This dataset you received contains information on over 800+ fashion products. The data includes fields such as product ID,identifier ,selling_price,original_price,currency,availability,color,category,breadcrumbs,country,

Task 1: Please analyze the data in order to find which colors are the most popular among different genders and age groups, in the end please create a dashboard to show your results.

Task 2: Please analyze the correlation between ratings and reviews to determine which factors are most important to customers, in the end please create a dashboard to show your results.

Task 3: Please create a modell to predict which combinatin of selling_price,color,category,breadcrumbs could be most popular products for next year?

In [1]:
```
#Requirements

#pandas==1.4.4
#numpy==1.21.5
#scipy==1.9.1
#seaborn==0.11.2
#matplotlib==3.5.2
#statsmodels==0.13.2
#scikit-learn==1.0.2
#seaborn==0.11.2
```

In [82]:
```
#importing data and observe
import pandas as pd
import scipy.stats
import numpy as np
from sklearn.impute import KNNImputer
from sklearn import preprocessing
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import statsmodels.api as sm


import seaborn as sns
sns.set()
import matplotlib.pyplot as plt

nan = np.nan


import warnings
warnings.filterwarnings("ignore")


FILE_PATH = 'C:\учеба\Siemens\Fashion_Retail.csv'
raw_data = pd.read_csv(FILE_PATH, encoding= 'unicode_escape')
raw_data.head()
```

Out[82]:

	product ID	identifier	selling_price	original_price	currency	availability	color	category	bre
0	100	FJ5089	40	NaN	euro	InStock	Black	Clothing	Wome
1	101	BC0770	150	NaN	euro	InStock	Grey	Shoes	Won
2	102	GC7946	70	NaN	euro	InStock	White	Clothing	Kid
3	103	FV4744	160	NaN	euro	InStock	Black	Shoes	Five
4	104	GM0239	65	NaN	euro	InStock	Blue	Clothing	Me

Task 1: Please analyze the data in order to find which colors are the most popular among different genders and age groups, in the end please create a dashboard to show your results.

In [3]:
```
#get general information on our data
raw_data.info()
#so we observe that we have missing value on original_price
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 845 entries, 0 to 844
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   product ID      845 non-null    int64  
 1   identifier      845 non-null    object 
 2   selling_price   845 non-null    int64  
 3   original_price  829 non-null    float64
 4   currency        845 non-null    object 
 5   availability    845 non-null    object 
 6   color           845 non-null    object 
 7   category        845 non-null    object 
 8   breadcrumbs     845 non-null    object 
 9   country         845 non-null    object 
 10  language        845 non-null    object 
 11  average_rating  845 non-null    float64
 12  reviews_count   845 non-null    int64  
dtypes: float64(2), int64(3), object(8)
memory usage: 85.9+ KB
```

In [5]:
```
#get some general statistic first glance
raw_data.describe()
```

Out[5]:

	product ID	selling_price	original_price	average_rating	reviews_count
count	845.000000	845.000000	829.000000	845.000000	845.000000
mean	522.000000	53.192899	69.008444	4.608402	426.178698
std	244.074784	31.411645	40.490127	0.293795	1229.158277
min	100.000000	9.000000	14.000000	1.000000	1.000000
25%	311.000000	28.000000	35.000000	4.500000	19.000000
50%	522.000000	48.000000	65.000000	4.700000	68.000000
75%	733.000000	70.000000	90.000000	4.800000	314.000000
max	944.000000	240.000000	300.000000	5.000000	11750.000000

Preprocessing the data

In [6]:
```
#from the column 'breadcrumbs' we can get information about gender, and also get ac
raw_data['gender'] = raw_data['breadcrumbs'].str.split("/").str[0]
raw_data['clothesType'] = raw_data['breadcrumbs'].str.split("/").str[1]

raw_data.head()
```

Out[6]:

	product ID	identifier	selling_price	original_price	currency	availability	color	category	bre
0	100	FJ5089	40	NaN	euro	InStock	Black	Clothing	Women
1	101	BC0770	150	NaN	euro	InStock	Grey	Shoes	Wom
2	102	GC7946	70	NaN	euro	InStock	White	Clothing	Kid
3	103	FV4744	160	NaN	euro	InStock	Black	Shoes	Five
4	104	GM0239	65	NaN	euro	InStock	Blue	Clothing	Mer

In [7]:
```
#checking the data in new column, how much of each value we have
raw_data['gender'].value_counts()
```

Out[7]:
```
Women          347
Men            285
Kids           101
Originals       58
Training        25
Soccer          12
Swim             7
Running          6
Essentials       2
Five Ten         1
Sportswear       1
Name: gender, dtype: int64
```

In [8]:
```
#also take a look at the number of each value
raw_data['color'].value_counts()
```

Out[8]:
```
White         222
Black         187
Blue          104
Grey           81
Pink           62
Green          59
Purple         31
Red            25
Multicolor     20
Yellow         17
Orange         11
Burgundy        9
Beige           6
Multi           4
Gold            3
Turquoise       2
Silver          1
Brown           1
Name: color, dtype: int64
```

In [9]:
```
#as soon as we have Multicolor in different way of typing - make the common name fo
raw_data.loc[raw_data['color']=='Multicolor', ['color']]='Multi'
raw_data['color'].value_counts()
```

Out[9]:
```
White         222
Black         187
Blue          104
Grey           81
Pink           62
Green          59
Purple         31
Red            25
Multi          24
Yellow         17
Orange         11
Burgundy        9
Beige           6
Gold            3
Turquoise       2
Silver          1
Brown           1
Name: color, dtype: int64
```

In [10]:
```
#Making additional column Gender
raw_data['ageGroup']  = raw_data['gender'].replace(['Women', 'Men'], 'adult')
raw_data.head()
```

Out[10]:

	product ID	identifier	selling_price	original_price	currency	availability	color	category	bre
0	100	FJ5089	40	NaN	euro	InStock	Black	Clothing	Wome
1	101	BC0770	150	NaN	euro	InStock	Grey	Shoes	Wo
2	102	GC7946	70	NaN	euro	InStock	White	Clothing	Kid
3	103	FV4744	160	NaN	euro	InStock	Black	Shoes	Five
4	104	GM0239	65	NaN	euro	InStock	Blue	Clothing	Me

In [11]:
```
raw_data['ageGroup']  = raw_data['ageGroup'].replace(['Kids'], 'children')
```

In [12]:
```
#Checking if our new column, extracted from breadcrumbsis the same as category colu
(raw_data['category']==raw_data['clothesType']).unique()
```

Out[12]:
```
array([ True])
```

In [13]:
```
#As soon as it is the same, we can drop it
# Dropping all columns, which has no useful information for us
# As soon as columns currency', 'country', 'language' has the only one value - we d

significant_columns_data = raw_data.drop([ 'currency', 'clothesType',  'country',
significant_columns_data.head()
```

Out[13]:

	product ID	identifier	selling_price	original_price	availability	color	category	breadcrumbs
0	100	FJ5089	40	NaN	InStock	Black	Clothing	Women/Clothing
1	101	BC0770	150	NaN	InStock	Grey	Shoes	Women/Shoes
2	102	GC7946	70	NaN	InStock	White	Clothing	Kids/Clothing
3	103	FV4744	160	NaN	InStock	Black	Shoes	Five Ten/Shoes
4	104	GM0239	65	NaN	InStock	Blue	Clothing	Men/Clothing

In [14]:
```
#dropping all values from ageGroup which is not adultn or children
significant_columns_data_clean = significant_columns_data.loc[~((significant_columı
                                                                 &(significant_colu

#and reseting the index
significant_columns_data_clean.reset_index().drop(['index'], axis=1).head()
```

Out[14]:

	product ID	identifier	selling_price	original_price	availability	color	category	breadcrumbs
0	100	FJ5089	40	NaN	InStock	Black	Clothing	Women/Clothing
1	101	BC0770	150	NaN	InStock	Grey	Shoes	Women/Shoes
2	102	GC7946	70	NaN	InStock	White	Clothing	Kids/Clothing
3	104	GM0239	65	NaN	InStock	Blue	Clothing	Men/Clothing
4	105	FX7449	110	NaN	InStock	Grey	Shoes	Women/Shoes

In [15]:
```
#checking the value of new column ageGroup, it should be 'adult' and 'children'
significant_columns_data_clean['ageGroup'].unique()
```

Out[15]:
```
array(['adult', 'children'], dtype=object)
```

In [16]:
```
#checking the value of new column gender, it should be 'Women', 'Kids', 'Men'
significant_columns_data_clean['gender'].unique()
```

Out[16]:
```
array(['Women', 'Kids', 'Men'], dtype=object)
```

In [17]:
```
#Checking information on our new table
significant_columns_data_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 733 entries, 0 to 844
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   product ID      733 non-null    int64  
 1   identifier      733 non-null    object 
 2   selling_price   733 non-null    int64  
 3   original_price  718 non-null    float64
 4   availability    733 non-null    object 
 5   color           733 non-null    object 
 6   category        733 non-null    object 
 7   breadcrumbs     733 non-null    object 
 8   average_rating  733 non-null    float64
 9   reviews_count   733 non-null    int64  
 10  gender          733 non-null    object 
 11  ageGroup        733 non-null    object 
dtypes: float64(2), int64(3), object(7)
memory usage: 74.4+ KB
```

In [18]:
```
#we still have missing values. So we need to fill it, cause dropping can be crueti
```

In [19]:
```
#As soon as missing values is in original_price column, we will use columns sellin
#for prediction of missing values
to_predict_nulls_array = np.array([significant_columns_data_clean['selling_price'],
             significant_columns_data_clean['selling_price'], significant_columns_

to_predict_nulls_array = np.transpose(to_predict_nulls_array)
to_predict_nulls_array
```

Out[19]:
```
array([[ 40.,  nan,  40.,  35.],
       [150.,  nan, 150.,   4.],
       [ 70.,  nan,  70.,  42.],
       ...,
       [ 35.,  50.,  35., 190.],
       [ 40.,  50.,  40., 190.],
       [ 70., 100.,  70., 135.]])
```

In [20]:
```
# The KNNImputer class provides imputation for filling in missing values using the

imputer = KNNImputer(n_neighbors=3, weights="uniform")
```

In [21]:
```
predicted_nulls_array = imputer.fit_transform(to_predict_nulls_array)
predicted_nulls_array
```

