MSIS 2431: Machine Learning

Team Members

Tejaswini Kambhampati

Jeffrey Monticelli

Swetha Sarma

Elyse Wu

Company Overview

- Wish is an online e-commerce platform that allows consumers to shop for a wide range of products at discounted prices
- Founded in 2011, it became one of the world's largest online marketplaces, with over 100 million monthly active users and over 500,000 merchants selling products on the platform in 2020
- The platform is known for its low prices, which are made possible through a variety of cost-cutting measures, such as working directly with manufacturers, sourcing products from around the world, and offering unbranded items
- The Wish app lends itself to scrolling and discovery, like a social network, unlike Amazon’s, in which a specific item is usually sought out
- Wish uses an algorithm that tailors product recommendations to individual users based on their gender, interests, age, location, as well as browsing and purchasing history to customize the set of products shown to each user

Project Idea

We aim to understand the factors that drive product sales by building a model that can predict the sales of products (units sold) based on features such as product type, price, shipping, color, customer rating, etc.

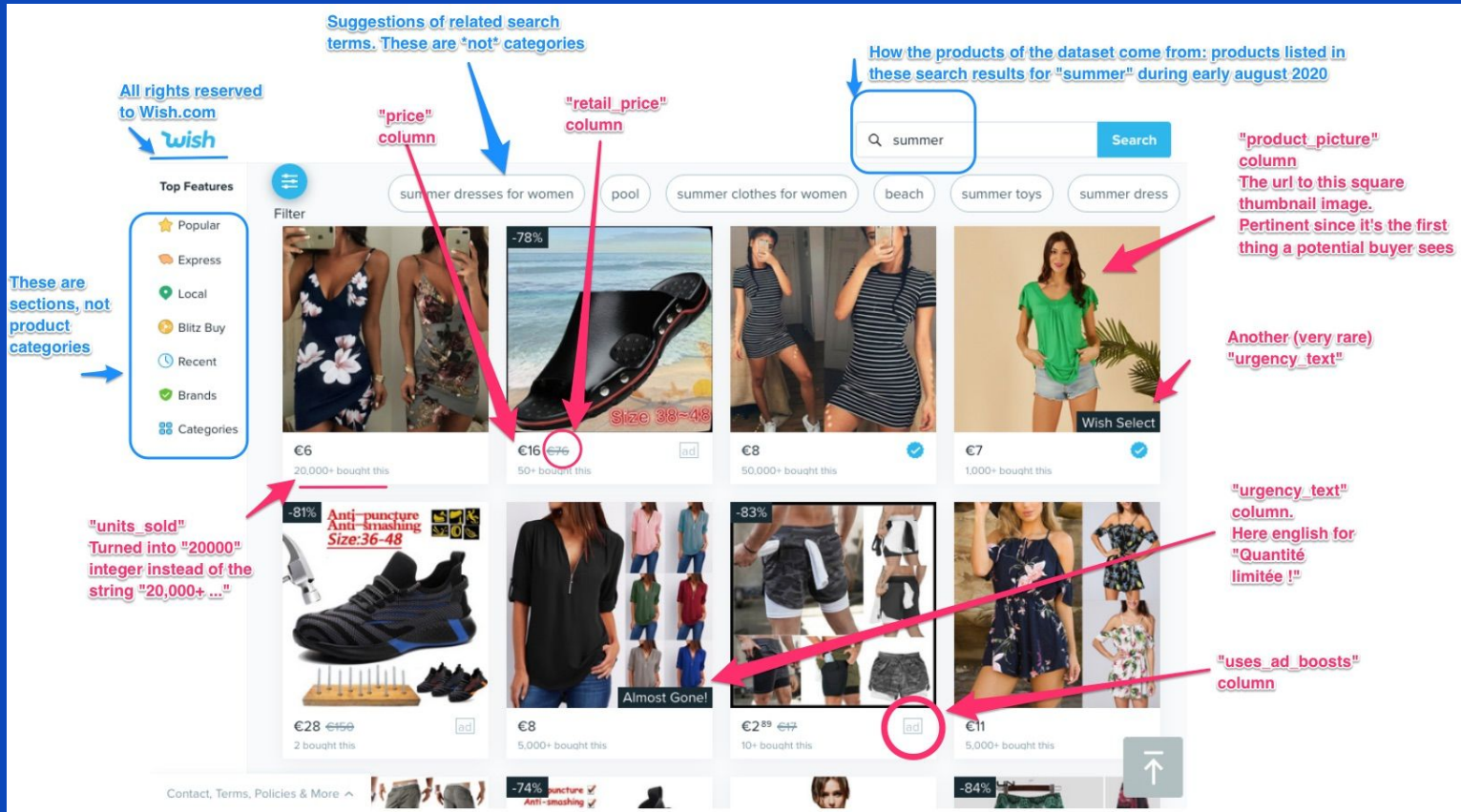
Wish Dataset

Our dataset was downloaded from Kaggle. It contains a snapshot of some of the apparel available on the Wish platform during the summer of 2020

```
In [4]: data.shape
Out[4]: (1573, 43)
```

```
data.columns
```

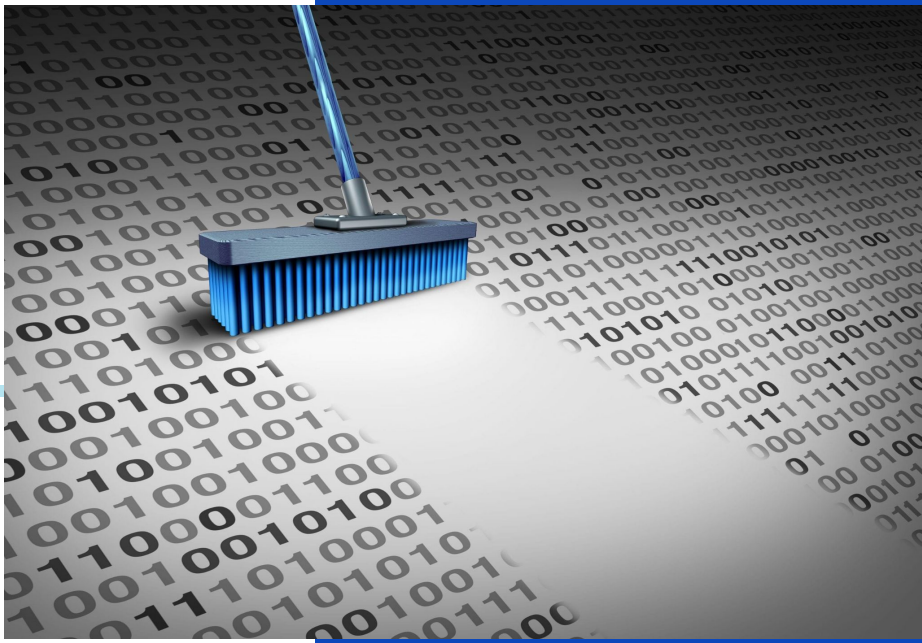
```
Index(['title', 'title_orig', 'price', 'retail_price', 'currency_buyer',
       'units_sold', 'uses_ad_boosts', 'rating', 'rating_count',
       'rating_five_count', 'rating_four_count', 'rating_three_count',
       'rating_two_count', 'rating_one_count', 'badges_count',
       'badge_local_product', 'badge_product_quality', 'badge_fast_shipping',
       'tags', 'product_color', 'product_variation_size_id',
       'product_variation_inventory', 'shipping_option_name',
       'shipping_option_price', 'shipping_is_express', 'countries_shipped_to',
       'inventory_total', 'has_urgency_banner', 'urgency_text',
       'origin_country', 'merchant_title', 'merchant_name',
       'merchant_info_subtitle', 'merchant_rating_count', 'merchant_rating',
       'merchant_id', 'merchant_has_profile_picture',
       'merchant_profile_picture', 'product_url', 'product_picture',
       'product_id', 'theme', 'crawl_month'],
      dtype='object')
```



All in all, we see that the website actually does not have the concept of categories.
Rather, it uses tags (keywords/compound keywords) sellers can set.
When the keyword is searched for (like "summer"), products related to that tag appear.

