROI'], ax=axes[2], color='chocolate', label='Brand: Case XX'
```

In [ ]:
```
new_cutoff = df_new['converted_price'].mean() + (2*df_new['converted_price'].std(
```

In [ ]:
```
df_new2 = df_new.loc[df_new['converted_price'] < new_cutoff].copy()
```

In [ ]:
```
fig, axes  = plt.subplots(figsize=(15,15), nrows=3, sharey=True)
sns.histplot(df_new2['converted_price'], ax=axes[0], color='lightcoral', label='E
sns.histplot(df_new2['profit'], ax=axes[1], color='blue', label='Brand: Buck')
sns.histplot(df_new2['ROI'], ax=axes[2], color='chocolate', label='Brand: Case XX
```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```
# fig, axes  = plt.subplots(figsize=(15,15), ncols=4, nrows=2, sharey=True)
# sns.histplot(tera_bench2['converted_price'], ax=axes[0][0], color='lightcoral',
# sns.histplot(tera_case2['converted_price'], ax=axes[0][1], color='chocolate', l
# sns.histplot(tera_crkt2['converted_price'], ax=axes[0][2],  color='khaki', labe
# sns.histplot(tera_kershaw2['converted_price'], ax=axes[0][3], color='palegreen
# sns.histplot(tera_leatherman2['converted_price'], ax=axes[1][0], color='turquoi
# sns.histplot(tera_sog2['converted_price'], ax=axes[1][1], color='firebrick', la
# sns.histplot(tera_spyderco2['converted_price'], ax=axes[1][2],  color='plum', l
# sns.histplot(tera_victorinox2['converted_price'], ax=axes[1][3], color='mediumb
```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```
# fig, axes  = plt.subplots(figsize=(15,15), ncols=4, nrows=2, sharey=True)
# sns.histplot(df_bench['profit'], ax=axes[0][0], color='lightcoral', label='Brar
# sns.histplot(df_buck['profit'], ax=axes[0][1], color='chocolate', label='Brand:
# sns.histplot(df_case['profit'], ax=axes[0][2],  color='khaki', label='Brand: Cc
# sns.histplot(df_crkt['profit'], ax=axes[0][3], color='palegreen', label='Brand:
# sns.histplot(df_kershaw['profit'], ax=axes[1][0], color='turquoise', label='Brc
# sns.histplot(df_leatherman['profit'], ax=axes[1][1], color='firebrick', label=
# sns.histplot(df_spyderco['profit'], ax=axes[1][2],  color='plum', label='Brand:
# sns.histplot(df_victorinox['profit'], ax=axes[1][3], color='mediumblue', label=


# for n in range(8):
#     row = n//4
#     col = n%4
#     axes[row][col].set_xlabel('Expected Profit (US Dollars)', fontsize=15)
#     axes[row][col].set_ylabel('number of knives listed (ebay)', fontsize=15)
#     axes[row][col].tick_params(axis='both', labelsize=13)
#     axes[row][col].axvline(x = 0, color = 'black')

# axes[0][0].axvline(x = 0, color = 'black', label= 'Break Even Point')
# fig.suptitle("Expected Profit of Surplus Store Knives by Brand", fontsize=24, x
# fig.legend(loc=(.82, .80), fontsize='large')
# plt.show();
```

In [ ]:
```
# fig, axes  = plt.subplots(figsize=(15,15), ncols=4, nrows=2, sharey=True)
# sns.histplot(df_bench2['profit'], ax=axes[0][0], color='lightcoral', label='Brc
# sns.histplot(df_buck2['profit'], ax=axes[0][1], color='chocolate', label='Brand
# sns.histplot(df_case2['profit'], ax=axes[0][2],  color='khaki', label='Brand: C
# sns.histplot(df_crkt2['profit'], ax=axes[0][3], color='palegreen', label='Brand
# sns.histplot(df_kershaw2['profit'], ax=axes[1][0], color='turquoise', label='Br

# sns.histplot(df_spyderco2['profit'], ax=axes[1][2],  color='plum', label='Brand
# sns.histplot(df_victorinox2['profit'], ax=axes[1][3], color='mediumblue', label


# for n in range(8):
#     row = n//4
#     col = n%4
#     axes[row][col].set_xlabel('Expected Profit (US Dollars)', fontsize=15)
#     axes[row][col].set_ylabel('number of knives listed (ebay)', fontsize=15)
#     axes[row][col].tick_params(axis='both', labelsize=13)
#     axes[row][col].axvline(x = 0, color = 'black')

# axes[0][0].axvline(x = 0, color = 'black', label= 'Break Even Point')
# fig.suptitle("Expected Profit of Surplus Store Knives by Brand", fontsize=24, x
# fig.legend(loc=(.82, .80), fontsize='large')
# plt.show();
```

In [ ]:
```
# fig, axes  = plt.subplots(figsize=(15,15), ncols=4, nrows=2, sharey=True)
# sns.histplot(df_bench['ROI'], ax=axes[0][0], color='lightcoral', label='Brand:
# sns.histplot(df_buck['ROI'], ax=axes[0][1], color='chocolate', label='Brand: Bu
# sns.histplot(df_case['ROI'], ax=axes[0][2],  color='khaki', label='Brand: Case
# sns.histplot(df_crkt['ROI'], ax=axes[0][3], color='palegreen', label='Brand: CR
# sns.histplot(df_kershaw['ROI'], ax=axes[1][0], color='turquoise', label='Brand:
# sns.histplot(df_leatherman['ROI'], ax=axes[1][1], color='firebrick', label='Bra
# sns.histplot(df_spyderco['ROI'], ax=axes[1][2],  color='plum', label='Brand: Sp
# sns.histplot(df_victorinox['ROI'], ax=axes[1][3], color='mediumblue', label='Br


# for n in range(8):
#     row = n//4
#     col = n%4
#     axes[row][col].set_xlabel('Expected ROI(Percent US Dollars)', fontsize=12)
#     axes[row][col].set_ylabel('number of knives listed (ebay)', fontsize=15)
#     axes[row][col].tick_params(axis='both', labelsize=13)
#     axes[row][col].axvline(x = 0, color = 'black')

# axes[0][0].axvline(x = 0, color = 'black', label= 'Break Even Point')
# fig.suptitle("Expected Return On Investment of Surplus Store Knives by Brand",
# fig.legend(loc=(.82, .80), fontsize='large')
# plt.show();
```

In [ ]:
```

```

In [ ]:
```
fig, axes  = plt.subplots(figsize=(15,15), ncols=3, nrows=3, sharey=True)
sns.histplot(tera_bench['ROI'], ax=axes[0][0], color='lightcoral', label='Brand:
sns.histplot(tera_buck['ROI'], ax=axes[0][1], color='blue', label='Brand: Buck')
sns.histplot(tera_case['ROI'], ax=axes[0][2], color='chocolate', label='Brand: Ca
sns.histplot(tera_crkt['ROI'], ax=axes[1][0],  color='khaki', label='Brand: CRKT'
sns.histplot(tera_kershaw['ROI'], ax=axes[1][1], color='palegreen', label='Brand:
sns.histplot(tera_leatherman['ROI'], ax=axes[1][2], color='turquoise', label='Bra
sns.histplot(tera_sog['ROI'], ax=axes[2][0], color='firebrick', label='Brand: Sog
sns.histplot(tera_spyderco['ROI'], ax=axes[2][1],  color='plum', label='Brand: Sp
sns.histplot(tera_victorinox['ROI'], ax=axes[2][2], color='mediumblue', label='Br



for n in range(9):
    row = n//3
    col = n%3
    axes[row][col].set_xlabel('Expected ROI(Percent US Dollars)', fontsize=12)
    axes[row][col].set_ylabel('number of knives listed (ebay)', fontsize=15)
    axes[row][col].tick_params(axis='both', labelsize=13)
    axes[row][col].axvline(x = 0, color = 'black')

axes[0][0].axvline(x = 0, color = 'black', label= 'Break Even Point')
fig.suptitle("Expected Return On Investment of Surplus Store Knives by Brand", fo
fig.legend(loc=(.82, .80), fontsize='large')
plt.show();
```

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

In [ ]:
```
df.info()
```

In [ ]:
```
df[['converted_price', 'profit', 'ROI']].describe()
```

In [ ]:
```
tera_spyderco[['converted_price', 'profit', 'ROI']].describe()
```

In [ ]:
```
df_spyder2[['converted_price','profit', 'ROI']].describe()
```

In [ ]:
```
df_spyder[['converted_price','profit', 'ROI']].describe()
```

In [ ]:
```
df_spyder2.loc[df_spyder2['condition'] != 1000.0][['converted_price','profit', 'F
```

In [ ]:
```
sns.histplot(df_spyder['converted_price'])
```

In [ ]:
```
df_spyder.info()
```

In [ ]:
```
tera_spyderco.info()
```

In [ ]:
```
concat_df = pd.concat([tera_spyderco, df_spyder])
```

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

In [ ]:
```
concat_df.to_csv('data/concat_df.csv', index=False)
```

In [ ]:
```
concat_df['profit'].describe()
```

In [ ]:
```
len(concat_df.loc[concat_df['profit'] > 70])/len(concat_df)
```

In [ ]:
```
concat_df['target'] = (concat_df['profit'] > 70)
```

In [ ]:
```
concat_df['target'].replace({True: 1, False:0}, inplace=True)
```

In [ ]:
```
concat_df['target'].value_counts()
```

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

In [ ]:
```
concat_df.to_csv('data/concat_df_extra.csv', index=False)
```

In [ ]:
```
concat_df = pd.concat([tera_bench, df_bench])
concat_df.reset_index(drop=True, inplace=True)
```

In [ ]:
```
concat_df['PictureURL'].fillna(concat_df['Image'], inplace=True)
```

In [ ]:
```
concat_df.to_csv('data/concat_df_bench.csv', index=False)
```

In [ ]:
```
concat_df.info()
```

In [ ]:
```

```

In [ ]:
```
# tera_spyderco['target'] = (tera_spyderco['profit'] > 52)
# tera_spyderco['target'].replace({True: 1, False:0}, inplace=True)
```

In [ ]:
```
# tera_spyderco.reset_index(drop=True, inplace=True)
```

In [ ]:
```
# tera_spyderco.to_csv('data_spyder_ready.csv', index=False)
```

In [ ]:
```
fig, axes  = plt.subplots(figsize=(15,15), ncols=3, nrows=3, sharey=True, sharex=
sns.boxplot(x=tera_bench['ROI'], ax=axes[0][0], color='lightcoral')
sns.boxplot(x=tera_buck['ROI'], ax=axes[0][1], color='blue')
sns.boxplot(x=tera_case['ROI'], ax=axes[0][2], color='chocolate')
sns.boxplot(x=tera_crkt['ROI'], ax=axes[1][0],  color='khaki')
sns.boxplot(x=tera_kershaw['ROI'], ax=axes[1][1], color='palegreen')
sns.boxplot(x=tera_leatherman['ROI'], ax=axes[1][2], color='turquoise')
sns.boxplot(x=tera_sog['ROI'], ax=axes[2][0], color='firebrick')
sns.boxplot(x=tera_spyderco['ROI'], ax=axes[2][1],  color='plum')
sns.boxplot(x=tera_victorinox['ROI'], ax=axes[2][2], color='mediumblue')


