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

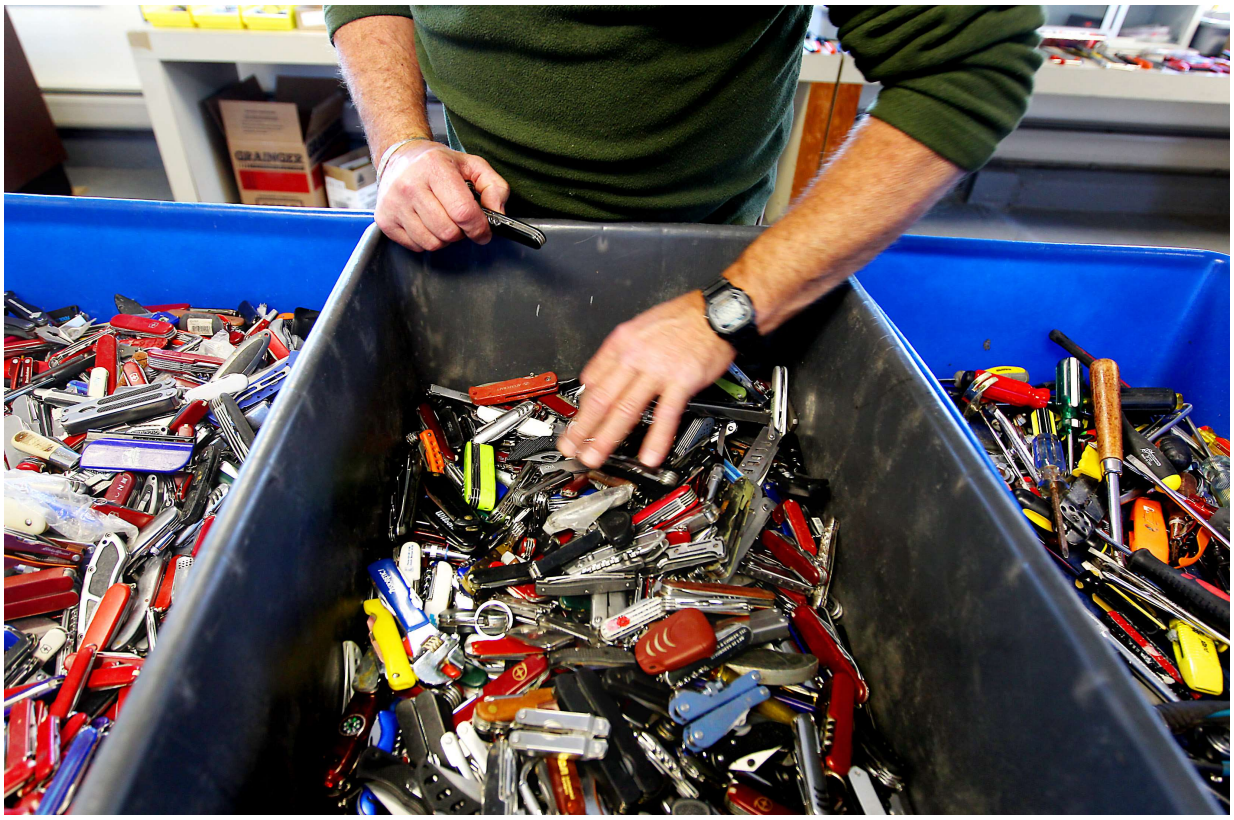
Author: Dylan Dey

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)

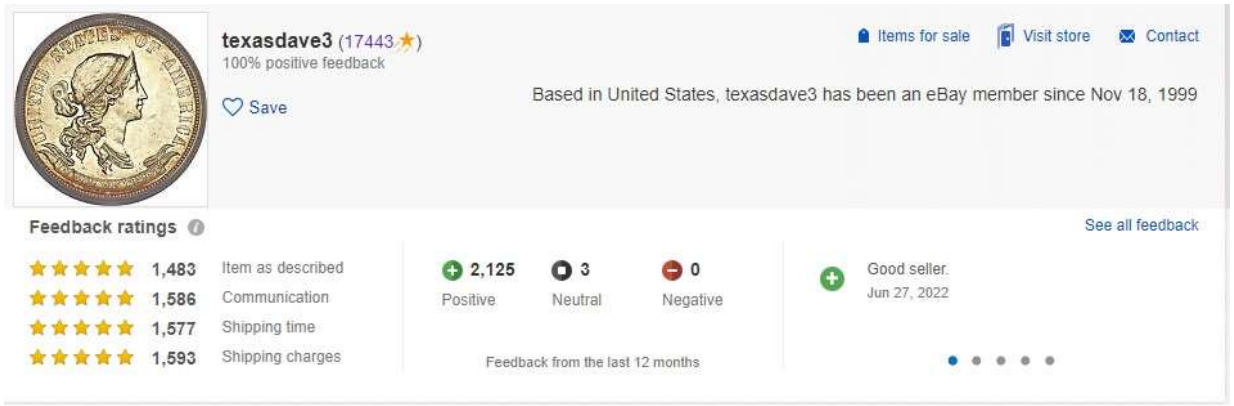


[Everything that doesn't make it through Texas airports can be found at one Austin store (https://cbsaustin.com/news/local/everything-that-doesnt-make-it-through-texas-airports-can-be-found-at-one-austin-store)](https://cbsaustin.com/news/local/everything-that-doesnt-make-it-through-texas-airports-can-be-found-at-one-austin-store)

> 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)](https://developer.ebay.com/devzone/finding/CallRef/index.html)

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

```
#import sdk and establish connnection
from ebaysdk.finding import Connection

if __name__ == '__main__':
    api = Connection(config_file='ebay.yaml', debug=True, siteid="EBAY-U
S")
#create the request
    request = {
        'keywords': 'knife',
        'itemFilter': [
            {'name': 'condition', 'value': 'Used'},
            {'name': 'ListingType', 'value': 'FixedPrice'}
        ],
        'aspectFilter': [
            {'aspectName': 'Brand', 'aspectValueName': 'Benchmade'},
            {'aspectName': 'Brand', 'aspectValueName': 'Buck'},
            {'aspectName': 'Brand', 'aspectValueName': 'Case'},
            {'aspectName': 'Brand', 'aspectValueName': 'CRKT'},
            {'aspectName': 'Brand', 'aspectValueName': 'Kershaw'},
            {'aspectName': 'Brand', 'aspectValueName': 'Leatherman'},
```

```
                {'aspectName': 'Brand', 'aspectValueName': 'Spyderco'},
                {'aspectName': 'Brand', 'aspectValueName': 'Victorinox'},
            ],
            'paginationInput': {
                'entriesPerPage': 10,
                'pageNumber': 1
            },
            'sortOrder': 'PricePlusShippingLowest'
        }

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



#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_d
ata
        # with an associated value of None

        keys = ['itemId', 'title', 'galleryURL',
                'viewItemURL', 'postalCode', 'sellingStatus',
                'shippingInfo', 'listingInfo']

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


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

    return results
```

Ebay only allows 100 pages and 100 items max

```
# Create an empty list for the full prepared dataset
full_dataset = []

for page in range(1, 100):
    # Add or update the "offset" key-value pair in url_params
    request['paginationInput']['pageNumber'] = page

    # Make the query and get the response

    api = Connection(config_file='ebay.yaml', debug=True, siteid="EBAY-U
S")

    request = {
                'keywords': 'knife',
                'itemFilter': [
                                {'name': 'condition', 'value': 'Used'},
                                {'name': 'ListingType', 'value': 'FixedP
rice'}
                              ],
                'aspectFilter': [
                                  {'aspectName': 'Brand', 'aspectValueNa
me': 'Benchmade'},
                                  {'aspectName': 'Brand', 'aspectValueNa
me': 'Buck'},
                                  {'aspectName': 'Brand', 'aspectValueNa
me': 'Case'},
                                  {'aspectName': 'Brand', 'aspectValueNa
me': 'CRKT'},
                                  {'aspectName': 'Brand', 'aspectValueNa
me': 'Kershaw'},
                                  {'aspectName': 'Brand', 'aspectValueNa
me': 'Leatherman'},
                                  {'aspectName': 'Brand', 'aspectValueNa
me': 'Spyderco'},
                                  {'aspectName': 'Brand', 'aspectValueNa
me': 'Victorinox'},
                                 ],
                'paginationInput': {
                                    'entriesPerPage': 100,
                                    'pageNumber': page

                                    },


                }
```

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

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



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

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

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

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


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

After succesfully going through 10,000 items on ebay's website and extracting everything possible, there is still a little bit more extracting to do from the json dictionary before saving the dataframe again. We need to get the price of the knives out of the nested dictionary in the dataframe as well as the shipping cost. After that, I would like to create a new feature called **"converted price," which is simply the price of the knife listed on the ebay's website plus shipping.**

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

    df['price_in_US'] = price_list



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

```
['value'])
    shipping_cost_list.append(shipping_cost)

df['shipping_price'] = shipping_cost_list



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

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

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

Data Preparation

Describe and justify the process for preparing the data for analysis.

Questions to consider:

- Were there variables you dropped or created?
- How did you address missing values or outliers?
- Why are these choices appropriate given the data and the business problem?

---

here you run your code to clean the data

```
import code.data_cleaning as dc

full_dataset = dc.full_clean()
```

Data Modeling

Describe and justify the process for analyzing or modeling the data.

Questions to consider:

- How did you analyze or model the data?

- How did you iterate on your initial approach to make it better?
- Why are these choices appropriate given the data and the business problem?

---

here you run your code to model the data

Evaluation

Evaluate how well your work solves the stated business problem.

Questions to consider:

- How do you interpret the results?
- How well does your model fit your data? How much better is this than your baseline model?
- How confident are you that your results would generalize beyond the data you have?
- How confident are you that this model would benefit the business if put into use?

Conclusions

Provide your conclusions about the work you've done, including any limitations or next steps.

Questions to consider:

- What would you recommend the business do as a result of this work?
- What are some reasons why your analysis might not fully solve the business problem?
- What else could you do in the future to improve this project?

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]:
```
df = pd.read_csv('data/full_dataset.csv')
```

In [3]:
```
#dictionary for cost of knives at surplus store
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
              }

#lowercase titles
df.title = df.title.apply(str.lower)

#remove special characters
# df.title.apply(pp.remove_special_chars)

#function for finding brands using regex
def find_brand(pattern):
    df[pattern] = df.title.apply(lambda x: re.findall(pattern, x.lower()))
    df[pattern] = df[pattern].apply(lambda x: np.nan if len(x)==0 else 1)
    df[pattern].fillna(0, inplace=True)
    return df


#apply find_brand function
for key in list(bucket_dict.keys()):
    find_brand(key)
```

In [4]:
```
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')
df.victorinox.value_counts(normalize=False)


display(df.info())
```

```
Benchmade Value Counts

0.0    7391
1.0     354
Name: benchmade, dtype: int64

-----------------------------------------
Buck Value Counts

0.0    6938
1.0     807
Name: buck, dtype: int64

-----------------------------------------
Case Value Counts

0.0    6153
1.0    1592
Name: case, dtype: int64

-----------------------------------------
CRKT Value Counts
```

```
0.0    7290
1.0     455
Name: crkt, dtype: int64

----------------------------------------
Kershaw Value Counts

0.0    6154
1.0    1591
Name: kershaw, dtype: int64

----------------------------------------
Leatherman Value Counts

0.0    7682
1.0      63
Name: leatherman, dtype: int64

----------------------------------------
Spyderco Value Counts

0.0    6550
1.0    1195
Name: spyderco, dtype: int64

----------------------------------------
Victorinox Value Counts
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7745 entries, 0 to 7744
Data columns (total 19 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   itemId           7745 non-null   int64
 1   title            7745 non-null   object
 2   galleryURL       7745 non-null   object
 3   viewItemURL      7745 non-null   object
 4   postalCode       7745 non-null   object
 5   sellingStatus    7745 non-null   object
 6   shippingInfo     7745 non-null   object
 7   listingInfo      7745 non-null   object
 8   price_in_US      7745 non-null   float64
 9   shipping_cost    7745 non-null   float64
 10  converted_price  7745 non-null   float64
 11  benchmade        7745 non-null   float64
 12  buck             7745 non-null   float64
 13  case             7745 non-null   float64
 14  crkt             7745 non-null   float64
 15  kershaw          7745 non-null   float64
 16  leatherman       7745 non-null   float64
 17  spyderco         7745 non-null   float64
 18  victorinox       7745 non-null   float64
dtypes: float64(11), int64(1), object(7)
memory usage: 1.1+ MB