Data Cleaning

Preparing the data for transformation



Data Cleaning - Dropping Columns

```
title
title_orig
price
retail_price
currency_buyer
units_sold
uses_ad_boosts
rating
rating_count
rating_five_count
rating_four_count
rating_three_count
rating_two_count
rating_one_count
badges_count
badge_local_product
badge_product_quality
badge_fast_shipping
tags
product_color
product_variation_size_id
product_variation_inventory
shipping_option_name
shipping_option_price
shipping_is_express
countries_shipped_to
inventory_total
has_urgency_banner
urgency_text
origin_country
merchant_title
merchant_name
merchant_info_subtitle
merchant_rating_count
merchant_rating
merchant_id
merchant_has_profile_picture
merchant_profile_picture
product_url
product_picture
product_id
theme
crawl_month
```

Data Cleaning - Dropping Columns

title
title_orig
price
retail_price
currency_buyer
units_sold
uses_ad_boosts
rating
rating_count
rating_five_count
rating_four_count
rating_three_count
rating_two_count
rating_one_count
badges_count
badge_local_product
badge_product_quality
badge_fast_shipping
tags
product_color
product_variation_size_id
product_variation_inventory
shipping_option_name
shipping_option_price
shipping_is_express
countries_shipped_to
inventory_total
has_urgency_banner
urgency_text
origin_country
merchant_title
merchant_name
merchant_info_subtitle
merchant_rating_count
merchant_rating
merchant_id
merchant_has_profile_picture
merchant_profile_picture
product_url
product_picture
product_id
theme
crawl_month

43

title
title_orig
price
retail_price
units_sold
uses_ad_boosts
rating
rating_count
rating_five_count
rating_four_count
rating_three_count
rating_two_count
rating_one_count
badges_count
badge_local_product
badge_product_quality
badge_fast_shipping
tags
product_color
product_variation_size_id
product_variation_inventory
shipping_option_price
countries_shipped_to
inventory_total
has_urgency_banner
origin_country
merchant_info_subtitle
merchant_rating_count
merchant_rating
merchant_has_profile_picture

30

Data Cleaning - Dropping Columns

```
title
title_orig
price
retail_price
currency_buyer
units_sold
uses_ad_boosts
rating
rating_count
rating_five_count
rating_four_count
rating_three_count
rating_two_count
rating_one_count
badges_count
badge_local_product
badge_product_quality
badge_fast_shipping
tags
product_color
product_variation_size_id
product_variation_inventory
shipping_option_name
shipping_option_price
shipping_is_express
countries_shipped_to
inventory_total
has_urgency_banner
urgency_text
origin_country
merchant_title
merchant_name
merchant_info_subtitle
merchant_rating_count
merchant_rating
merchant_id
merchant_has_profile_picture
merchant_profile_picture
product_url
product_picture
product_id
theme
crawl_month
```

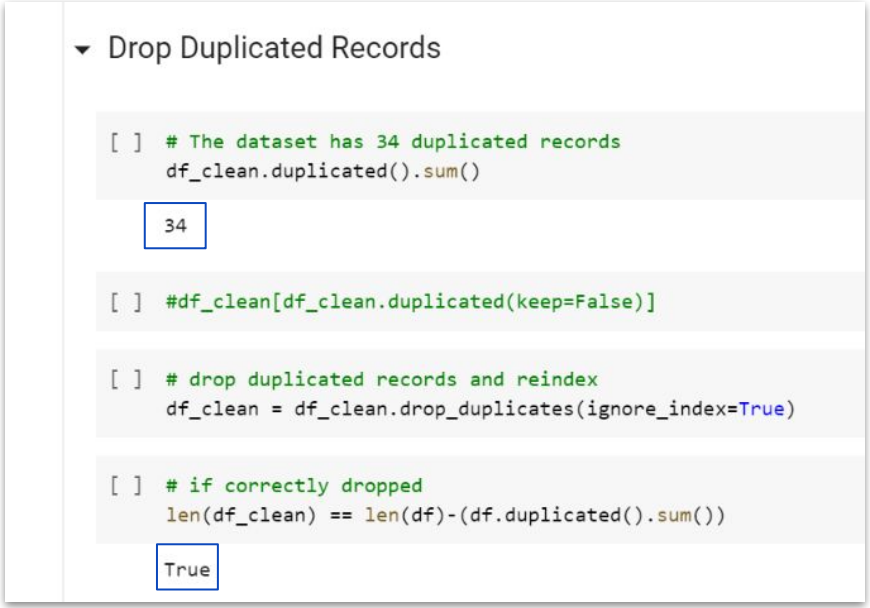
43

```
title
title_orig
price
retail_price
units_sold
uses_ad_boosts
rating
rating_count
rating_five_count
rating_four_count
rating_three_count
rating_two_count
rating_one_count
badges_count
badge_local_product
badge_product_quality
badge_fast_shipping
tags
product_color
product_variation_size_id
product_variation_inventory
shipping_option_price
countries_shipped_to
inventory_total
has_urgency_banner
origin_country
merchant_info_subtitle
merchant_rating_count
merchant_rating
merchant_has_profile_picture
```