Out[21]:
```
array([[ 40.        ,  46.66666667,  40.        ,  35.        ],
       [150.        , 156.66666667, 150.        ,   4.        ],
       [ 70.        ,  86.66666667,  70.        ,  42.        ],
       ...,
       [ 35.        ,  50.        ,  35.        , 190.        ],
       [ 40.        ,  50.        ,  40.        , 190.        ],
       [ 70.        , 100.        ,  70.        , 135.        ]])
```

In [22]:
```
#adding values to our table
significant_columns_data_clean.loc[:, ('original_price')]=predicted_nulls_array[:,
```

In [23]:
```
significant_columns_data_clean.head(10)
```

Out[23]:

	product ID	identifier	selling_price	original_price	availability	color	category	breadcrumbs
0	100	FJ5089	40	47	InStock	Black	Clothing	Women/Clothing
1	101	BC0770	150	157	InStock	Grey	Shoes	Women/Shoes
2	102	GC7946	70	87	InStock	White	Clothing	Kids/Clothing
4	104	GM0239	65	87	InStock	Blue	Clothing	Men/Clothing
5	105	FX7449	110	143	InStock	Grey	Shoes	Women/Shoes
6	106	GM5505	80	100	InStock	Black	Clothing	Men/Clothing
7	107	GP4975	60	75	InStock	Black	Clothing	Kids/Clothing
8	108	FS3923	40	47	InStock	Black	Clothing	Women/Clothing
9	109	GP4967	65	85	InStock	Black	Clothing	Men/Clothing
10	110	GL1127	60	72	InStock	Black	Clothing	Women/Clothing

In [24]:
```
significant_columns_data_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 733 entries, 0 to 844
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   product ID      733 non-null    int64
 1   identifier      733 non-null    object
 2   selling_price   733 non-null    int64
 3   original_price  733 non-null    int32
 4   availability    733 non-null    object
 5   color           733 non-null    object
 6   category        733 non-null    object
 7   breadcrumbs     733 non-null    object
 8   average_rating  733 non-null    float64
 9   reviews_count   733 non-null    int64
 10  gender          733 non-null    object
 11  ageGroup        733 non-null    object
dtypes: float64(1), int32(1), int64(3), object(7)
memory usage: 71.6+ KB
```

In [25]:
```
#lets check the corelation between average_rating and reviews_count
np.corrcoef(significant_columns_data_clean['average_rating'], significant_columns_
```

Out[25]:
```
array([[1.        , 0.01708148],
       [0.01708148, 1.        ]])
```

In [26]:
```
#we see that correlation is not null
```

In [27]:
```
significant_columns_data_clean.describe()
```

Out[27]:

	product ID	selling_price	original_price	average_rating	reviews_count
count	733.000000	733.000000	733.000000	733.000000	733.000000
mean	522.335607	52.107776	68.010914	4.600546	346.541610
std	247.603920	28.363528	37.331295	0.295503	1030.046448
min	100.000000	10.000000	14.000000	1.000000	1.000000
25%	298.000000	32.000000	40.000000	4.500000	18.000000
50%	531.000000	48.000000	65.000000	4.700000	67.000000
75%	740.000000	68.000000	87.000000	4.800000	251.000000
max	944.000000	240.000000	300.000000	5.000000	11750.000000

In [28]:
```
#make sure there are no duplicate rows in our table
no_duplicates_data = significant_columns_data_clean.drop_duplicates(subset = ['proc
```

In [29]:
```
no_duplicates_data.reset_index().drop(['index'], axis=1).head()
```

Out[29]:

	product ID	identifier	selling_price	original_price	availability	color	category	breadcrumbs
0	100	FJ5089	40	47	InStock	Black	Clothing	Women/Clothing
1	101	BC0770	150	157	InStock	Grey	Shoes	Women/Shoes
2	102	GC7946	70	87	InStock	White	Clothing	Kids/Clothing
3	104	GM0239	65	87	InStock	Blue	Clothing	Men/Clothing
4	105	FX7449	110	143	InStock	Grey	Shoes	Women/Shoes

In [30]:
```
#it can be logically assumed that the popularity of a particular color is made up r
#but also of the number of reviews for the product. Therefore, we introduce a new v
#To do this, we need to standardize these two columns, since the data is too differ

#Locking our two columns
no_duplicates_data_check=no_duplicates_data.loc[:,  ['reviews_count','average_ratin
no_duplicates_data_check
```

Out[30]:

	reviews_count	average_rating
0	35	4.5
1	4	4.8
2	42	4.9
4	11	4.7
5	30	4.9
...	...	...
840	151	4.3
841	135	4.7
842	190	4.7
843	190	4.7
844	135	4.7

733 rows × 2 columns

In [31]:
```
#Now we need to stundartise our data('reviews_count','average_rating') to the range

X_train = np.array(no_duplicates_data_check)
X_train
```

Out[31]:
```
array([[ 35. ,   4.5],
       [  4. ,   4.8],
       [ 42. ,   4.9],
       ...,
       [190. ,   4.7],
       [190. ,   4.7],
       [135. ,   4.7]])
```

In [32]:
```
#going to stundartise our data to the range of (0,1)
min_max_scaler = preprocessing.MinMaxScaler()
X_train_minmax = min_max_scaler.fit_transform(X_train)
X_train_minmax
```

Out[32]:
```
array([[2.89386331e-03, 8.75000000e-01],
       [2.55340880e-04, 9.50000000e-01],
       [3.48965869e-03, 9.75000000e-01],
       ...,
       [1.60864754e-02, 9.25000000e-01],
       [1.60864754e-02, 9.25000000e-01],
       [1.14052260e-02, 9.25000000e-01]])
```

In [33]:
```
#And now, as we standartised the data, making new column 'popularity', and putting
no_duplicates_data['popularity'] = np.sum(X_train_minmax, axis = 1)
no_duplicates_data
```

Out[33]:

	product ID	identifier	selling_price	original_price	availability	color	category	breadcrumb
0	100	FJ5089	40	47	InStock	Black	Clothing	Women/Clothin
1	101	BC0770	150	157	InStock	Grey	Shoes	Women/Shoe
2	102	GC7946	70	87	InStock	White	Clothing	Kids/Clothin
4	104	GM0239	65	87	InStock	Blue	Clothing	Men/Clothin
5	105	FX7449	110	143	InStock	Grey	Shoes	Women/Shoe
...	...	...	...	...	...	...	...	
840	940	FX2858	72	120	InStock	White	Shoes	Women/Shoe
841	941	H00667	70	100	InStock	White	Shoes	Women/Shoe
842	942	GZ7705	35	50	InStock	Black	Shoes	Kids/Shoe
843	943	GZ7706	40	50	InStock	Pink	Shoes	Kids/Shoe
844	944	FY6503	70	100	InStock	Black	Shoes	Women/Shoe

733 rows × 13 columns

In [34]:
```
#Now we need to plot the box_plot to find the outliners and delete it

no_duplicates_data_scaled = no_duplicates_data
```

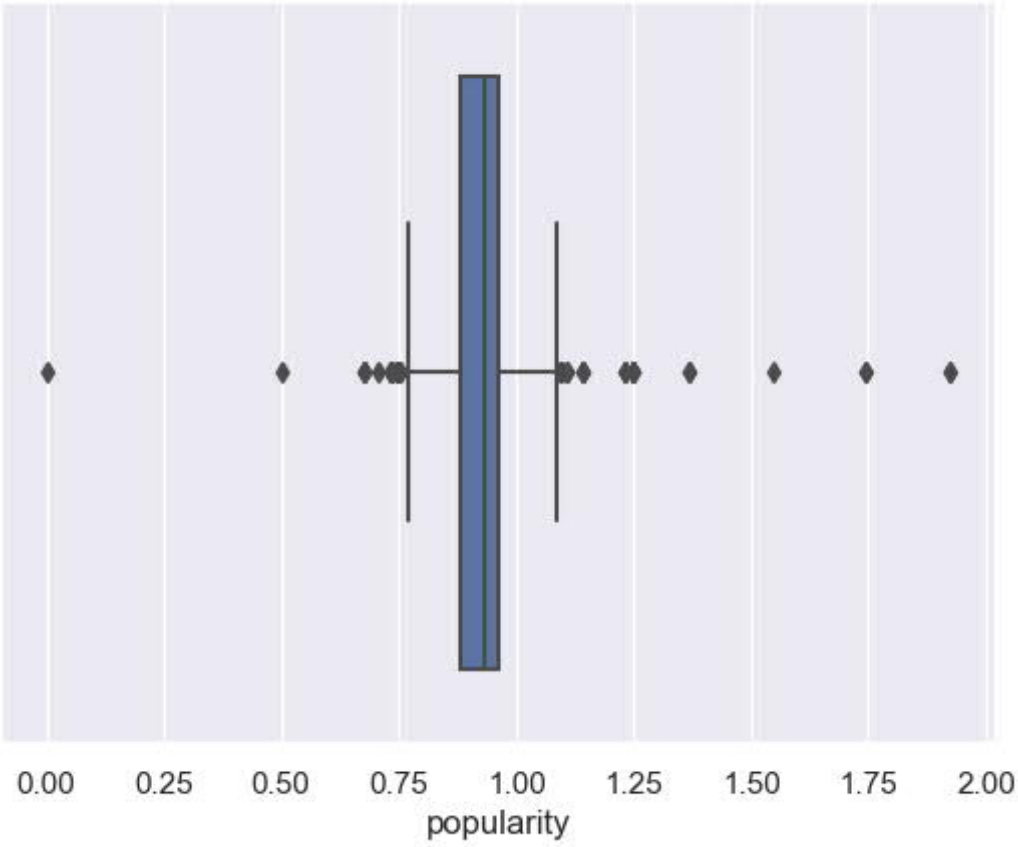
In [35]:
```
no_duplicates_data_scaled.head(10)
```

Out[35]:

	product ID	identifier	selling_price	original_price	availability	color	category	breadcrumbs
0	100	FJ5089	40	47	InStock	Black	Clothing	Women/Clothing
1	101	BC0770	150	157	InStock	Grey	Shoes	Women/Shoes
2	102	GC7946	70	87	InStock	White	Clothing	Kids/Clothing
4	104	GM0239	65	87	InStock	Blue	Clothing	Men/Clothing
5	105	FX7449	110	143	InStock	Grey	Shoes	Women/Shoes
6	106	GM5505	80	100	InStock	Black	Clothing	Men/Clothing
7	107	GP4975	60	75	InStock	Black	Clothing	Kids/Clothing
8	108	FS3923	40	47	InStock	Black	Clothing	Women/Clothing
9	109	GP4967	65	85	InStock	Black	Clothing	Men/Clothing
10	110	GL1127	60	72	InStock	Black	Clothing	Women/Clothing

In [36]:
```
sns.boxplot(no_duplicates_data_scaled['popularity'])
```

Out[36]:
```
<AxesSubplot:xlabel='popularity'>
```



We can see that some data points are outliers in the boxplot

Will use IQR method to display the data and the outliers (the shape of the data). Will use a mathematical formula to retrieve it. Find the outliers of the popularity column using the IQR method: Q1 = quantile(0.25) Q3 = quantile(0.75) IQR = Q3 - Q1

In [37]:
```
q1 = no_duplicates_data_scaled['popularity'].quantile(0.25)
q3 = no_duplicates_data_scaled['popularity'].quantile(0.75)

iqr = q3-q1
```

Find the upper fence and lower fence by adding the following code, and print all the data above the upper fence and below the lower fence.