for n in range(9):
    row = n//3
    col = n%3
    axes[row][col].set_xlabel('ROI(percent US Dollars)', fontsize=15)
    axes[row][col].tick_params(axis='both', labelsize=13)
    axes[row][col].axvline(x = 0, color = 'black')

axes[0][0].axvline(x = 0, color = 'black', label= 'Break Even Point')
fig.suptitle("Expected Return On Investment of Surplus Store Knives by Brand", fo
fig.legend(loc=(.82, .80), fontsize='large')
plt.show();
```

In [ ]:
```
tera_bench[['converted_price', 'profit', 'ROI']].describe()
```

In [ ]:
```
tera_buck[['converted_price', 'profit', 'ROI']].describe()
```

In [ ]:
```
tera_case[['converted_price', 'profit', 'ROI']].describe()
```

In [ ]:
```
tera_spyderco[['converted_price', 'profit', 'ROI']].describe()
```

In [ ]:
```
tera_crkt[['converted_price', 'profit', 'ROI']].describe()
```

In [ ]:
```
tera_sog[['converted_price', 'profit', 'ROI']].describe()
```

In [ ]:
```
tera_kershaw[['converted_price', 'profit', 'ROI']].describe()
```

In [ ]:
```
tera_leatherman[['converted_price', 'profit', 'ROI']].describe()
```

In [ ]:
```
tera_victorinox[['converted_price', 'profit', 'ROI']].describe()
```

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

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```
df_spyderco['profit'].min() - df_case['profit'].min()
```

In [ ]:
```
df_spyderco['profit'].max() - df_case['profit'].max()
```

In [ ]:
```
df_spyderco['converted_price'].max() - df_case['converted_price'].max()
```

In [ ]:
```
df_case['profit'].mean() + 2 *df_case['profit'].std()
```

In [ ]:
```
df_spyderco['profit'].mean() + 2 *df_spyderco['profit'].std()
```

In [ ]:
```
sns.boxplot(x=df_case['profit'])
```

In [ ]:
```
sns.boxplot(x=df_spyderco['profit'])
```

In [ ]:
```
sns.boxplot(x=df_case['ROI'])
```

In [ ]:
```
sns.boxplot(x=df_spyderco['ROI'])
```

In [ ]:
```
df_case['viewItemURL'].sample(10).apply(print)
```

Smaller standard deviation for case and boxplot and distrubtion plot show less variance in mean profit and ROI returns. Better pick for better returns. However, must see which knife has better patterns to capture.

I was able to get data for nearly 6000 spyderco knives for a fixed price listing on ebay. I was able to get 7000 Case XX brand knives for a fixed price listing on ebay. At the surplus store, Spyderco knives cost 30 dollars, while Case XX knives cost 20 dollars. That is only a 10 dollar difference. However, the mean price of the listed spydercos is about 25 dollars higher. That means even though I would have to pay 10 dollars more to get a

spyderco at the store, they have a higher probability of generating higher profit margins and higher returns. As you can see from the charts above, just risking ten dollars more leads to a higher profit ceiling of 89 dollars more. The mean return on investment values are fairly close, at 218.76 for case knives and about 200 mean ROI for spyderco. I feel that both knives would really do well from a business standpoint. However, Spyderco knives also have much more distinctive features and branding patterns for my model to catch when building my CNN. Therefore, for the first iteration of my project I will make a binary classification for spyderco knives, and expand first to Case XX knives and then other knives when I can gather additional data.

After reviewing the distrubtions above, it seems that benchmades and case knives might be the most apporpiate targets. With a new determined target of ROI to account for buying a large number of cheap knives or a small number of expensive knives for max profit, I am going to filter the data again to remove anything outside of the range -1.96 and +1.96 z-score for ROI (this would mean there would be less than a 5% chance of ever seeing those knives listed on ebay... and probably a much lower probability they would ever end up at the surplus store. Therefore, the data is irrelevant and can be removed.

JK WENT WITH IQR RANGES FOR PRICE PROFIT AND ROI FROM RAW DF. THIS REMOVED 10.95% of outlier data.

In [ ]:
```
# concat_df['PictureURL'].fillna(concat_df['Image'], inplace=True)
```

In [ ]:
```
import os
import requests
```

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

In [ ]:
```
df.info()
```

In [ ]:
```
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')

root_folder = 'C:/Users/12108/Documents/GitHub/Neural_Network_Predicting_Reseller
im_extension = '.jpg'  # or whatever type of images you are downloading

df_filtered.apply(download, axis=1)
```

In [ ]:
```
# import os
# import requests


# 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.PictureURL
#     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')

# root_folder = 'C:/Users/12108/Documents/GitHub/Neural_Network_Predicting_Resell
# im_extension = '.jpg'  # or whatever type of images you are downloading

# concat_df.apply(download, axis=1)
```

In [ ]:
```
# import os
# import requests


# 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')

# root_folder = 'C:/Users/12108/Documents/GitHub/Neural_Network_Predicting_Resell
# im_extension = '.jpg'  # or whatever type of images you are downloading

# # df.apply(download, axis=1)
```

In [ ]:
```
# df2.apply(download, axis=1)
```

In [ ]:
```
# import os
# import requests


# 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')

# root_folder = 'C:/Users/12108/Documents/GitHub/Neural_Network_Predicting_Resell
# im_extension = '.jpg'  # or whatever type of images you are downloading

# tera_spyderco.apply(download, axis=1)
```

In [ ]:
```

```

In [ ]:
```
# import os
# import requests


# 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}")
#     r = requests.get(url, allow_redirects=True)
#     with open(filename, 'wb') as f:
#         f.write(r.content)

# root_folder = 'C:/Users/12108/Documents/GitHub/Neural_Network_Predicting_Resell
# im_extension = '.jpg'  # or whatever type of images you are downloading

# df.apply(download, axis=1)
```

In [ ]:
```
import os, shutil
import tensorflow as tf
from tensorflow import keras
from keras.models import load_model
from keras.preprocessing import image
import numpy as np
import cv2
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras.layers import Input, Dropout, Conv2D, Dense, Flatten, Globa

img_array = cv2.imread('nn_images/105.jpg')  # convert to array

img_rgb = cv2.resize(img_array,(256,256),3)
plt.imshow(img_rgb)  # graph it
plt.show();


def image_checker(index):
    img_array = cv2.imread('knife_images/'+str(index)+'.jpg')
    img_rgb = cv2.resize(img_array,(256,256),3)
    plt.imshow(img_rgb)  # graph it
    plt.show();
```

In [ ]:
```
sns.histplot(df['profit'])
```

In [ ]:
```
sns.histplot(df['ROI'])
```

In [ ]:
```
sns.histplot(df['converted_price'])
```

In [ ]:
```
range(1, len(df))
```

In [ ]:
```
#final processing steps for images

image_list = []
for x in range(len(df)):
    img_array = cv2.imread('nn_images/'+str(x)+'.jpg')  # convert to array
    try:
        img_rgb = cv2.resize(img_array,(256,256),3)  # resize
        img_rgb = np.array(img_rgb).astype(np.float64)/255.0  # scaling
        image_list.append(img_rgb)
    except:
        print('exception error has occured at ' + 'nn_images/'+str(x)+'.jpg')

    # img_rgb = np.expand_dims(img_rgb, axis=0)  # expand dimension
```

In [ ]:
```
to_drop = [1864,1887,6660,10837,17409,18351,19350,21949,21953,21965,23616,25521,2
```

In [ ]:
```
X = np.array(image_list)
```

In [ ]:
```

```

In [ ]:
```
y = df['profit']
```

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

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, test_si

print("Xtrain:", X_train.shape)
print("y_train:", y_train.shape)
print("X_test:", X_test.shape)
print("y_test:", y_test.shape)

X_test, X_val, y_test, y_val = train_test_split(X_test, y_test, test_size=0.5, ra

print("Xtrain:", X_train.shape)
print("y_train:", y_train.shape)
print("X_test:", X_test.shape)
print("y_test:", y_test.shape)
print("X_val:", X_test.shape)
print("y_val:", y_test.shape)
```

In [ ]:
```
from keras import models
from keras import layers
```

In [ ]:
```
#small batch

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.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.Conv2D(256, (3, 3), activation='relu', padding='same'))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(256, (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(Dense(128, activation='relu'))
model.add(Dense(1, activation='linear'))

model.compile(loss='mean_squared_error',
              optimizer='Adam',
               metrics=['mse'])

history = model.fit(X_train,
                    y_train,
                    epochs=30,
                    batch_size=32,
                    validation_data=(X_val, y_val))
```

In [ ]:
```
# 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)
```

In [ ]:
```
# 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('-----------------------------------')
```

In [ ]:
```
# new_df = df[(df['converted_price'] < price_upper_limit) &
#              (df['profit'] <profit_upper_limit) &
#              (df['ROI'] < ROI_upper_limit) &
#              (df['profit'] < profit_upper_limit) &
#              (df['ROI'] > ROI_lower_limit)]

# new_df.shape
```

In [ ]:
```
# (len(new_df) - len(df))/len(df)
```

In [ ]:
```
# new_df.reset_index(drop=True, inplace=True)
```

In [ ]:
```
# df_benchmade2 = new_df.loc[new_df['benchmade'] != 0]
# df_buck2 = new_df.loc[new_df['buck'] != 0]
# df_case2 = new_df.loc[new_df['case'] != 0]
# df_crkt2 = new_df.loc[new_df['crkt'] != 0]
# df_kershaw2 = new_df.loc[new_df['kershaw'] != 0]
# df_leatherman2 = new_df.loc[new_df['leatherman'] != 0]
# df_spyderco2 = new_df.loc[new_df['spyderco'] != 0]
# df_victorinox2 = new_df.loc[new_df['victorinox'] != 0]
```

In [ ]:
```
# fig, ax = plt.subplots(figsize=(12,8))
# sns.histplot(df_benchmade2['ROI'], ax=ax, color='lime')


# fig.suptitle("Expected ROI(as percent dollars invested) By Brand for $45 Knives
# ax.set_xlabel('Return On Investment (% US Dollars)', fontsize=20)
# ax.set_ylabel('number of knives listed (ebay)', fontsize=15)