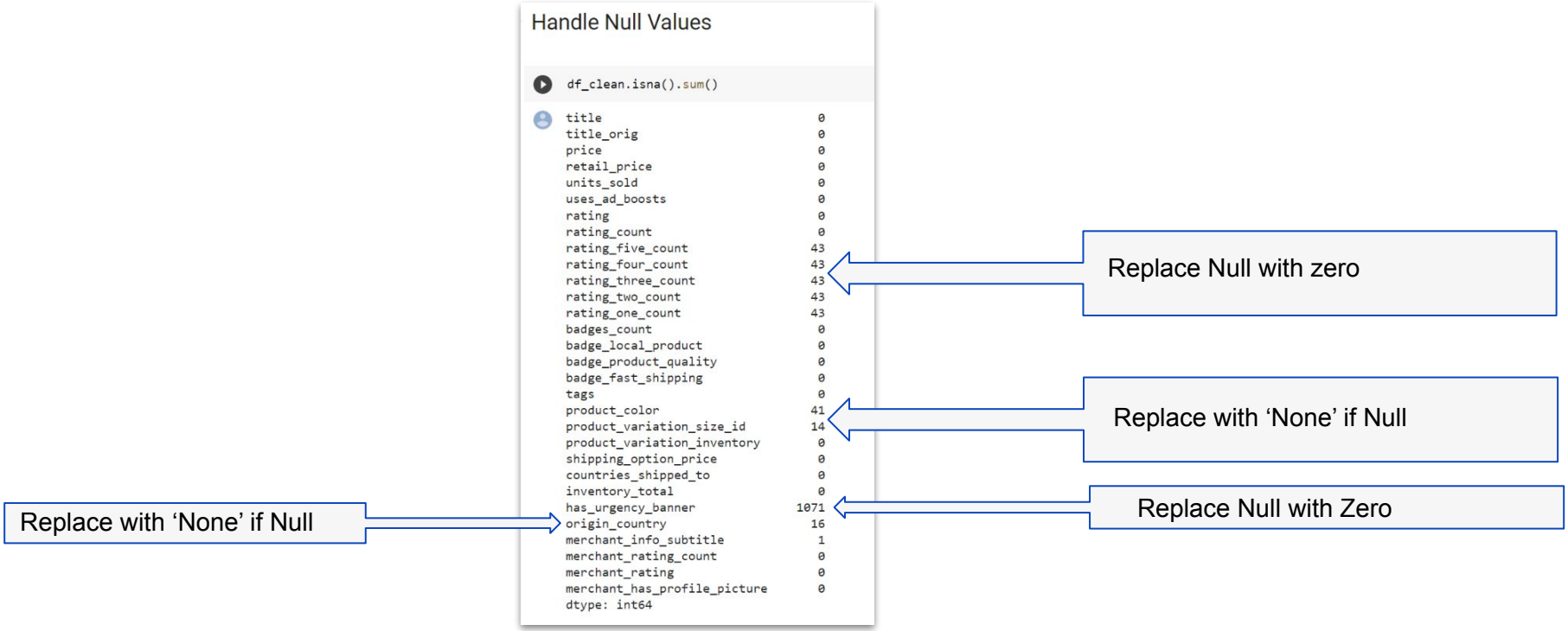
30

Column	Reason	Column	Reason
'Currency_buyer'	All Euros	'Product_id'	Not pertinent
'Shipping_option_name'	Most Standard	'Theme'	All summer
'Urgency_text'	We have binary column	'crawl_month'	Meta data not needed
'Merchant_title'	Not pertinent	shipping_is_express	Most data is Standard shipping
'Merchant_name'	Not pertinent	merchant_profile_picture	Already have binary column
'Merchant_id'	Not pertinent		
'Product_url'	Not pertinent		
'Product_picture'	Not needed for our Models		

Data Cleaning - Dropping Duplicate Rows



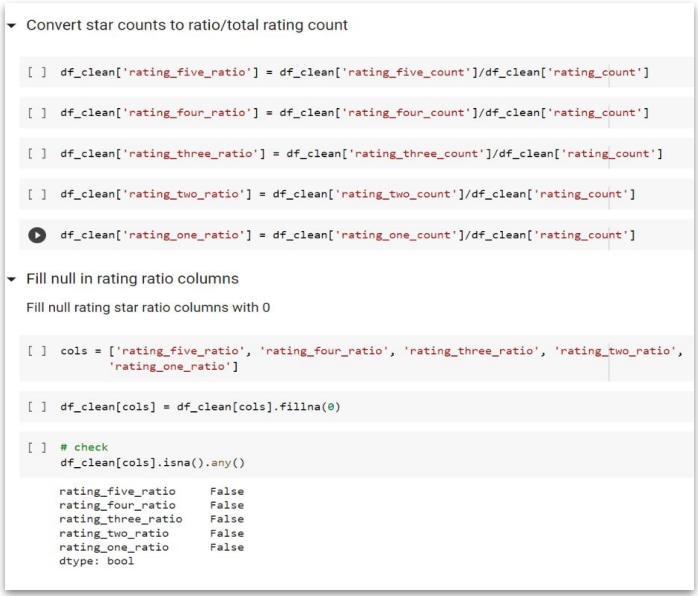
Data Cleaning - Filling NaN Values



Data Cleaning - Reformatting

We reformatted:

1. Rating_X_count → from count to proportion (five, four, three, two, & one)
    a. Dropped original star rating_X_count and included the results of their respective proportions



Convert star counts to ratio/total rating count

```
[ ] df_clean['rating_five_ratio'] = df_clean['rating_five_count']/df_clean['rating_count']

[ ] df_clean['rating_four_ratio'] = df_clean['rating_four_count']/df_clean['rating_count']

[ ] df_clean['rating_three_ratio'] = df_clean['rating_three_count']/df_clean['rating_count']

[ ] df_clean['rating_two_ratio'] = df_clean['rating_two_count']/df_clean['rating_count']

    df_clean['rating_one_ratio'] = df_clean['rating_one_count']/df_clean['rating_count']
```

Fill null in rating ratio columns

Fill null rating star ratio columns with 0

```
[ ] cols = ['rating_five_ratio', 'rating_four_ratio', 'rating_three_ratio', 'rating_two_ratio',
            'rating_one_ratio']

[ ] df_clean[cols] = df_clean[cols].fillna(0)

[ ] # check
    df_clean[cols].isna().any()

    rating_five_ratio     False
    rating_four_ratio     False
    rating_three_ratio    False
    rating_two_ratio      False
    rating_one_ratio      False
    dtype: bool
```

Data Cleaning - Reformatting

We reformatted:

2. Convert merchant_info_subtitle → extract only the percentage % value from row

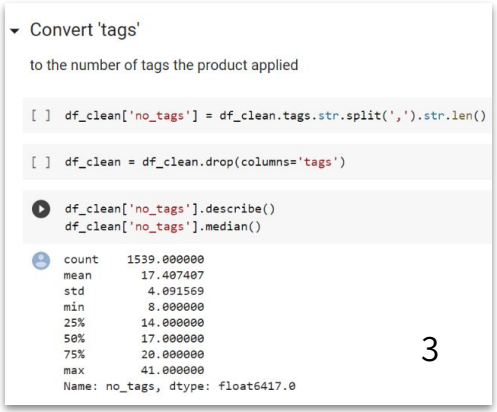
Convert format

```
[ ] import re

    # get percentage numbers

    pattern = re.compile("([0-9\.]+)%")

    df_clean['percent_positive'] = df_clean['merchant_info_subtitle'].apply(lambda x: max(pattern.findall(x), default=0))

[ ] # convert to float number
    df_clean['percent_positive'] = df_clean['percent_positive'].astype(float)/100

[ ] df_clean['percent_positive'].value_counts()

    0.00    1485
    0.83      11
    0.87       6
    0.86       6
    0.81       5
    0.88       4
    0.89       3
    0.82       3
    0.84       3
    0.91       3
    0.93       3
    0.80       2
    0.92       2
    0.90       1
    0.79       1
    0.85       1
    Name: percent_positive, dtype: int64
```

Data Cleaning - Reformat

We reformatted:

3. tags → extracted number of tags
4. title → extracted length of title
5. Price & discount→ discount column (price/retail price)