None
```

In [5]:
```
df.columns
```

Out[5]:
```
Index(['itemId', 'title', 'galleryURL', 'viewItemURL', 'postalCode',
       'sellingStatus', 'shippingInfo', 'listingInfo', 'price_in_US',
       'shipping_cost', 'converted_price', 'benchmade', 'buck', 'case', 'crkt',
       'kershaw', 'leatherman', 'spyderco', 'victorinox'],
      dtype='object')
```

In [6]:
```
#replace 1 with cost of knife from surplus store to create new features
for key, val in bucket_dict.items():
    df[key] = df[key].replace(1.0, val)
```

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 [7]:
```
#define an  estimate for ovearhead_cost per knife
#in order to define new profit feature
overhead_cost = 7
```

In [8]:
```
# Binary Classification target (make 30 bucks per knife all costs considered)

df['Binary_Target'] = (df['converted_price'] - df[list(bucket_dict.keys())].sum(a

df['Binary_Target'].value_counts()
```

Out[8]:
```
True     4053
False    3692
Name: Binary_Target, dtype: int64
```

In [9]:
```
df['profit'] = (df['converted_price'] - df[list(bucket_dict.keys())].sum(axis=1)
```

In [10]:
```
# df['ROI'] = ((df['converted_price'] - df[list(bucket_dict.keys())].sum(axis=1))
```

In [11]:
```
df.drop(columns=['shippingInfo', 'listingInfo', 'sellingStatus'], inplace=True)
df.head()
```

Out[11]:

	itemId	title	galleryURL	
0	324946521772	victorinox classic sd mini swiss army pocket k...	https://i.ebayimg.com/thumbs/images/g/smEAAOSw...	https://www.ebay.com
1	113729888404	kershaw misdirect gray assisted open framelock...	https://i.ebayimg.com/thumbs/images/g/zCgAAOSw...	https://www.ebay.cor
2	175006871402	kershaw kuro linerlock folding knife 3.1" 8cr1...	https://i.ebayimg.com/thumbs/images/g/2o4AAOSw...	https://www.ebay.cor
3	125406365289	benchmade 556 mini griptilian folding pocket k...	https://i.ebayimg.com/thumbs/images/g/oLQAAOSw...	https://www.ebay.com/it
4	283684293533	1660 kershaw leek knife silver plain blade new...	https://i.ebayimg.com/thumbs/images/g/VQ0AAOSw...	https://www.ebay Kersh

In [12]:
```
bucket_dict.keys()
```

Out[12]:
```
dict_keys(['benchmade', 'buck', 'case', 'crkt', 'kershaw', 'leatherman', 'spyderco', 'victorinox'])
```

In [13]:
```
#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 [14]:
```
#there are only 4 bins to work with. Much easier.
flipped
```

Out[14]:
```
{45.0: ['benchmade'],
 20.0: ['buck', 'case', 'victorinox'],
 15.0: ['crkt', 'kershaw'],
 30.0: ['leatherman', 'spyderco']}
```

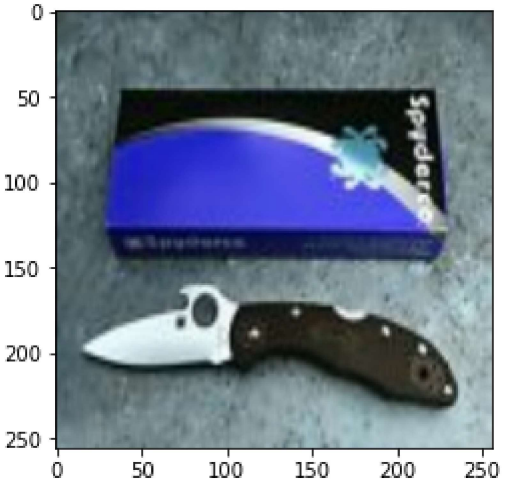
In [15]:
```
df_15 = df.loc[(df['crkt'] != 0) | (df['kershaw'] != 0)]
df_20 = df.loc[(df['buck'] != 0) | (df['case'] != 0)| (df['victorinox'] != 0)]
df_30 = df.loc[(df['leatherman'] != 0) | (df['spyderco'] != 0)]
df_45 = df.loc[df['benchmade'] != 0]
```

In [65]:
```
top_30_index = df_30.sort_values(by=['profit'], ascending=False).index
```

In [66]:
```
top_30_index[:50]
```

Out[66]:
```
Int64Index([4249, 3846, 1980,  464, 2177, 6936, 6761,  218, 1820, 7311, 4680,
             192, 7055, 3515, 3861, 1686, 5739, 5866, 6045, 6739, 3566, 6895,
            5179, 4840, 6842, 3437, 6630, 6009, 6813,  846, 1642, 3718, 3494,
            2506, 6206, 6703, 5056, 7318, 5443, 7716, 2538, 6241, 5160, 5657,
             266, 6977, 5339, 7136, 4129, 2998],
           dtype='int64')
```

In [71]:
```
image_example = image_checker(5739)
```



In [69]:
```
df_30.loc[6206]
```

Out[69]:
```
itemId                                                  304404774358
title              spyderco stovepipe knife c260tip stonewash 2.7...
galleryURL         https://i.ebayimg.com/thumbs/images/g/leoAAOSw... (https://
i.ebayimg.com/thumbs/images/g/leoAAOSw...)
viewItemURL        https://www.ebay.com/itm/Spyderco-Stovepipe-Kn... (https://w
ww.ebay.com/itm/Spyderco-Stovepipe-Kn...)
postalCode                                                      287**
price_in_US                                                       420
shipping_cost                                                       0
converted_price                                                   420
benchmade                                                           0
buck                                                                0
case                                                                0
crkt                                                                0
kershaw                                                             0
leatherman                                                          0
spyderco                                                           30
victorinox                                                          0
Binary_Target                                                    True
profit                                                            383
Name: 6206, dtype: object
```

In [ ]:
```
```

In [ ]:
```
```

In [16]:
```
display(df_15.info())
display(df_20.info())
display(df_30.info())
display(df_45.info())
```

```
 11  crkt             3987 non-null   float64
 12  kershaw          3987 non-null   float64
 13  leatherman       3987 non-null   float64
 14  spyderco         3987 non-null   float64
 15  victorinox       3987 non-null   float64
 16  Binary_Target    3987 non-null   bool
 17  profit           3987 non-null   float64
dtypes: bool(1), float64(12), int64(1), object(4)
memory usage: 564.6+ KB

None
```

There is a considerbly long tail on the distrubtion of of price for this dataset. In order to clean the dataset, I have decided to drop the values that are statistically the least likely to occur. According to my dataset, the probability of selling a knife on ebay for more than $270 is less than 5 14.52. I have provided a simple formula to show the exact percentage of data dropped from the original dataset when removing the tails from the dataset in this way, and have also provided clean visualizions to show how it has seemingly imrpoved the distrubtion.

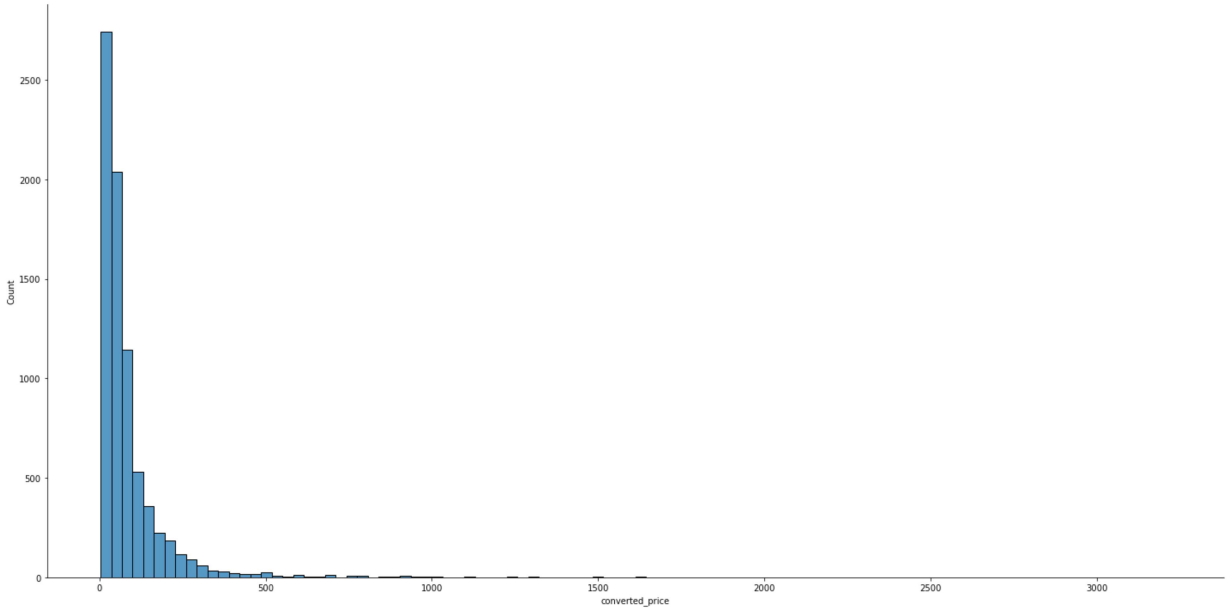
In [17]:
```
#there is five percent chance of selling a knife for less than $14.52 on ebay
df['converted_price'].quantile(.05)
```

Out[17]: 14.516000000000004

In [18]:
```
#There is a five percent chance of selling a knife for 269.15 dollars on ebay.
df['converted_price'].quantile(.95)
```

Out[18]: 269.15

In [19]:
```
sns.displot(df['converted_price'], height=10,aspect=2, bins=100)
plt.show();
```

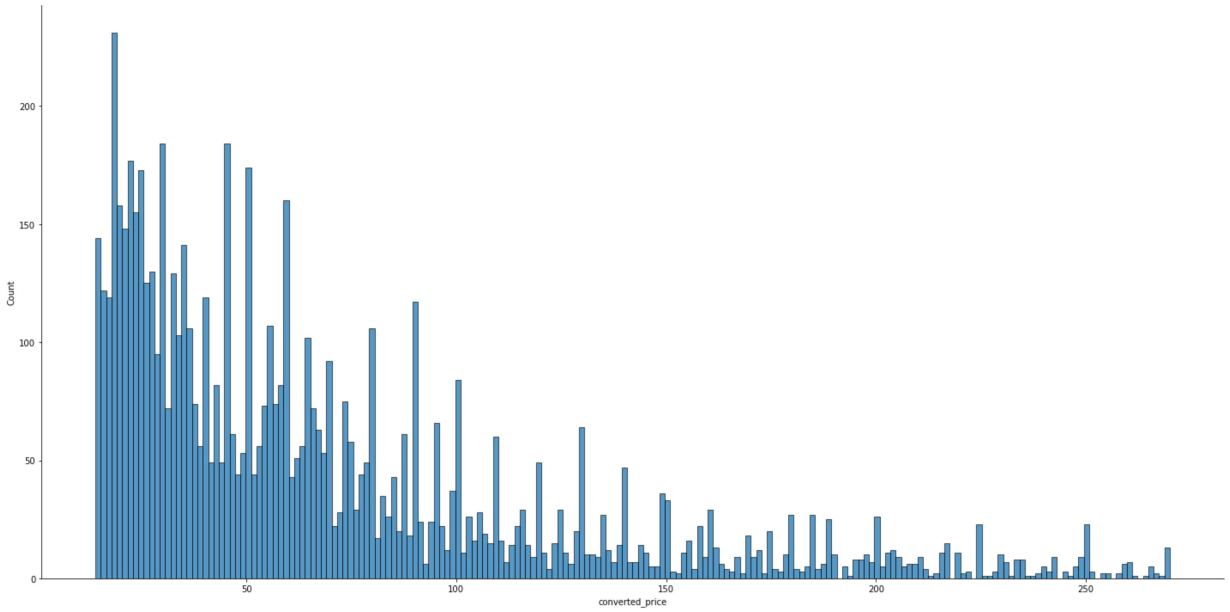


In [73]:
```
df_scrub = df[(df['converted_price'] > 14) & (df['converted_price'] < 270)]
```

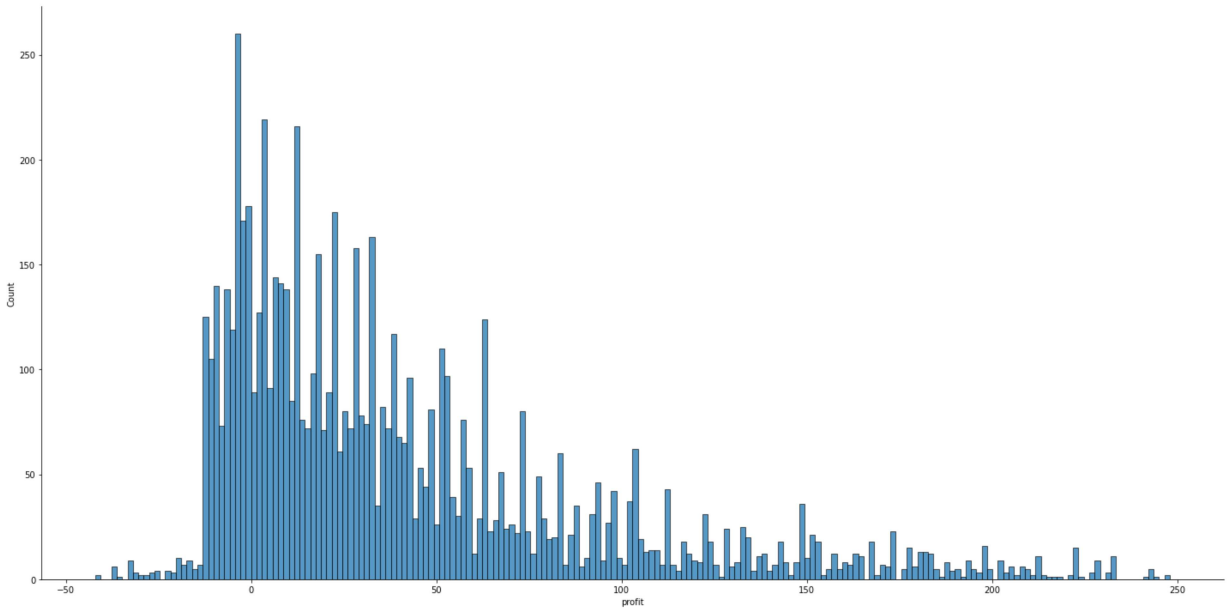
In [74]:
```
((len(df) - len(df_scrub)) / len(df)) *100
```

