In [49]:

```
import numpy as np
import pandas as pd
import seaborn as sns

import matplotlib.pyplot as plt
from matplotlib import style
style.use('ggplot')

import plotly.express as px
import plotly.graph_objects as go
```

NOte : Execute below cells only if running on google colab for getting the dataset

In [50]:

```
!gdown --id 192h8LDFatS7r8GHI6trlegkheGKgBVdE --output CleanDF.csv
```

```
Downloading...
From: https://drive.google.com/uc?id=192h8LDFatS7r8GHI6trlegkheGKgBV
dE
To: /content/CleanDF.csv
100% 747k/747k [00:00<00:00, 50.3MB/s]
```

In [51]:

```
df = pd.read_csv('CleanDF.csv')
df.head(3)
```

Out[51]:

| | title_translated | listed_price | retail_price | units_sold | uses_ad_boosts | rating | rating_count |
|---|---|---|---|---|---|---|---|
| **0** | 2020 Summer Vintage Flamingo Print Pajamas Se... | 16.0 | 14 | 100 | 0 | 3.76 | 54 |
| **1** | Women's Casual Summer Sleeveless Sexy Mini Dress | 8.0 | 22 | 20000 | 1 | 3.45 | 6135 |
| **2** | 2020 New Arrival Women Spring and Summer Beach... | 8.0 | 43 | 100 | 0 | 3.57 | 14 |

## EDA

group the features by whether they are categorical, numerical, or 'other'.

In [52]:

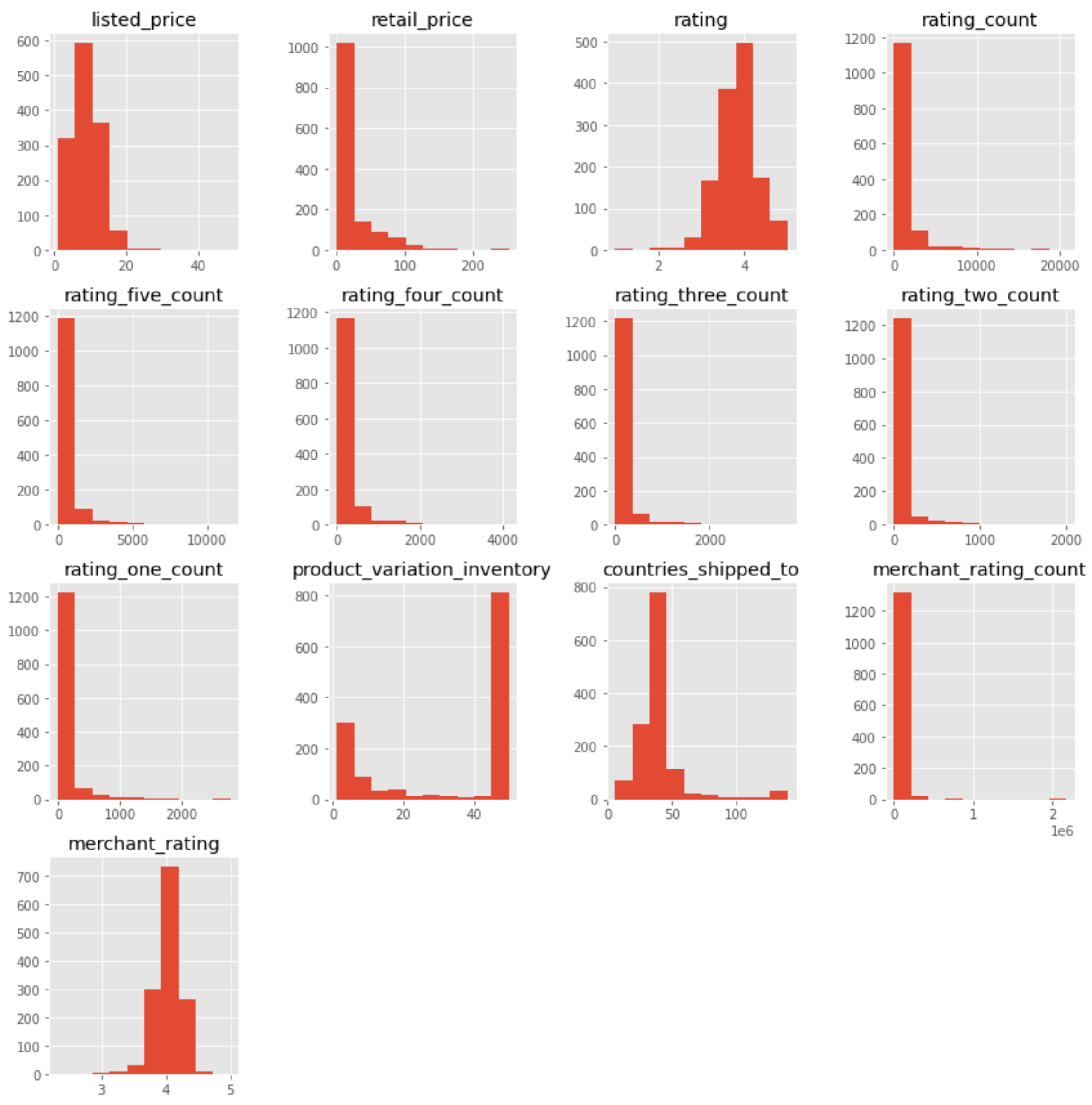
```
numerical_features = ['listed_price', 'retail_price', 'rating', 'rating_count', 
'rating_five_count', 'rating_four_count',
                      'rating_three_count', 'rating_two_count', 'rating_one_coun
t', 'product_variation_inventory',
                      'countries_shipped_to', 'merchant_rating_count', 'merchant
_rating']

categorical_features = ['units_sold', 'uses_ad_boosts', 'badge_product_quality', 
'product_color',
                        'product_variation_size_id', 'shipping_option_price','ha
s_urgency_banner',
                        'urgency_text']

other_features = ['title_orig', 'tags', 'merchant_id', 'product_picture', 'produ
ct_id']
```

Let's use `DataFrame.hist()` to visualize the distribution of the numerical features:

- The `price` and `countries_shipped_to` features are positively skewed.
- The `rating` and `merchant_rating` features are negatively skewed.
- The `retail_price` , `rating_count` ... and `merchant_rating_count` features are exponential distributions rather than Gaussian.
- The `product_variation_inventory` is organic; not obeying any probability distribution.