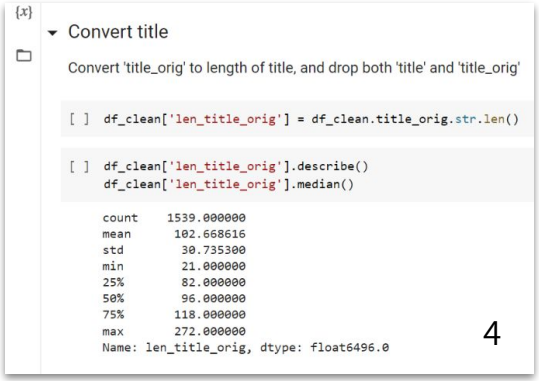
Convert 'tags'

to the number of tags the product applied

```
[ ] df_clean['no_tags'] = df_clean.tags.str.split(',').str.len()

[ ] df_clean = df_clean.drop(columns='tags')

df_clean['no_tags'].describe()
df_clean['no_tags'].median()
```

```
count    1539.000000
mean       17.407407
std         4.091569
min         8.000000
25%        14.000000
50%        17.000000
75%        20.000000
max        41.000000
Name: no_tags, dtype: float6417.0
```

3



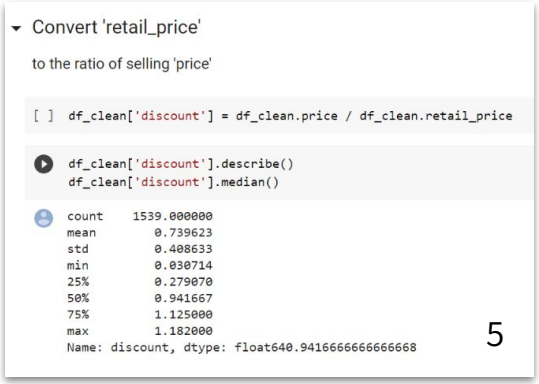
Convert title

Convert 'title_orig' to length of title, and drop both 'title' and 'title_orig'

```
[ ] df_clean['len_title_orig'] = df_clean.title_orig.str.len()

[ ] df_clean['len_title_orig'].describe()
    df_clean['len_title_orig'].median()
```

```
count    1539.000000
mean      102.668616
std        30.735300
min        21.000000
25%        82.000000
50%        96.000000
75%       118.000000
max       272.000000
Name: len_title_orig, dtype: float6496.0
```

4



Convert 'retail_price'

to the ratio of selling 'price'

```
[ ] df_clean['discount'] = df_clean.price / df_clean.retail_price

df_clean['discount'].describe()
df_clean['discount'].median()
```

```
count    1539.000000
mean        0.739623
std         0.408633
min         0.030714
25%         0.279070
50%         0.941667
75%         1.125000
max         1.182000
Name: discount, dtype: float640.9416666666666668
```

5

Data Cleaning - Reformat

We reformatted:

6. Group the same/similar size, and encode as ordinal values
7. Group similar or same product_colors together

Encode sizes

```
[ ] size_encoder = {'xxs':1.0,'xs':2.0,'s':3.0,'m':4.0,'l':5.0,'xl':6.0,'xxl':7.0,'3xl':8.0,
                    '4xl':9.0,'5xl':10.0,'6xl':11.0,'Other':12.0}

[ ] df_clean['size'] = df_clean["size"].replace(size_encoder)

    # check if repleced properly
    df_clean['size'].unique()
    df_clean['size'].value_counts()

    array([12.,  2.,  3.,  1.,  5.,  7., 10.,  8.,  6.,  9., 11.,  4.])3.0     676
    2.0     345
    12.0    312
    1.0      94
    5.0      54
    7.0      19
    6.0      19
    9.0       9
    8.0       4
    10.0      3
    4.0       3
    11.0      1
    Name: size, dtype: int64
```

6

```
[ ] df_clean['color'].unique()

    array(['white', 'green', 'Other', 'black', 'yellow', 'blue', 'gray',
           'orange', 'red', 'pink', 'multicolor', 'purple', 'silver', 'brown'],
          dtype=object)

[ ] df_clean['color'].value_counts()

    black         304
    white         251
    blue          159
    red           145
    green         134
    pink          105
    yellow        103
    Other          84
    gray           82
    multicolor     72
    purple         55
    orange         30
    brown          13
    silver          2
    Name: color, dtype: int64
```

7

Needs More Cleaning? YES!

Final check on Cleaned dataset

check data types of features

```
df_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1539 entries, 0 to 1538
Data columns (total 29 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   price                         1539 non-null   float64
 1   uses_ad_boosts                1539 non-null   int64
 2   rating                        1539 non-null   float64
 3   rating_count                  1539 non-null   int64
 4   badges_count                  1539 non-null   int64
 5   badge_local_product           1539 non-null   int64
 6   badge_product_quality         1539 non-null   int64
 7   badge_fast_shipping           1539 non-null   int64
 8   product_variation_inventory   1539 non-null   int64
 9   shipping_option_price         1539 non-null   int64
 10  countries_shipped_to          1539 non-null   int64
 11  inventory_total               1539 non-null   int64
 12  has_urgency_banner            1539 non-null   float64
 13  origin_country                1539 non-null   object
 14  merchant_rating_count         1539 non-null   int64
 15  merchant_rating               1539 non-null   float64
 16  merchant_has_profile_picture  1539 non-null   int64
 17  rating_five_ratio             1539 non-null   float64
 18  rating_four_ratio             1539 non-null   float64
 19  rating_three_ratio            1539 non-null   float64
 20  rating_two_ratio              1539 non-null   float64
 21  rating_one_ratio              1539 non-null   float64
 22  percent_positive              1539 non-null   float64
 23  no_tags                       1539 non-null   int64
 24  len_title_orig                1539 non-null   int64
 25  discount                      1539 non-null   float64
 26  size                          1539 non-null   float64
 27  color                         1539 non-null   object
 28  target                        1539 non-null   int64
dtypes: float64(12), int64(15), object(2)
```

Get Dummies

Get Dummies

```
[ ] cols = ['origin_country', 'color']

[ ] df_clean_dummy = pd.get_dummies(df_clean, prefix=cols, columns=cols, drop_first=True)

[ ] df_clean_dummy.head(3)
```

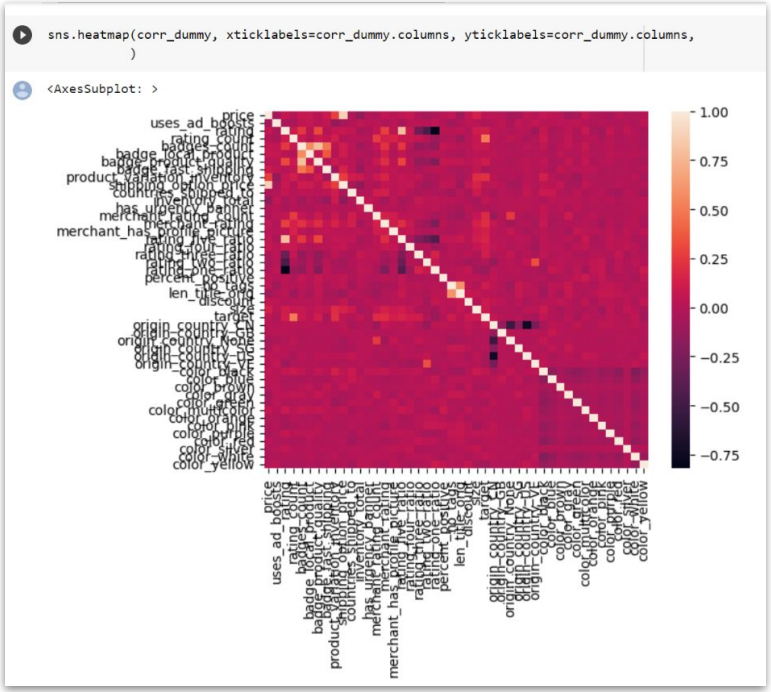
	price	uses_ad_boosts	rating	rating_count	badges_count	badge_local_product	badge_product_quality	badge_fast_shipping	product_variation_inventory	shipping_option_price	countries_shipped_to	invent
0	16.0	0	3.76	54	0	0	0	0	50	4	34	
1	8.0	1	3.45	6135	0	0	0	0	50	2	41	
2	8.0	0	3.57	14	0	0	0	0	1	3	36	