# # Add a legend to the figure
# # (in general, these are quite nitpicky to style and position)
# fig.legend(labels=["benchmade"], loc=(.68, .78), fontsize='x-large')
# plt.show();
```

In [ ]:
```
# df_benchmade = ROI_df.loc[ROI_df['benchmade'] != 0]
# df_buck = ROI_df.loc[ROI_df['buck'] != 0]
# df_case = ROI_df.loc[ROI_df['case'] != 0]
# df_crkt = ROI_df.loc[ROI_df['crkt'] != 0]
# df_kershaw = ROI_df.loc[ROI_df['kershaw'] != 0]
# df_leatherman = ROI_df.loc[ROI_df['leatherman'] != 0]
# df_spyderco = ROI_df.loc[ROI_df['spyderco'] != 0]
# df_victorinox = ROI_df.loc[ROI_df['victorinox'] != 0]
```

In [ ]:
```
# fig, (ax1, ax2) = plt.subplots(figsize=(10,12), nrows=2)
# sns.histplot(df_crkt['ROI'], ax=ax1, color='lime')
# sns.histplot(df_kershaw['ROI'], ax=ax2, color='aqua')
# fig.suptitle("Expected ROI(as percent dollars invested) By Brand for $15 Knives
# ax1.set_xlabel('Return on Investment(% US Dollars)', fontsize=18)
# ax1.set_ylabel('number of knives for sale', fontsize=18)
# ax2.set_xlabel('Return on Investment(% US Dollars)', fontsize=18)
# ax2.set_ylabel('number of knives for sale', fontsize=18)

# # Add a legend to the figure
# # (in general, these are quite nitpicky to style and position)
# fig.legend(labels=["crkt", "kershaw"], loc=(.68, .78), fontsize='x-large')
# plt.show();
```

In [ ]:
```
# fig, (ax1, ax2) = plt.subplots(figsize=(10,12), nrows=2)
# sns.boxplot(x=df_crkt['ROI'], ax=ax1, color='lime')
# sns.boxplot(x=df_kershaw['ROI'], ax=ax2, color='aqua')


# fig.suptitle("Expected ROI(as percent dollars invested) By Brand for $15 Knives
# ax1.set_xlabel('Return on Investment(% US Dollars)', fontsize=18)
# ax1.set_ylabel('number of knives for sale', fontsize=18)
# ax2.set_xlabel('Return on Investment(% US Dollars)', fontsize=18)
# ax2.set_ylabel('number of knives for sale', fontsize=18)

# # Add a legend to the figure
# # (in general, these are quite nitpicky to style and position)
# fig.legend(labels=["crkt", "kershaw"], loc=(.68, .78), fontsize='x-large')
# plt.show();
```

In [ ]:
```
# fig, ax = plt.subplots(figsize=(12,8))
# sns.histplot(df_benchmade['ROI'], ax=ax, color='lime')


# fig.suptitle("Expected ROI(as percent dollars invested) By Brand for $45 Knives
# ax1.set_xlabel('Return On Investment (% US Dollars)', fontsize=20)
# ax1.set_ylabel('number of knives listed (ebay)', fontsize=15)
# ax2.set_xlabel('Return On Investment (% US Dollars)', fontsize=20)
# ax2.set_ylabel('number of knives listed (ebay)', fontsize=15)


# # Add a legend to the figure
# # (in general, these are quite nitpicky to style and position)
# fig.legend(labels=["benchmade"], loc=(.68, .78), fontsize='x-large')
# plt.show();
```

In [ ]:
```
# fig, ax = plt.subplots(figsize=(12,8))
# sns.boxplot(df_benchmade['ROI'], ax=ax, color='lime')


# fig.suptitle("Expected ROI(as percent dollars invested) By Brand for $45 Knives
# ax1.set_xlabel('Return On Investment (% US Dollars)', fontsize=20)
# ax1.set_ylabel('number of knives listed (ebay)', fontsize=15)
# ax2.set_xlabel('Return On Investment (% US Dollars)', fontsize=20)
# ax2.set_ylabel('number of knives listed (ebay)', fontsize=15)


# # Add a legend to the figure
# # (in general, these are quite nitpicky to style and position)
# fig.legend(labels=["benchmade"], loc=(.68, .78), fontsize='x-large')
# plt.show();
```

In [ ]:
```
# #final processing steps for images

# image_list = []
# for x in range(len(df_CNN_regression)):

#     img_array = cv2.imread('knife_images/'+str(x)+'.jpg')  # convert to array
#     img_rgb = cv2.resize(img_array,(256,256),3)  # resize
#     img_rgb = np.array(img_rgb).astype(np.float64)/255.0  # scaling
#     image_list.append(img_rgb)

#     # img_rgb = np.expand_dims(img_rgb, axis=0)  # expand dimension
```

In [ ]:
```
len(df_case.loc[df_case['ROI'] >= df_case['ROI'].mean()])/len(df_case)
```

In [ ]:
```
len(df_case.loc[df_case['ROI'] >= 188])/len(df_case)
```

In [ ]:
```

```

In [ ]:
```
df_case['ROI'].mean()
```

In [ ]:
```
len(df_spyderco.loc[df_spyderco['ROI'] >= df_spyderco['ROI'].mean()])/len(df_spyc
```

In [ ]:
```
len(df_spyderco.loc[df_spyderco['ROI'] >= 168])/len(df_spyderco)
```

In [ ]:
```
df_case.loc[df_case['ROI'] >= 188, 'profit'].describe()
```

In [ ]:
```
df_spyderco.loc[df_spyderco['ROI'] >= 168, "profit"].describe()
```

In [ ]:
```
df_case.loc[df_case['ROI'] >= 188, 'ROI'].describe()
```

In [ ]:
```
(df_spyderco['ROI'] >= 168).value_counts()
```

In [ ]:
```
df_spyderco['target'].replace(True,1.0,inplace=True)
```

In [ ]:
```
df_spyderco['target'].replace(False,0.0,inplace=True)
```

In [ ]:
```
df_spyderco['target'].value_counts()
```

In [ ]:
```
buy_df = df_spyderco.loc[df_spyderco['target'] == 1.0]
dont_buy_df = df_spyderco.loc[df_spyderco['target'] == 0.0]
```

In [ ]:
```
buy_df.info()
```

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

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

In [ ]:
```
buy_df.info()
```

In [ ]:
```
dont_buy_df.info()
```

In [ ]:
```

```

Both knives are very similar from a business standpoint. Spyderco knives have higher profits, but the slightly higher cost of the knives (10 dollars more each) make the return on the invesment even out to be around the same as the case knives. Choosing to optimize for either knife would be a great bet, as long as the model can find distintcive features and there isnt alot of "trash" data, ie. pictures that do not show images of knives. Even though I was careful only to use the ebay API to get the URL for the images, it seems there were still alot of images of actual

knife cases that snuck through for Case XX knives that are hard to deal with. I believe not enough to ruin the distrubtions, but perhaps enough to throw off my learning rates a bit when training my model. Therefore, I am going to go with spyderco knives. Also, with first hand experience at the store, the spyderco bucket always seemed to have a wider selection than the case knives and there seemed to be more chinese fake "case" knives that might trick my model. So based on the statistics and a bit of domain understanding, I choose my target of Spyderco knives.

In [ ]:

In [ ]:
```
import os
import requests


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.galleryURL
    print(f"Downloading {url} to {filename}")
    r = requests.get(url, allow_redirects=True)
    with open(filename, 'wb') as f:
        f.write(r.content)

root_folder = 'data/not_buy/'
im_extension = '.jpg'  # or whatever type of images you are downloading

dont_buy_df.apply(download, axis=1)
```

In [ ]:
```
# import os
# import requests


# 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.galleryURL
#     print(f"Downloading {url} to {filename}")
#     r = requests.get(url, allow_redirects=True)
#     with open(filename, 'wb') as f:
#         f.write(r.content)

# root_folder = 'data/buy/'
# im_extension = '.jpg'  # or whatever type of images you are downloading

# buy_df.apply(download, axis=1)
```

In [ ]:
```
data_buy_dir = 'data/buy'
data_not_buy_dir = 'data/not_buy'
new_dir = 'split'
```

You can use `os.listdir()` to create an object that stores all the relevant image names.

In [ ]:
```
imgs_buy = [file for file in os.listdir(data_buy_dir) if file.endswith('.jpg')]
```

In [ ]:
```
import re
def sorted_nicely(image_list):
    """
    Sorts the given iterable in the way that is expected.

    Required arguments:
    l -- The iterable to be sorted.

    """
    convert = lambda text: int(text) if text.isdigit() else text
    alphanum_key = lambda key: [convert(c) for c in re.split('([0-9]+)', key)]
    return sorted(image_list, key = alphanum_key)

# Driver code
a = ["v1_0001.jpg","v1_0002.jpg","v1_0003.jpg","v1_00017.jpg","v1_00015.jpg"]
print(sorted_nicely(a))
```

In [ ]:
```
imgs_buy[0:10]
```

In [ ]:
```
imgs_buy = sorted_nicely(imgs_buy)
```

Let's see how many images there are in the `santa` directory.

In [ ]:
```
print('There are', len(imgs_buy), 'images of knives that optimize returns.')
```

Now, repeat this for the `not_santa` directory:

In [ ]:
```
imgs_not_buy = [file for file in os.listdir(data_not_buy_dir) if file.endswith('
```

In [ ]:
```
imgs_not_buy = sorted_nicely(imgs_not_buy)
```

In [ ]:
```
imgs_not_buy[:10]
```

In [ ]:
```
print('There are', len(imgs_not_buy), 'images with knives not worth buying.')
```

Create all the folders and subfolders in order to get the structure represented above. You can use `os.path.join()` to create strings that will be used later on to generate new directories.

In [ ]:
```
os.mkdir(new_dir)
```

In [ ]:
```
train_folder = os.path.join(new_dir, 'train')
train_buy = os.path.join(train_folder, 'buy')
train_not_buy = os.path.join(train_folder, 'not_buy')

test_folder = os.path.join(new_dir, 'test')
test_buy = os.path.join(test_folder, 'buy')
test_not_buy = os.path.join(test_folder, 'not_buy')

val_folder = os.path.join(new_dir, 'validation')
val_buy = os.path.join(val_folder, 'buy')
val_not_buy = os.path.join(val_folder, 'not_buy')
```

In [ ]:
```
val_buy
```

In [ ]:
```
data_buy_dir
```

Now use all the path strings you created to make new directories. You can use `os.mkdir()` to do this. Go have a look at your directory and see if this worked!

In [ ]:
```
os.mkdir(test_folder)
os.mkdir(test_buy)
os.mkdir(test_not_buy)