Out[74]: 9.296320206584893

In [75]:
```
sns.displot(df_scrub['converted_price'], height=10,aspect=2, bins=200)
plt.show();
```

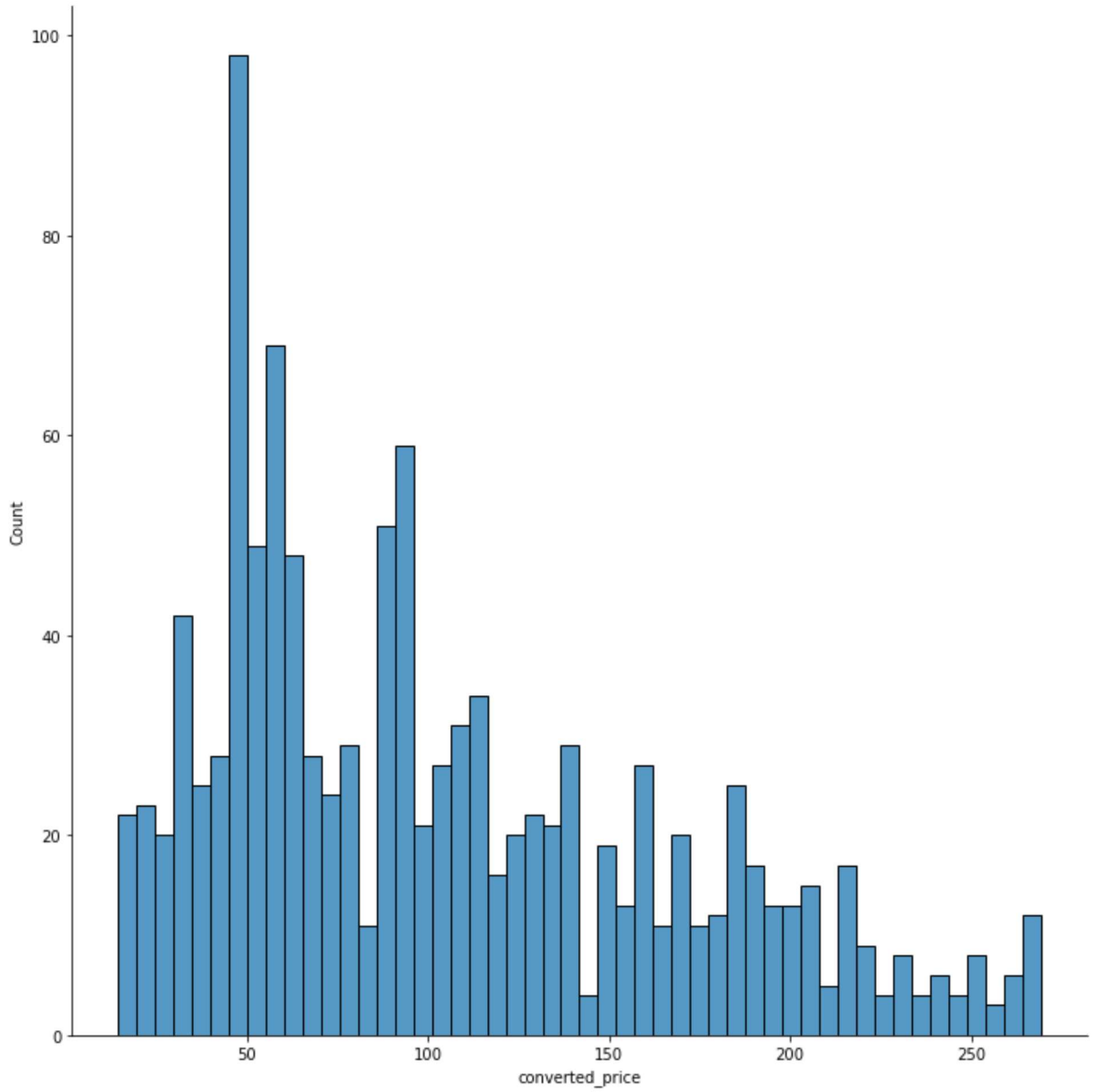


In [76]:
```
sns.displot(df_scrub['profit'], height=10,aspect=2, bins=200)
plt.show();
```

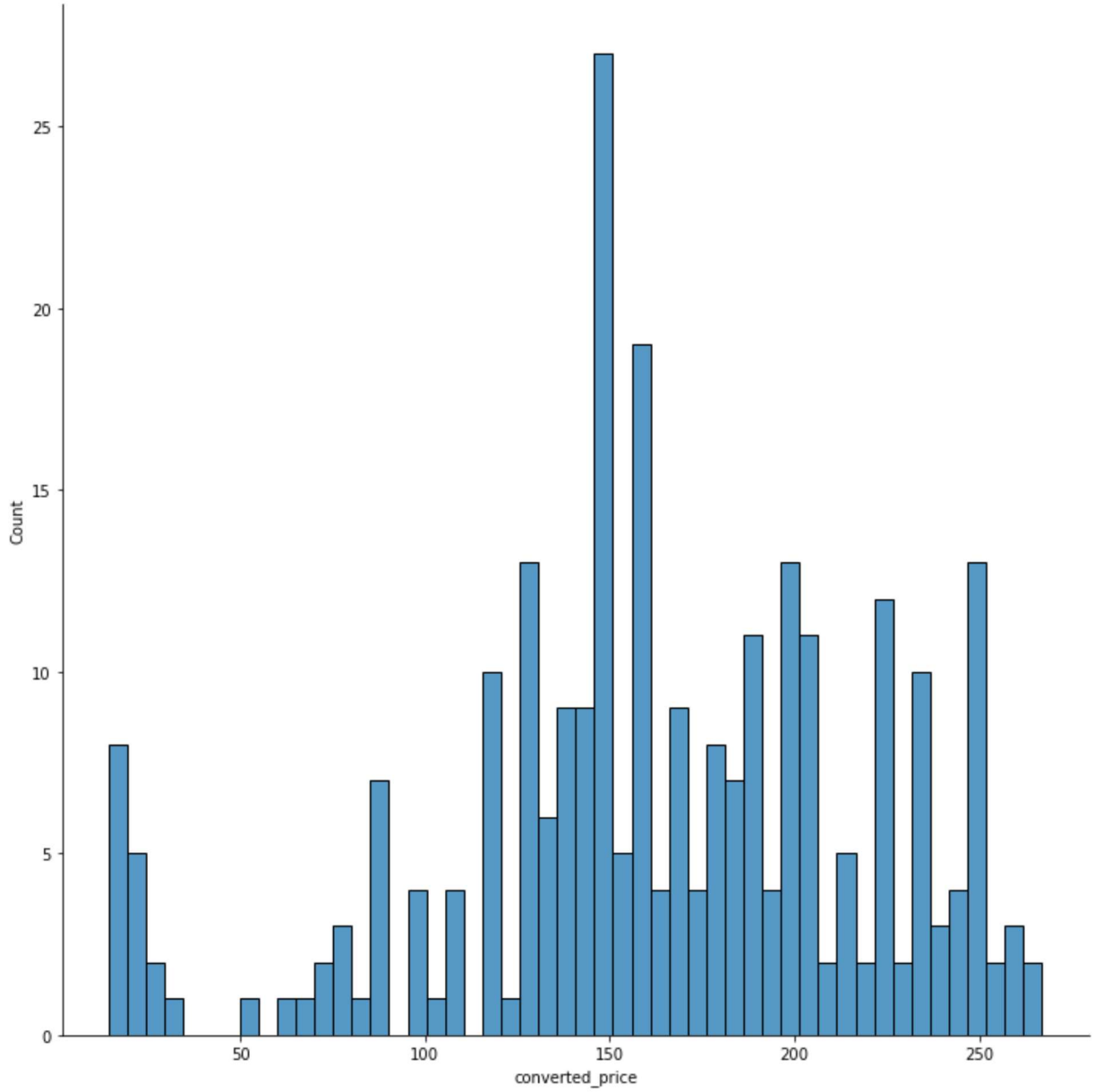


In [77]:
```
df_15 = df_scrub.loc[(df_scrub['crkt'] != 0) | (df_scrub['kershaw'] != 0)]
df_20 = df_scrub.loc[(df_scrub['buck'] != 0) | (df_scrub['case'] != 0)| (df_scrub
df_30 = df_scrub.loc[(df_scrub['leatherman'] != 0) | (df_scrub['spyderco'] != 0)]
df_45 = df_scrub.loc[df_scrub['benchmade'] != 0]
```

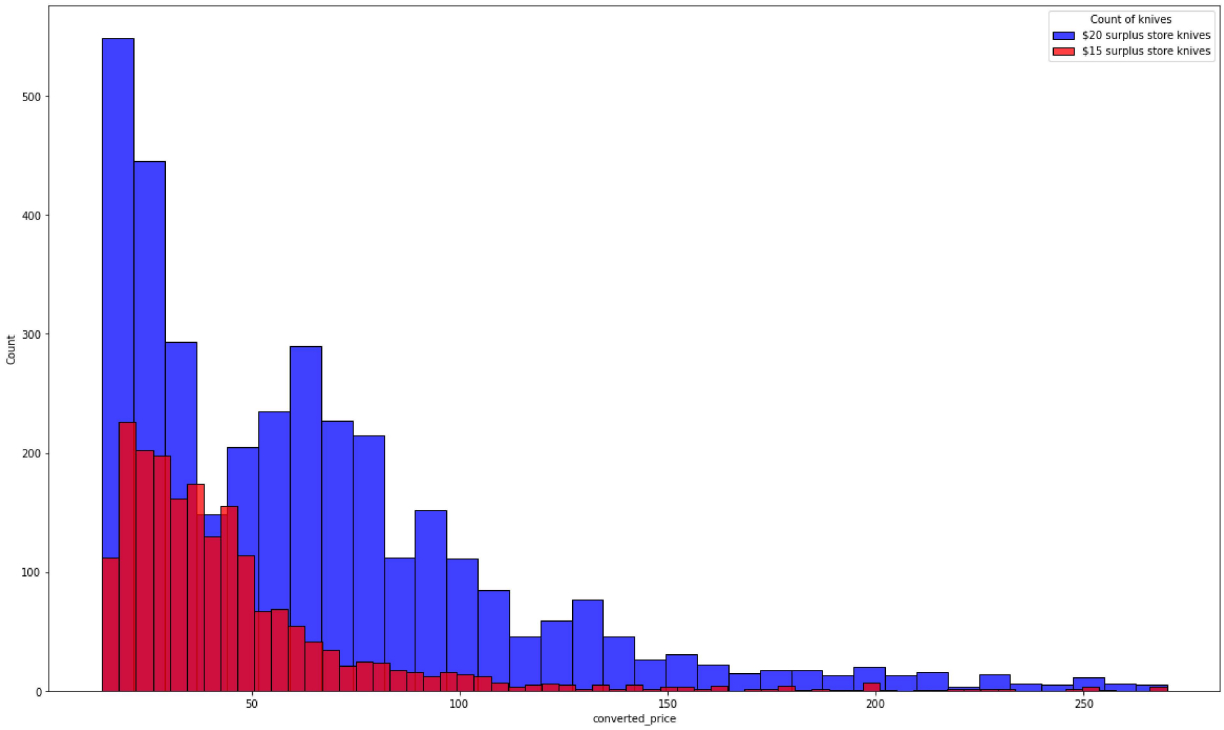
In [78]:
```
sns.displot(df_30.converted_price, height=10,aspect=1, bins=50)
plt.show();
```