Final Dataset All Cleaned

```
# check
df_clean_dummy.info()
```

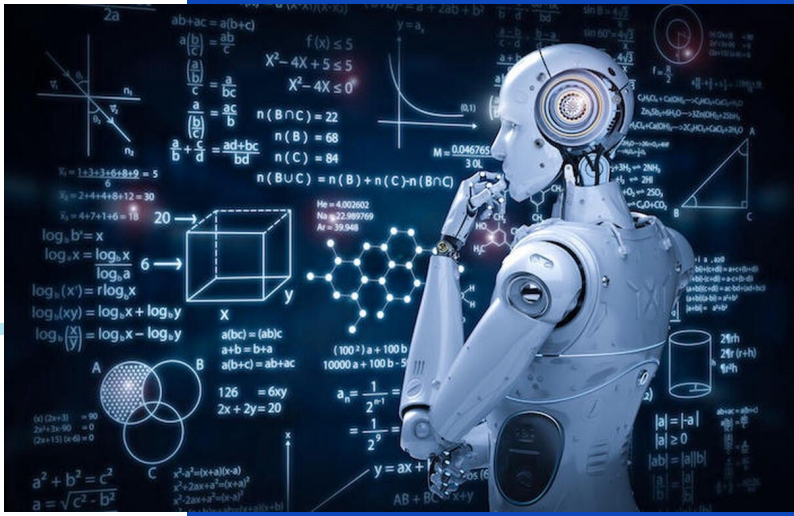
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1539 entries, 0 to 1538
Data columns (total 46 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   price                        1539 non-null   float64
 1   uses_ad_boosts               1539 non-null   int64  
 2   rating                       1539 non-null   float64
 3   rating_count                 1539 non-null   int64  
 4   badges_count                 1539 non-null   int64  
 5   badge_local_product          1539 non-null   int64  
 6   badge_product_quality        1539 non-null   int64  
 7   badge_fast_shipping          1539 non-null   int64  
 8   product_variation_inventory  1539 non-null   int64  
 9   shipping_option_price        1539 non-null   int64  
 10  countries_shipped_to         1539 non-null   int64  
 11  inventory_total              1539 non-null   int64  
 12  has_urgency_banner           1539 non-null   float64
 13  merchant_rating_count        1539 non-null   int64  
 14  merchant_rating              1539 non-null   float64
 15  merchant_has_profile_picture 1539 non-null   int64  
 16  rating_five_ratio            1539 non-null   float64
 17  rating_four_ratio            1539 non-null   float64
 18  rating_three_ratio           1539 non-null   float64
 19  rating_two_ratio             1539 non-null   float64
 20  rating_one_ratio             1539 non-null   float64
 21  percent_positive             1539 non-null   float64
 22  no_tags                      1539 non-null   int64  
 23  len_title_orig               1539 non-null   int64  
 24  discount                     1539 non-null   float64
 25  size                         1539 non-null   float64
 26  target                       1539 non-null   int64  
 27  origin_country_CN            1539 non-null   uint8  
 28  origin_country_GB            1539 non-null   uint8  
 29  origin_country_None          1539 non-null   uint8  
 30  origin_country_SG            1539 non-null   uint8  
 31  origin_country_US            1539 non-null   uint8  
 32  origin_country_VE            1539 non-null   uint8  
 33  color_black                  1539 non-null   uint8  
 34  color_blue                   1539 non-null   uint8  
 35  color_brown                  1539 non-null   uint8  
 36  color_gray                   1539 non-null   uint8  
 37  color_green                  1539 non-null   uint8  
 38  color_multicolor             1539 non-null   uint8  
 39  color_orange                 1539 non-null   uint8  
 40  color_pink                   1539 non-null   uint8  
 41  color_purple                 1539 non-null   uint8  
 42  color_red                    1539 non-null   uint8  
 43  color_silver                 1539 non-null   uint8  
 44  color_white                  1539 non-null   uint8  
 45  color_yellow                 1539 non-null   uint8  
dtypes: float64(12), int64(15), uint8(19)
memory usage: 353.3 KB
```

Correlation Matrix



Model Selection

Choosing the right model for our data



Model Selection - Split train/test

Split dataset to Train and Test

```
[ ] X = df_clean_dummy.drop(columns='target')
    y = df_clean_dummy['target']
```

```
[ ] X.columns
```

```
Index(['price', 'uses_ad_boosts', 'rating', 'rating_count', 'badges_count',
       'badge_local_product', 'badge_product_quality', 'badge_fast_shipping',
       'product_variation_inventory', 'shipping_option_price',
       'countries_shipped_to', 'inventory_total', 'has_urgency_banner',
       'merchant_rating_count', 'merchant_rating',
       'merchant_has_profile_picture', 'rating_five_ratio',
       'rating_four_ratio', 'rating_three_ratio', 'rating_two_ratio',
       'rating_one_ratio', 'percent_positive', 'no_tags', 'len_title_orig',
       'discount', 'size', 'origin_country_CN', 'origin_country_GB',
       'origin_country_None', 'origin_country_SG', 'origin_country_US',
       'origin_country_VE', 'color_black', 'color_blue', 'color_brown',
       'color_gray', 'color_green', 'color_multicolor', 'color_orange',
       'color_pink', 'color_purple', 'color_red', 'color_silver',
       'color_white', 'color_yellow'],
      dtype='object')
```

```
len(X.columns)
y.value_counts()
```

```
450    618
2      518
1      403
Name: target, dtype: int64
```

```
[ ] from sklearn.model_selection import train_test_split

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)
```

Model Selection - StandardScaler

Perform PCA

Scale dataset

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

```
[ ] X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)
```

```
[ ] X_train_scaled
    X_test_scaled
```

```
array([[-0.59703164, -0.87529458, -0.576262  , ..., -0.0285133 ,
        -0.44960921, -0.26539552],
       [-0.34426443,  1.14247251,  0.05290937, ..., -0.0285133 ,
         2.22415375, -0.26539552],
       [ 0.66680443,  1.14247251,  0.66987928, ..., -0.0285133 ,
        -0.44960921, -0.26539552],
       ...,
       [-0.84979885, -0.87529458,  0.30463097, ..., -0.0285133 ,
        -0.44960921, -0.26539552],
       [-0.59703164, -0.87529458,  0.06829383, ..., -0.0285133 ,
        -0.44960921, -0.26539552],
       [-0.09149721,  1.14247251,  0.43354214, ..., -0.0285133 ,
        -0.44960921, -0.26539552]])array([[-0.84979885,  1.14247251, -1.09190667, ..., -0.0285133 ,
        -0.44960921, -0.26539552],
       [ 0.16127   ,  1.14247251,  1.50780186, ..., -0.0285133 ,
         2.22415375, -0.26539552],
       [ 0.16127   ,  1.14247251, -0.59774719, ..., -0.0285133 ,
        -0.44960921, -0.26539552],
       ...,
       [-0.63241905, -0.87529458,  1.03512758, ..., -0.0285133 ,
        -0.44960921, -0.26539552],
       [-0.59703164, -0.87529458, -1.3282438 , ..., -0.0285133 ,
        -0.44960921, -0.26539552],
       [-0.34426443, -0.87529458,  1.14255356, ..., -0.0285133 ,
         2.22415375, -0.26539552]])
```

Model Selection - Oversampling

Oversampling