os.mkdir(train_folder)
os.mkdir(train_buy)
os.mkdir(train_not_buy)

os.mkdir(val_folder)
os.mkdir(val_buy)
os.mkdir(val_not_buy)
```

In [ ]:
```
len(imgs_buy)*.15
```

In [ ]:
```
2115+453
```

In [ ]:
```
train_buy
```

Copy the Santa images in the three santa subfolders. Let's put the first 271 images in the training set, the next 100 images in the validation set and the final 90 images in the test set.

In [ ]:
```
# train santa
imgs = imgs_buy[:2115]
for img in imgs:
    origin = os.path.join(data_buy_dir, img)
    destination = os.path.join(train_buy, img)
    shutil.copyfile(origin, destination)
```

In [ ]:
```
# validation santa
imgs = imgs_buy[2115:2568]
for img in imgs:
    origin = os.path.join(data_buy_dir, img)
    destination = os.path.join(val_buy, img)
    shutil.copyfile(origin, destination)
```

In [ ]:
```
# test santa
imgs = imgs_buy[2568:]
for img in imgs:
    origin = os.path.join(data_buy_dir, img)
    destination = os.path.join(test_buy, img)
    shutil.copyfile(origin, destination)
```

In [ ]:
```
len(imgs_not_buy)*.15
```

In [ ]:
```
len(imgs_not_buy)*.70
```

In [ ]:
```
2071+443
```

Now, repeat all this for the `not_santa` images!

In [ ]:
```
# train not_santa
imgs = imgs_not_buy[:2071]
for img in imgs:
    origin = os.path.join(data_not_buy_dir, img)
    destination = os.path.join(train_not_buy, img)
    shutil.copyfile(origin, destination)
# validation not_santa
imgs = imgs_not_buy[2071:2514]
for img in imgs:
    origin = os.path.join(data_not_buy_dir, img)
    destination = os.path.join(val_not_buy, img)
    shutil.copyfile(origin, destination)
# test not_santa
imgs = imgs_not_buy[2514:]
for img in imgs:
    origin = os.path.join(data_not_buy_dir, img)
    destination = os.path.join(test_not_buy, img)
    shutil.copyfile(origin, destination)
```

Let's print out how many images we have in each directory so we know for sure our numbers are right!

In [ ]:
```
print('There are', len(os.listdir(train_buy)), 'images of knives that optimize re
```

In [ ]:
```
print('There are', len(os.listdir(val_buy)), 'images of knives that optimize retu
```

In [ ]:
```
print('There are', len(os.listdir(test_buy)), 'images of knives that optimize ret
```

In [ ]:
```
print('There are', len(os.listdir(train_not_buy)), 'images not worth buying in th
```

In [ ]:
```
print('There are', len(os.listdir(val_not_buy)), 'images of spyderco knives not w
```

In [ ]:
```
print('There are', len(os.listdir(test_not_buy)), 'images not worth buying in the
```

Use a densely connected network as a baseline

Now that we've a handle on our data, we can easily use Keras' module with image-processing tools. Let's import the necessary libraries below.

In [ ]:
```
import time
import matplotlib.pyplot as plt
import scipy
import numpy as np
from PIL import Image
from scipy import ndimage
from keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_ar

np.random.seed(123)
```

In [ ]:
```
test_folder
```

In [ ]:
```
val_folder
```

In [ ]:
```
# get all the data in the directory split/test (180 images), and reshape them
test_generator = ImageDataGenerator(rescale=1./255).flow_from_directory(
        test_folder,
        target_size=(256, 256), batch_size = 200)

# get all the data in the directory split/validation (200 images), and reshape th
val_generator = ImageDataGenerator(rescale=1./255).flow_from_directory(
        val_folder,
        target_size=(256, 256), batch_size = 200)

# get all the data in the directory split/train (542 images), and reshape them
train_generator = ImageDataGenerator(rescale=1./255).flow_from_directory(
        train_folder,
        target_size=(256, 256), batch_size = 200)
```

In [ ]:
```
# create the data sets
train_images, train_labels = next(train_generator)
test_images, test_labels = next(test_generator)
val_images, val_labels = next(val_generator)
```

In [ ]:
```
train_folder
```

In [ ]:
```
import os
from PIL import Image
folder_path = r'split/train/'
extensions = []
for fldr in os.listdir(folder_path):
    sub_folder_path = os.path.join(folder_path, fldr)
    for filee in os.listdir(sub_folder_path):
        file_path = os.path.join(sub_folder_path, filee)
        print('** Path: {}  **'.format(file_path), end="\r", flush=True)
        im = Image.open(file_path)
        rgb_im = im.convert('RGB')
        if filee.split('.')[1] not in extensions:
            extensions.append(filee.split('.')[1])
```

In [ ]:
```

```

In [ ]:
```
train_img = train_images.reshape(train_images.shape[0], -1)
test_img = test_images.reshape(test_images.shape[0], -1)
val_img = val_images.reshape(val_images.shape[0], -1)

print(train_img.shape)
print(test_img.shape)
print(val_img.shape)
```

In [ ]:
```
train_y = np.reshape(train_labels[:,0], (896,1))
test_y = np.reshape(test_labels[:,0], (896,1))
val_y = np.reshape(val_labels[:,0], (896,1))
```

In [ ]:
```
# Build a baseline fully connected model
from keras import models
from keras import layers
np.random.seed(123)
model = models.Sequential()
model.add(layers.Dense(20, activation='relu', input_shape=(196608,))) # 2 hidden
model.add(layers.Dense(7, activation='relu'))
model.add(layers.Dense(5, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

In [ ]:
```
model.compile(optimizer='sgd',
              loss='binary_crossentropy',
              metrics=['accuracy'])

histoire = model.fit(train_img,
                     train_y,
                     epochs=50,
                     batch_size=32,
                     validation_data=(val_img, val_y))
```

In [ ]:
```
results_train = model.evaluate(train_img, train_y)
```

In [ ]:
```
results_test = model.evaluate(test_img, test_y)
```

In [ ]:
```
results_train
```

In [ ]:
```
results_test
```

In [ ]:
```
import PIL
```

In [ ]:
```
# Explore your dataset again
m_train = train_images.shape[0]
num_px = train_images.shape[1]
m_test = test_images.shape[0]
m_val = val_images.shape[0]

print ("Number of training samples: " + str(m_train))
print ("Number of testing samples: " + str(m_test))
print ("Number of validation samples: " + str(m_val))
print ("train_images shape: " + str(train_images.shape))
print ("train_labels shape: " + str(train_labels.shape))
print ("test_images shape: " + str(test_images.shape))
print ("test_labels shape: " + str(test_labels.shape))
print ("val_images shape: " + str(val_images.shape))
print ("val_labels shape: " + str(val_labels.shape))
```

In [ ]:
```
train_img = train_images.reshape(train_images.shape[0], -1)
test_img = test_images.reshape(test_images.shape[0], -1)
val_img = val_images.reshape(val_images.shape[0], -1)

print(train_img.shape)
print(test_img.shape)
print(val_img.shape)
```

Remember that, in our previous lab on building deeper neural networks from scratch, we obtained a training accuracy of 95%, and a test set accuracy of 74.23%.

This result is similar to what we got building our manual "deeper" dense model. The results are not entirely different. This is not a surprise!

- Before, we only had a training and a validation set (which was at the same time the test set). Now we have split up the data 3-ways.
- We didn't use minibatches before, yet we used mini-batches of 32 units here.

Build a CNN

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.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.Conv2D(256, (3, 3), activation='relu', padding='same'))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(256, (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(Dense(128, activation='relu'))
model.add(Dense(1, activation='linear'))

model.compile(loss='mean_squared_error',
              optimizer='Adam',
               metrics=['mse'])

history = model.fit(train_img,
                    train_y,
                    epochs=30,
                    batch_size=32,
                    validation_data=(val_img, val_y))
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:
```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(64 ,64,  3)))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(32, (4, 4), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

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

model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer="sgd",
              metrics=['acc'])
```

In [ ]:
```
history = model.fit(train_images,
                    train_y,
                    epochs=30,
                    batch_size=32,
                    validation_data=(val_images, val_y))
```

In [ ]:
```
results_train = model.evaluate(train_images, train_y)
```

In [ ]:
```
results_test = model.evaluate(test_images, test_y)
```

In [ ]:
```
results_train
```

In [ ]:
```
results_test
```

Data Augmentation

`ImageDataGenerator()` becomes really useful when we *actually* want to generate more data. We'll show you how this works.

In [ ]:
```
train_datagen = ImageDataGenerator(rescale=1./255,
                                   rotation_range=40,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   shear_range=0.3,
                                   zoom_range=0.1,
                                   horizontal_flip=False)
```

In [ ]:
```
names = [os.path.join(train_santa, name) for name in os.listdir(train_santa)]
img_path = names[91]
img = load_img(img_path, target_size=(64, 64))

reshape_img = img_to_array(img)
reshape_img = reshape_img.reshape((1,) + reshape_img.shape)
i=0
for batch in train_datagen.flow(reshape_img, batch_size=1):
    plt.figure(i)
    imgplot = plt.imshow(array_to_img(batch[0]))
    i += 1
    if i % 3 == 0:
        break
plt.show()
```

In [ ]:
```
# get all the data in the directory split/test (180 images), and reshape them
test_generator = ImageDataGenerator(rescale=1./255).flow_from_directory(
        test_folder,
        target_size=(64, 64),
        batch_size = 180,
        class_mode='binary')

# get all the data in the directory split/validation (200 images), and reshape th
val_generator = ImageDataGenerator(rescale=1./255).flow_from_directory(
        val_folder,
        target_size=(64, 64),
        batch_size = 32,
        class_mode='binary')

# get all the data in the directory split/train (542 images), and reshape them
train_generator = train_datagen.flow_from_directory(
        train_folder,
        target_size=(64, 64),
        batch_size = 32,
        class_mode='binary')
```

In [ ]:
```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(64 ,64,  3)))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(32, (4, 4), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

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

model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer= 'sgd',
              metrics=['acc'])
```

In [ ]:
```
history_2 = model.fit_generator(train_generator,
                                steps_per_epoch=25,
                                epochs=30,
                                validation_data=val_generator,
                                validation_steps=25)
```

In [ ]:
```
test_x, test_y = next(test_generator)
```

In [ ]:
```
results_test = model.evaluate(test_x, test_y)
```

In [ ]:
```
results_test
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:
```
# data_knife_dir = 'knife_images'
# data_profit_dir = 'data/profit'
# new_dir = 'split'
```

In [ ]:
```
# os.mkdir(new_dir)
```

In [ ]:
```
# train_folder = os.path.join(new_dir, 'train')
# train_profit = os.path.join(train_folder, 'profit')
# os.mkdir(train_folder)
# os.mkdir(train_profit)