Lower_Fence = Q1 - (1.5 * IQR)

Upper_Fence = Q3 + (1.5 * IQR)

In [38]:
```
lower_fence = q1-1.5*iqr
upper_fence = q3+1.5*iqr

#Lets see what we have got
print('q1_1=', q1)
print('q3_1=', q3)

print('iqr_1=', iqr)

print('lower_fence_1=',lower_fence)
print('upper_fence_1=',upper_fence)
```

```
q1_1= 0.8802770448548812
q3_1= 0.9639586347774278
iqr_1= 0.0836815899225466
lower_fence_1= 0.7547546599710613
upper_fence_1= 1.0894810196612477
```

In [39]:
```
#outliers_data are all the data above the upper fence and below the lower fence
outliers_data = no_duplicates_data_scaled.loc[(no_duplicates_data_scaled['popularit
                                               (no_duplicates_data_scaled['popularit

outliers_data.head()
```

Out[39]:

	product ID	identifier	selling_price	original_price	availability	color	category	breadcruml
11	111	B93219	20	25	InStock	White	Accessories	Men/Accessori
35	135	HG6676	120	150	InStock	Black	Clothing	Women/Clothir
36	136	HG6677	120	150	InStock	White	Clothing	Women/Clothir
42	142	GS1343	20	25	InStock	Black	Clothing	Women/Clothir
52	152	GV2043	20	28	InStock	Pink	Clothing	Kids/Clothir

In [40]:
```
#examine outliers_data

outliers_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 50 entries, 11 to 776
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   product ID      50 non-null     int64
 1   identifier      50 non-null     object
 2   selling_price   50 non-null     int64
 3   original_price  50 non-null     int32
 4   availability    50 non-null     object
 5   color           50 non-null     object
 6   category        50 non-null     object
 7   breadcrumbs     50 non-null     object
 8   average_rating  50 non-null     float64
 9   reviews_count   50 non-null     int64
 10  gender          50 non-null     object
 11  ageGroup        50 non-null     object
 12  popularity      50 non-null     float64
dtypes: float64(2), int32(1), int64(3), object(7)
memory usage: 5.3+ KB
```

In [41]:
```
#We see that we have 50 outliners, due to the amount of rows we can just delete it
```

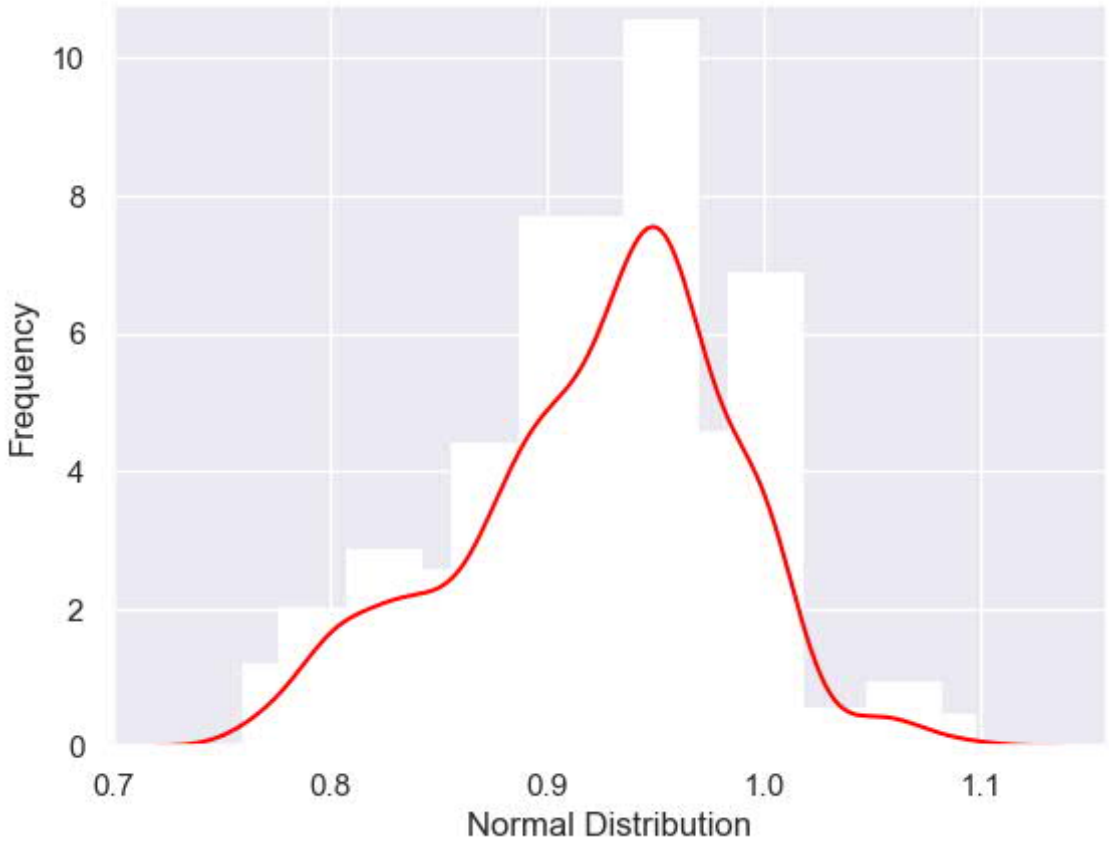
In [42]:
```
non_outliners_data = no_duplicates_data_scaled.loc[~((no_duplicates_data_scaled['p
                                                      (no_duplicates_data_scaled['p

#and reseting indexes
non_outliners_data.reset_index().drop(['index'], axis=1).head()
```

Out[42]:

	product ID	identifier	selling_price	original_price	availability	color	category	breadcrumbs
0	100	FJ5089	40	47	InStock	Black	Clothing	Women/Clothing
1	101	BC0770	150	157	InStock	Grey	Shoes	Women/Shoes
2	102	GC7946	70	87	InStock	White	Clothing	Kids/Clothing
3	104	GM0239	65	87	InStock	Blue	Clothing	Men/Clothing
4	105	FX7449	110	143	InStock	Grey	Shoes	Women/Shoes

In [43]:
```
#Lets check if we can say that distribution is normal
ax = sns.distplot(non_outliners_data['popularity'],
                  bins=20,
                  kde=True,
                  color='red',
                  hist_kws={"linewidth": 15,'alpha':1})
ax.set(xlabel='Normal Distribution', ylabel='Frequency')
#Let it say we can assume it is normal(bel shapped)
```

Out[43]:
```
[Text(0.5, 0, 'Normal Distribution'), Text(0, 0.5, 'Frequency')]
```



Now we can state our hypothesis

- H_0 : $\mu\_1 = \mu\_2 = \mu\_3$ (the three group means are equal: adult Women, Adult Men, Kids)
- H_1 : At least one of the means differ

In [44]:
```
#Lets separate the three samples (one for age and gender category) into a variable

Women = non_outliners_data[non_outliners_data['gender'] == 'Women']['popularity']
```

```
Men = non_outliners_data[non_outliners_data['gender'] == 'Men']['popularity']
Kids = non_outliners_data[non_outliners_data['gender'] == 'Kids']['popularity']
```

In [45]:
```
#Now, run a one-way ANOVA.
f_statistic, p_value = scipy.stats.f_oneway(Women, Men, Kids)
print("F_Statistic: {0}, P-Value: {1}".format(f_statistic,p_value))
```

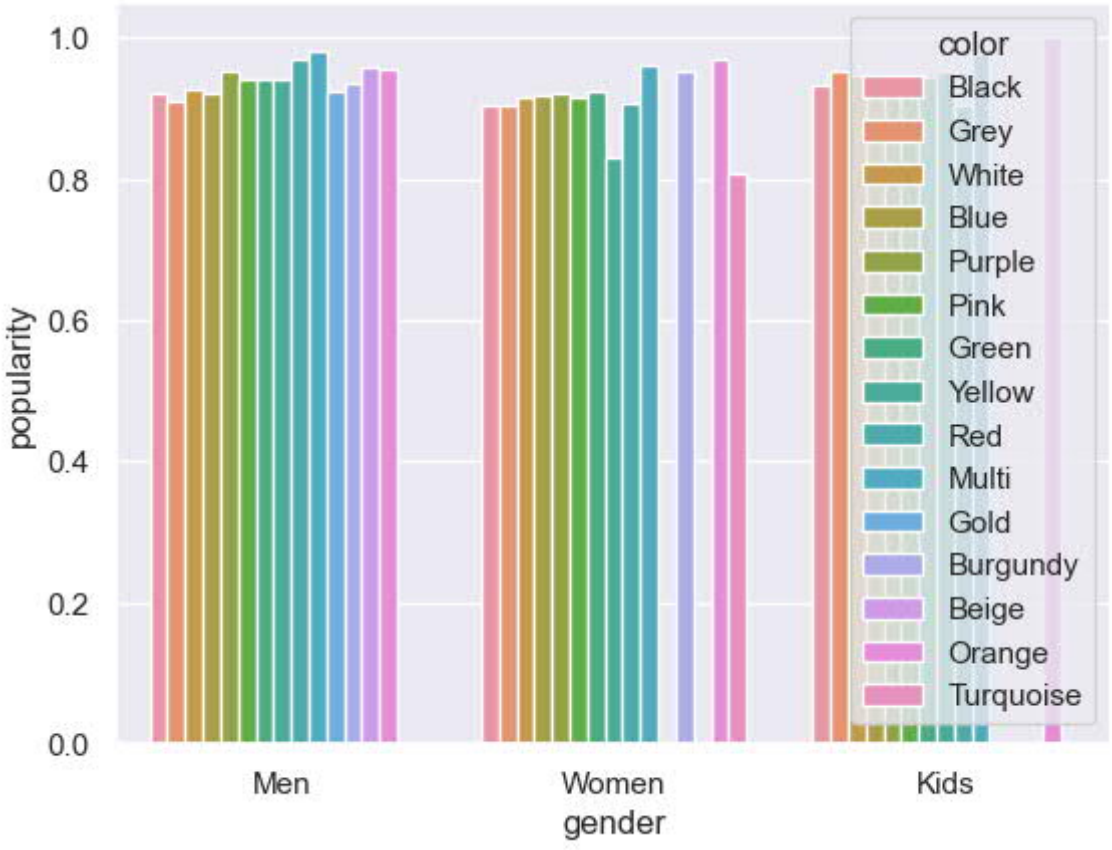
```
F_Statistic: 7.315097363652796, P-Value: 0.0007190957528595708
```