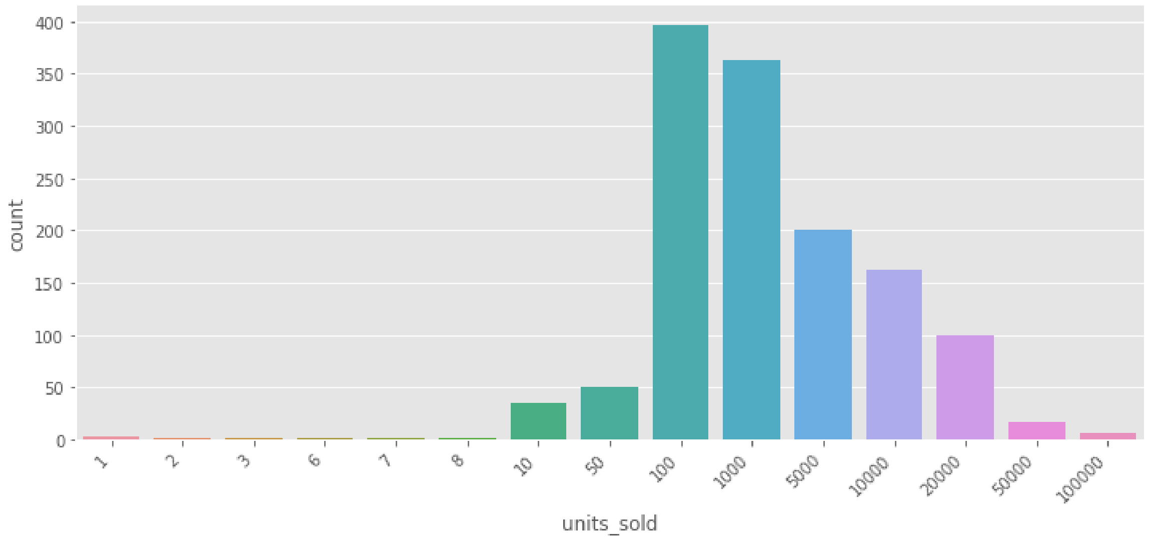
In [53]:

```
hist = df[numerical_features].hist(figsize=(12, 12))

plt.tight_layout()
plt.show()
```


listed_price
retail_price
rating
rating_count
rating_five_count
rating_four_count
rating_three_count
rating_two_count
rating_one_count
product_variation_inventory
countries_shipped_to
merchant_rating_count
1e6
merchant_rating

In [54]:

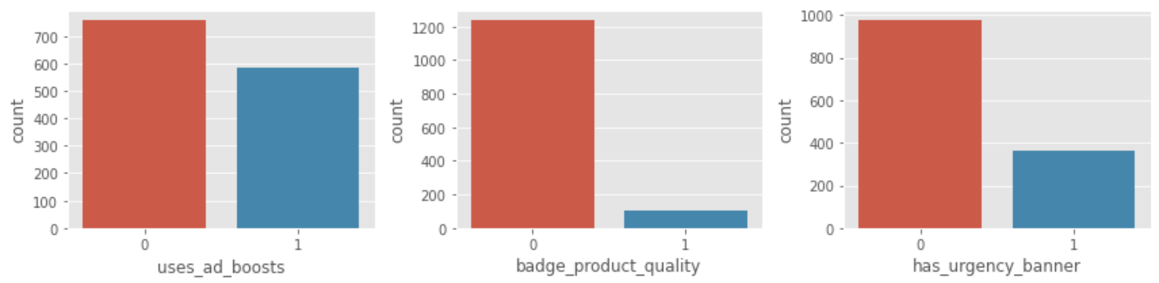
```
# visualize the categorical features.

fig, ax = plt.subplots(figsize=(12, 5))

hist = sns.countplot(x='units_sold', data=df, order=sorted(df['units_sold'].uniq
ue()), ax=ax)

plt.xticks(rotation=45, ha='right')
plt.show()
```



In [55]:

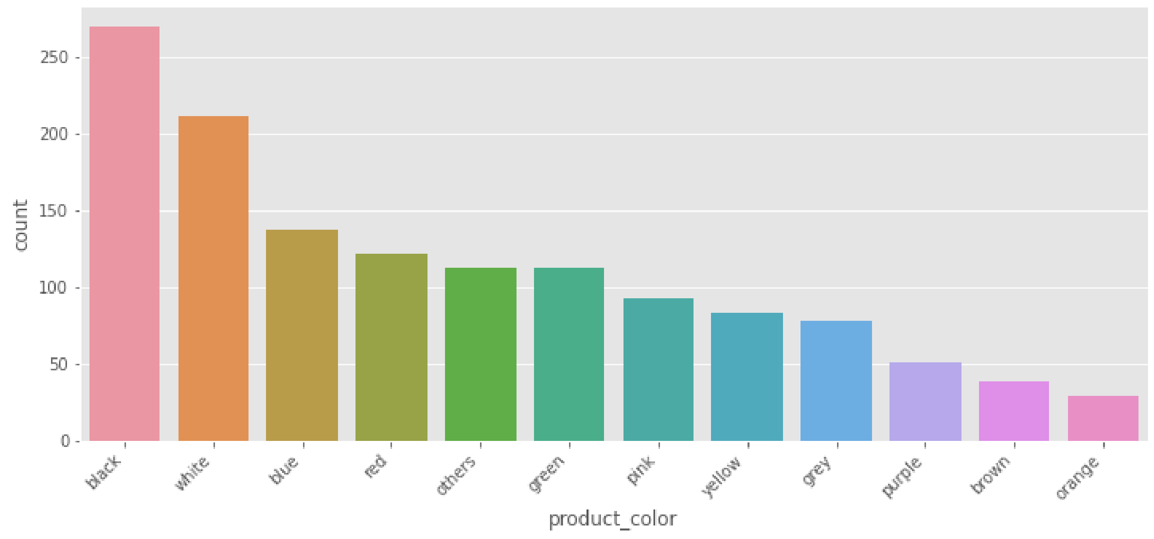
```
# The features: `uses_ad_boosts`, `badge_product_quality`, and `has_urgency_bann
er` are binary.

fig = plt.figure(figsize=(12, 3))

boolean_features = ['uses_ad_boosts', 'badge_product_quality', 'has_urgency_bann
er']
for i in range(3):
    true_percentage = (df[boolean_features[i]].value_counts()[1] / len(df[boolea
n_features[i]])) * 100
    print("Percent that the '%s' flag is True: %f" % (boolean_features[i], true_
percentage))

    fig.add_subplot(1, 3, i + 1)
    sns.countplot(df[boolean_features[i]])

plt.tight_layout()
plt.show()
```

```
Percent that the 'uses_ad_boosts' flag is True: 43.549590
Percent that the 'badge_product_quality' flag is True: 7.755406
Percent that the 'has_urgency_banner' flag is True: 27.293065

/usr/local/lib/python3.6/dist-packages/seaborn/_decorators.py:43: Fu
tureWarning:

Pass the following variable as a keyword arg: x. From version 0.12,
the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or mis
interpretation.

/usr/local/lib/python3.6/dist-packages/seaborn/_decorators.py:43: Fu
tureWarning:

Pass the following variable as a keyword arg: x. From version 0.12,
the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or mis
interpretation.

/usr/local/lib/python3.6/dist-packages/seaborn/_decorators.py:43: Fu
tureWarning:

Pass the following variable as a keyword arg: x. From version 0.12,
the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or mis
interpretation.
```