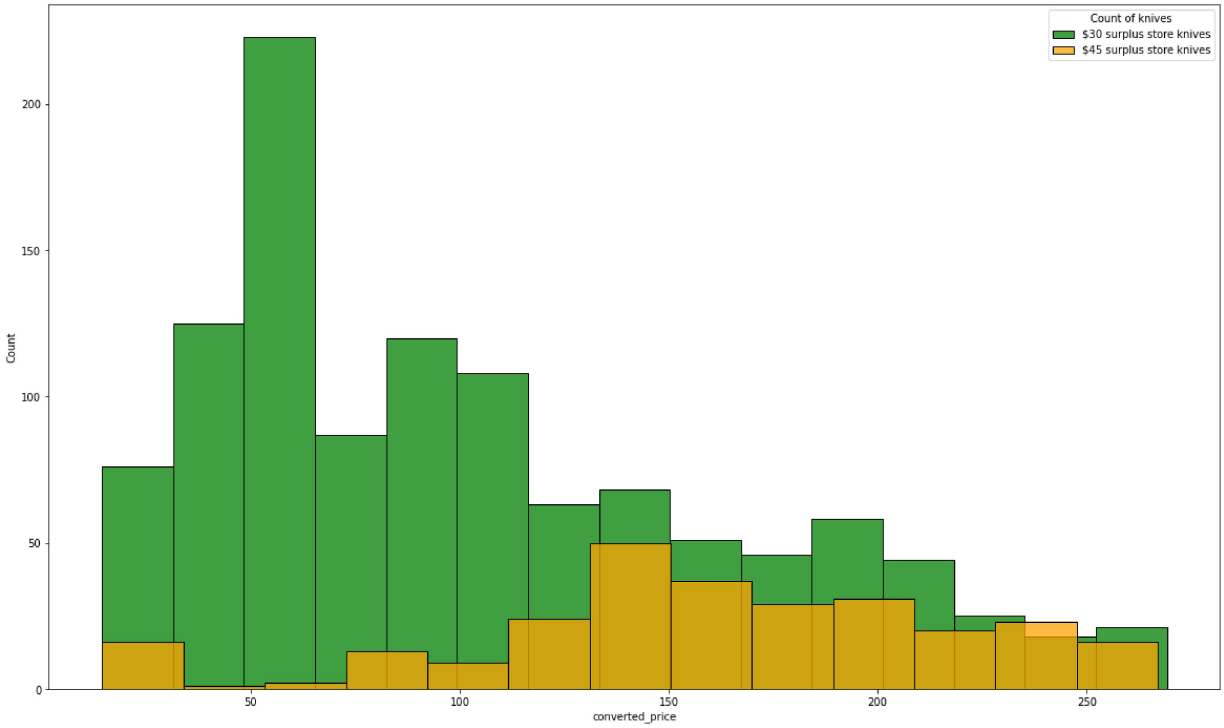
In [79]:
```
sns.displot(df_45.converted_price, height=10,aspect=1, bins=50)
plt.show();
```



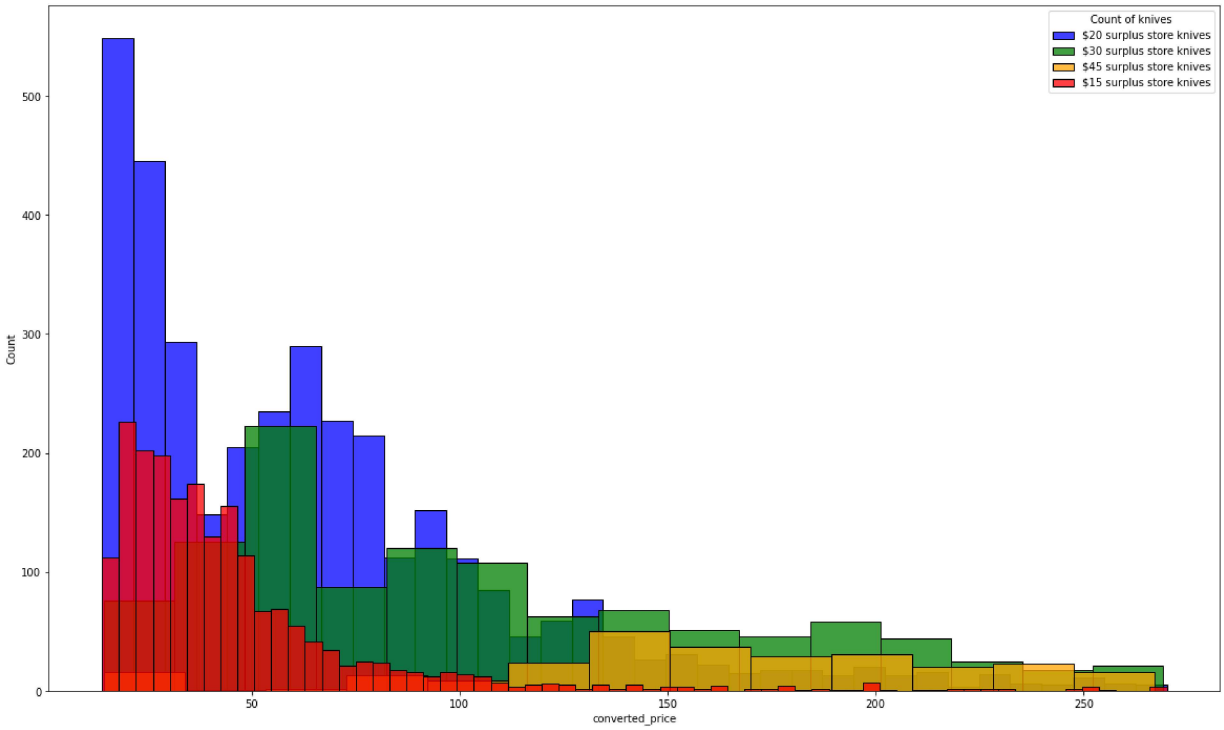
In [80]:
```
fig, ax = plt.subplots(figsize=(20,12))
sns.histplot(df_20['converted_price'], ax=ax, color='b', label='$20 surplus store
sns.histplot(df_15['converted_price'], ax=ax, thresh=.5, color='r', label='$15 su
# sns.histplot(df_30['converted_price'], ax=ax, thresh=.7, color='b')
# sns.histplot(df_45['converted_price'], ax=ax, thresh=.7, color='r')
ax.legend(title="Count of knives")
plt.show();
```



In [81]:
```
fig, ax = plt.subplots(figsize=(20,12))
# sns.histplot(df_15['converted_price'], ax=ax, thresh=.7, color='r')
# sns.histplot(df_20['converted_price'], ax=ax, thresh=.1, color='b')
sns.histplot(df_30['converted_price'], ax=ax, thresh=1, color='g', label='$30 sur
sns.histplot(df_45['converted_price'], ax=ax, thresh=.6, color='orange', label='$
ax.legend(title="Count of knives")
plt.show();
```

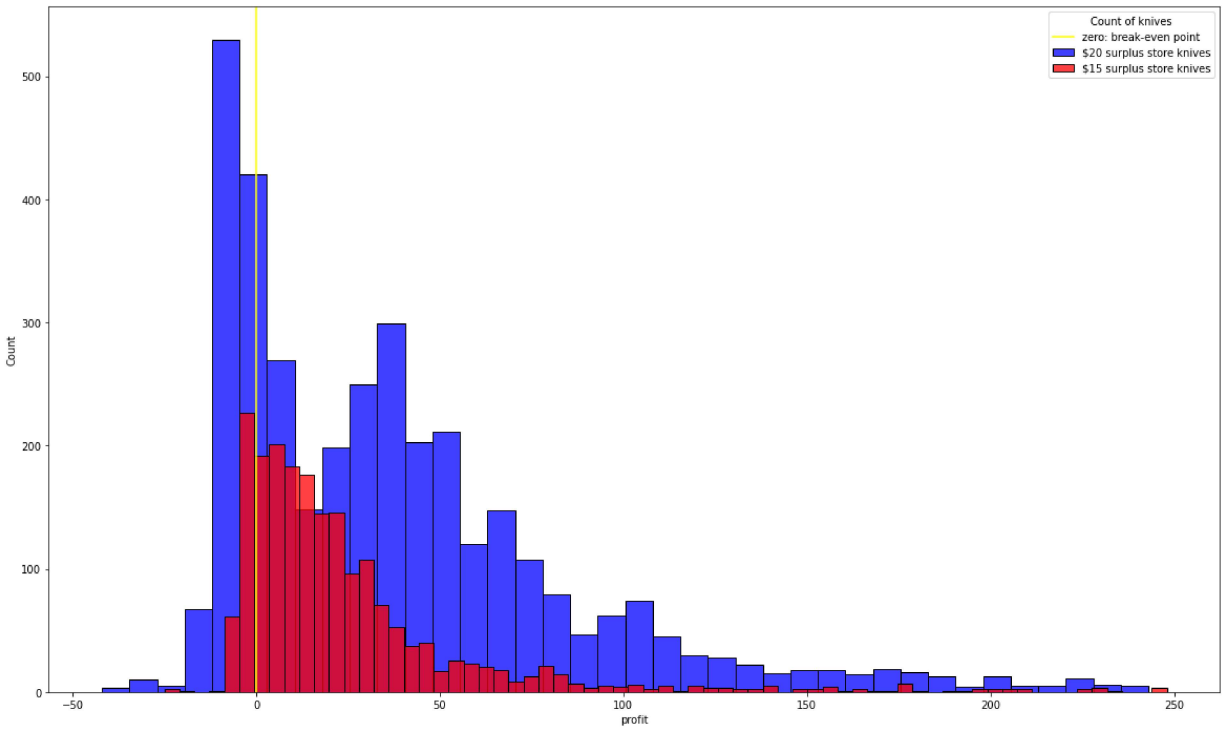


In [82]:
```
fig, ax = plt.subplots(figsize=(20,12))
sns.histplot(df_20['converted_price'], ax=ax, thresh=.1, color='b', label='$20 su
sns.histplot(df_30['converted_price'], ax=ax, thresh=.5, color='g', label='$30 su
sns.histplot(df_45['converted_price'], ax=ax, thresh=.3, color='orange', label='$
sns.histplot(df_15['converted_price'], ax=ax, thresh=.2, color='r', label='$15 su
ax.legend(title="Count of knives")
plt.show();
```