Conclusion: Since the p-value is less than 0.05, we will reject the null hypothesis as there is significant evidence that at least one of the means differ.

In [46]:
```
#Lets examine Popularity by gender Group mean
genderGroupMeanPopularity = non_outliners_data.groupby(['gender']).popularity.mean
genderGroupMeanPopularity
```

Out[46]:
```
gender
Kids     0.938583
Men      0.928335
Women    0.914788
Name: popularity, dtype: float64
```

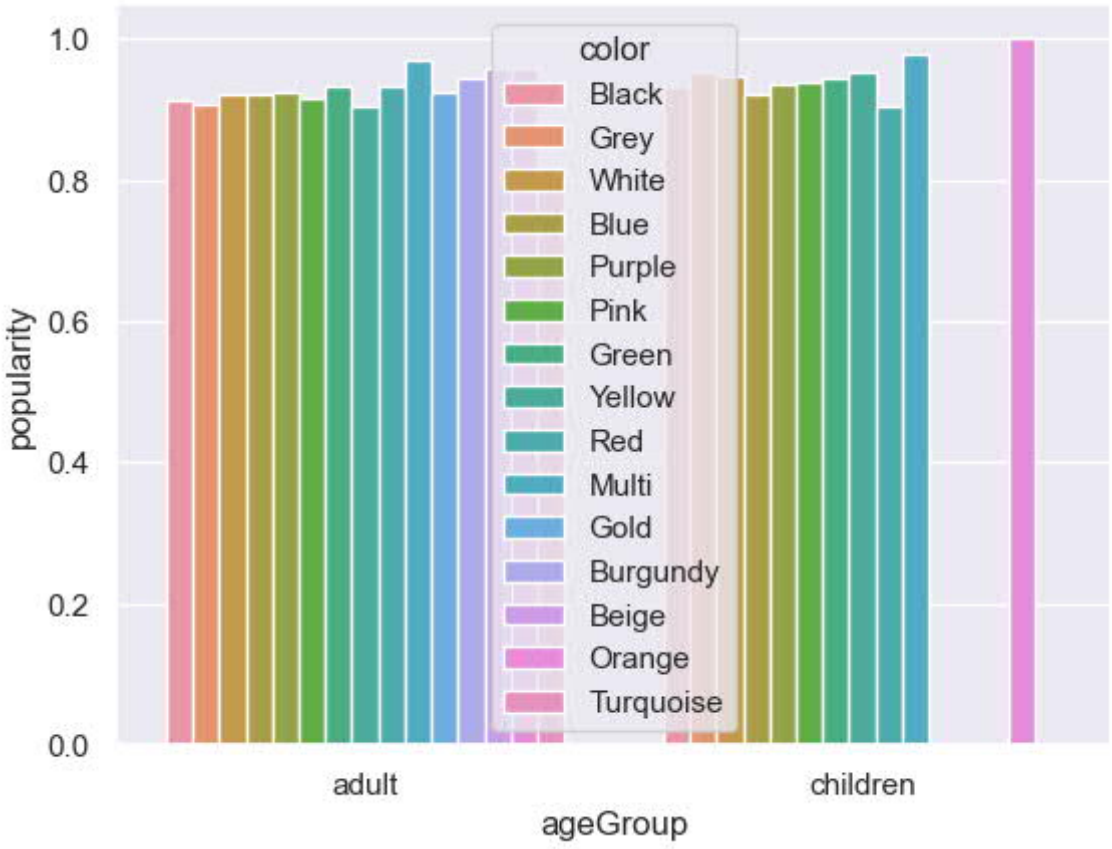
In [47]:
```
#Lets make visualisation on the popularity of colors amoung Women, Men, Kids

sns.barplot(
    x="gender",
    y="popularity",
    hue="color",
    ci=None,
    order=["Men","Women", "Kids"],
    data=non_outliners_data
)
```



In [97]:
```
sns.barplot(
    x="ageGroup",
    y="popularity",
```

```
    hue="color",
    ci=None,
    data=non_outliners_data
)
```

Out[97]: <AxesSubplot:xlabel='ageGroup', ylabel='popularity'>



Conclusion on Task 1: analyze the data in order to find which colors are the most popular among different genders and age groups

Due to the data - we observe, that the three group means are not equal: adult Women, Adult Men, Kids(there is significant evidence that at least one of the means differ due to the Anova)

As we can see, among the adult - high popularity is at multicolor option. while for kids high popularity is for orange color

We also see, that orange is also popular for Women/adult.As much as multicolor is the high option for kids as well

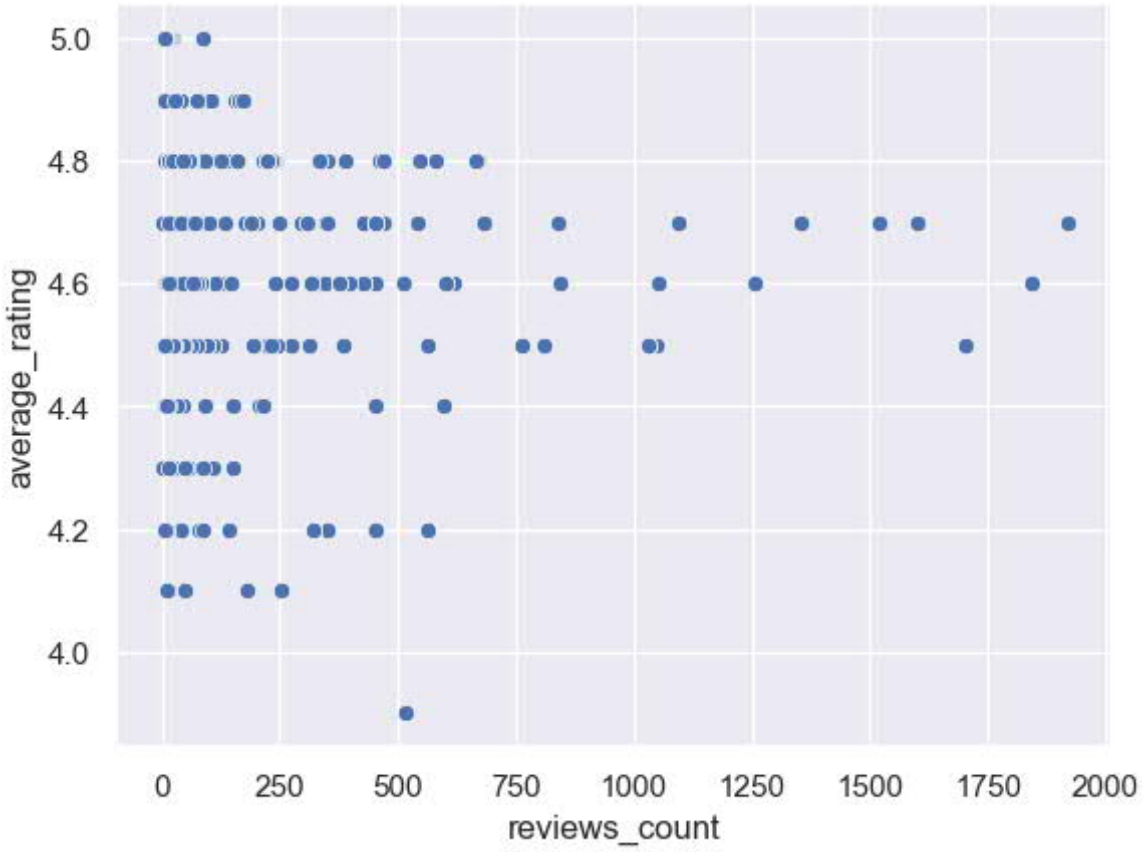
Task 2: Please analyze the correlation between ratings and reviews to determine which factors are most important to customers, in the end please create a dashboard to show your results.

Correlation: Using the dataset, Is rating evaluation score correlated with review count?