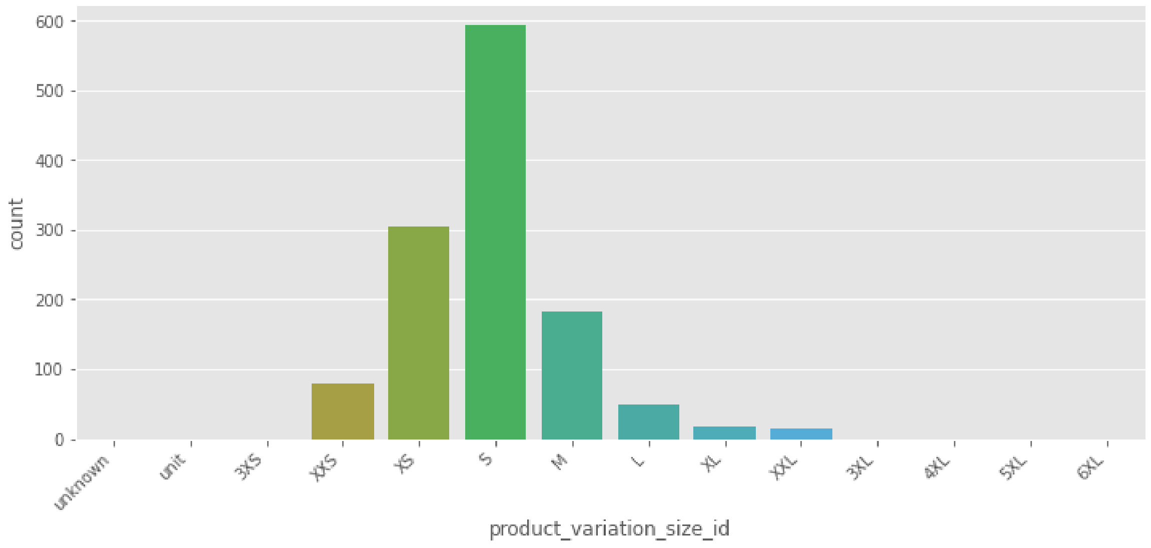
In [56]:

```
# visualize the Product Colours.

fig, ax = plt.subplots(figsize=(12, 5))

sns.countplot(x='product_color', data=df, order=df['product_color'].value_counts
().index, ax=ax)

ax.set(xlabel='product_color', ylabel='count')
plt.xticks(rotation=45, ha='right')

plt.show()
```



In [57]:

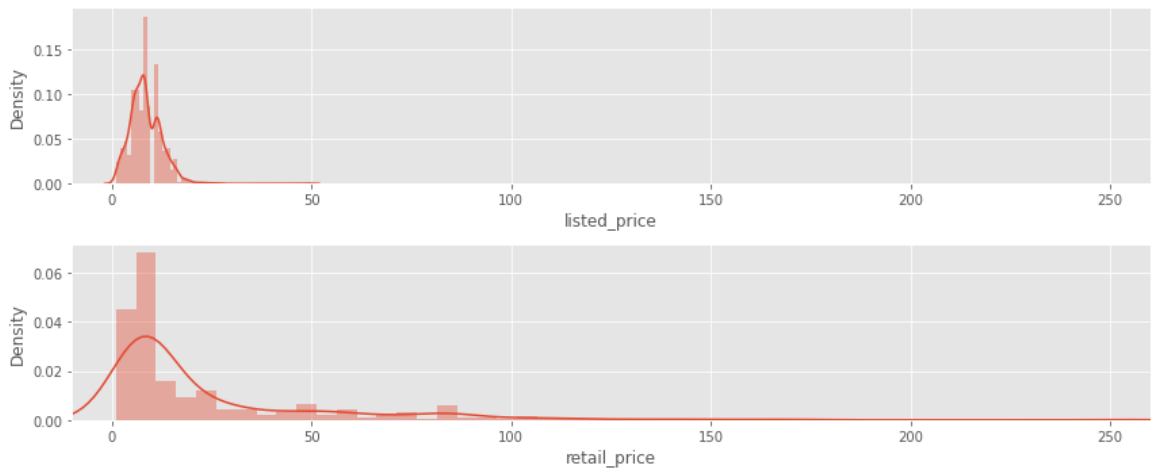
```
# visualize the product_variation_size_id.


fig, ax = plt.subplots(figsize=(12, 5))

size_order = ['unknown', 'unit', '3XS', 'XXS', 'XS', 'S', 'M', 'L', 'XL', 'XXL',
'3XL', '4XL', '5XL', '6XL']

sns.countplot(x='product_variation_size_id', data=df, order=size_order, ax=ax)

plt.xticks(rotation=45, ha='right')
plt.show()
```



In [58]:

```
# Visualizing product prices (prediction target).

fig = plt.figure(figsize=(12, 5))

fig.add_subplot(2,1,1)
ax1 = sns.distplot(df['listed_price'])
ax1.set_xlim(-10, 260)

fig.add_subplot(2,1,2)
ax2 = sns.distplot(df['retail_price'])
ax2.set_xlim(-10, 260)

plt.tight_layout()
```

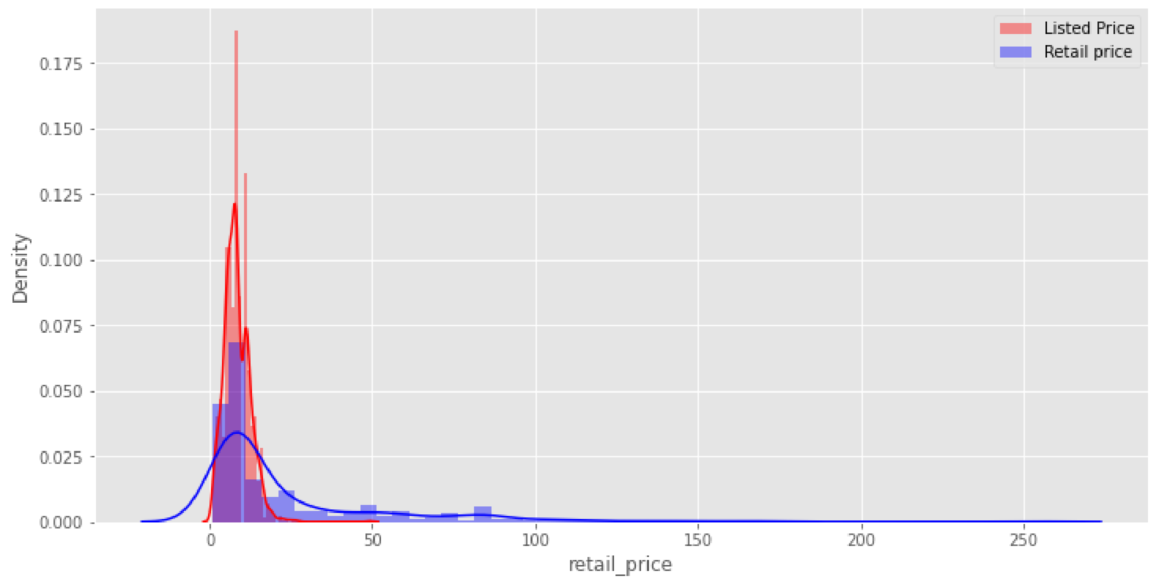
```
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:255
1: FutureWarning:

`distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-le
vel function with similar flexibility) or `histplot` (an axes-level
function for histograms).

/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:255
1: FutureWarning:

`distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-le
vel function with similar flexibility) or `histplot` (an axes-level
function for histograms).
```



```
In [59]:
```

```
plt.figure(figsize=(12,6))
sb.distplot(df['listed_price'], color='red', label='Listed Price')
sb.distplot(df['retail_price'], color='blue', label='Retail price')
plt.legend();
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:255
1: FutureWarning:

`distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-le
vel function with similar flexibility) or `histplot` (an axes-level
function for histograms).

/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:255
1: FutureWarning:

`distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-le
vel function with similar flexibility) or `histplot` (an axes-level
function for histograms).
```