# test_folder = os.path.join(new_dir, 'test')
# test_profit = os.path.join(test_folder, 'profit')
# os.mkdir(test_folder)
# os.mkdir(test_profit)


# val_folder = os.path.join(new_dir, 'validation')
# val_profit = os.path.join(val_folder, 'profit')
# os.mkdir(val_folder)
# os.mkdir(val_profit)
```

In [ ]:
```
# val_profit
```

In [ ]:
```
# # train knife regression images
# #80% of data
# imgs = knife_images[:5620]
# for img in imgs:
#     origin = os.path.join(data_knife_dir, img)
#     destination = os.path.join(train_profit, img)
#     shutil.copyfile(origin, destination)

# # test knife regression images
# #10% of data
# imgs = knife_images[5620:6322]
# for img in imgs:
#     origin = os.path.join(data_knife_dir, img)
#     destination = os.path.join(test_profit, img)
#     shutil.copyfile(origin, destination)


# # validation knife regression images
# #10% of data
# imgs = knife_images[6322:]
# for img in imgs:
#     origin = os.path.join(data_knife_dir, img)
#     destination = os.path.join(val, img)
#     shutil.copyfile(origin, destination)
```

In [ ]:
```
df_45 = df_scrub.loc[df_scrub['benchmade'] != 0]
```

In [ ]:
```
top_benchmade_index = df_45.sort_values(by=['profit'], ascending=False).index
```

In [ ]:
```
```

In [ ]:
```
import tensorflow as tf
from tensorflow import keras
from keras.models import load_model
from keras.preprocessing import image
import numpy as np
import cv2
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras.layers import Input, Dropout, Conv2D, Dense, Flatten, Globa

img_array = cv2.imread('data/knife_images/918.jpg')  # convert to array

img_rgb = cv2.resize(img_array,(256,256),3)
plt.imshow(img_rgb)  # graph it
plt.show();
```

In [ ]:
```
import os
import requests


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.galleryURL
    print(f"Downloading {url} to {filename}")
    r = requests.get(url, allow_redirects=True)
    with open(filename, 'wb') as f:
        f.write(r.content)

root_folder = 'data/knife_images/'
im_extension = '.jpg'  # or whatever type of images you are downloading

dont_buy_df.apply(download, axis=1)
```

In [ ]:
```
df_spyderco['buy'] = df_spyderco['ROI'] > 168
```

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

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

In [ ]:
```
img_array = cv2.imread('data/knife_images/918.jpg')  # convert to array

img_rgb = cv2.resize(img_array,(256,256),3)
plt.imshow(img_rgb)  # graph it
plt.show();
```

In [46]:
```
import os
```

In [ ]:
```
# image_list = []
# files = glob.glob ("C:/Users/12108/OneDrive/Documents/GitHub/Neural_Network_Pre
# for myFile in files:
#     print(myFile)
#     image_array = cv2.imread (myFile)
# #     img_rgb = cv2.resize(image_array,(256,256),3)
#     image_list.append (img_array)

# print('image_data:', np.array(image_list).shape)
```

In [47]:
```
path = 'C:/Users/12108/Documents/GitHub/Neural_Network_Predicting_Reseller_Succes
```

In [67]:
```
imgs_buy = [file for file in os.listdir(path) if file.endswith('.jpg')]
```

In [68]:
```
import re
def sorted_nicely(image_list):
    """
    Sorts the given iterable in the way that is expected.

    Required arguments:
    l -- The iterable to be sorted.

    """
    convert = lambda text: int(text) if text.isdigit() else text
    alphanum_key = lambda key: [convert(c) for c in re.split('([0-9]+)', key)]
    return sorted(image_list, key = alphanum_key)

# Driver code
a = ["v1_0001.jpg","v1_0002.jpg","v1_0003.jpg","v1_00017.jpg","v1_00015.jpg"]
print(sorted_nicely(a))
```

```
['v1_0001.jpg', 'v1_0002.jpg', 'v1_0003.jpg', 'v1_00015.jpg', 'v1_00017.jpg']
```

In [69]:
```
len(imgs_buy)
```

Out[69]:
```
60248
```

In [70]:
```
sorted_image_list = sorted_nicely(imgs_buy)
```

In [71]:
```
df_new = pd.DataFrame(sorted_image_list)
```

In [72]:
```
pattern = re.compile('(\D)')
```

In [73]:
```
df_new[0] = df_new[0].apply(lambda x: re.sub(pattern, "",x))
```

In [74]:
```
df_new[0] = df_new[0].apply(lambda x: int(x))
```

In [75]:
```
new_df_indx = df_new[0] .values
```

In [77]:
```
old_df_indx = df_filtered.index.values
```

In [78]:
```
missing_imgs = list(set(old_df_indx) - set(new_df_indx))
```

In [79]:
```
df_ready = df_filtered.drop(missing_imgs)
```

In [80]:
```
df_ready.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 60248 entries, 0 to 60279
Data columns (total 12 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Image            60248 non-null  object 
 1   date_sold        47133 non-null  object 
 2   title            53273 non-null  object 
 3   avg_price        60248 non-null  float64
 4   avg_shipping     60248 non-null  float64
 5   converted_price  60248 non-null  float64
 6   profit           60248 non-null  float64
 7   ROI              60248 non-null  float64
 8   brand            60248 non-null  object 
 9   cost             60248 non-null  float64
 10  url              2317 non-null   object 
 11  binary_target    60248 non-null  int64  
dtypes: float64(6), int64(1), object(5)
memory usage: 6.0+ MB
```

In [81]:
```
import cv2
```

In [ ]:
```
for filename in os.listdir('C:/Users/12108/Documents/GitHub/Neural_Network_Predic
    if filename.endswith('.jpg'):
        try:
            img = Image.open('C:/Users/12108/Documents/GitHub/Neural_Network_Pred
            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_
```

In [ ]:
```
imgs_buy = [file for file in os.listdir(path) if file.endswith('.jpg')]
sorted_image_list = sorted_nicely(imgs_buy)
```

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

In [82]:
```
image_list = []
for image in sorted_image_list:
    image_array = cv2.imread(path + image)
    try:
        img_rgb = cv2.resize(image_array,(100,100),3)
    except Exception as e:
        print(str(e) + f'error at {image}')

    image_list.append(img_rgb)


print('image_data:', np.array(image_list).shape)
```

```
OpenCV(4.6.0) D:\a\opencv-python\opencv-python\opencv\modules\imgproc\src\resiz
e.cpp:4052: error: (-215:Assertion failed) !ssize.empty() in function 'cv::resi
ze'
error at 44701.jpg
image_data: (60248, 100, 100, 3)
```

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

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

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

In [ ]:
```
df_filtered.drop([562, 2167, 4171, 5780, 6722, 7687, 8170, 8548, 8640, 9084, 9996
```

In [ ]:
```
df_new = pd.DataFrame(sorted_image_list)
```

In [ ]:
```
pattern = re.compile('(\d)')
```

In [ ]:
```
df_new
```

In [ ]:
```
df_new['num'] = df_new[0].apply(lambda x: re.findall(pattern, str(x)))
```

In [ ]:
```

```

In [84]:
```
df_ready['target'] = (df_ready['ROI']/df_ready['ROI'].mean())
```

In [85]:
```
df_ready['target'].describe()
```

Out[85]:
```
count    60248.000000
mean         1.000000
std          1.133773
min         -0.871252
25%          0.137035
50%          0.741346
75%          1.607150
max          4.907001
Name: target, dtype: float64
```

In [86]:
```
y = df_ready['target']
```

In [ ]:
```

```

In [ ]:
```

```

In [87]:
```
X = np.array(image_list)
```

In [ ]:
```
# len(df_ready.loc[df_ready['profit'] > 110]) / len(df_ready)
```

In [ ]:
```
# df_ready['target'] = (df_ready['profit'] > 110)
```

In [ ]:
```
# df_ready['target'].replace({True: 1, False:0}, inplace=True)
```

In [ ]:
```
# df_ready.drop([1806,7527,12286,13101], inplace=True)
```

In [ ]:
```
# y = df_ready['target']
```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```
# train_buy = os.listdir('split/train/buy')
# train_buy = sorted_nicely(train_buy)
# train_not_buy = os.listdir('split/train/not_buy')
# train_not_buy = sorted_nicely(train_buy)
```

In [ ]:
```
# test_buy = os.listdir('split/test/buy')
# test_buy = sorted_nicely(test_buy)
```

In [ ]:
```
# test_not_buy = os.listdir('split/test/not_buy')
# test_not_buy = sorted_nicely(test_not_buy)
```

In [ ]:
```
# import cv2
# import glob
# import numpy as np

# val_buy_data = []
# files = glob.glob ("C:/Users/12108/OneDrive/Documents/GitHub/Neural_Network_Pre
# for myFile in files:
#     print(myFile)
#     image = cv2.imread (myFile)
#     img_rgb = cv2.resize(image,(256,256),3)
#     val_buy_data.append (img_rgb)

# print('val_buy_data:', np.array(val_buy_data).shape)
```

In [ ]:
```
# val_not_buy_data = []
# files = glob.glob ("C:/Users/12108/OneDrive/Documents/GitHub/Neural_Network_Pre
# for myFile in files:
#     print(myFile)
#     image = cv2.imread (myFile)
#     img_rgb = cv2.resize(image,(256,256),3)
#     val_not_buy_data.append (img_rgb)

# print('val__notbuy_data:', np.array(val_not_buy_data).shape)
```

In [ ]:
```
# train_buy_data = []
# files = glob.glob ("C:/Users/12108/OneDrive/Documents/GitHub/Neural_Network_Pre
# for myFile in files:
#     print(myFile)
#     image = cv2.imread (myFile)
#     img_rgb = cv2.resize(image,(256,256),3)
#     train_buy_data.append (img_rgb)

# print('train_buy_data:', np.array(train_buy_data).shape)
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:
```
df_CNN_regression['mean_profit']= (df_CNN_regression['profit']/df_CNN_regression|
```

In [ ]:
```
df_CNN_regression['mean_profit'].describe()
```

In [ ]:
```
X = np.array(image_list)
```

In [ ]:
```
y=  df_CNN_regression['mean_profit']
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:
```
!pip install sklearn
```

In [ ]:

In [ ]:
```
 import numpy as np
from sklearn.model_selection import train_test_split
```

In [ ]:
```
from random import sample
```

In [ ]:
```
len(X)*.7
```

In [ ]:
```
X = np.array(image_list)
```

In [ ]:

In [ ]:

In [177]:
```
y=df_ready['binary_target']
```

In [ ]:

In [ ]:

In [ ]:
```
y= df_spyderco['buy']
```

In [ ]:
```
y.shape
```

In [178]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, test_si
```

In [179]:
```
print("Xtrain:", X_train.shape)
print("y_train:", y_train.shape)
print("X_test:", X_test.shape)
print("y_test:", y_test.shape)
```

```
Xtrain: (42173, 100, 100, 3)
y_train: (42173,)
X_test: (18075, 100, 100, 3)
y_test: (18075,)
```

In [180]:
```
X_test, X_val, y_test, y_val = train_test_split(X_test, y_test, test_size=0.5, ra

print("Xtrain:", X_train.shape)
print("y_train:", y_train.shape)
print("X_test:", X_test.shape)
print("y_test:", y_test.shape)
print("X_val:", X_test.shape)
print("y_val:", y_test.shape)
```

```
Xtrain: (42173, 100, 100, 3)
y_train: (42173,)
X_test: (9037, 100, 100, 3)
y_test: (9037,)
X_val: (9037, 100, 100, 3)
y_val: (9037,)
```

In [181]:
```
from keras import models
from keras import layers
```

In [92]:
```
model = models.Sequential()

model.add(layers.Conv2D(32, (3, 3), padding='same', activation='relu',
                        input_shape=(100 ,100,  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.Conv2D(64, (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(500, activation='relu'))
model.add(Dropout(0.1))

model.add(Dense(100, activation='relu'))
model.add(Dropout(0.1))

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

model.compile(loss='mean_absolute_error',
              optimizer='Adam',
               metrics=['mae'])
history = model.fit(X_train,
                    y_train,
                    epochs=5,
                    batch_size=32,
                    validation_data=(X_val, y_val))
```

```
Epoch 1/5
1318/1318 [==============================] - 1581s 1s/step - loss: 0.9418 - m
ae: 0.9418 - val_loss: 0.8969 - val_mae: 0.8969
Epoch 2/5
1318/1318 [==============================] - 1647s 1s/step - loss: 0.8395 - m
ae: 0.8395 - val_loss: 0.8306 - val_mae: 0.8306
Epoch 3/5
1318/1318 [==============================] - 1735s 1s/step - loss: 0.8283 - m
```

```
ae: 0.8283 - val_loss: 0.8316 - val_mae: 0.8316
Epoch 4/5
1318/1318 [==============================] - 1740s 1s/step - loss: 0.8150 - m
ae: 0.8150 - val_loss: 0.8768 - val_mae: 0.8768
Epoch 5/5
1318/1318 [==============================] - 1737s 1s/step - loss: 0.8002 - m
ae: 0.8002 - val_loss: 0.7936 - val_mae: 0.7936
```

In [93]:
```
results_train = model.evaluate(X_test, y_test)
```

```
283/283 [==============================] - 67s 236ms/step - loss: 0.7851 - mae:
0.7851
```

In [94]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 100, 100, 32)      896

 batch_normalization (BatchN  (None, 100, 100, 32)     128
 ormalization)

 conv2d_1 (Conv2D)           (None, 100, 100, 32)      9248

 batch_normalization_1 (Batc  (None, 100, 100, 32)     128
 hNormalization)

 max_pooling2d (MaxPooling2D  (None, 50, 50, 32)       0
 )

 conv2d_2 (Conv2D)           (None, 50, 50, 64)        18496

 batch_normalization_2 (Batc  (None, 50, 50, 64)       256
 hNormalization)

 conv2d_3 (Conv2D)           (None, 50, 50, 64)        36928

 batch_normalization_3 (Batc  (None, 50, 50, 64)       256
 hNormalization)

 conv2d_4 (Conv2D)           (None, 50, 50, 64)        36928

 batch_normalization_4 (Batc  (None, 50, 50, 64)       256
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 25, 25, 64)       0
 2D)

 conv2d_5 (Conv2D)           (None, 25, 25, 100)       57700

 batch_normalization_5 (Batc  (None, 25, 25, 100)      400
 hNormalization)

 conv2d_6 (Conv2D)           (None, 25, 25, 100)       90100

 batch_normalization_6 (Batc  (None, 25, 25, 100)      400
 hNormalization)

 max_pooling2d_2 (MaxPooling  (None, 12, 12, 100)      0
 2D)

 flatten (Flatten)           (None, 14400)             0

 dense (Dense)               (None, 500)               7200500

 dropout (Dropout)           (None, 500)               0

 dense_1 (Dense)             (None, 100)               50100
```

```
 dropout_1 (Dropout)          (None, 100)               0

 dense_2 (Dense)              (None, 1)                 101

=================================================================
Total params: 7,502,821
Trainable params: 7,501,909
Non-trainable params: 912
_________________________________________________________________
```

In [95]:
```
df_ready['ROI'].describe()
```

Out[95]:
```
count    60248.000000
mean       109.095834
std        123.689947
min        -95.050000
25%         14.950000
50%         80.877778
75%        175.333333
max        535.333333
Name: ROI, dtype: float64
```

In [96]:
```
df_ready['target'].describe()
```

Out[96]:
```
count    60248.000000
mean         1.000000
std          1.133773
min         -0.871252
25%          0.137035
50%          0.741346
75%          1.607150
max          4.907001
Name: target, dtype: float64
```

In [98]:
```
df_ready['ROI'].mean() * 0.7851
```

Out[98]:
```
85.65113961874253
```

In [ ]:
```
model.predict()
```

In [107]:
```
df_ready.loc[df_ready['ROI'] > 100].sample(10)
```

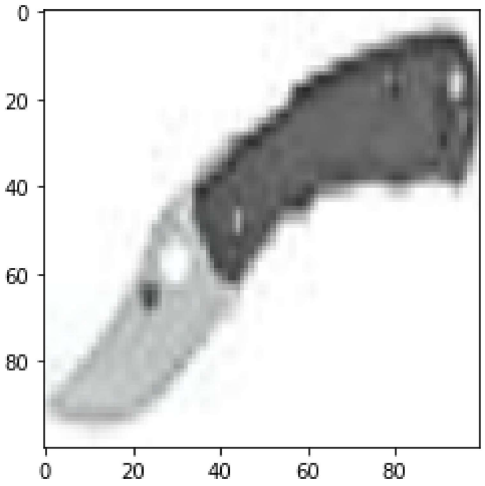
Out[107]:

	Image	date_sold	title	avg_price	avg
18414	https://thumbs.ebaystatic.com/pict/17515863347...	Feb 22, 2022	2007 Case XX JOHN DEERE COPPERLOCK Knife 61549...	66.05	
5019	https://thumbs.ebaystatic.com/pict/39392640105...	Feb 17, 2022	NTSA BENCHMADE USA " BARRAGE" 4 3/4" CLOSED A...	72.01	
21169	https://thumbs.ebaystatic.com/pict/25544485233...	Mar 26, 2022	Vintage Case XX TB6339 Bone Sowbelly Stockman ...	81.00	
12127	https://thumbs.ebaystatic.com/pict/23437431265...	NaN	BUCK RUSH 290 ASSISTED OPENING FOLDING POCKET ...	80.50	
29245	https://thumbs.ebaystatic.com/pict/27531424632...	May 24, 2022	NaN	24.60	
25600	https://i.ebayimg.com/00/s/MTYwMFgxMjAw/z/RI8A...	Jun 14, 2022	CRKT Seismic 5401 Folding Pocket Knife, read d...	46.00	
23021	https://thumbs.ebaystatic.com/pict/29452244622...	Nov 14, 2021	Case XX TB62110 SS President's Knife Saddlehor...	46.25	
4531	https://i.ebayimg.com/00/s/MTYwMFgxMjAw/z/TwEA...	Jun 20, 2022	, preview full size imageBenchmade 915 Triage ...	100.00	
2309	https://i.ebayimg.com/00/s/MTYwMFgxMTk5/z/YioA...	Aug 28, 2022	, preview full size imageBenchmade Mini Bugout...	109.49	
14615	https://thumbs.ebaystatic.com/pict/28457934166...	NaN	1990Vintage Buck 539 TruBlue Fish Fillet Knife...	50.00	

In [139]:
```
np.shape([X[48825]])
```

Out[139]:
```
(1, 100, 100, 3)
```

In [138]:
```
from matplotlib import pyplot as plt
plt.imshow(X[48825], interpolation='nearest')
plt.show()
```



In [124]:
```
df_ready.reset_index(drop=True,inplace=True)
```

In [125]:
```
df_ready.loc[df_ready['ROI'] > 100].sample(10)
```

Out[125]:

	Image	date_sold	title	avg_price	av
20663	https://thumbs.ebaystatic.com/pict/23433403966...	Dec 18, 2021	NM VINTAGE '75 CASE XX USA G137 2137 SOD BUSTE...	47.00	
20791	https://thumbs.ebaystatic.com/pict/23448386379...	Apr 6, 2022	CASE XX 2006 CARIBBEAN BLUE BONE MUSKRAT KNI...	55.99	
34368	https://thumbs.ebaystatic.com/pict/22471602170...	Dec 5, 2021	NaN	24.50	
26295	https://i.ebayimg.com/00/s/MTU4OFgxNjAw/z/ZGYA...	Aug 14, 2022	Scarce CRKT Nirk Tighe 2 knife unused SHARP!	81.00	
48825	https://thumbs.ebaystatic.com/pict/30428216051...	Jan 5, 2022	Spyderco Reinhold Rhino Knife Carbon Fiber G-1...	117.52	
15696	https://thumbs.ebaystatic.com/pict/11504167635...	Oct 16, 2021	CASE XX STAG KNIFE TRAPPER LIMITED EDITION	83.00	
34933	https://thumbs.ebaystatic.com/pict/23448434000...	Apr 3, 2022	NaN	26.54	
14005	https://thumbs.ebaystatic.com/pict/27499449685...	NaN	Pocket Knife Buck 112 USA Lock Back Folding Si...	39.00	
1462	https://thumbs.ebaystatic.com/pict/17491157872...	Aug 31, 2021	Benchmade knife 154CMBenchmade knife 154CM	125.00	
56388	https://i.ebayimg.com/00/s/MTE5NVgxNjAw/z/LAMA...	Aug 25, 2022	VICTORINOX SWISS ARMY KNIFE STAINLESS POCKET K...	50.00	

In [126]:
```
df_ready.loc[48825].apply(print)
```

```
https://thumbs.ebaystatic.com/pict/3042821605106464_1.jpg (https://thumbs.ebays
tatic.com/pict/3042821605106464_1.jpg)
Jan 5, 2022
Spyderco Reinhold Rhino Knife Carbon Fiber G-10 C210CFP XHP DISCONTINUED
117.52
0.0
117.52
87.52
291.73333333333335
spyderco
30.0
nan
1
2.674101489128121
```