State the hypothesis:

- H_0 : rating evaluation score is not correlated with review count
- H_1 : rating evaluation score is correlated with review count

In [48]:
```
#Since they are both continuous variables we can use a pearson correlation test and
ax = sns.scatterplot(x="reviews_count", y="average_rating", data=non_outliners_data
```



In [49]:
```
#Lets see the pearson correlation
scipy.stats.pearsonr(non_outliners_data['reviews_count'], non_outliners_data['aver
```

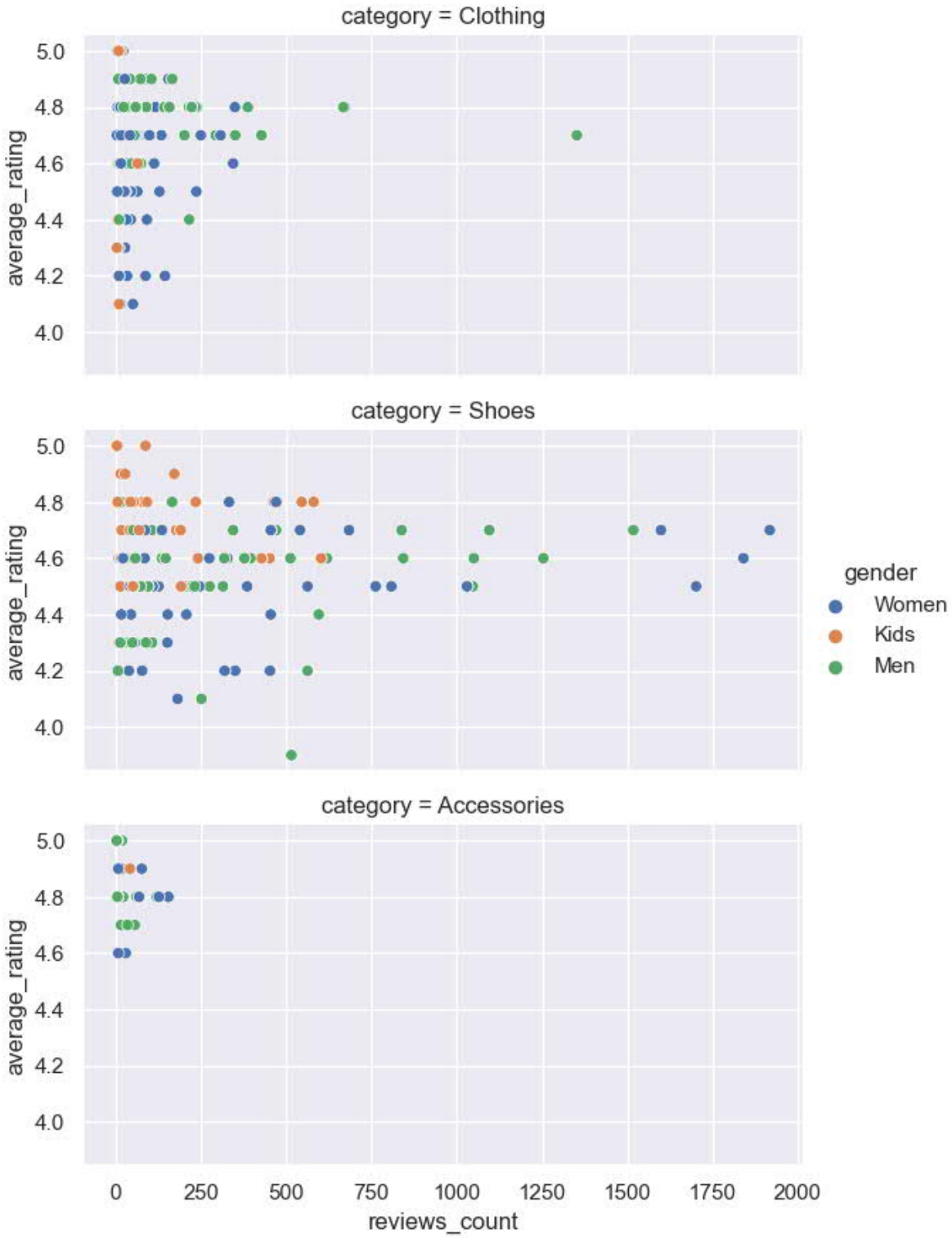
Out[49]:
```
PearsonRResult(statistic=-0.14807598309176218, pvalue=0.00010266684983972688)
```

Conclusion: Since the p-value < 0.05, we reject the Null hypothesis and conclude that there exists a relationship between reviews_count and average_rating score.

A negative correlation between two variables means that the variables move in opposite directions. An increase in one variable leads to a decrease in the other variable and vice versa

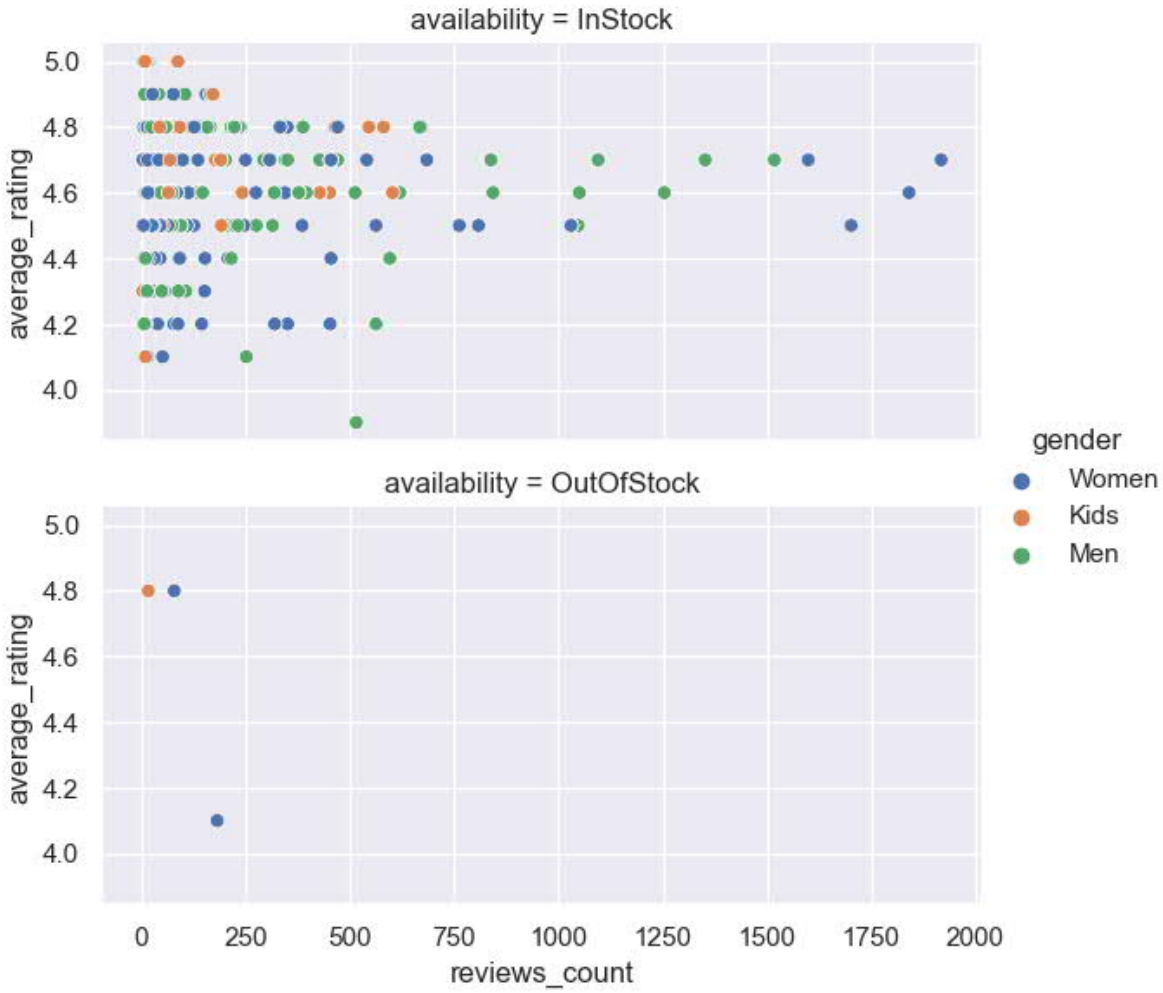
In [50]:
```
sns.relplot(x="reviews_count", y="average_rating", hue="gender",
            row="category",
            data=non_outliners_data, height = 3, aspect = 2)
```

Out[50]:
```
<seaborn.axisgrid.FacetGrid at 0x1eaeeb1ed30>
```



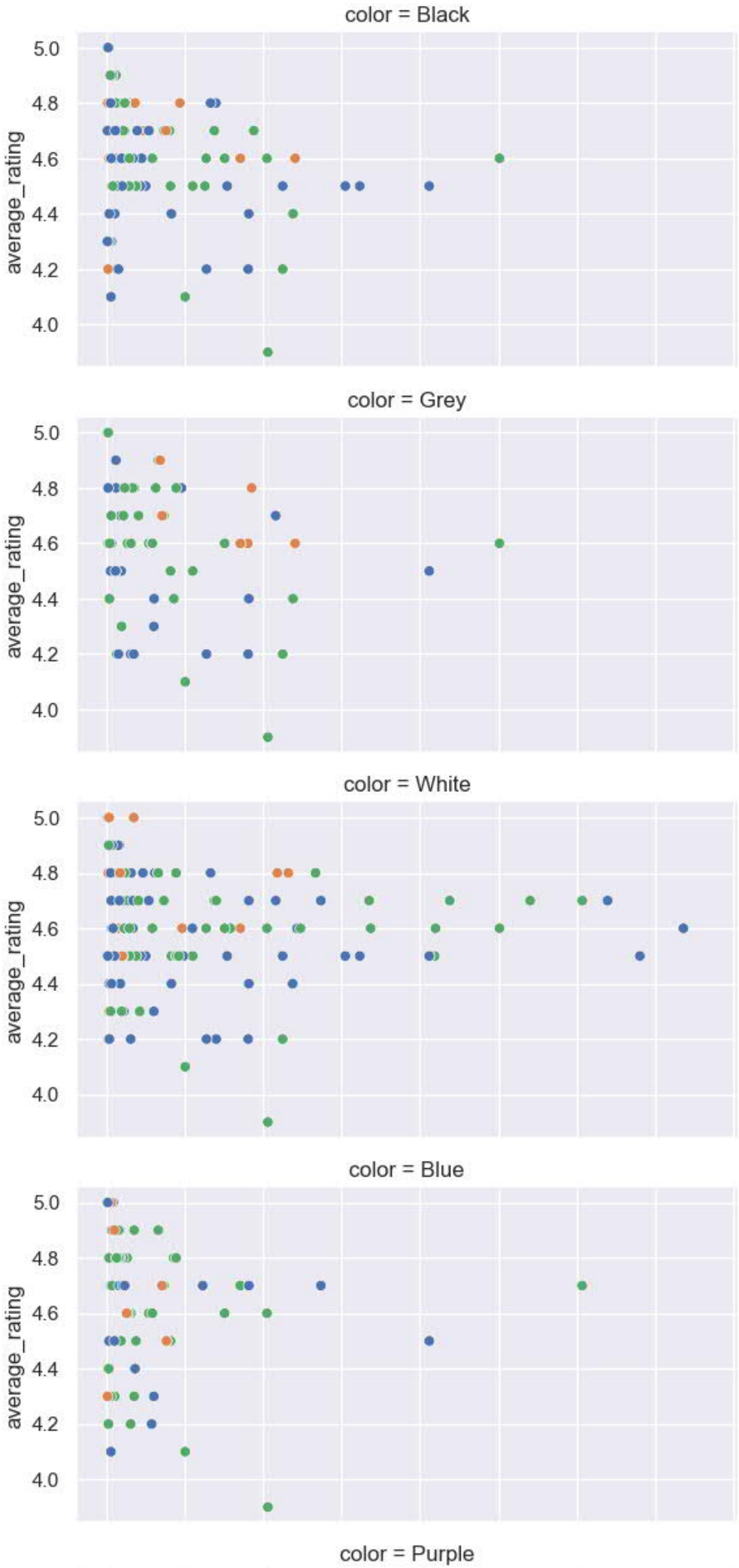
In [51]:
```
sns.relplot(x="reviews_count", y="average_rating", hue="gender",
            row="availability",
            data=non_outliners_data, height = 3, aspect = 2)
```