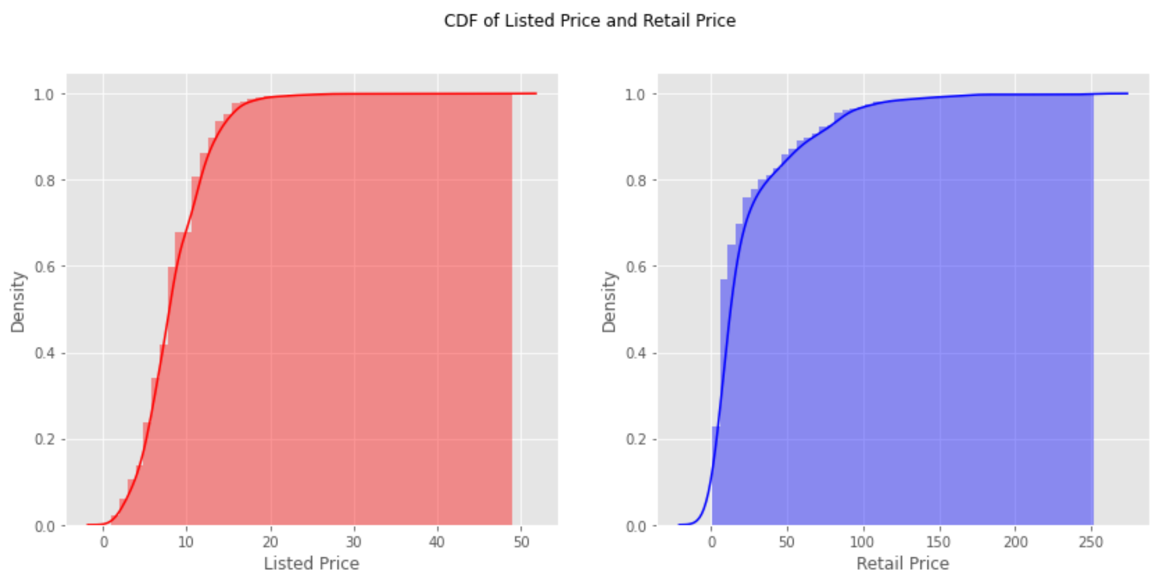


**The plot indicates a right skewed distribution but not very Clear**

In [60]:

```
kwargs = {'cumulative':True}
f, axes = plt.subplots(1,2, figsize=(14,6))
f.suptitle('CDF of Listed Price and Retail Price')
sb.distplot(df['listed_price'].values,kde_kws=kwargs, hist_kws=kwargs, color='re
d', label='Listed Price', ax=axes[0]);
sb.distplot(df['retail_price'].values,kde_kws=kwargs, hist_kws=kwargs, color='bl
ue', label='Retail Price', ax=axes[1]);
axes[0].set(xlabel='Listed Price');
axes[1].set(xlabel='Retail Price');
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:255
1: FutureWarning:

`distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-le
vel function with similar flexibility) or `histplot` (an axes-level
function for histograms).

/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:255
1: FutureWarning:

`distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-le
vel function with similar flexibility) or `histplot` (an axes-level
function for histograms).
```



CDFs are more useful inorder to visualize the data more efficiently.

CDF of price reveals that 97% of products are listed for less than aproximate price 19-20
CDF of Price closely represents the CDF curve of Normal distribution which can be summarized efficiently except for the 3% data
Incase of Retail price the distribution is not very much smooth and contains price gaps

In [61]:

```
fig = go.Figure()
fig.add_trace(go.Box(x=df['retail_price'], name='Retail Price'))
fig.add_trace((go.Box(x=df['listed_price'], name='Listed Price')))
fig['layout']['title'] = 'Distribution of Listed Price and Retail Price'
fig.show()
```

With Boxplots we can easily spot the outliers and quartiles

- The Upper fence of Price is at 18 i.e most of the data is priced less tha 18
- There an item wiht price of 49 i.e clearly an oulier as it is far away from the Inter Quartile Range (Q3 - Q1)
- Box plot of Retail price is much more spread out, there is huge difference of 195 between the upper fence and max data point

In [62]:

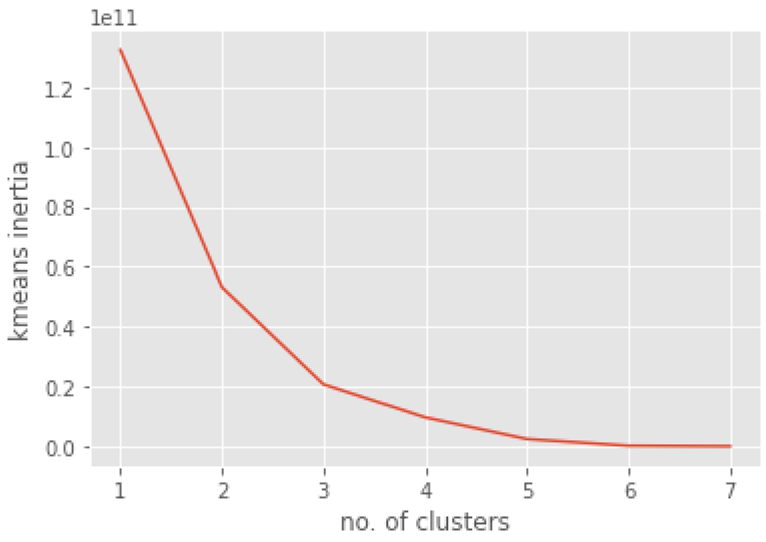
```
#range for units sold
sorted(df['units_sold'].unique())
```

Out[62]:

```
[1, 2, 3, 6, 7, 8, 10, 50, 100, 1000, 5000, 10000, 20000, 50000, 100
000]
```

In [63]:

```
from sklearn.cluster import KMeans

clusters = {}
for i in range(1,8):
    kmeans = KMeans(n_clusters=i).fit(df[['units_sold']])
    clusters[i] = kmeans.inertia_

plt.plot(list(clusters.keys()), list(clusters.values()));
plt.xlabel('no. of clusters');
plt.ylabel('kmeans inertia');
```



By performing clustering we can see that units_sold can be clustered in 3 categories (optimal) as the inertia curve smooths out after 3 clusters

In [64]:

```
#order cluster method

def order_cluster(cluster_field_name, target_field_name,df,ascending):
    new_cluster_field_name = 'new_' + cluster_field_name
    df_new = df.groupby(cluster_field_name)[target_field_name].mean().reset_inde
x()
    df_new = df_new.sort_values(by=target_field_name,ascending=ascending).reset_
index(drop=True)
    df_new['index'] = df_new.index
    df_final = pd.merge(df,df_new[[cluster_field_name,'index']], on=cluster_fiel
d_name)
    df_final = df_final.drop([cluster_field_name],axis=1)
    df_final = df_final.rename(columns={"index":cluster_field_name})
    return df_final

df['units_sold_cluster'] = KMeans(n_clusters=3).fit(df[['units_sold']]).predict(
df[['units_sold']])
df = order_cluster('units_sold_cluster','units_sold',df,True)
df.groupby(['units_sold_cluster'])['units_sold'].describe()
```