```
[ ] from imblearn.over_sampling import SMOTE

    # Apply SMOTE to the train set
    smote = SMOTE(random_state=42)
    X_train_SMOTE, y_train_SMOTE = smote.fit_resample(X_train_scaled, y_train)
```

```
y_train.value_counts()
y_train_SMOTE.value_counts()
```

```
0    494
2    414
1    323
Name: target, dtype: int641    494
0    494
2    494
Name: target, dtype: int64
```

```
[ ] X_train_SMOTE.shape
```

```
(1482, 45)
```

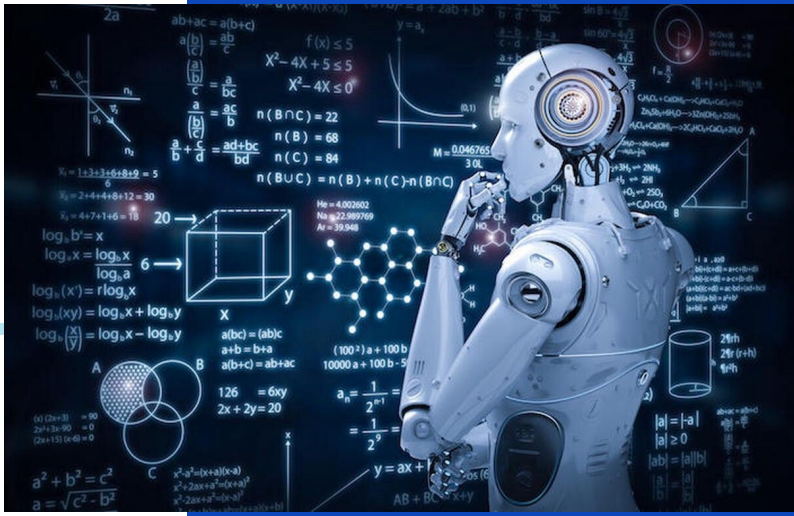
Model Selection - Two Approaches

Because the target column could be interpreted as BOTH categorical and continuous, we decided to implement two approaches.

1. Classification (Multi Class)
2. Regression

Classification

Choosing the right model for our data



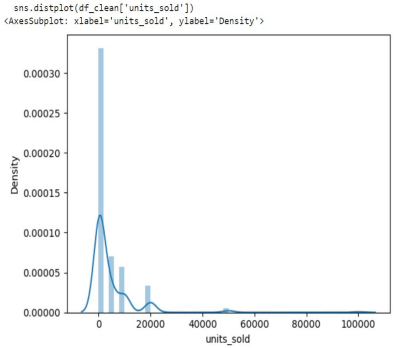
Target - Units Sold Bin

Group targets

```
df_clean.units_sold.value_counts().sort_index()
```

```
1          3
2          2
3          2
6          1
7          2
8          3
10        44
50        68
100      493
1000     403
5000     216
10000    176
20000    103
50000     17
100000     6
Name: units_sold, dtype: int64
```

```
sns.distplot(df_clean['units_sold'])
<AxesSubplot: xlabel='units_sold', ylabel='Density'>
```



BEFORE

```
df.units_sold.mean()
df.units_sold.median()
```

```
4339.005085823268

1000.0
```

Encode target

0 - 'low_sales' if 'units_sold' <= 100

1 - 'regular_sales' if 'units_sold' <= 1000

2 - 'high_sales' if 'units_sold' > 1000

```
df_clean['target'] = df_clean.units_sold.apply(lambda x: 0 if x <= 100
                                                else 1 if x <= 1000
                                                else 2)
```

```
df_clean['target'].value_counts().sort_index()
```

```
0    618
1    403
2    518
Name: target, dtype: int64
```

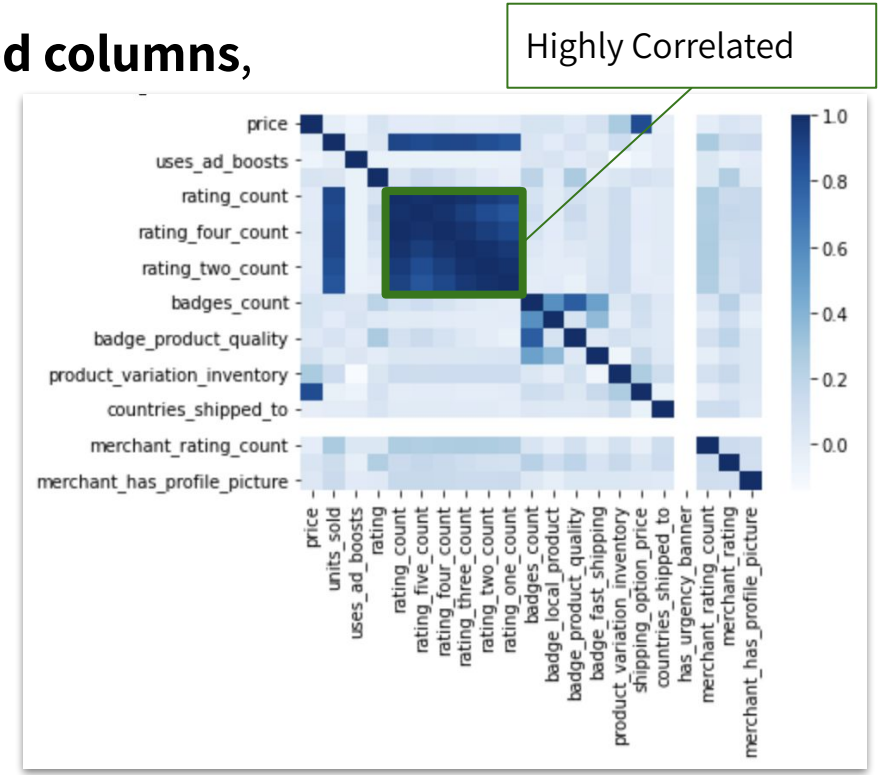
```
df_clean['target'].value_counts().sort_index()/len(df_clean)
```