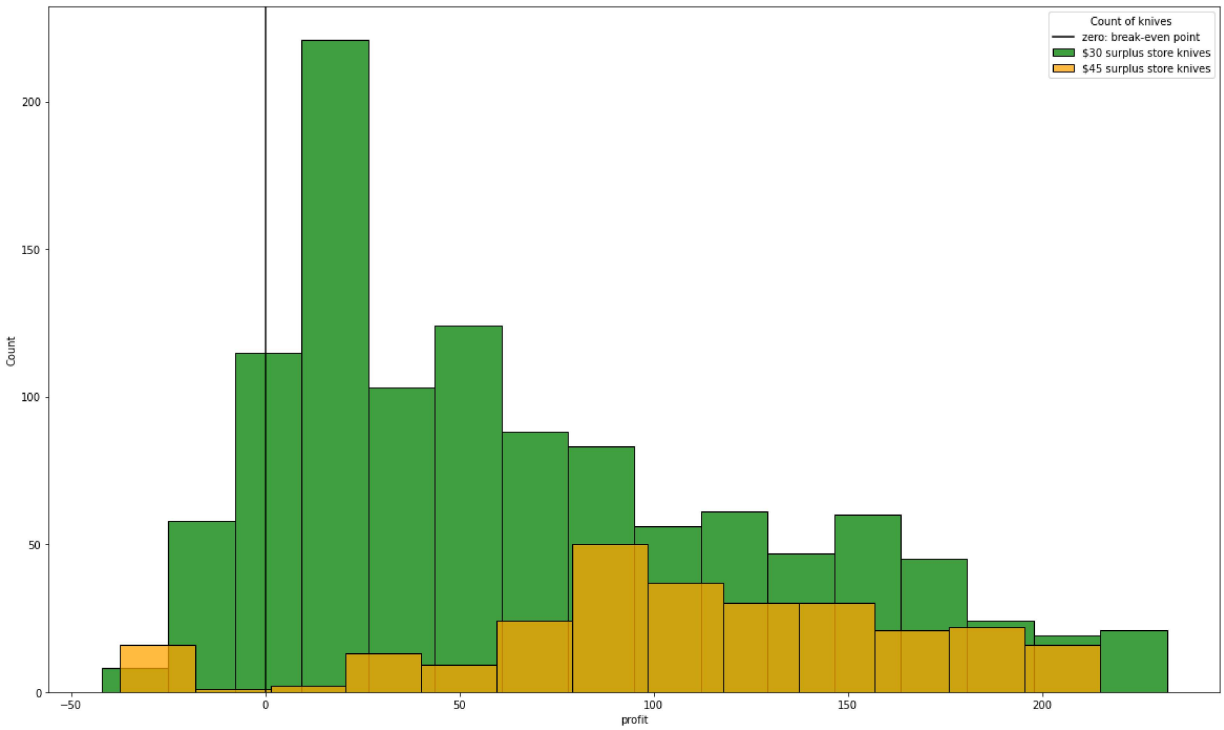


PROFIT GRAPHS

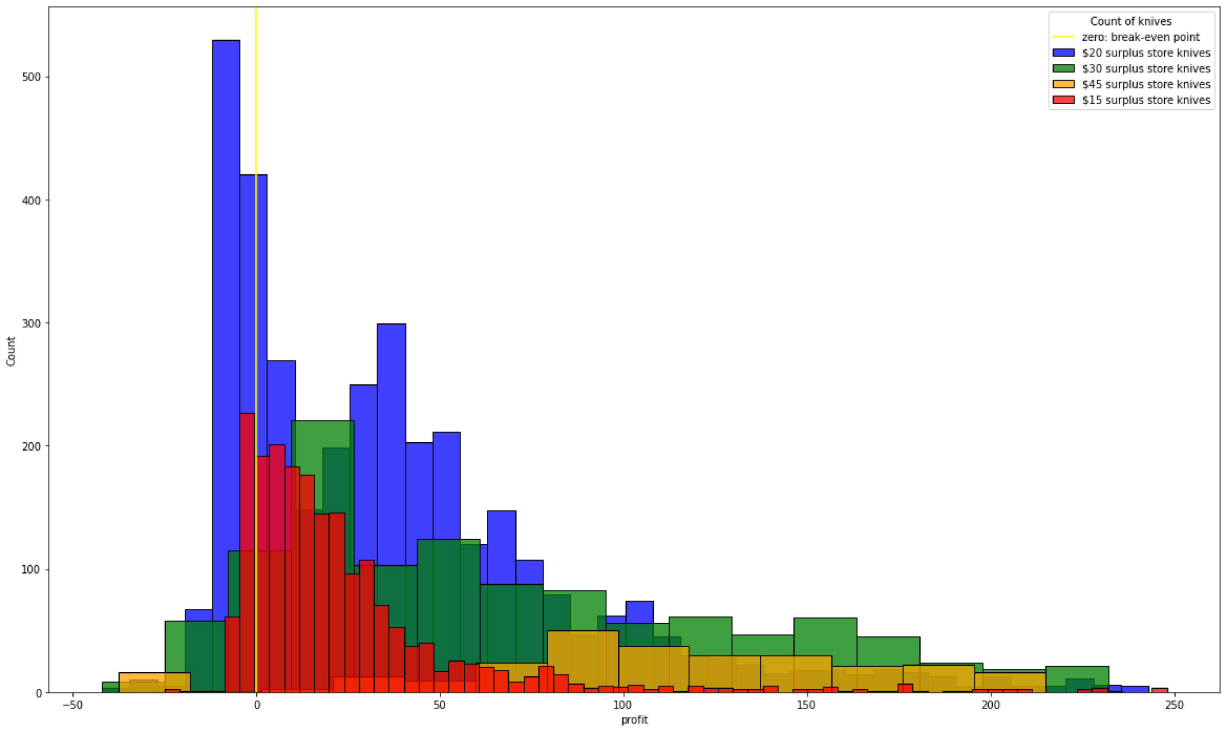
In [83]:
```
fig, ax = plt.subplots(figsize=(20,12))
sns.histplot(df_20['profit'], ax=ax, color='b', label='$20 surplus store knives')
sns.histplot(df_15['profit'], ax=ax, thresh=.5, color='r', label='$15 surplus sto
plt.axvline(x = 0, color = 'yellow', label = 'zero: break-even point')
# sns.histplot(df_30['converted_price'], ax=ax, thresh=.7, color='b')
# sns.histplot(df_45['converted_price'], ax=ax, thresh=.7, color='r')
ax.legend(title="Count of knives")
plt.show();
```



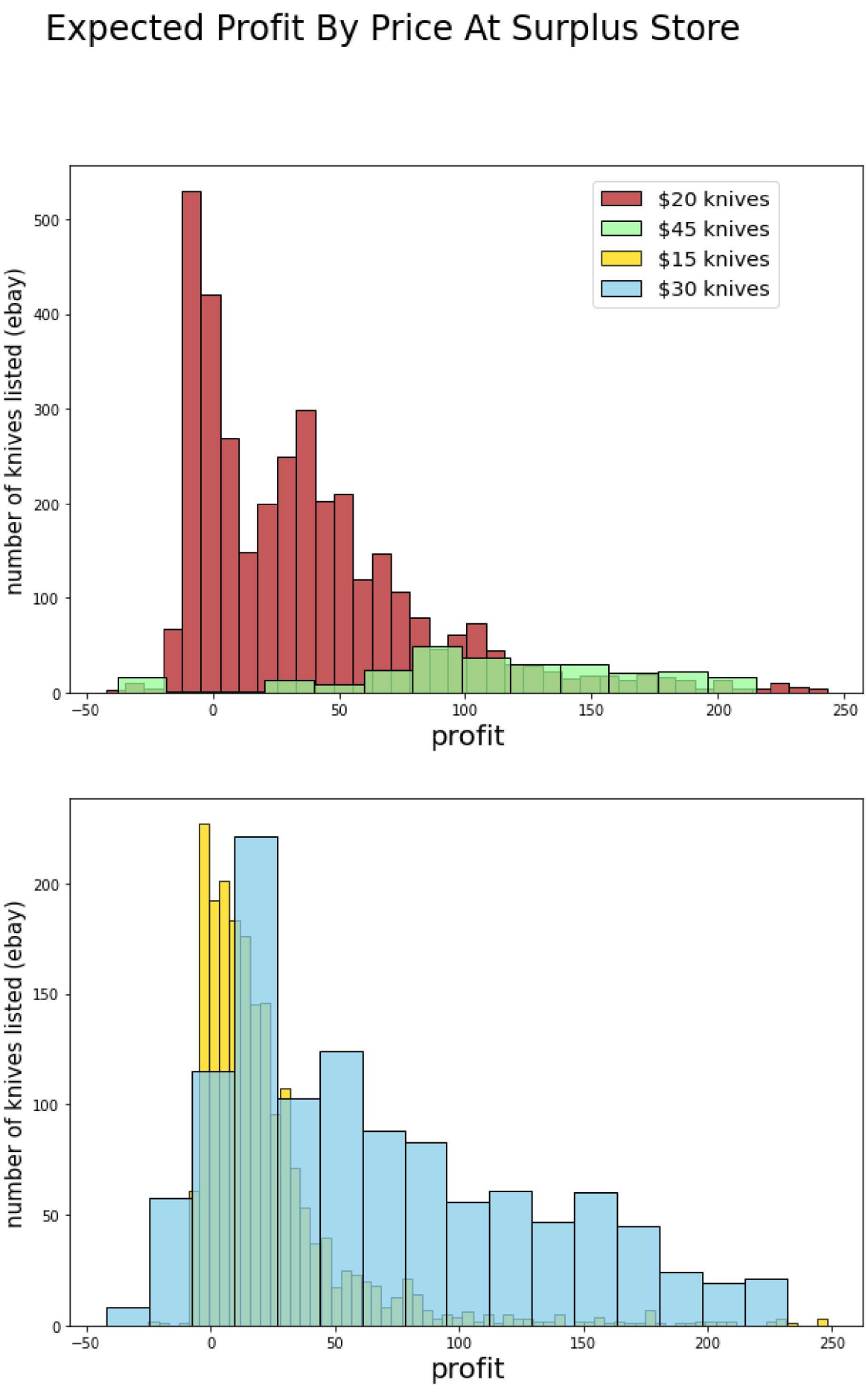
In [84]:
```
fig, ax = plt.subplots(figsize=(20,12))
# sns.histplot(df_15['converted_price'], ax=ax, thresh=.7, color='r')
# sns.histplot(df_20['converted_price'], ax=ax, thresh=.1, color='b')
plt.axvline(x = 0, color = 'black', label = 'zero: break-even point')
sns.histplot(df_30['profit'], ax=ax, thresh=1, color='g', label='$30 surplus stor
sns.histplot(df_45['profit'], ax=ax, thresh=.6, color='orange', label='$45 surplu
ax.legend(title="Count of knives")
plt.show();
```



In [85]:
```
fig, ax = plt.subplots(figsize=(20,12))
plt.axvline(x = 0, color = 'yellow', label = 'zero: break-even point')
sns.histplot(df_20['profit'], ax=ax, thresh=.1, color='b', label='$20 surplus sto
sns.histplot(df_30['profit'], ax=ax, thresh=.5, color='g', label='$30 surplus sto
sns.histplot(df_45['profit'], ax=ax, thresh=.3, color='orange', label='$45 surplu
sns.histplot(df_15['profit'], ax=ax, thresh=.2, color='r', label='$15 surplus sto
ax.legend(title="Count of knives")
plt.show();
```



In [86]:
```
fig, (ax1, ax2) = plt.subplots(figsize=(10,15), nrows=2)

sns.histplot(df_15['profit'], ax=ax2, color='gold')
sns.histplot(df_20['profit'], ax=ax1, color='firebrick')
sns.histplot(df_30['profit'], ax=ax2,  color='skyblue')
sns.histplot(df_45['profit'], ax=ax1, color='palegreen')

fig.suptitle("Expected Profit By Price At Surplus Store", fontsize=24, x=0.44)
ax1.set_xlabel('profit', fontsize=20)
ax1.set_ylabel('number of knives listed (ebay)', fontsize=15)
ax2.set_xlabel('profit', 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=["$20 knives", "$45 knives", "$15 knives", "$30 knives"], loc=(
plt.show();
```

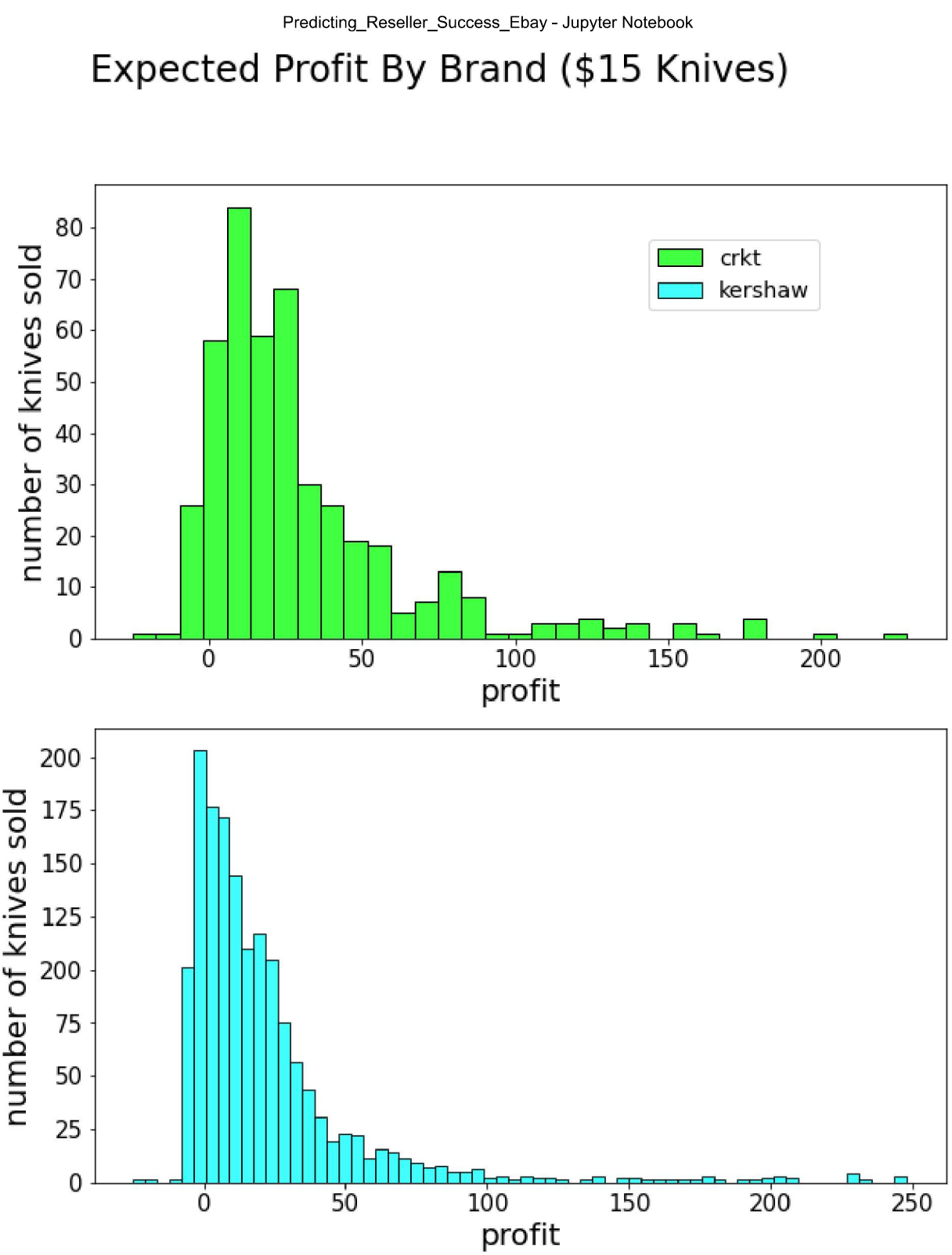

Expected Profit By Price At Surplus Store
$20 knives
$45 knives
$15 knives
$30 knives
number of knives listed (ebay)
profit
0
100
200
300
400
500
−50
0
50
100
150
200
250
number of knives listed (ebay)
profit
0
50
100
150
200
−50
0
50
100
150
200
250

In [87]:
```
df_benchmade =  df_scrub.loc[df_scrub['benchmade'] != 0]
df_buck = df_scrub.loc[df_scrub['buck'] != 0]
df_case = df_scrub.loc[df_scrub['case'] != 0]
df_crkt = df_scrub.loc[df_scrub['crkt'] != 0]
df_kershaw = df_scrub.loc[df_scrub['kershaw'] != 0]
df_leatherman = df_scrub.loc[df_scrub['leatherman'] != 0]
df_spyderco = df_scrub.loc[df_scrub['spyderco'] != 0]
df_victorinox = df_scrub.loc[df_scrub['victorinox'] != 0]
```