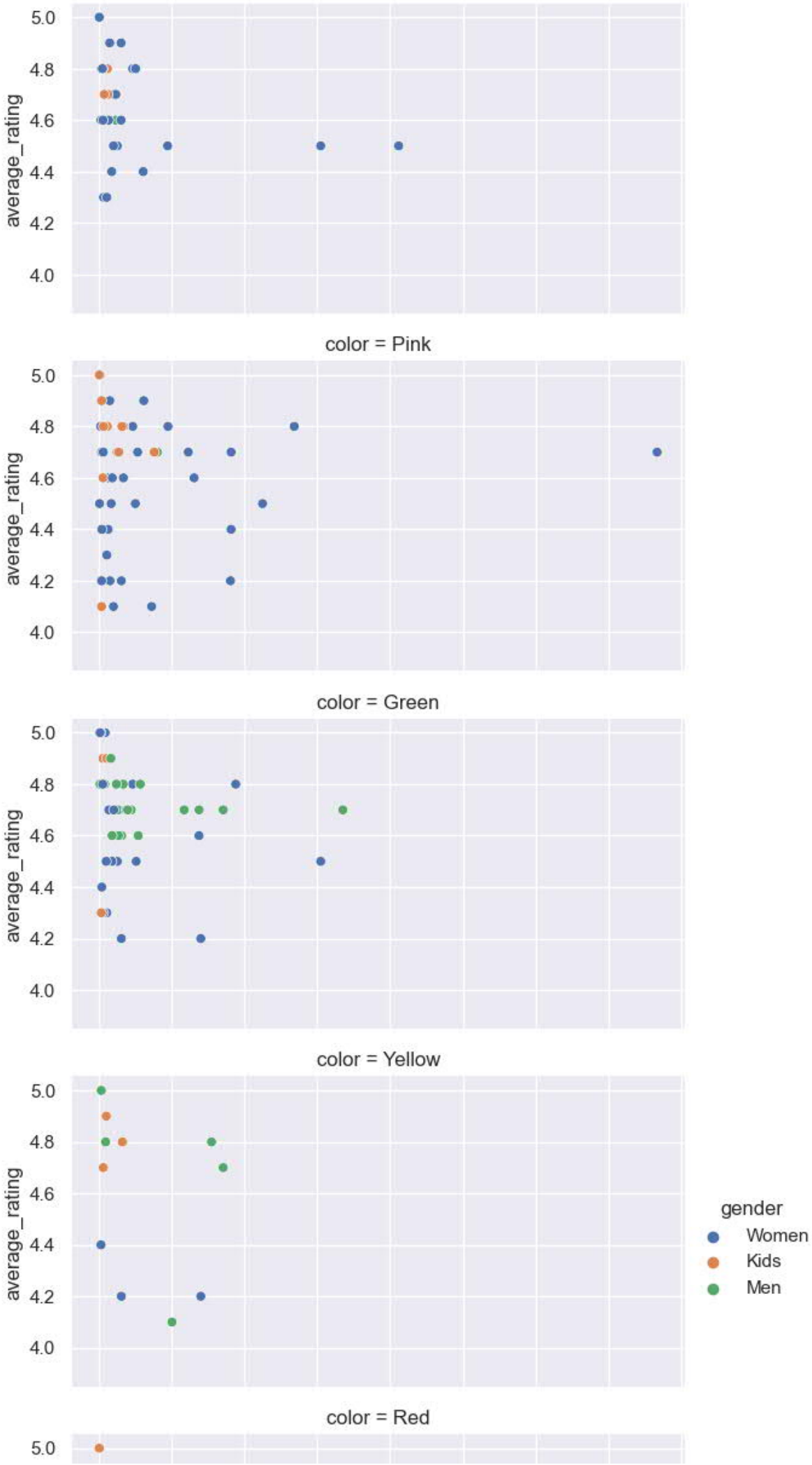
Out[51]:
```
<seaborn.axisgrid.FacetGrid at 0x1eaeec2f1c0>
```



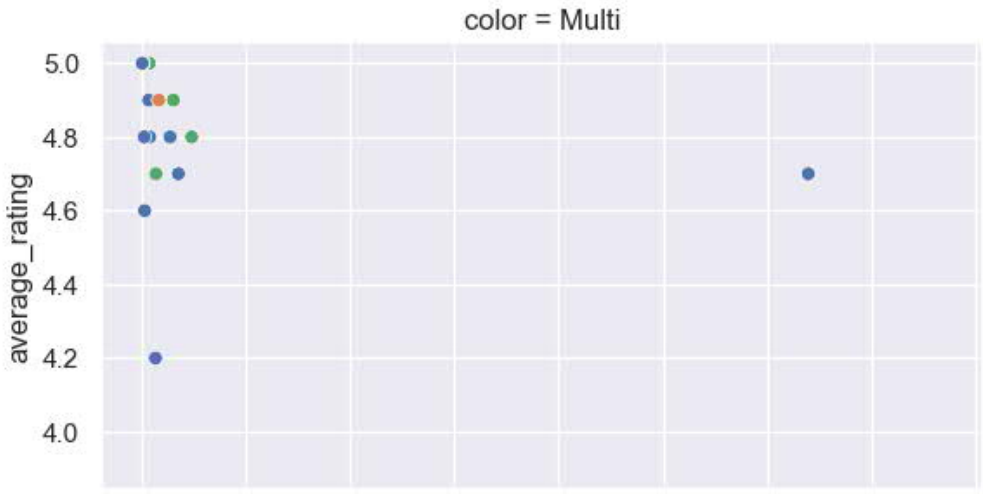
In [52]:
```
sns.relplot(x="reviews_count", y="average_rating", hue="gender",
            row="color",
            data=non_outliners_data, height = 3, aspect = 2)
```

Out[52]:
```
<seaborn.axisgrid.FacetGrid at 0x1eaeeb393d0>
```

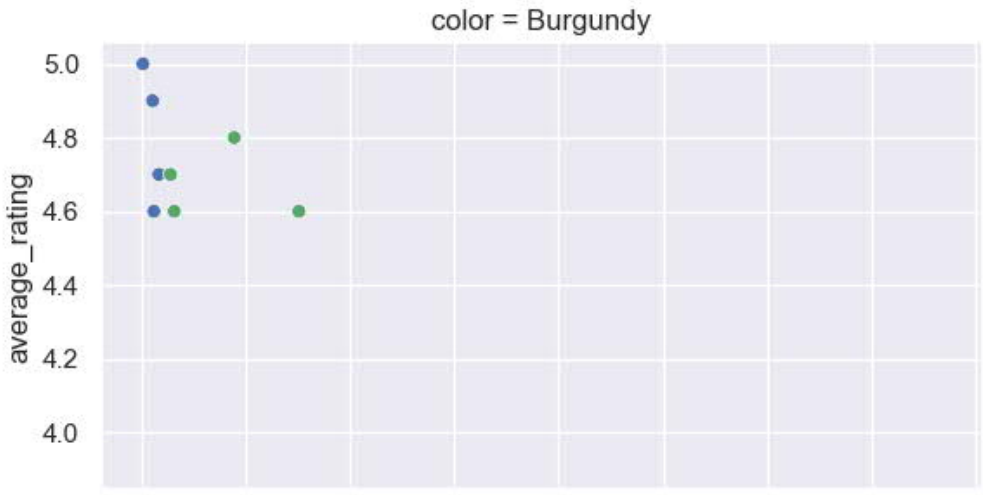

color = Black
color = Grey
color = White
color = Blue
color = Purple
average_rating
5.0
4.8
4.6
4.4
4.2
4.0


average_rating
5.0
4.8
4.6
4.4
4.2
4.0
color = Pink
color = Green
color = Yellow
color = Red
gender
Women
Kids
Men