Out[64]:

| | count | mean | std | min | 25% | 50% | 75% | |
|---|---|---|---|---|---|---|---|---|
| **units_sold_cluster** | | | | | | | | |
| **0** | 1056.0 | 1330.027462 | 1820.903726 | 1.0 | 100.0 | 1000.0 | 1000.0 | |
| **1** | 262.0 | 13778.625954 | 4857.809727 | 10000.0 | 10000.0 | 10000.0 | 20000.0 | 2 |
| **2** | 23.0 | 63043.478261 | 22448.887927 | 50000.0 | 50000.0 | 50000.0 | 75000.0 | 1 |

now we have a clear picture of top selling, and price range of products

In [65]:

```
px.scatter(df,x='units_sold',y='rating', color='units_sold_cluster', marginal_y
='box',title='Rating vs units sold')
```

- Median for rating is 3.85 and the products in top selling cluster has rating between 3.35 to 4.1 seems very reasonable
- Rating is very important to determine the potential of product
- Still there are some products with 5 star rating yet unable to cross the 100-1000 unit sold line
- there are some really bad performing products with rating below 3

In [66]:

```
px.scatter(df,x='rating',y='merchant_rating', color='units_sold_cluster', margin
al_y ='box',title='Merchant Rating vs units sold', opacity=0.7)
```

In [67]:

```
px.scatter(df,x='rating', y='product_variation_inventory', color='units_sold_clu
ster', title='Product variation vs Rating')
```

In [68]:

```
fig = px.scatter(df,x='rating_count',y='rating', color='units_sold_cluster', tit
le='Rating vs Rating count')
fig.add_trace(go.Scatter(x=np.ones((len(df)))*1103,y=df['rating'],name='Threshol
d 1'))
fig.add_trace(go.Scatter(x=np.ones((len(df)))*7773, y=df['rating'],name='Thresho
ld 2'))
fig.update_layout(showlegend=False)
```

**From above visualization we can conclude that products sold by merchants belonging to cluster 2 and 1 are Top selling,most liked and trusted by buyers**

- There's some kind of thresholding that can be done on rating and rating count to separate the 3 categories of products
- still there are few overlapping data points

In [69]:

```
px.scatter(df,x='retail_price', y='listed_price',color='units_sold_cluster',marg
inal_y='box')
```

Most of the top selling products seems be concentrated to the left where the price difference is much siginificant

In [70]:

```
px.scatter(df, x='listed_price', y='shipping_option_price', color= 'units_sold_c
luster', title='Shipping price vs Price')
```

People always prefer paying less shipping charges we can see that most selling products has low shipping charges

### The distribution of prices and defining a successful product..

In [71]:

```
print('Median of units sold is',df['units_sold'].median())
print('Mean of units sold is',df['units_sold'].mean())
df['units_sold'].value_counts()
```

```
Median of units sold is 1000.0
Mean of units sold is 4820.662938105891
```

Out[71]:

```
100       396
1000      362
5000      200
10000     163
20000      99
50         50
10         36
50000      17
100000      6
1           3
8           2
7           2
3           2
2           2
6           1
Name: units_sold, dtype: int64
```

In [72]:

```
def below_ten(units_sold):
    if units_sold < 10:
        return 10
    else:
        return units_sold
```

In [73]:

```
df['units_sold'] = df['units_sold'].apply(below_ten)
df['units_sold'].value_counts()
```

Out[73]:

```
100       396
1000      362
5000      200
10000     163
20000      99
50         50
10         48
50000      17
100000      6
Name: units_sold, dtype: int64
```

- The median is 1000 units sold. I will consider for those products with units sold over 1000 units as successful products.
- A quick look at the products with 100000 sales.

In [74]:

```
df[df['units_sold'] == 100000]
```

Out[74]:

| | title_translated | listed_price | retail_price | units_sold | uses_ad_boosts | rating | rating_coun |
|---|---|---|---|---|---|---|---|
| **1319** | 2018 New Fashion Women's Tops Sexy Strappy Sle... | 5.00 | 25 | 100000 | 1 | 3.83 | 1798 |
| **1322** | Women Stretchy Camisole Spaghetti Strap Long T... | 5.77 | 48 | 100000 | 0 | 4.10 | 2074 |
| **1323** | New Aeeival Women Clothing Long Sleeve Autumn ... | 8.00 | 7 | 100000 | 1 | 3.76 | 1106 |
| **1324** | Womens Summer Red White and Blue Chiffon Short... | 5.00 | 33 | 100000 | 0 | 3.98 | 1378 |
| **1335** | Women Lace Short Sleeve Long Tops Blouse Shirt... | 7.00 | 22 | 100000 | 1 | 3.82 | 1191 |
| **1337** | Women's Summer Sexy Sleeveless Turtleneck Mini... | 5.67 | 19 | 100000 | 0 | 3.53 | 1839 |

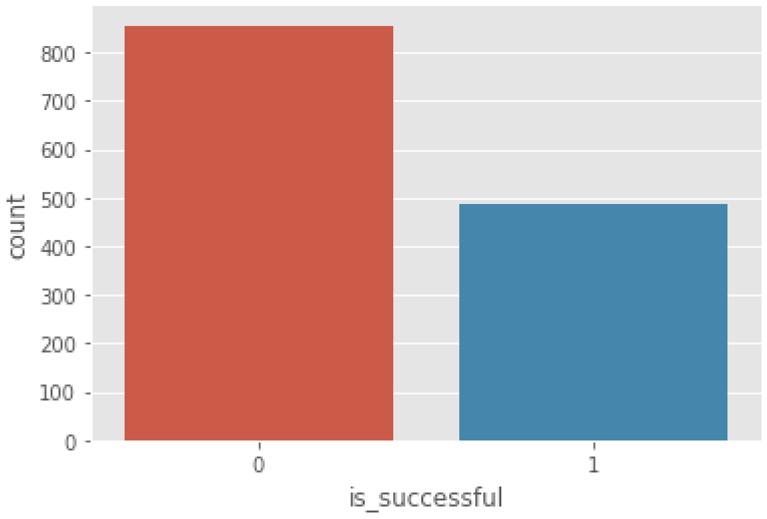
In [75]:

```
# A simple function to create a new label 'is_successful':A simple function to c
reate a new label 'is_successful':

def is_successful(units_sold):
    if units_sold > 1000:
        return 1
    else:
        return 0
```

In [76]:

```
df['is_successful'] = df['units_sold'].apply(is_successful)
#df['is_successful'] = df['units_sold'].apply(is_successful).astype('category')
print('Percent of successful products: ', df['is_successful'].value_counts()[1]
/ len(df['is_successful'])*100)
sb.countplot(data=df, x='is_successful')
plt.show()
```

```
Percent of successful products:  36.16703952274422
```



I believe 33% is an appropriate % to be defined as successful sales.