```
0    0.401559
1    0.261858
2    0.336582
Name: target, dtype: float64
```

AFTER

A Challenging Approach

We started with **dropping all rating related columns**, then tried different models …



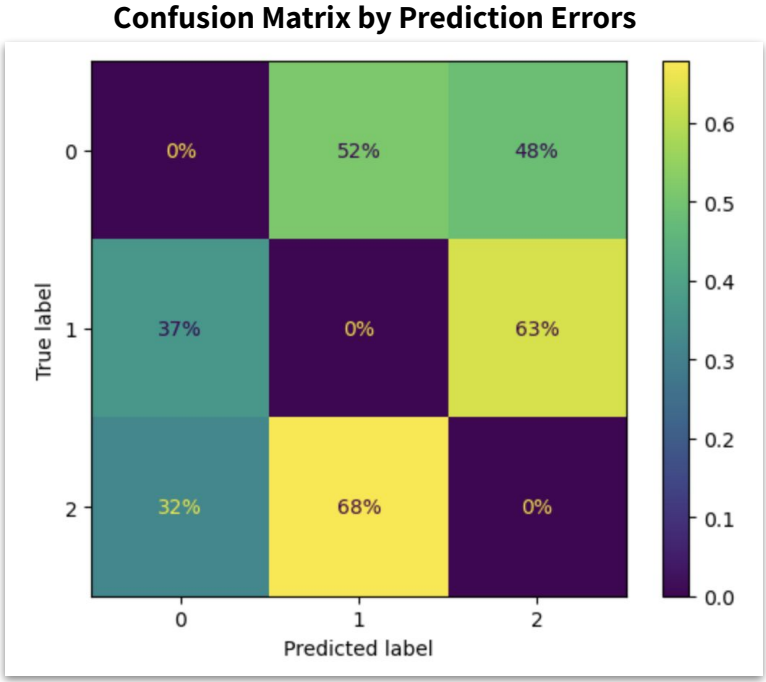
A Challenging Approach

We started with **dropping all rating related columns**, then tried different models …

Random Forest	OVR Random Forest	OVO Random Forest	Logistic Regression	Kernel SVM	XGBoost	Stacking (XGB+RFC)
0.63	0.65	0.63	0.52	0.47	0.66	0.67

A Challenging Approach

We tried to find out what is happening …



Binary Classification Approach

Since the model is not good at distinguishing Class 1 and Class 2
We tried group these two classes …

Random Forest	OVR Random Forest	OVO Random Forest	Logistic Regression	Kernel SVM	XGBoost	Stacking
0.77	N/A	N/A	0.68	0.69	0.79	0.79

Binary Classification Approach

Since the model is not good at distinguishing between Class 1 and Class 2
We tried group these two classes …

Random Forest	OVR Random Forest	OVO Random Forest	Logistic Regression	Kernel SVM	XGBoost	Stacking
0.77	N/A	N/A	0.68	0.69	0.79	0.79

However, we don't think binary classification is a good approach from the business perspective

3 Classes with Rating Ratios

We are back to 3-Class Classification, but adding Rating Stars Ratio columns

```
df_clean['rating_five_ratio'] = df_clean['rating_five_count']/df_clean['rating_count']
df_clean['rating_four_ratio'] = df_clean['rating_four_count']/df_clean['rating_count']
df_clean['rating_three_ratio'] = df_clean['rating_three_count']/df_clean['rating_count']
df_clean['rating_two_ratio'] = df_clean['rating_two_count']/df_clean['rating_count']
df_clean['rating_one_ratio'] = df_clean['rating_one_count']/df_clean['rating_count']
```

3 Classes with Rating Ratios

We are back to 3-Class Classification, but adding Rating Stars Ratio columns
And it resulted a raise on the accuracy scores :

Random Forest	OVR Random Forest	OVO Random Forest	Logistic Regression	Kernel SVM	XGBoost	Stacking
0.76	0.75	0.74	0.51	0.58	0.75	0.74

3 Classes with Rating Counts

After discussing with Professor Lin, we decided to add ‘rating count’ back

3 Classes with Rating Counts

After discussing with Professor Lin, we decided to **add ‘rating count’ back**
Here’s what we have :

Random Forest	OVR Random Forest	OVO Random Forest	Logistic Regression	Kernel SVM	XGBoost	Stacking
0.87	0.87	0.85	0.76	0.67	0.85	0.86

3 Classes with Rating Counts

Random Forest	OVR Random Forest	OVO Random Forest	Logistic Regression	Kernel SVM	XGBoost	Stacking
0.87	0.87	0.85	0.76	0.67	0.85	0.86

We have a weak class

	precision	recall	f1-score	support
0	0.94	0.94	0.94	124
1	0.72	0.79	0.75	80
2	0.90	0.84	0.87	104
accuracy			0.87	308
macro avg	0.85	0.86	0.85	308
weighted avg	0.87	0.87	0.87	308

3 Classes with Rating Counts

	precision	recall	f1-score	support
0	0.94	0.94	0.94	124
1	0.72	0.79	0.75	80
2	0.90	0.84	0.87	104
accuracy			0.87	308
macro avg	0.85	0.86	0.85	308
weighted avg	0.87	0.87	0.87	308

We have a weak class

We tune the model by adding class weights

Class weights

```
class_weights = {0: 1, 1:3, 2: 2}
```

```
rfc_weighted = RandomForestClassifier(n_estimators=764, class_weight=class_weights,
rfc_weighted.fit(X_train_scaled, y_train)
```

3 Classes with Rating Counts

After adjusting class weights:

Random Forest	OVR Random Forest	OVO Random Forest	Logistic Regression	Kernel SVM	XGBoost	Stacking
0.88	0.87	0.85	0.76	0.67	0.85	0.86

Accuracy Scores Comparison

	Random Forest	OVR Random Forest	OVO Random Forest	Logistic Regression	Kernel SVM	XGBoost	Stacking
W/O Rating Counts Cols	0.63	0.65	0.63	0.52	0.47	0.66	0.67
W/ Rating Star Ratios	0.76*	0.75	0.74	0.51	0.58	0.75	0.74
Rating Counts + Star Ratios	0.88*	0.87	0.85	0.76	0.67	0.85	0.86
Binary Classification (w/o Rating Cols)	0.77	N/A	N/A	0.68	0.69	0.79	0.79

* with manually adjusted class weights

Machine Learning - Classification

Random Forest

```
[ ] rfc = RandomForestClassifier(n_estimators=764, random_state=42, n_jobs=-1)
    rfc.fit(X_train_scaled, y_train)
```

```
RandomForestClassifier(n_estimators=764, n_jobs=-1, random_state=42)
```

```
[ ] y_pred_rfc = rfc.predict(X_test_scaled)
```

```
[ ] from sklearn.metrics import classification_report

    report = classification_report(y_test, y_pred_rfc)
    print(report)
```

	precision	recall	f1-score	support
0	0.94	0.94	0.94	124
1	0.72	0.79	0.75	80
2	0.90	0.84	0.87	104
accuracy			0.87	308
macro avg	0.85	0.86	0.85	308
weighted avg	0.87	0.87	0.87	308

Balanced

```
rfc_balanced = RandomForestClassifier(n_estimators=764, class_weight='balanced', random_state=42, n_jobs=-1)
rfc_balanced.fit(X_train_scaled, y_train)
```