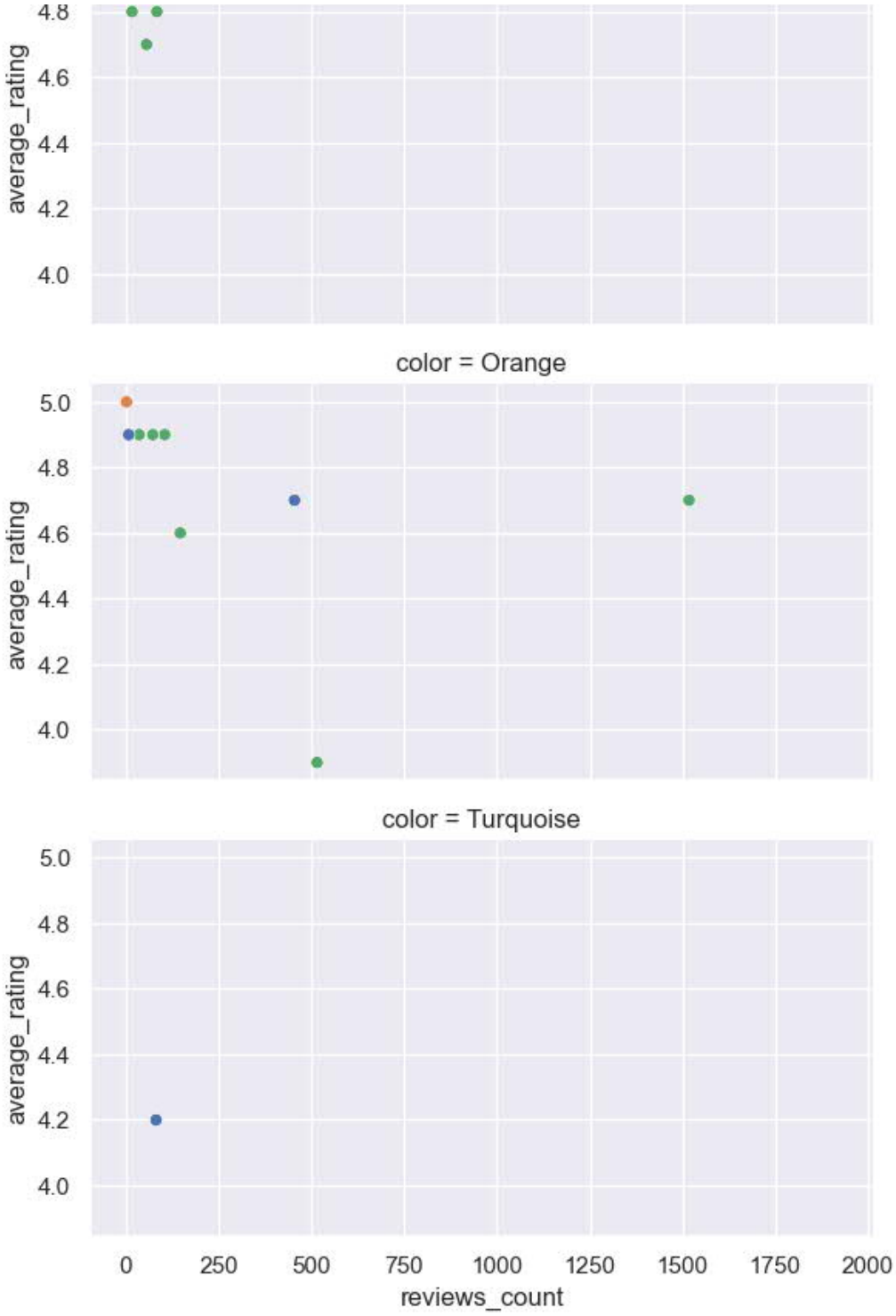

average_rating
4.8
4.6
4.4
4.2
4.0


color = Multi
average_rating
5.0
4.8
4.6
4.4
4.2
4.0


color = Gold
average_rating
5.0
4.8
4.6
4.4
4.2
4.0


color = Burgundy
average_rating
5.0
4.8
4.6
4.4
4.2
4.0


color = Beige
5.0



Conclusion on Task 2: correlation between ratings and reviews

There exists a relationship between reviews_count and average_rating score.

A negative correlation between two variables means that the variables move in opposite directions. An increase in one variable leads to a decrease in the other variable and vice versa

Due to the visualisation, we see that type of corelation on shoes category stronger, than on Clothes. Depending on color - we see negotive corelation between reviews_count and average_rating score on white, Black and Grey color goods

Task 3: Please create a modell to predict which combinatin of selling_price,color,category,breadcrumbs could be most popular products for next year?

Lets make the regression analysis for that purpose

The goal of regression analysis is to describe the relationship between one set of variables called the dependent variables, and another set of variables, called independent or explanatory variables

In [53]:
```
# Descriptive statistics are very useful for initial exploration of the variables
non_outliners_data.describe(include='all')

# categorical variables don't have some types of numerical descriptives
# and numerical variables don't have some types of categorical descriptives
```

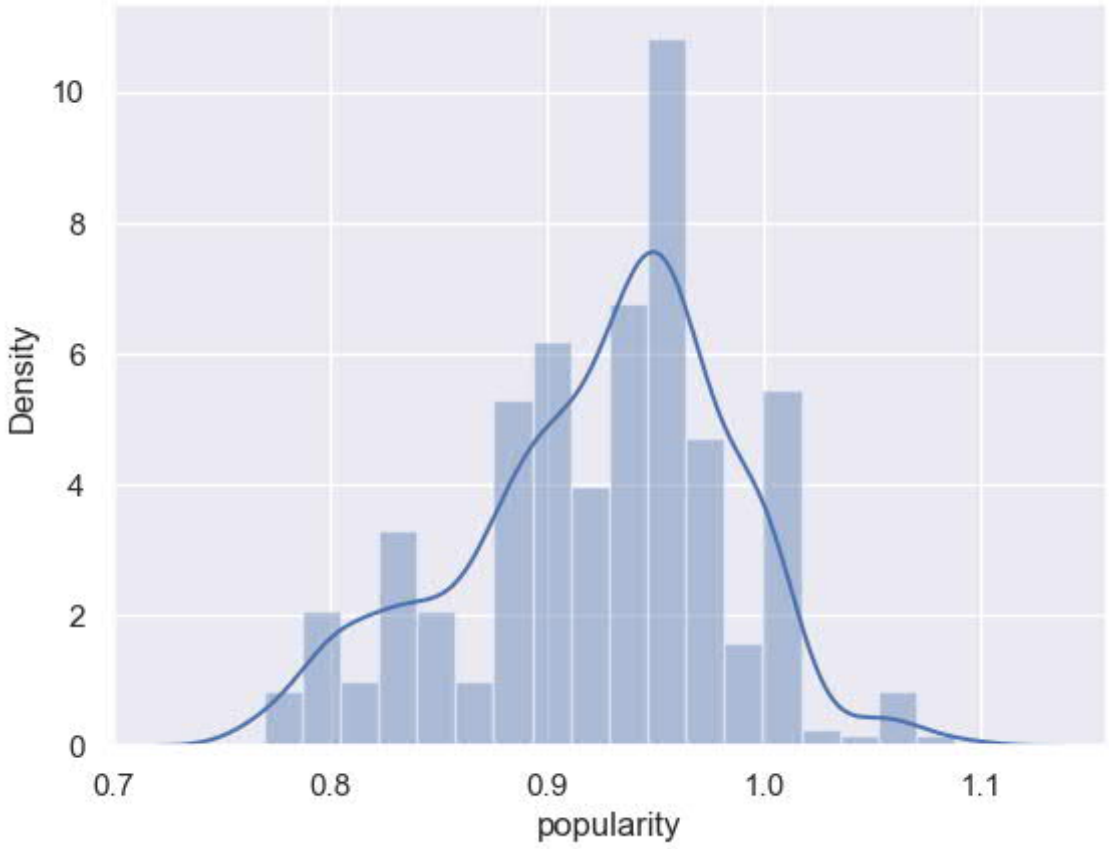
Out[53]:

	product ID	identifier	selling_price	original_price	availability	color	category	bread
count	683.000000	683	683.000000	683.000000	683	683	683	
unique	NaN	683	NaN	NaN	2	15	3	
top	NaN	FJ5089	NaN	NaN	InStock	White	Shoes	Women/(
freq	NaN	1	NaN	NaN	680	175	337	
mean	528.638360	NaN	51.754026	67.527086	NaN	NaN	NaN	
std	249.230857	NaN	27.602825	36.131769	NaN	NaN	NaN	
min	100.000000	NaN	10.000000	14.000000	NaN	NaN	NaN	
25%	299.500000	NaN	32.000000	40.000000	NaN	NaN	NaN	
50%	543.000000	NaN	48.000000	65.000000	NaN	NaN	NaN	
75%	746.500000	NaN	64.000000	85.000000	NaN	NaN	NaN	
max	944.000000	NaN	240.000000	300.000000	NaN	NaN	NaN	

Determining the variables of interest

In [54]:
```
#display the probability distribution function (PDF) of a variable
# The PDF will show us how that variable is distributed
# This makes it very easy to spot anomalies, such as outliers

sns.distplot(non_outliners_data['popularity'])
```

Out[54]:
```
<AxesSubplot:xlabel='popularity', ylabel='Density'>
```



Checking the OLS assumptions

In [55]:
```
f, (ax1, ax2, ax3, ax4) = plt.subplots(1, 4, sharey=True, figsize =(15,3))
ax1.scatter(non_outliners_data['selling_price'],non_outliners_data['popularity'])
ax1.set_title('Popularity and selling price')
ax2.scatter(non_outliners_data['color'],non_outliners_data['popularity'])
ax2.set_title('Popularity and color')
ax3.scatter(non_outliners_data['category'],non_outliners_data['popularity'])
ax3.set_title('Popularity and category')
ax4.scatter(non_outliners_data['breadcrumbs'],non_outliners_data['popularity'])
ax4.set_title('Popularity and breadcrumbs')


plt.show()
```

In [56]:
```
regressionAnalys = non_outliners_data.loc[:, ['selling_price', 'color', 'category'

regressionAnalys
```

Out[56]:

	selling_price	color	category	gender	popularity
0	40	Black	Clothing	Women	0.877894
1	150	Grey	Shoes	Women	0.950255
2	70	White	Clothing	Kids	0.978490
4	65	Blue	Clothing	Men	0.925851
5	110	Grey	Shoes	Women	0.977468
...	...	...	...	...	...
840	72	White	Shoes	Women	0.837767
841	70	White	Shoes	Women	0.936405
842	35	Black	Shoes	Kids	0.941086
843	40	Pink	Shoes	Kids	0.941086
844	70	Black	Shoes	Women	0.936405

683 rows × 5 columns

In [57]:
```
regressionAnalys.columns.values
```

Out[57]:
```
array(['selling_price', 'color', 'category', 'gender', 'popularity'],
      dtype=object)
```

Create dummy variables

In [58]:
```
# To include the categorical data in the regression, we create dummies
data_with_dummies = pd.get_dummies(regressionAnalys, drop_first=True)

# Here's the result
data_with_dummies.head()
```

Out[58]:

	selling_price	popularity	color_Black	color_Blue	color_Burgundy	color_Gold	color_Green	col
0	40	0.877894	1	0	0	0	0	
1	150	0.950255	0	0	0	0	0	
2	70	0.978490	0	0	0	0	0	
4	65	0.925851	0	1	0	0	0	
5	110	0.977468	0	0	0	0	0	