In [88]:
```
ax1x_labels = [0,0,50,100,150,200]
ax1y_labels = [0,10,20,30,40,50,60,70,80]
ax2x_labels = [0,0,50,100,150,200, 250]
ax2y_labels = [0,25,50,75,200,125,150,175,200]
```

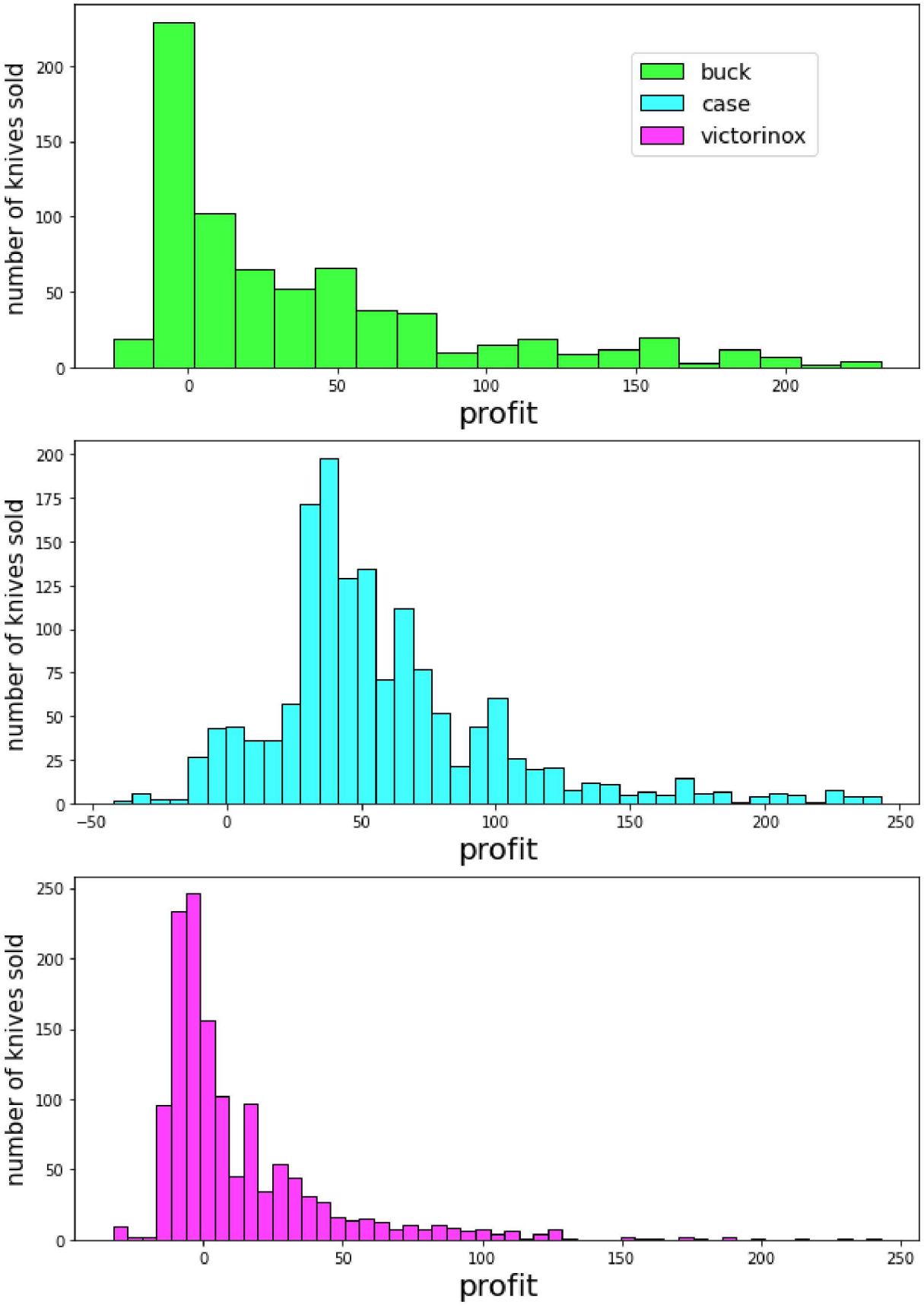
In [89]:
```
fig, (ax1, ax2) = plt.subplots(figsize=(10,12), nrows=2)
sns.histplot(df_crkt['profit'], ax=ax1, color='lime')
sns.histplot(df_kershaw['profit'], ax=ax2, color='aqua')
fig.suptitle("Expected Profit By Brand ($15 Knives)", fontsize=24, x=0.44)
ax1.set_xlabel('profit', fontsize=20)
ax1.set_ylabel('number of knives sold', fontsize=20)
ax2.set_xlabel('profit', fontsize=20)
ax2.set_ylabel('number of knives sold', fontsize=20)
ax1.set_xticklabels(ax1x_labels, fontsize=15)
ax1.set_yticklabels(ax1y_labels, fontsize=15)
ax2.set_xticklabels(ax2x_labels, fontsize=15)
ax2.set_yticklabels(ax2y_labels, fontsize=15)
# 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();
```

```
<ipython-input-89-041f8b402783>:9: UserWarning: FixedFormatter should only be u
sed together with FixedLocator
  ax1.set_xticklabels(ax1x_labels, fontsize=15)
<ipython-input-89-041f8b402783>:10: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax1.set_yticklabels(ax1y_labels, fontsize=15)
<ipython-input-89-041f8b402783>:11: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax2.set_xticklabels(ax2x_labels, fontsize=15)
<ipython-input-89-041f8b402783>:12: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax2.set_yticklabels(ax2y_labels, fontsize=15)
```

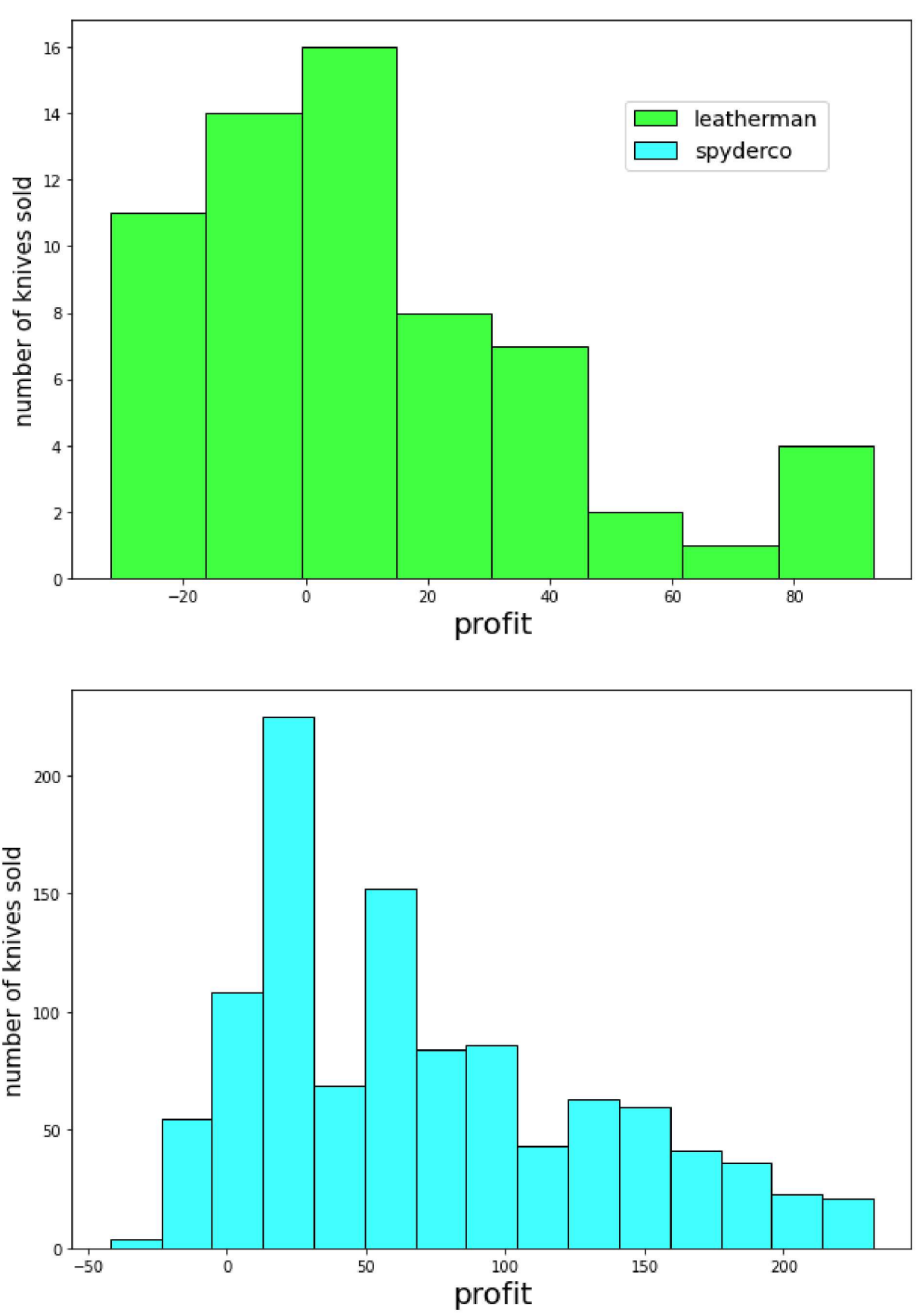

Expected Profit By Brand ($15 Knives)
number of knives sold
profit
crkt
kershaw
0
10
20
30
40
50
60
70
80
0
50
100
150
200
number of knives sold
profit
0
25
50
75
200
125
150
175
200
0
50
100
150
200
250

In [90]:
```
fig, (ax1, ax2, ax3) = plt.subplots(figsize=(10,15), nrows=3)
sns.histplot(df_buck['profit'], ax=ax1, color='lime')
sns.histplot(df_case['profit'], ax=ax2, color='aqua')
sns.histplot(df_victorinox['profit'], ax=ax3, color='magenta')

fig.suptitle("Expected Profit By Brand ($20 Knives)", fontsize=24, x=0.44)
ax1.set_xlabel('profit', fontsize=20)
ax1.set_ylabel('number of knives sold', fontsize=15)
ax2.set_xlabel('profit', fontsize=20)
ax2.set_ylabel('number of knives sold', fontsize=15)
ax3.set_xlabel('profit', fontsize=20)
ax3.set_ylabel('number of knives sold', fontsize=15)

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

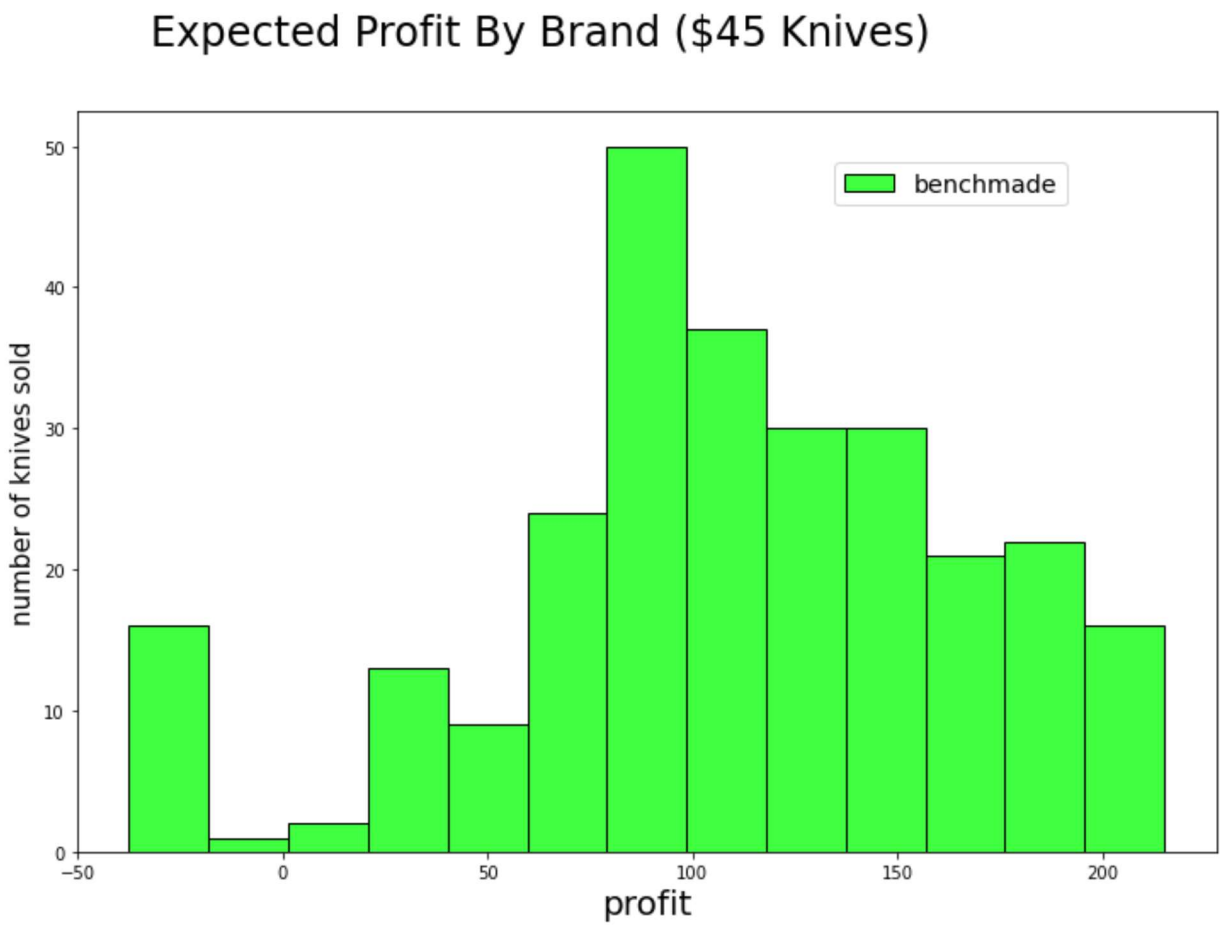
Expected Profit By Brand ($20 Knives)



In [91]:
```
fig, (ax1, ax2) = plt.subplots(figsize=(10,15), nrows=2)
sns.histplot(df_leatherman['profit'], ax=ax1, color='lime')
sns.histplot(df_spyderco['profit'], ax=ax2, color='aqua')

fig.suptitle("Expected Profit By Brand ($30 Knives)", fontsize=24, x=0.44)
ax1.set_xlabel('profit', fontsize=20)
ax1.set_ylabel('number of knives sold', fontsize=15)
ax2.set_xlabel('profit', fontsize=20)
ax2.set_ylabel('number of knives sold', fontsize=15)