```
RandomForestClassifier(class_weight='balanced', n_estimators=764, n_jobs=-1,
                       random_state=42)
```

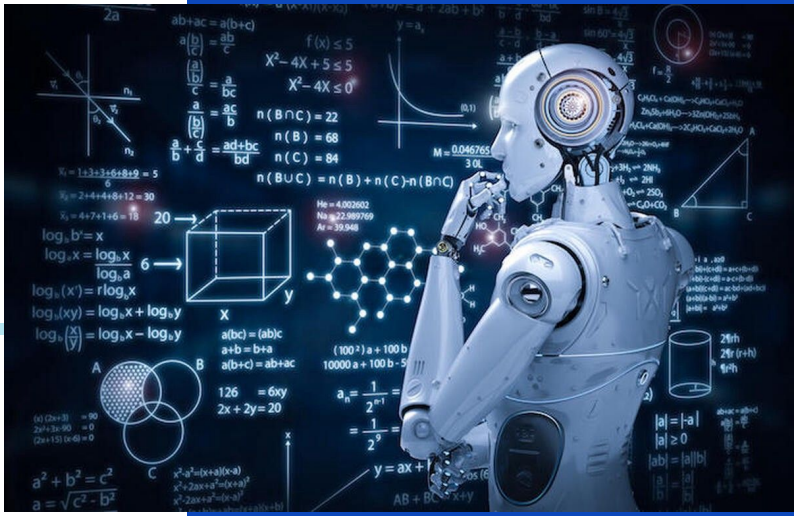
```
[ ] y_pred_rfc_balanced = rfc_balanced.predict(X_test_scaled)
```

```
[ ] report = classification_report(y_test, y_pred_rfc_balanced)
    print(report)
```

	precision	recall	f1-score	support
0	0.95	0.94	0.95	124
1	0.72	0.80	0.76	80
2	0.90	0.83	0.86	104
accuracy			0.87	308
macro avg	0.86	0.86	0.85	308
weighted avg	0.87	0.87	0.87	308

Regression

Choosing the right model for our data



Machine Learning - Regression

From Cleaned_dataset:

- 'Units_sold' - target column
- Train_test_split - 80/20
- StandardScaler - (X_train_scaled , y_train)

Feature selection:

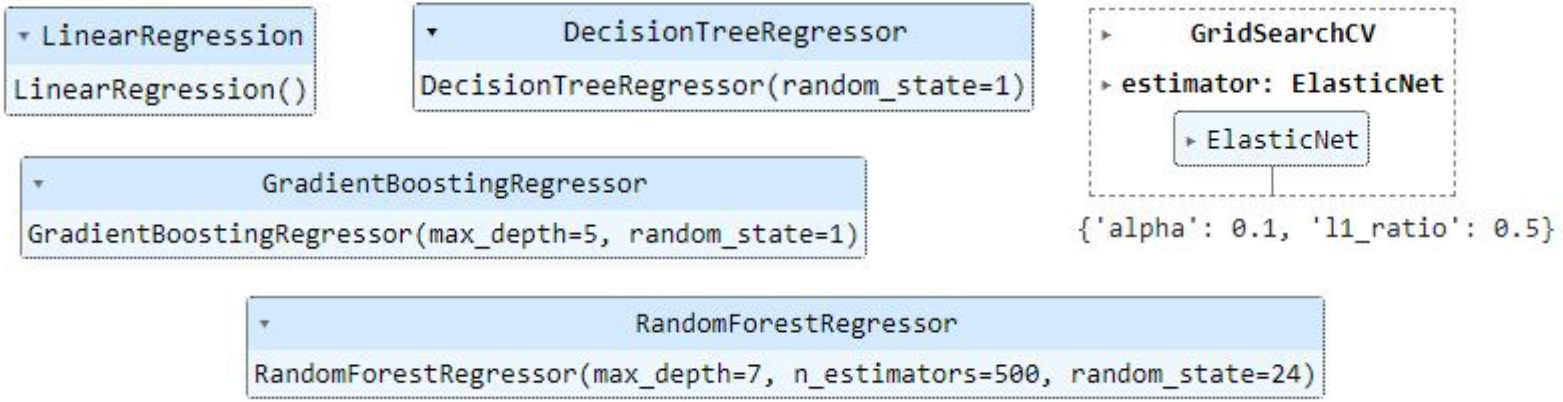
- RandomForestRegressor()
- Obtained ranked feature_importances_
- Extracted top 15 features

```
RandomForestRegressor
RandomForestRegressor(n_estimators=500, n_jobs=-1, random_state=0)
```

	features	rf_importance	rf_rank
3	badges_count	0.862988	1
13	merchant_has_profile_picture	0.019921	2
12	merchant_rating	0.016614	3
0	uses_ad_boosts	0.016071	4
16	rating_three_ratio	0.008849	5
19	percent_positive	0.008084	6
9	countries_shipped_to	0.007768	7
10	has_urgency_banner	0.007059	8
21	no_of_tags	0.006878	9
8	shipping_option_price	0.005930	10
17	rating_two_ratio	0.005381	11
22	CN	0.005197	12
18	rating_one_ratio	0.003589	13
15	rating_four_ratio	0.003480	14
2	rating_count	0.002818	15

Machine Learning - Regression

Model Building:



Machine Learning - Regression

Metrics:

Number of Features	Metrics	Linear Regression	Lasso Regression('alpha':0.1)	ElasticNet with Grid Search ('alpha': 0.1, 'l1_ratio': 0.5,cv=10)	Random Forest Regressor(max_depth=7, n_estimators=500)	GradientBoost Regressor(max_depth=5)
All the 43 Features	Training score	0.83	0.83	0.83	0.95	0.99
	Mean Squared Error	22107692.77	22112420.96	-	23927992.11	29146096.6
	Mean Absolute Error	2082.57	2076.05	-	1805	1903.1
	R- Squared	0.75	0.75	0.81*	0.73	0.67
Top 15 Features	Training score	0.82	0.83	0.83	0.96	0.99
	Mean Squared Error	21962413.51	21962579.97	-	24551260.95	27647955.82
	Mean Absolute Error	1960.78	1960.73	-	1813.37	1849.67
	R- Squared	0.75	0.87	0.81*	0.73	0.69

* Highest scores

Conclusion

Ending Thoughts…

Limitations

Inclusion of the rating counts variables introduces the issue of simultaneity (a rating can only be left after the product has been sold, and high ratings result in more sales)

Wish dataset was crawled all at once, which gives us an incomplete picture of seasonality, the impact of review scores, and the long-tail of sales

95% of the product merchants are in China, which may skew the available inventory, such as product sizing

The data used in this project was a smaller subset of the potential sales data available. If given another chance, we'd recommend tracking the same products over multiple years to increase the dataset size and thus increasing the success rate of our model.

Insights

Unsurprisingly, product review ratings are highly correlated with the volume of units sold

Not all clothing is created equal. Certain colors are correlated with higher sales (e.g., Black, Grey, and Orange) while others are correlated with lower sales (e.g., Yellow, Green, and Pink). Similarly, sizing has an impact on the number of units sold (M > S > L).

The number of tags associated with a product is correlated with the number of units sold, with more tags yielding higher sales

Future Steps

Recommendations for Wish:

- By using an ML model to predict which units will be top-sellers, Wish can prioritize these items in its product recommendation algorithm, thereby increasing user conversion rate and maximizing revenue
    - Because Wish does not own or manage its product inventory, the risk of stocking an unpopular item is minimal, but Wish stands to benefit from giving more visibility to products it predicts will be top-sellers
        - Tweak its user interface to give greater visibility to highly reviewed products (e.g., “Top-rated” badge) from the home and search pages
- Wish could use convolutional neural networks on its product images to recommend tags to the seller, thereby aiding product discovery

How to improve model:

- We could drop the rating column and try to create a model that would address the newer products because they may not have high number of reviews
- The company could track lifetime sales for the same products in order to do a time-series analysis for predicting future sales of specific products by season.

Thank You