## The use of ad boosts to boost the success..

There is an almost equal use of ad boosts by the products.

In [77]:

```
print('Percent of products using ad boosts: ', df['uses_ad_boosts'].value_counts
()[1] / len(df['uses_ad_boosts'])*100)
```

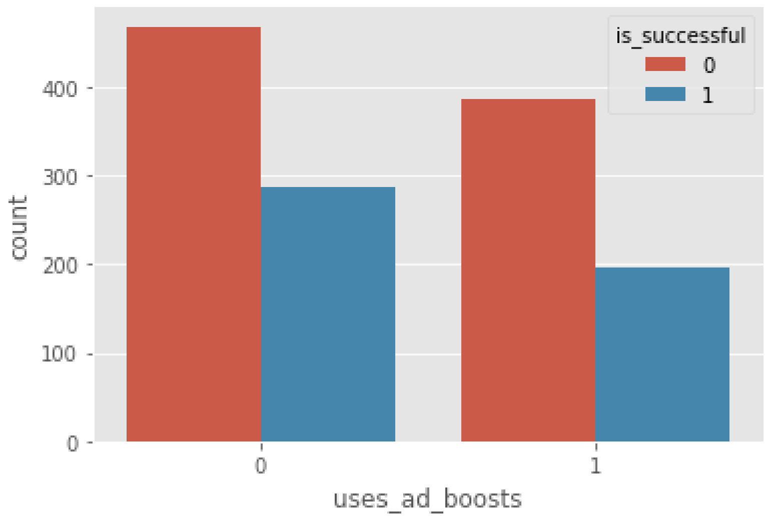
```
Percent of products using ad boosts:  43.54958985831469
```

In [78]:

```
sb.countplot(data=df, x='uses_ad_boosts', hue='is_successful')
```

Out[78]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c736b96d8>
```



In [79]:

```
pd.crosstab(df['uses_ad_boosts'], df['is_successful'])
```

Out[79]:

| is_successful | 0 | 1 |
|---|---|---|
| **uses_ad_boosts** | | |
| **0** | 469 | 288 |
| **1** | 387 | 197 |

## Higher ratings means higher units sold?

In [80]:

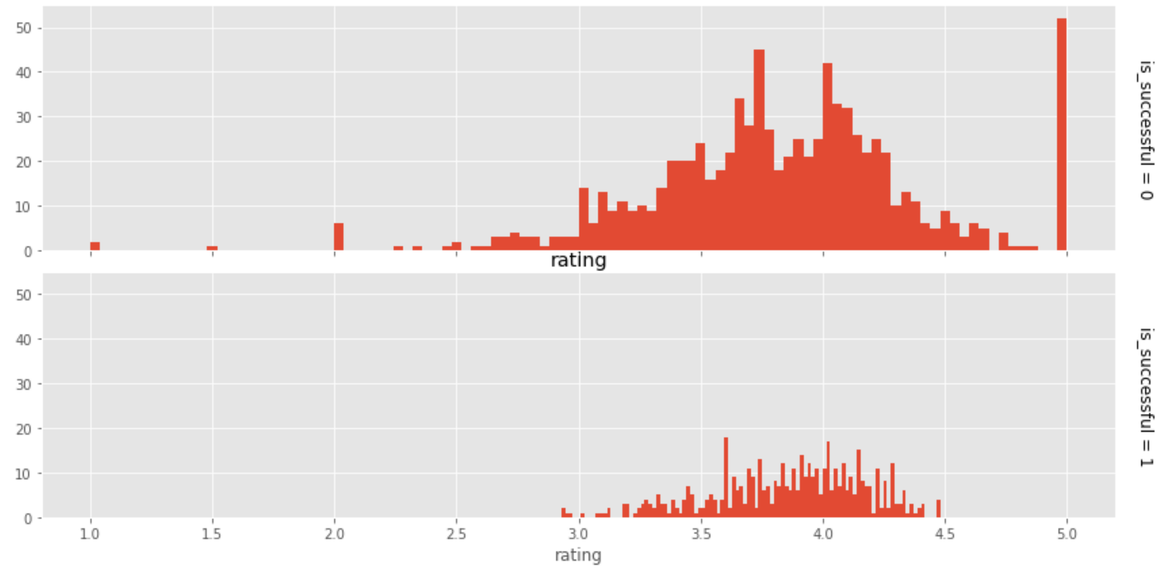
```
df['rating']
```

Out[80]:

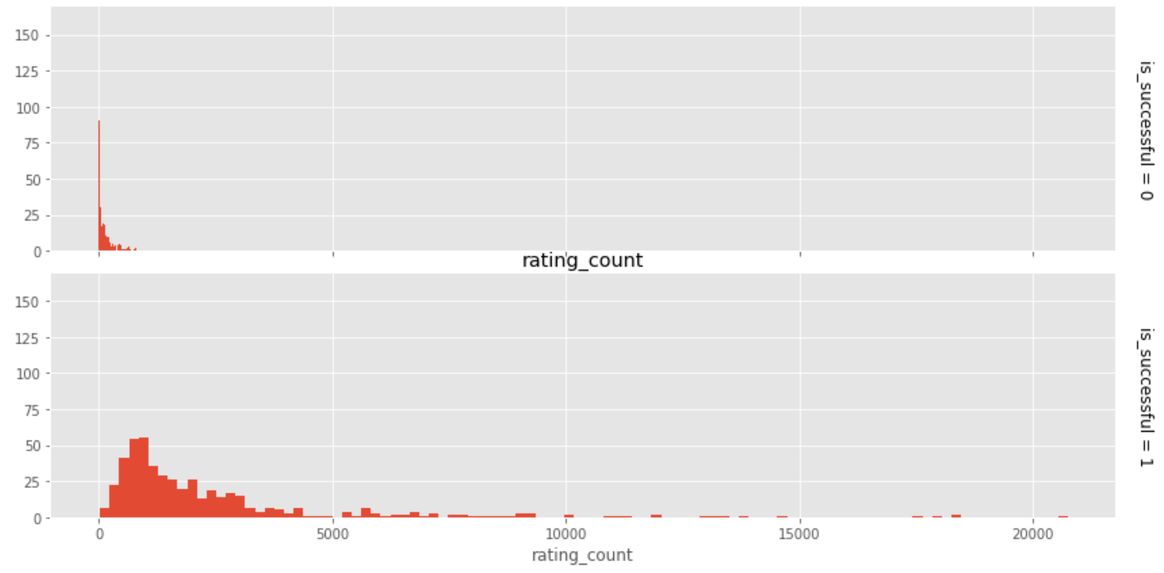
```
0       3.76
1       3.57
2       4.03
3       3.10
4       5.00
        ... 
1336    3.91
1337    3.53
1338    4.01
1339    4.01
1340    3.60
Name: rating, Length: 1341, dtype: float64
```

In [81]:

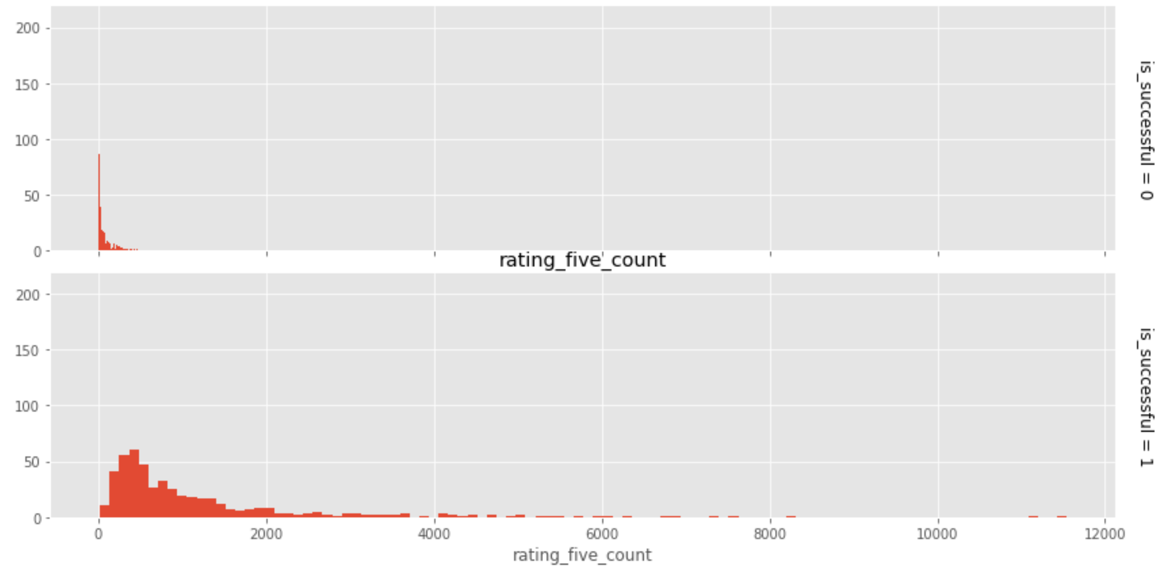
```
ratings_column = ['rating', 'rating_count',
       'rating_five_count', 'rating_four_count', 'rating_three_count',
       'rating_two_count', 'rating_one_count']