Out[126]:
```
Image              None
date_sold          None
title              None
avg_price          None
avg_shipping       None
converted_price    None
profit             None
ROI                None
brand              None
cost               None
url                None
binary_target      None
target             None
Name: 48825, dtype: object
```

In [171]:
```
model.predict(np.array(X[45388].reshape(1,100,100,3)))
```

```
1/1 [==============================] - 0s 48ms/step
```

Out[171]:
```
array([[0.57079]], dtype=float32)
```

In [143]:
```
0.6280405*df_ready['ROI'].mean()
```

Out[143]:
```
68.51660240953365
```

In [168]:
```
df_ready.sample(10)
```

Out[168]:

	Image	date_sold	title	avg_price	avg_sh
1667	https://thumbs.ebaystatic.com/pict/18507438397...	Sep 26, 2021	Benchmade Griptilian AXIS Lock Knife Olive Dra...	90.00	
28146	https://thumbs.ebaystatic.com/pict/37394636126...	Feb 25, 2022	CRKT M16 -10KZ Folding Lock Back Knife Carson ...	10.00	
45388	https://i.ebayimg.com/00/s/MTYwMFgxMjAw/z/ij4A...	Jul 26, 2022	SPYDERCO AUS-8 SERATED BLADE SEKI CITY JAPAN C...	32.00	
34629	https://i.ebayimg.com/00/s/OTYwWDEyODA=/z/oLUA...	Aug 22, 2022	KERSHAW POCKET KNIFE USA 1660SWBLK Assisted O...	11.05	
22018	https://i.ebayimg.com/00/s/MTYwMFgxMjAw/z/gxoA...	Aug 26, 2022	VINTAGE CASE XX WOOD HANDLE LOCK BLADE POCKET ...	66.00	
24156	https://thumbs.ebaystatic.com/pict/13398254538...	Jan 9, 2022	CRKT M16-14Z tactical knife serrated blade	24.00	
9993	https://thumbs.ebaystatic.com/pict/19438391274...	NaN	Buck 780 Exert Folding Pocket knife	16.01	
55551	https://thumbs.ebaystatic.com/pict/25552452599...	May 8, 2022	Victorinox Recruit Swiss Army Knife Multi-Tool!	12.00	
20390	https://thumbs.ebaystatic.com/pict/22493905715...	Apr 25, 2022	PAIR CASE XX FIGURAL HAT LAPEL PIN, RUSSLOCK P...	38.00	
34206	https://i.ebayimg.com/00/s/MTIwMFgxNTk5/z/NJ4A...	Jul 31, 2022	Kershaw Scallion 1620NB Folding Pocket Knife -...	36.00	

In [158]:
```
mean = df_ready['ROI'].mean()
```

In [172]:
```
0.57079 * mean
```

Out[172]:
```
62.27081133993383
```

In [ ]:
```

```

In [182]:
```
#small batch

model = models.Sequential()

model.add(layers.Conv2D(32, (3, 3), padding='same', activation='relu',
                        input_shape=(100,100,3)))
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.Conv2D(64, (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(500, activation='relu'))

model.add(Dropout(0.1))

model.add(Dense(2, activation='softmax'))
#2 classes
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
               metrics=['accuracy'])
#switching from regression to classifier.. no longer mse loss
history = model.fit(X_train,
                    y_train,
                    epochs=10,
                    batch_size=100,
                    validation_data=(X_val, y_val))
```

```
Epoch 1/10
 25/422 [>.............................] - ETA: 15:59 - loss: 3.6015 - accurac
y: 0.5104

---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-182-d97ca3ce89a1> in <module>
     39                metrics=['accuracy'])
```

```
     40 #switching from regression to classifier.. no longer mse loss
---> 41 history = model.fit(X_train,
     42                     y_train,
     43                     epochs=10,

~\anaconda3\envs\learn-env\lib\site-packages\keras\utils\traceback_utils.py i
n error_handler(*args, **kwargs)
     63         filtered_tb = None
     64         try:
---> 65             return fn(*args, **kwargs)
     66         except Exception as e:
     67             filtered_tb = _process_traceback_frames(e.__traceback__)

~\anaconda3\envs\learn-env\lib\site-packages\keras\engine\training.py in fit
(self, x, y, batch_size, epochs, verbose, callbacks, validation_split, valida
tion_data, shuffle, class_weight, sample_weight, initial_epoch, steps_per_epo
ch, validation_steps, validation_batch_size, validation_freq, max_queue_size,
workers, use_multiprocessing)
   1562                         ):
   1563                             callbacks.on_train_batch_begin(step)
-> 1564                             tmp_logs = self.train_function(iterator)
   1565                             if data_handler.should_sync:
   1566                                 context.async_wait()

~\anaconda3\envs\learn-env\lib\site-packages\tensorflow\python\util\traceback
_utils.py in error_handler(*args, **kwargs)
    148     filtered_tb = None
    149     try:
--> 150       return fn(*args, **kwargs)
    151     except Exception as e:
    152       filtered_tb = _process_traceback_frames(e.__traceback__)

~\anaconda3\envs\learn-env\lib\site-packages\tensorflow\python\eager\def_func
tion.py in __call__(self, *args, **kwds)
    913
    914       with OptionalXlaContext(self._jit_compile):
--> 915         result = self._call(*args, **kwds)
    916
    917       new_tracing_count = self.experimental_get_tracing_count()

~\anaconda3\envs\learn-env\lib\site-packages\tensorflow\python\eager\def_func
tion.py in _call(self, *args, **kwds)
    945       # In this case we have created variables on the first call, so
 we run the
    946       # defunned version which is guaranteed to never create variable
s.
--> 947       return self._stateless_fn(*args, **kwds)  # pylint: disable=not
-callable
    948     elif self._stateful_fn is not None:
    949       # Release the lock early so that multiple threads can perform t
he call

~\anaconda3\envs\learn-env\lib\site-packages\tensorflow\python\eager\functio
n.py in __call__(self, *args, **kwargs)
   2494       (graph_function,
   2495        filtered_flat_args) = self._maybe_define_function(args, kwarg
s)
```

```
-> 2496     return graph_function._call_flat(
   2497         filtered_flat_args, captured_inputs=graph_function.captured_i
nputs)  # pylint: disable=protected-access
   2498

~\anaconda3\envs\learn-env\lib\site-packages\tensorflow\python\eager\functio
n.py in _call_flat(self, args, captured_inputs, cancellation_manager)
   1860         and executing_eagerly):
   1861       # No tape is watching; skip to running the function.
-> 1862       return self._build_call_outputs(self._inference_function.call(
   1863           ctx, args, cancellation_manager=cancellation_manager))
   1864     forward_backward = self._select_forward_and_backward_functions(

~\anaconda3\envs\learn-env\lib\site-packages\tensorflow\python\eager\functio
n.py in call(self, ctx, args, cancellation_manager)
    497       with _InterpolateFunctionError(self):
    498         if cancellation_manager is None:
--> 499           outputs = execute.execute(
    500               str(self.signature.name),
    501               num_outputs=self._num_outputs,

~\anaconda3\envs\learn-env\lib\site-packages\tensorflow\python\eager\execute.
py in quick_execute(op_name, num_outputs, inputs, attrs, ctx, name)
     52   try:
     53     ctx.ensure_initialized()
---> 54     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op_
name,
     55                                         inputs, attrs, num_outputs)
     56   except core._NotOkStatusException as e:

KeyboardInterrupt:
```

In [ ]:
```
results_test = model.evaluate(X_test, y_test)

model.summary()
```

In [ ]:
```
df_scrub['profit'].mean()
```

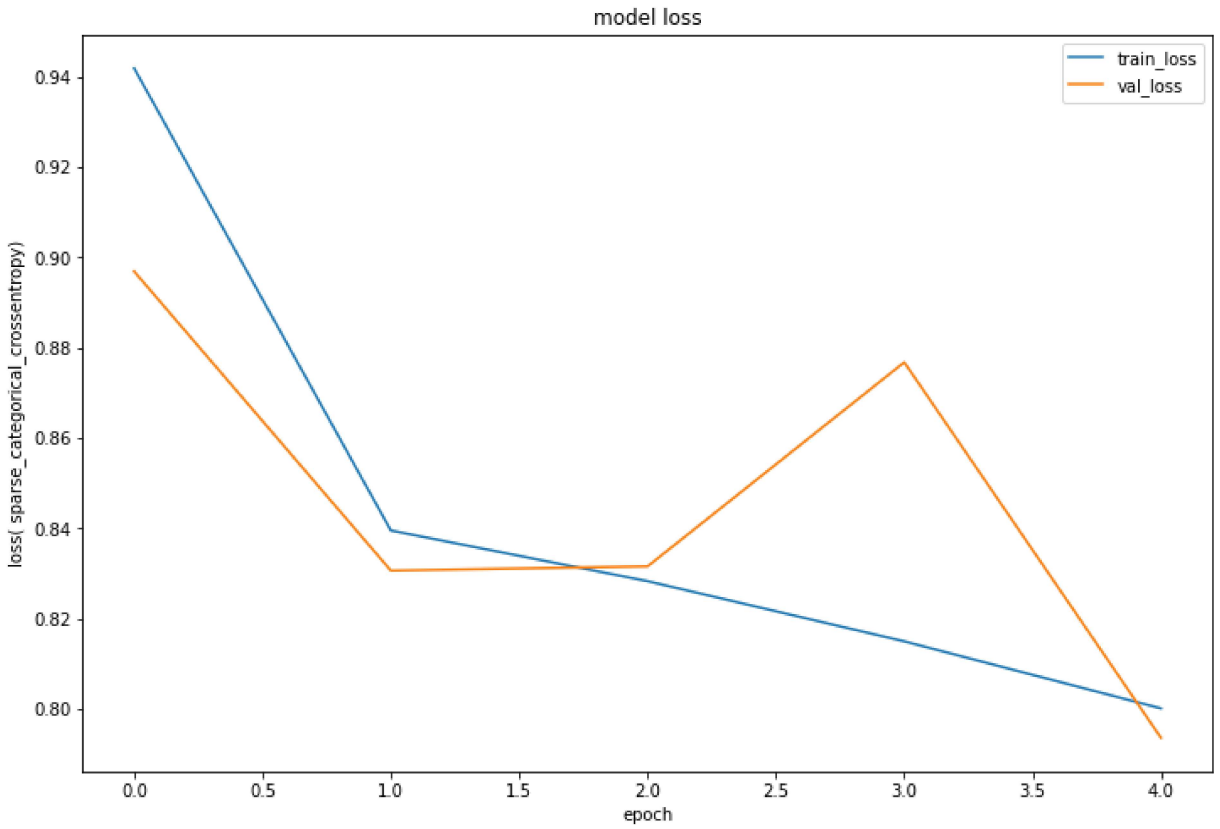
In [ ]:
```
2.2164 * 41.374303202846974
```