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


Expected Profit By Brand ($30 Knives)
leatherman
spyderco
number of knives sold
profit
0
2
4
6
8
10
12
14
16
−20
0
20
40
60
80
number of knives sold
profit
0
50
100
150
200
−50
0
50
100
150
200

In [92]:
```
fig, ax = plt.subplots(figsize=(12,8))
sns.histplot(df_benchmade['profit'], ax=ax, color='lime')


fig.suptitle("Expected Profit By Brand ($45 Knives)", fontsize=24, x=0.44)
ax.set_xlabel('profit', fontsize=20)
ax.set_ylabel('number of knives sold', 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 [93]:
```
def trial_run_45_15(trials):
    thousand_45 = (((df_45.sample(22)['profit'].sum())*trials)/trials)
    thousand_15 = (((df_15.sample(66)['profit'].sum())*trials)/trials)
    difference = thousand_45 - thousand_15
    print(f'The average price after buying 22 $45 dollar knives {trials} number o
    print(f'The average price after buying 66 $15 dollar knives {trials} number o
    print(f'The difference between the two numbers is: {difference}')



def trial_run_45_20(trials):
    thousand_45 = (((df_45.sample(22)['profit'].sum())*trials)/trials)
    thousand_20 = (((df_20.sample(50)['profit'].sum())*trials)/trials)
    difference = thousand_45 - thousand_20
    print(f'The average price after buying 22 $45 dollar knives {trials} number o
    print(f'The average price after buying 66 $15 dollar knives {trials} number o
    print(f'The difference between the two numbers is: {difference}')



def trial_run_45_30(trials):
    thousand_45 = (((df_45.sample(22)['profit'].sum())*trials)/trials)
    thousand_30 = (((df_30.sample(33)['profit'].sum())*trials)/trials)
    difference = thousand_45 - thousand_30
    print(f'The average price after buying 22 $45 dollar knives {trials} number o
    print(f'The average price after buying 33 $30 dollar knives {trials} number o
    print(f'The difference between the two numbers is: {difference}')
```

In [94]:
```
df_CNN_regression = df_scrub.drop(df_scrub.columns[4:17].values, axis=1)
```

In [95]:
```
df_CNN_regression.drop('index', axis=1, inplace=True)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-95-e47f6d683c24> in <module>
----> 1 df_CNN_regression.drop('index', axis=1, inplace=True)

~\anaconda3\envs\learn-env\lib\site-packages\pandas\core\frame.py in drop(self,
labels, axis, index, columns, level, inplace, errors)
   4161                 weight  1.0     0.8
   4162         """
-> 4163         return super().drop(
   4164             labels=labels,
   4165             axis=axis,

~\anaconda3\envs\learn-env\lib\site-packages\pandas\core\generic.py in drop(sel
f, labels, axis, index, columns, level, inplace, errors)
   3885         for axis, labels in axes.items():
   3886             if labels is not None:
-> 3887                 obj = obj._drop_axis(labels, axis, level=level, errors=
errors)
   3888
   3889         if inplace:

~\anaconda3\envs\learn-env\lib\site-packages\pandas\core\generic.py in _drop_ax
is(self, labels, axis, level, errors)
   3919                 new_axis = axis.drop(labels, level=level, errors=errors
)
   3920             else:
-> 3921                 new_axis = axis.drop(labels, errors=errors)
   3922             result = self.reindex(**{axis_name: new_axis})
   3923

~\anaconda3\envs\learn-env\lib\site-packages\pandas\core\indexes\base.py in dro
p(self, labels, errors)
   5280         if mask.any():
   5281             if errors != "ignore":
-> 5282                 raise KeyError(f"{labels[mask]} not found in axis")
   5283             indexer = indexer[~mask]
   5284         return self.delete(indexer)

KeyError: "['index'] not found in axis"
```

In [96]:
```
# 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 [25]:
```
df_45 = df_scrub.loc[df_scrub['benchmade'] != 0]
```

In [26]:
```
top_benchmade_index = df_45.sort_values(by=['profit'], ascending=False).index
```

In [ ]:
```

```

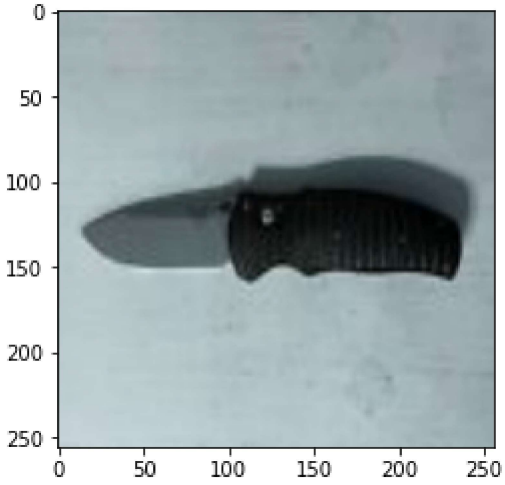
In [27]:
```
import tensorflow as tf
from tensorflow import keras
from keras.models import load_model
from keras.preprocessing import image
import numpy as np
import cv2
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras.layers import Input, Dropout, Conv2D, Dense, Flatten, Globa

img_array = cv2.imread('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 [28]:
```
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 [29]:
```
top_benchmade_index[:50]
```

Out[29]:
```
Int64Index([2850, 4497, 5857, 7371, 6974, 6786, 5049, 3722, 6837, 2072, 4641,
            4937,  563, 7068, 4739, 4925, 2087, 7073, 3810,  745, 3302,  864,
            2536, 1944,  824, 4365, 6461, 2428, 6787, 4778, 2253, 6796, 2349,
            2665, 6716, 6810, 5512,  286, 4976, 1502,  797, 5229, 6067, 3957,
             965, 6931, 1755, 1025, 1634, 4332],
           dtype='int64')
```

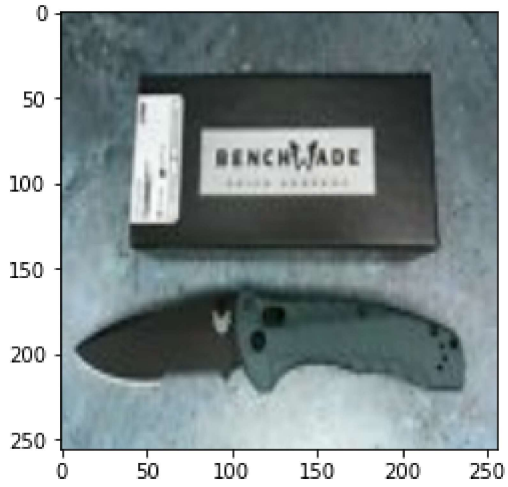
In [30]:
```
image_checker(6158)
```



In [31]:
```
image_checker(2286)
```

In [32]:
```
image_checker(1879)
```

In [33]:
```
image_checker(4326)
```



In [34]:
```
image_checker(6094)
```

In [35]:
```
#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 [36]:
```
df_CNN_regression['mean_profit']= (df_CNN_regression['profit']/df_CNN_regression|
```

In [37]:
```
df_CNN_regression['mean_profit'].describe()
```

Out[37]:
```
count    7025.000000
mean        1.000000
std         1.236768
min        -1.016331
25%         0.092811
50%         0.628409
75%         1.498515
max         5.993817
Name: mean_profit, dtype: float64
```

In [42]:
```
X = np.array(image_list)
```

In [43]:
```
y=  df_CNN_regression['mean_profit']
```

In [45]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.6, test_si
```

In [46]:
```
X.shape
```

Out[46]:
```
(7025, 256, 256, 3)
```

In [47]:
```
y.shape
```

Out[47]:
```
(7025,)
```

In [48]:
```
print("Xtrain:", X_train.shape)
print("y_train:", y_train.shape)
print("X_test:", X_test.shape)
print("y_test:", y_test.shape)
```

```
Xtrain: (4215, 256, 256, 3)
y_train: (4215,)
X_test: (2810, 256, 256, 3)
y_test: (2810,)
```

In [49]:
```
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: (4215, 256, 256, 3)
y_train: (4215,)
X_test: (1405, 256, 256, 3)
y_test: (1405,)
X_val: (1405, 256, 256, 3)
y_val: (1405,)
```

In [51]:
```
from keras import models
from keras import layers
```

In [ ]:
```

```

In [53]:
```
#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))
```

```
Epoch 1/30
132/132 [==============================] - 894s 7s/step - loss: 149.3835 - m
se: 149.3835 - val_loss: 61.2736 - val_mse: 61.2736
Epoch 2/30
```

```
132/132 [==============================] - 888s 7s/step - loss: 3.4374 - ms
e: 3.4374 - val_loss: 2.6859 - val_mse: 2.6859
Epoch 3/30
132/132 [==============================] - 878s 7s/step - loss: 1.9815 - ms
e: 1.9815 - val_loss: 2.0410 - val_mse: 2.0410
Epoch 4/30
132/132 [==============================] - 872s 7s/step - loss: 1.5425 - ms
e: 1.5425 - val_loss: 1.8453 - val_mse: 1.8453
Epoch 5/30
132/132 [==============================] - 875s 7s/step - loss: 1.5017 - ms
e: 1.5017 - val_loss: 1.8376 - val_mse: 1.8376
Epoch 6/30
132/132 [==============================] - 868s 7s/step - loss: 1.5847 - ms
e: 1.5847 - val_loss: 3.1610 - val_mse: 3.1610
Epoch 7/30
132/132 [==============================] - 871s 7s/step - loss: 1.3516 - ms
e: 1.3516 - val_loss: 1.9445 - val_mse: 1.9445
Epoch 8/30
132/132 [==============================] - 880s 7s/step - loss: 1.1749 - ms
e: 1.1749 - val_loss: 1.5204 - val_mse: 1.5204
Epoch 9/30
132/132 [==============================] - 874s 7s/step - loss: 1.1254 - ms
e: 1.1254 - val_loss: 1.8990 - val_mse: 1.8990
Epoch 10/30
132/132 [==============================] - 872s 7s/step - loss: 1.0284 - ms
e: 1.0284 - val_loss: 1.4243 - val_mse: 1.4243
Epoch 11/30
132/132 [==============================] - 873s 7s/step - loss: 0.9691 - ms
e: 0.9691 - val_loss: 1.7665 - val_mse: 1.7665
Epoch 12/30
132/132 [==============================] - 878s 7s/step - loss: 0.8852 - ms
e: 0.8852 - val_loss: 1.8338 - val_mse: 1.8338
Epoch 13/30
132/132 [==============================] - 879s 7s/step - loss: 0.8020 - ms
e: 0.8020 - val_loss: 1.4744 - val_mse: 1.4744
Epoch 14/30
132/132 [==============================] - 877s 7s/step - loss: 0.8273 - ms
e: 0.8273 - val_loss: 1.4506 - val_mse: 1.4506
Epoch 15/30
132/132 [==============================] - 875s 7s/step - loss: 0.8895 - ms
e: 0.8895 - val_loss: 1.4939 - val_mse: 1.4939
Epoch 16/30
132/132 [==============================] - 877s 7s/step - loss: 0.7312 - ms
e: 0.7312 - val_loss: 1.5110 - val_mse: 1.5110
Epoch 17/30
132/132 [==============================] - 876s 7s/step - loss: 0.6838 - ms
e: 0.6838 - val_loss: 1.4923 - val_mse: 1.4923
Epoch 18/30
132/132 [==============================] - 886s 7s/step - loss: 0.6667 - ms
e: 0.6667 - val_loss: 1.4072 - val_mse: 1.4072
Epoch 19/30
132/132 [==============================] - 880s 7s/step - loss: 0.6138 - ms
e: 0.6138 - val_loss: 1.4655 - val_mse: 1.4655
Epoch 20/30
132/132 [==============================] - 880s 7s/step - loss: 0.6816 - ms
e: 0.6816 - val_loss: 2.3128 - val_mse: 2.3128
Epoch 21/30
```

```
132/132 [==============================] - 878s 7s/step - loss: 0.8456 - ms
e: 0.8456 - val_loss: 1.5624 - val_mse: 1.5624
Epoch 22/30
132/132 [==============================] - 882s 7s/step - loss: 0.6001 - ms
e: 0.6001 - val_loss: 1.4482 - val_mse: 1.4482
Epoch 23/30
132/132 [==============================] - 878s 7s/step - loss: 1.0212 - ms
e: 1.0212 - val_loss: 517.5801 - val_mse: 517.5801
Epoch 24/30
132/132 [==============================] - 879s 7s/step - loss: 334.5583 - m
se: 334.5583 - val_loss: 36.9425 - val_mse: 36.9425
Epoch 25/30
132/132 [==============================] - 882s 7s/step - loss: 5.2716 - ms
e: 5.2716 - val_loss: 4.9364 - val_mse: 4.9364
Epoch 26/30
132/132 [==============================] - 872s 7s/step - loss: 3.6606 - ms
e: 3.6606 - val_loss: 3.1195 - val_mse: 3.1195
Epoch 27/30
132/132 [==============================] - 881s 7s/step - loss: 2.8978 - ms
e: 2.8978 - val_loss: 2.8636 - val_mse: 2.8636
Epoch 28/30
132/132 [==============================] - 881s 7s/step - loss: 2.4486 - ms
e: 2.4486 - val_loss: 2.5923 - val_mse: 2.5923
Epoch 29/30
132/132 [==============================] - 883s 7s/step - loss: 2.3793 - ms
e: 2.3793 - val_loss: 2.0809 - val_mse: 2.0809
Epoch 30/30
132/132 [==============================] - 883s 7s/step - loss: 2.5303 - ms
e: 2.5303 - val_loss: 2.0061 - val_mse: 2.0061
```