for column in ratings_column:
    g = sb.FacetGrid(df, row='is_successful', margin_titles=True, aspect=4, heig
ht=3)
    g.map(plt.hist, column, bins=100)
    plt.title(column)
    plt.show()
```

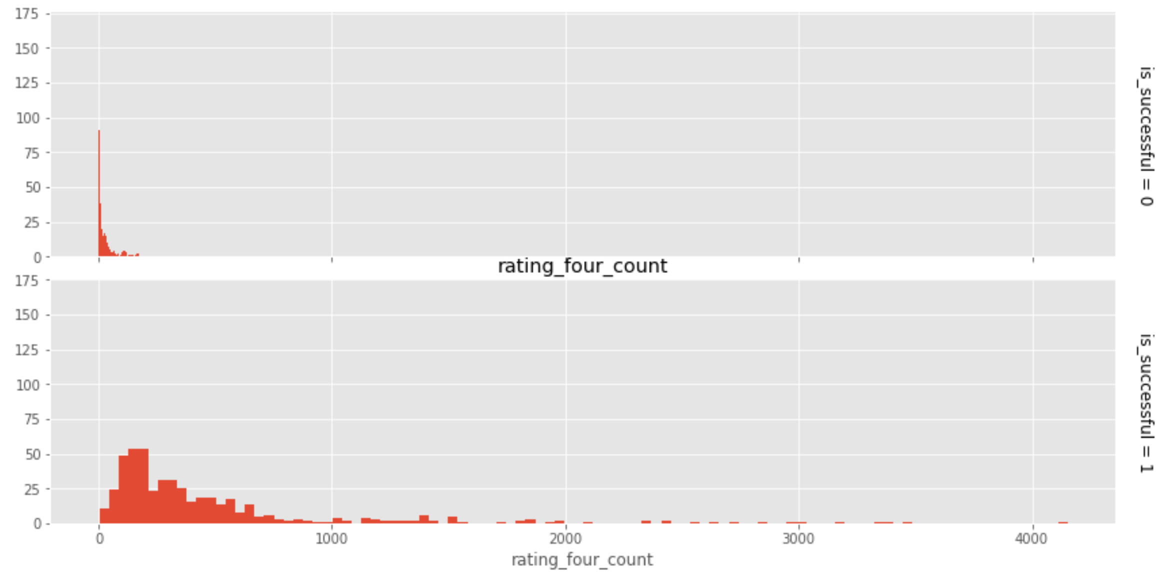


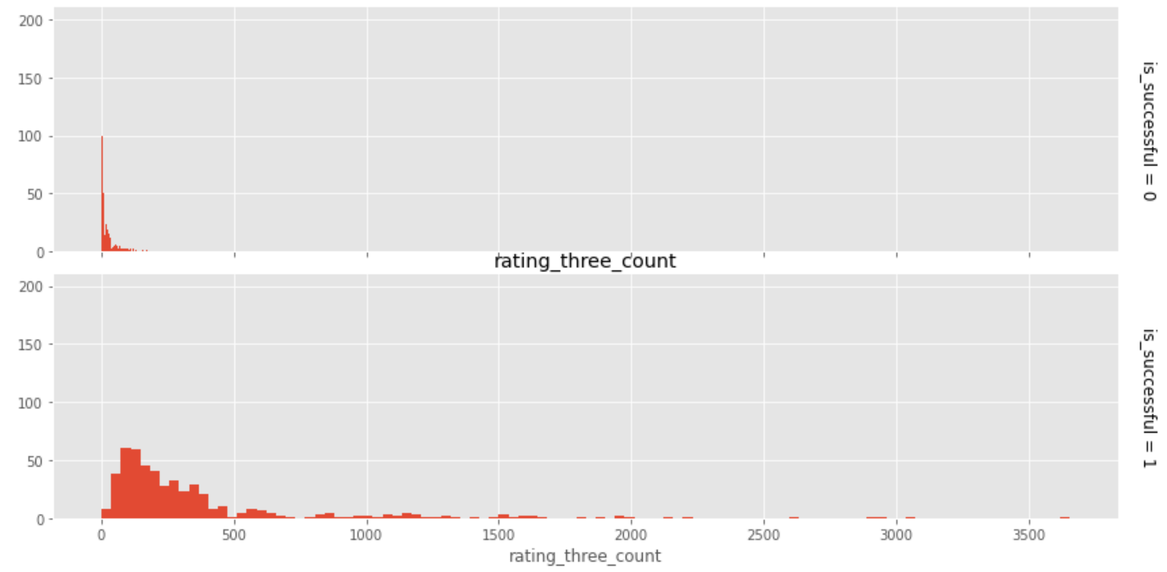
rating
is_successful = 0
is_successful = 1
rating

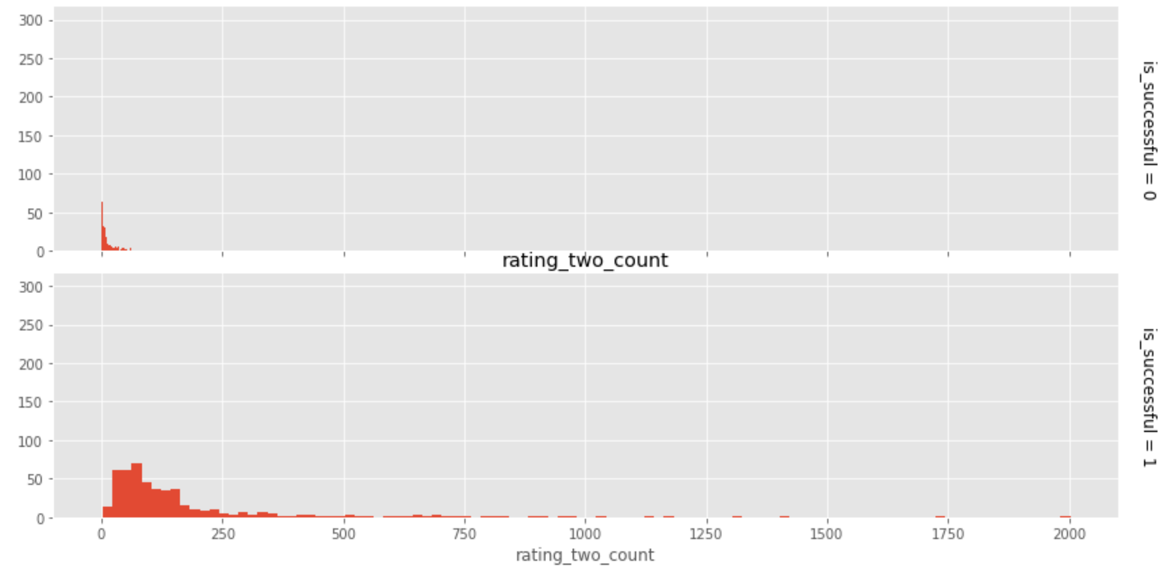


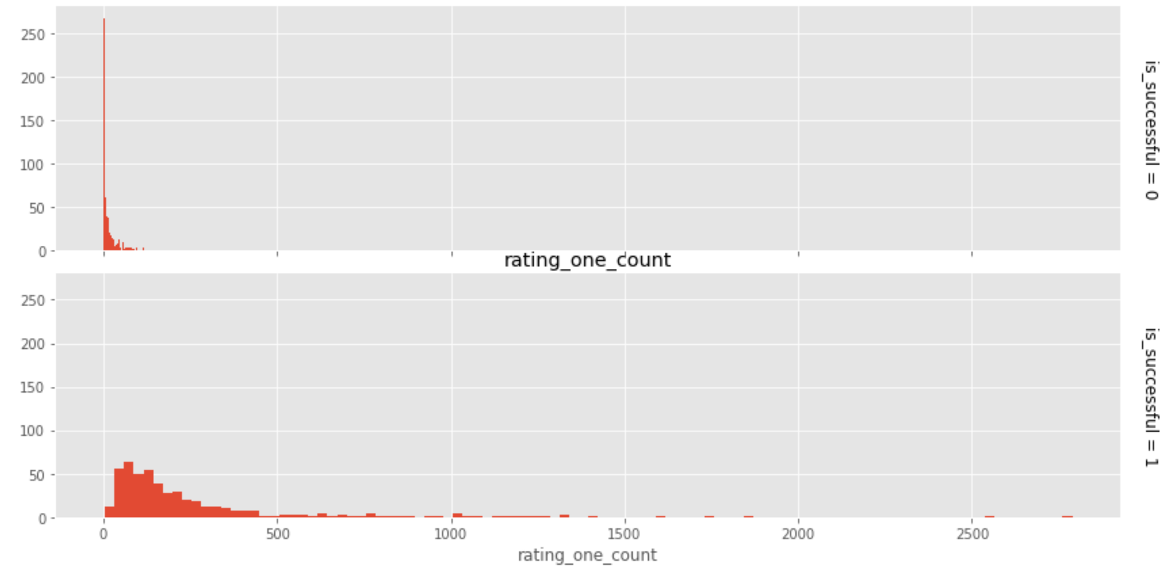
rating_count
is_successful = 0
is_successful = 1
rating_count


rating_five_count
is_successful = 0
is_successful = 1
rating_five_count
0
2000
4000
6000
8000
10000
12000
0
50
100
150
200


rating_four_count
is_successful = 0
is_successful = 1
rating_four_count
0
1000
2000
3000
4000
0
25
50
75
100
125
150
175


rating_three_count
is_successful = 0
is_successful = 1
rating_three_count
0
50
100
150
200
0
500
1000
1500
2000
2500
3000
3500


rating_two_count
is_successful = 0
is_successful = 1
rating_two_count
0
50
100
150
200
250
300
0
250
500
750
1000
1250
1500
1750
2000



In [82]:

```
df.groupby('is_successful').mean()[ratings_column]
```

Out[82]:

| | rating | rating_count | rating_five_count | rating_four_count | rating_three_count |
|---|---|---|---|---|---|
| **is_successful** | | | | | |
| **0** | 3.829603 | 136.151869 | 65.349299 | 26.147196 | 19.109813 |
| **1** | 3.858701 | 2474.486598 | 1196.740206 | 486.369072 | 365.517526 |

In [83]:

```
df.groupby('units_sold').mean()[ratings_column]
```

Out[83]:

| | rating | rating_count | rating_five_count | rating_four_count | rating_three_count | ra |
|---|---|---|---|---|---|---|
| **units_sold** | | | | | | |
| **10** | 4.388750 | 1.479167 | 0.625000 | 0.229167 | 0.229167 | |
| **50** | 3.977600 | 5.920000 | 3.000000 | 1.260000 | 0.720000 | |
| **100** | 3.752626 | 34.568182 | 16.002525 | 6.585859 | 4.823232 | |
| **1000** | 3.819227 | 283.121547 | 136.524862 | 54.419890 | 39.781768 | |
| **5000** | 3.818900 | 882.220000 | 421.290000 | 168.720000 | 127.325000 | |
| **10000** | 3.898282 | 1847.257669 | 922.674847 | 365.361963 | 258.582822 | |
| **20000** | 3.867576 | 4507.626263 | 2161.181818 | 889.747475 | 673.959596 | |
| **50000** | 3.903529 | 10731.941176 | 5216.764706 | 2104.882353 | 1614.058824 | |
| **100000** | 3.836667 | 15646.833333 | 7187.166667 | 3120.500000 | 2583.500000 | |

Successful products have more ratings. This is expected as units sold is higher so rating is also higher. However, if I am building a model, I don't think it would be a good idea to keep this column as the huge number of ratings must be the result of higher units sold.

## Some conclusions

- The site mainly sells female clothing.
- Higher units sold means higher rating count.
- The use of ad boosts does not seen to have any effect on the units sold and the site may lose revenue from this ads.
- More detailed units sold and inventory levels would have been more helpful for analysis.
- The tags can be improved so that products can be categorised more specfically. This can be done by reducing the number of tags per product, so the mechants are forced to choose their tags more wisely.
- Majority of the products are black and white. This might have been defined wrongly by the merchants. If not the case, the merchants can be encoured to include more variation to these.