In [ ]:
```
df_scrub.head()
```

In [ ]:
```

```

In [ ]:
```
model.summary()
```

In [173]:
```
#The model learned patterns wells until epoch 20
#after that the loss spikes signifcantly before dropping again
fig = plt.figure(figsize=(12,8))
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.plot
plt.title('model loss')
plt.ylabel('loss( sparse_categorical_crossentropy)')
plt.xlabel('epoch')
plt.legend(['train_loss', 'val_loss'], loc='upper right')
plt.show();
```



In [174]:
```
#The model learned patterns wells until epoch 20
#after that the loss spikes signifcantly before dropping again
fig = plt.figure(figsize=(12,8))
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.plot
plt.title('model loss')
plt.ylabel('loss( sparse_categorical_crossentropy)')
plt.xlabel('epoch')
plt.legend(['train_accuracy', 'val_accuracy'], loc='upper right')
plt.show();
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-174-1fb7478216be> in <module>
      2 #after that the loss spikes signifcantly before dropping again
      3 fig = plt.figure(figsize=(12,8))
----> 4 plt.plot(history.history['accuracy'])
      5 plt.plot(history.history['val_accuracy'])
      6 plt.plot

KeyError: 'accuracy'


<Figure size 864x576 with 0 Axes>
```

In [ ]:
```
model.save('my_model_bench100.h5')
```

In [ ]:
```

```

In [ ]:
```
#a train set of 60% and a val and test size of 20% each
```

In [ ]:
```
#this model showed a lot of indication that it was overfit
#need to retry how I split the data
#Instead of manul indexing, will use
# from sklearn model_selection train_test_split



# X_train = X[:4918]
# y_train = y[:4918]

# X_train = X[4918:5971]
# y_train = y[4918:5971]

# X_test = X[5971:]
# y_test = y[5971:]


# display(len(X_val)/len(X))
# display(len(X_train)/len(X))
# len(X_test)/len(X)



# model = models.Sequential()

# model.add(layers.Conv2D(32, (3, 3), padding='same', activation='relu',
#                         input_shape=(224 ,224,  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.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(Dense(128, activation='relu'))
```

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

# model.compile(loss='mean_squared_error',
#               optimizer='Adam',
#                metrics=['mse'])
# 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 [ ]:
```
results_train = model.evaluate(X_test, y_test)
```

In [ ]:
```
model.summary()
```

In [ ]:
```
# model.save('my_model_batch500.h5')
```

In [ ]:
```
history.history.keys()
```

In [ ]:
```
#The model is showing a lot of signs of overfitting
fig = plt.figure(figsize=(12,8))
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.plot
plt.title('model loss')
plt.ylabel('loss( mean square error)')
plt.xlabel('epoch')
plt.legend(['train_mse', 'val_mse'], loc='upper right')
plt.show();
```

In [ ]:
```
X_train.shape
```

In [ ]:
```
# results_train = model.evaluate(X_test, y_test)

#model.summary()


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

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```
df_scrub.to_csv('data/clean_dataframe.csv')
```

In [ ]:
```
# import glob
# import os


# path = r'C:\Users\12108\Desktop\ebay_knife_data\dsc-5-capstone-project\surplusS
# all_files = glob.glob(os.path.join(path, "*.csv"))     # advisable to use os.pa

# df_from_each_file = (pd.read_csv(f) for f in all_files)
# concatenated_df   = pd.concat(df_from_each_file, ignore_index=True)

# concatenated_df.head()

# concatenated_df.fillna(0, inplace=True)

# concatenated_df.info()

# concatenated_df.to_csv('surplusStore/workingDataFrame2.csv')
```

THIS CALL TO THE WEBSITE RETURNED NO SOG KNIVES OR CRKT KNIVES

MUST MAKE SEPERATE CALLS

NOT SHOWN IN THIS NOTEBOOK

In [ ]:
```


df_crkt = pd.read_csv('data/full_dataset_CRKT.csv')
df_sog = pd.read_csv('data/full_dataset_SOG.csv')



# df_SOG.info()

# df_SOG['sog'] = 1.0
# df_SOG['sog'] = 1.0

# mkdir surplusStore

# df_surplus.to_csv('surplusStore/workingDataFrame.csv', index=False)
# df_CRKT.to_csv('surplusStore/df_CRKT.csv', index=False)
# df_SOG.to_csv('surplusStore/df_SOG.csv', index=False)

# df_surplus.head()
```

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.

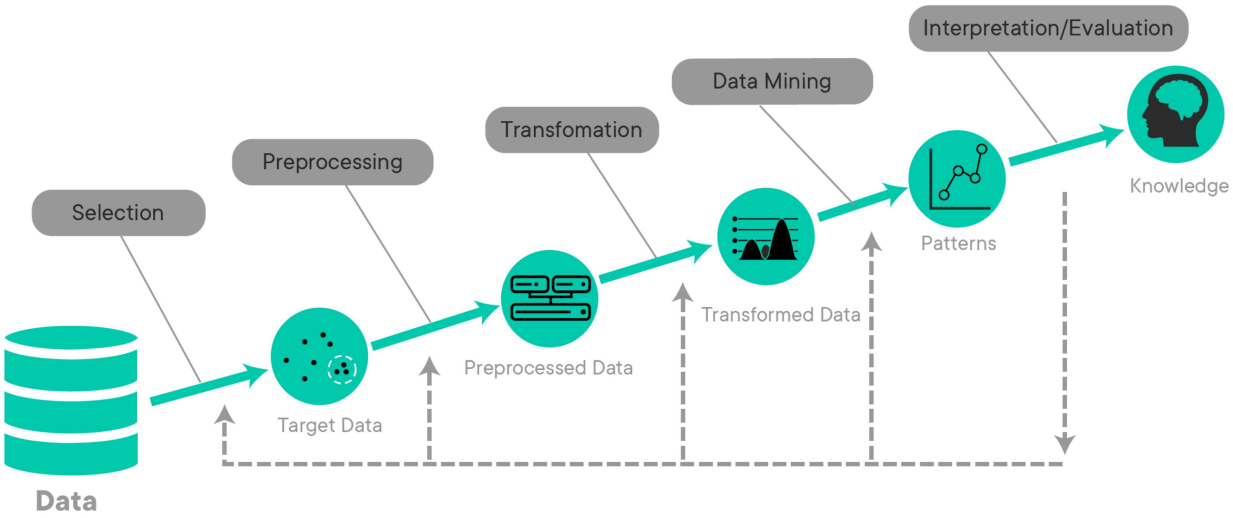
Of course, if the results are satisfactory, then we instead move onto deployment!

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.

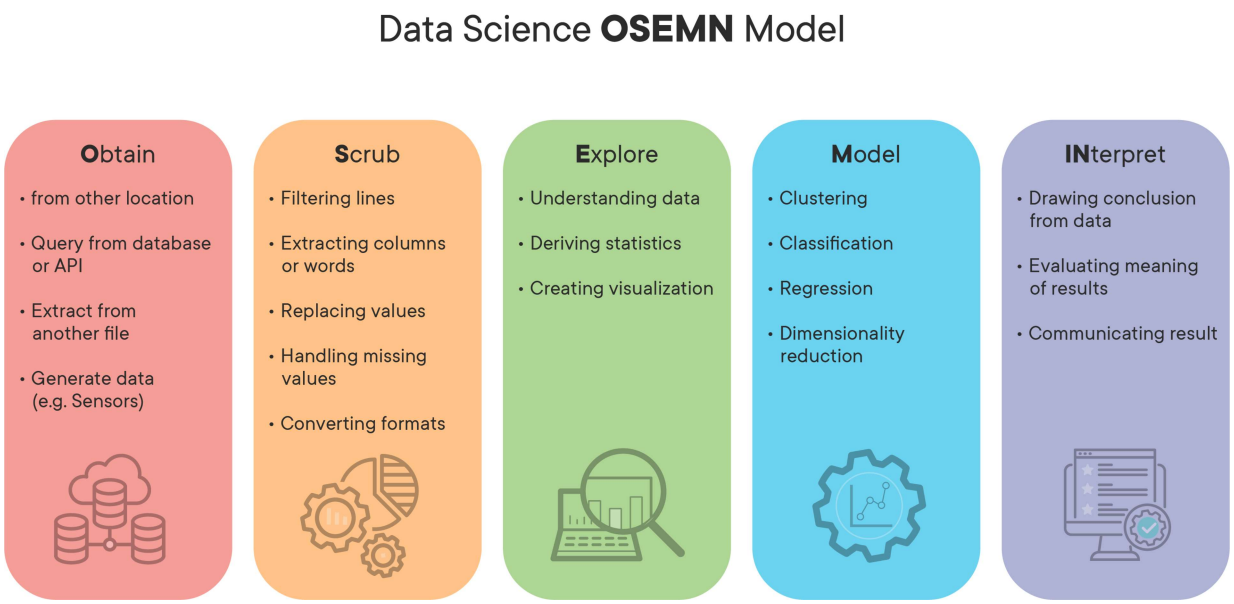
You may be wondering what Data Mining is, and how it relates to Data Science. In practice, it's just an older term that essentially means the same thing as Data Science. Dr. Piatetsky-Shapiro defines Data Mining as "the non-trivial extraction of implicit, previously unknown and potentially useful information 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)](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.

In [ ]:
```
# #REGEX BRAND PATTERNS AFTERN LOWERCASING TITLES AND REMOVING SPEVIAL CHARACTERS


#turn this into a dict bro

# benchmade_pattern = "benchmade"
# buck_pattern = "buck"
# case_pattern = "case"
# crkt_pattern = "crkt"
# kershaw_pattern = "kershaw"
# leatherman_pattern = "leatherman"
# sog_pattern = "sog"
# spyderco_pattern = "spyderco"
# victorinox_pattern = "victorinox"
```

In [ ]:
```
# df['is_profitable'] = (df['price_in_US'] - df[list(costs.keys())].sum(axis=1))>

# df.info()
```

In [ ]:
```
# #REGEX BRAND PATTERNS AFTERN LOWERCASING TITLES AND REMOVING SPEVIAL CHARACTERS


#turn this into a dict bro

# benchmade_pattern = "benchmade"
# buck_pattern = "buck"
# case_pattern = "case"
# crkt_pattern = "crkt"
# kershaw_pattern = "kershaw"
# leatherman_pattern = "leatherman"
# sog_pattern = "sog"
# spyderco_pattern = "spyderco"
# victorinox_pattern = "victorinox"

# df['is_profitable'] = (df['price_in_US'] - df[list(costs.keys())].sum(axis=1))>

# df.info()
```