In [54]:
```
results_test = model.evaluate(X_test, y_test)

#model.summary()
```

```
44/44 [==============================] - 32s 731ms/step - loss: 2.2164 - mse:
2.2164
```

In [56]:
```
df_scrub['profit'].mean()
```

Out[56]:
```
41.374303202846974
```

In [60]:
```
2.2164 * 41.374303202846974
```

Out[60]:
```
91.70200561879004
```

In [61]:
```
df_scrub.head()
```

Out[61]:

›ping_cost	converted_price	benchmade	buck	case	crkt	kershaw	leatherman	spyderco	victorino
0.0	26.84	0.0	0.0	0.0	0.0	15.0	0.0	0.0	0.
0.0	24.99	0.0	0.0	0.0	0.0	15.0	0.0	0.0	0.
0.0	89.99	45.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
0.0	39.95	0.0	0.0	0.0	0.0	15.0	0.0	0.0	0.
0.0	43.99	0.0	0.0	0.0	0.0	15.0	0.0	0.0	0.

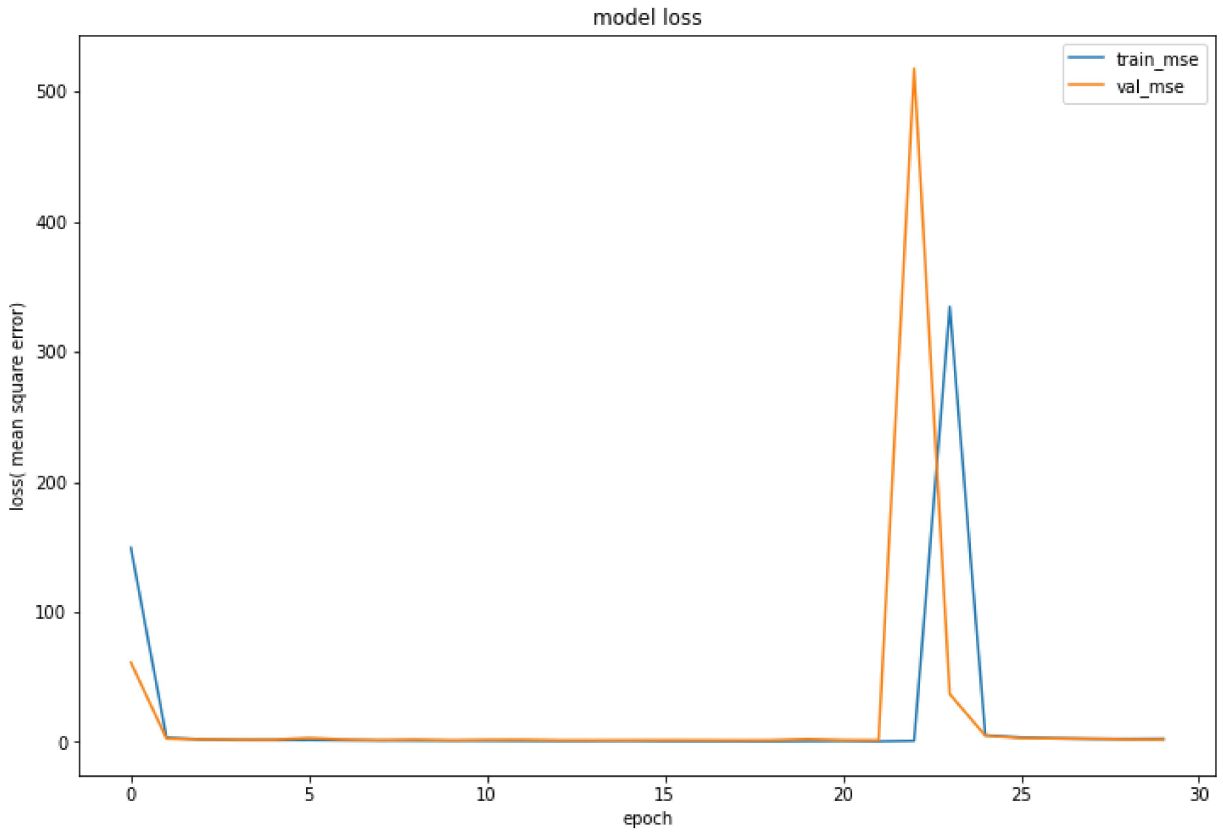
In [ ]:
```
```

In [57]:
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_9 (Conv2D)            (None, 256, 256, 32)      896
_________________________________________________________________
batch_normalization_9 (Batch (None, 256, 256, 32)      128
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 256, 256, 32)      9248
_________________________________________________________________
batch_normalization_10 (Batc (None, 256, 256, 32)      128
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 128, 128, 32)      0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 128, 128, 64)      18496
_________________________________________________________________
batch_normalization_11 (Batc (None, 128, 128, 64)      256
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 128, 128, 64)      36928
_________________________________________________________________
batch_normalization_12 (Batc (None, 128, 128, 64)      256
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 128, 128, 64)      36928
_________________________________________________________________
batch_normalization_13 (Batc (None, 128, 128, 64)      256
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 64, 64, 64)        0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 64, 64, 128)       73856
_________________________________________________________________
batch_normalization_14 (Batc (None, 64, 64, 128)       512
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 64, 64, 128)       147584
_________________________________________________________________
batch_normalization_15 (Batc (None, 64, 64, 128)       512
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 32, 32, 128)       0
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 32, 32, 256)       295168
_________________________________________________________________
batch_normalization_16 (Batc (None, 32, 32, 256)       1024
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 32, 32, 256)       590080
_________________________________________________________________
batch_normalization_17 (Batc (None, 32, 32, 256)       1024
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 16, 16, 256)       0
_________________________________________________________________
flatten_1 (Flatten)          (None, 65536)             0
_________________________________________________________________
dense_3 (Dense)              (None, 256)               16777472
_________________________________________________________________
dense_4 (Dense)              (None, 128)               32896
_________________________________________________________________
```

```
dense_5 (Dense)              (None, 1)                 129
=================================================================
Total params: 18,023,777
Trainable params: 18,021,729
Non-trainable params: 2,048
_________________________________________________________________
```

In [58]:
```
#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( mean square error)')
plt.xlabel('epoch')
plt.legend(['train_mse', 'val_mse'], loc='upper right')
plt.show();
```



In [59]:
```
model.save('my_model_batch32.h5')
```

In [ ]:

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 [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

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')
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:
```
df.isna().sum()
```

In [ ]:

In [ ]:

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.

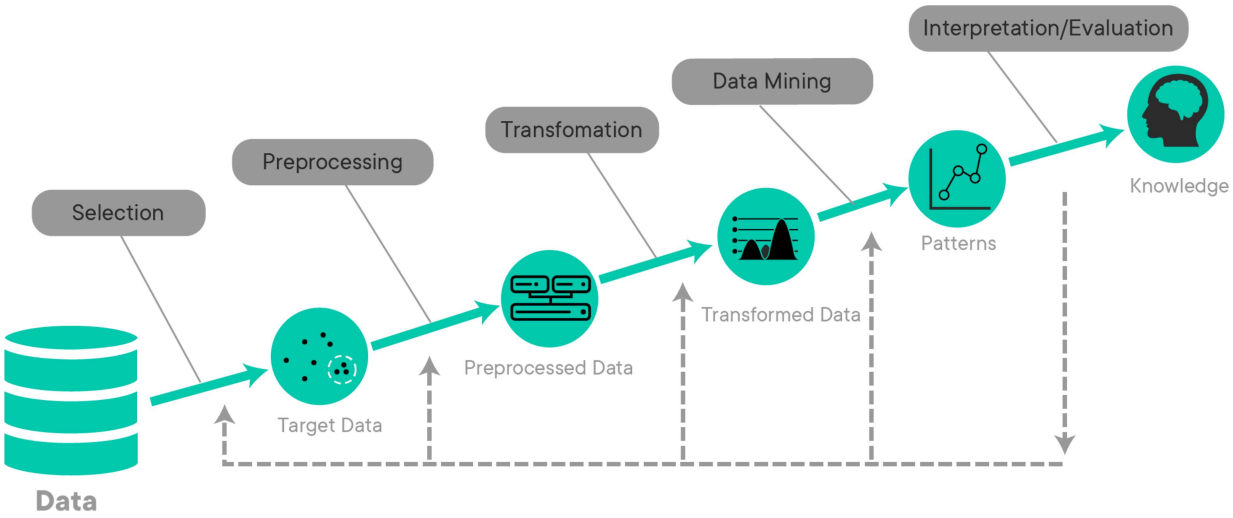
Of course, if the results are satisfactory, then we instead move onto deployment!

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.

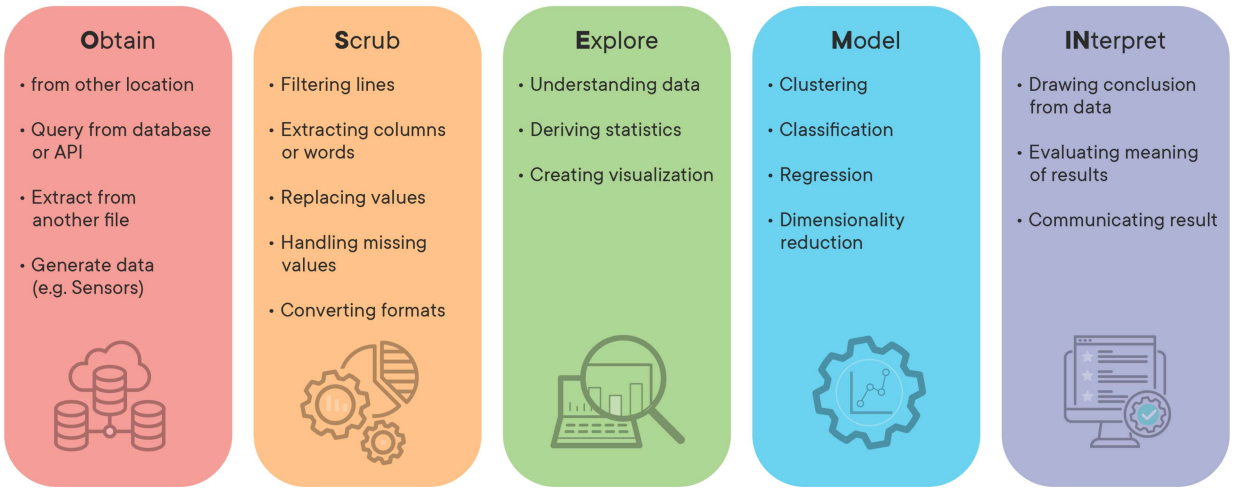
You may be wondering what Data Mining is, and how it relates to Data Science. In practice, it's just an older term that essentially means the same thing as Data Science. Dr. Piatetsky-Shapiro defines Data Mining as "the non-trivial extraction of implicit, previously unknown and potentially useful information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