In [59]:
```
# To make our data frame more organized rearrange a bit
data_with_dummies.columns.values
```

Out[59]:
```
array(['selling_price', 'popularity', 'color_Black', 'color_Blue',
       'color_Burgundy', 'color_Gold', 'color_Green', 'color_Grey',
       'color_Multi', 'color_Orange', 'color_Pink', 'color_Purple',
       'color_Red', 'color_Turquoise', 'color_White', 'color_Yellow',
       'category_Clothing', 'category_Shoes', 'gender_Men',
       'gender_Women'], dtype=object)
```

In [60]:
```
# To make the code a bit more parametrized,  declare a new variable that will conte

cols = ['popularity','selling_price', 'color_Black', 'color_Blue',
```

```
       'color_Burgundy', 'color_Gold', 'color_Green', 'color_Grey',
       'color_Multi', 'color_Orange', 'color_Pink', 'color_Purple',
       'color_Red', 'color_Turquoise', 'color_White', 'color_Yellow',
       'category_Clothing', 'category_Shoes', 'gender_Men',
       'gender_Women']
```

In [61]:
```
# To implement the reordering, create a new df = data_preprocessed,
#which is equal to the old one but with the new order of features

data_preprocessed = data_with_dummies[cols]
data_preprocessed.head()
```

Out[61]:

	popularity	selling_price	color_Black	color_Blue	color_Burgundy	color_Gold	color_Green	col
0	0.877894	40	1	0	0	0	0	
1	0.950255	150	0	0	0	0	0	
2	0.978490	70	0	0	0	0	0	
4	0.925851	65	0	1	0	0	0	
5	0.977468	110	0	0	0	0	0	

Linear regression model

In [62]:
```
#Declare the inputs and the targets
# The target(s) (dependent variable) is 'popularity'
targets = data_preprocessed['popularity']

# The inputs are everything BUT the dependent variable, so we can simply drop it
inputs = data_preprocessed.drop(['popularity'],axis=1)
```

In [63]:
```
#Scale the data

# Create a scaler object
scaler = StandardScaler()

# Fit the inputs (calculate the mean and standard deviation feature-wise)
scaler.fit(inputs)
```

Out[63]:
```
StandardScaler()
```

In [64]:
```
# Scale the features and store them in a new variable (the actual scaling procedure
inputs_scaled = scaler.transform(inputs)
```

Train Test Split

In [65]:
```
# Split the variables with an 80-20 split and some random state

x_train, x_test, y_train, y_test = train_test_split(inputs_scaled, targets, test_s
```

Create the regression

In [66]:
```
# Create a linear regression object
reg = LinearRegression()
# Fit the regression with the scaled TRAIN inputs and targets
reg.fit(x_train,y_train)
```

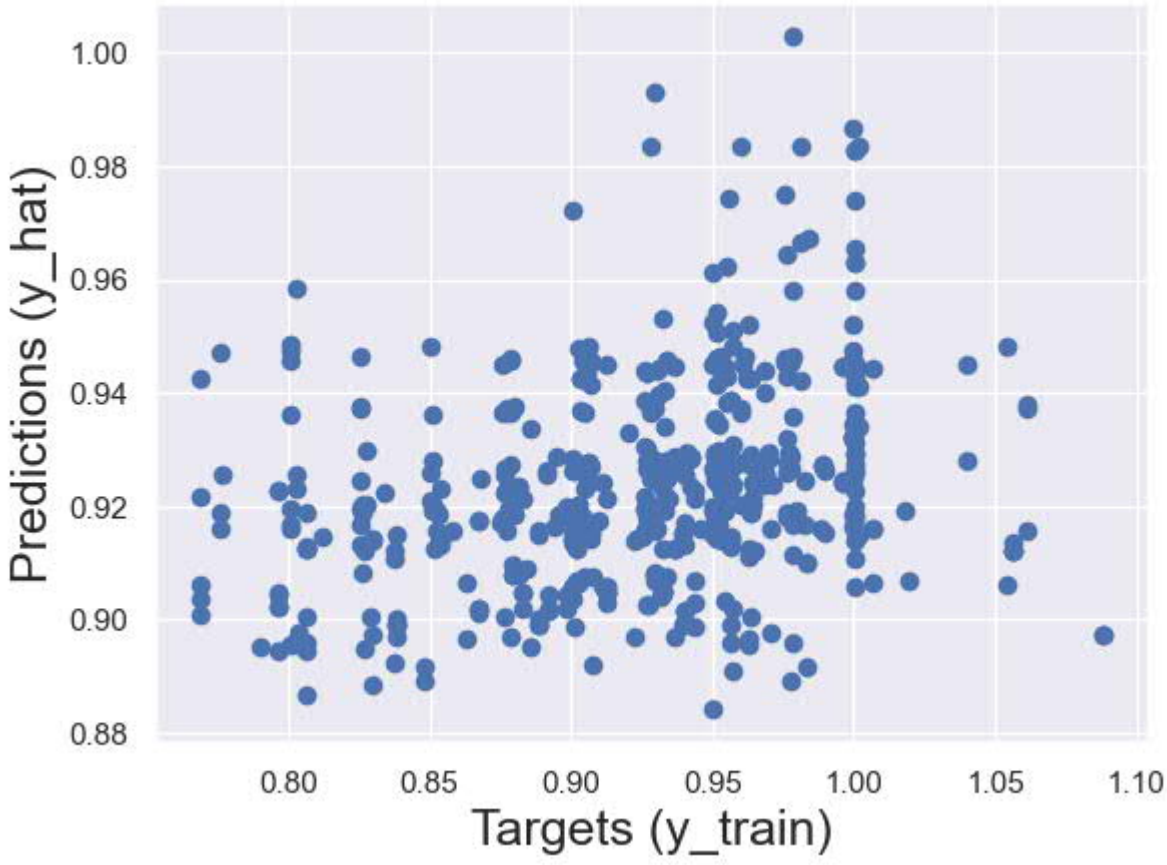
Out[66]:
```
LinearRegression()
```

In [67]:
```
# Let's check the outputs of the regression
# store them in y_hat
y_hat = reg.predict(x_train)
y_hat
```

Out[67]:
```
array([0.91435371, 0.91251221, 0.92147277, 0.94342415, 0.88856722,
       0.92474199, 0.92555311, 0.9189355 , 0.90056384, 0.90647606,
       0.89644223, 0.9386853 , 0.94538506, 0.91119685, 0.92147277,
       0.91946164, 0.92002007, 0.92410348, 0.89909351, 0.92644917,
       0.92265659, 0.94276291, 0.94499901, 0.92593916, 0.91762014,
       0.93393744, 0.91751056, 0.91487985, 0.92252505, 0.9160445 ,
       0.92747359, 0.94630855, 0.9396683 , 0.92500228, 0.92763424,
       0.92747359, 0.98266964, 0.9384258 , 0.91897359, 0.92642131,
       0.90251343, 0.9167048 , 0.92552842, 0.89552148, 0.90413144,
       0.91251221, 0.90740369, 0.93235902, 0.92895113, 0.92357734,
       0.91251221, 0.89910081, 0.91895745, 0.90790847, 0.92934574,
       0.94499901, 0.9350036 , 0.90623601, 0.90582526, 0.91435371,
       0.94447287, 0.94826823, 0.94565088, 0.92986221, 0.90700908,
       0.95252047, 0.9189355 , 0.92410348, 0.89502321, 0.91867521,
       0.93668757, 0.91474831, 0.89591609, 0.91910513, 0.92603525,
       0.92803615, 0.90266841, 0.92973756, 0.94538506, 0.94707433,
       0.90963291, 0.92593916, 0.9370943 , 0.98359039, 0.9193682 ,
       0.94513055, 0.92751759, 0.92460079, 0.89725201, 0.92789885,
       0.92656139, 0.9289569 , 0.92326445, 0.94577967, 0.93852108,
       0.92777308, 0.91461096, 0.91435371, 0.92104007, 0.92948304,
       0.92237157, 0.8922771 , 0.91800745, 0.93393744, 0.94630855,
       0.93629297, 0.92237157, 0.90831519, 0.92042049, 0.94579179,
       0.92147277, 0.92684503, 0.91907244, 0.90002156, 0.92699145,
       0.90882921, 0.90465758, 0.94538506, 0.90461358, 0.94368366,
       0.91590288, 0.91251221, 0.88672573, 0.9245102 , 0.91683093,
       0.92409767, 0.91330142, 0.9238346 , 0.91343296, 0.92011202,
       0.92011202, 0.89749452, 0.91527446, 0.90237748, 0.93790996,
       0.91855703, 0.91905974, 0.91630479, 0.94589781, 0.93195229,
       0.88935546, 0.94053037, 0.93235902, 0.9171321 , 0.91789935,
       0.93092993, 0.94420124, 0.9193682 , 0.93446359, 0.92541302,
       0.99289332, 0.92342164, 0.89696837, 0.946581  , 0.96109854,
       0.94801825, 0.92699145, 0.8968647 , 0.90698772, 0.91251221,
       0.90582526, 0.92736992, 0.92777308, 0.91435371, 0.94328905,
       0.98359039, 0.91590288, 0.92104007, 0.95419319, 0.91527446,
       0.91435371, 0.92302236, 0.94111438, 0.92122181, 0.9265807 ,
       0.89594395, 0.90766955, 0.92789885, 0.91382756, 0.90424684,
       0.91566325, 0.97226219, 0.9511751 , 0.91683093, 0.89922062,
       0.93668757, 0.90307916, 0.9230512 , 0.90461358, 0.8968574 ,
       0.89095101, 0.96312374, 0.9866082 , 0.9425034 , 0.91566325,
       0.93695065, 0.90740369, 0.91829396, 0.91831592, 0.91735707,
       0.91515504, 0.92278231, 0.92803893, 0.8987062 , 0.93669969,
       0.90307916, 0.90186305, 0.94526564, 0.92921142, 0.98359039,
       0.91776782, 0.98359039, 0.90582526, 0.89479242, 0.92462381,
       0.91933289, 0.94516896, 0.89739363, 0.92605456, 0.96558113,
       0.9238346 , 0.94839698, 0.92550911, 0.91318603, 0.92789731,
       0.93668757, 0.92828337, 0.92868527, 0.9278682 , 0.9423683 ,
       0.92512693, 0.91840936, 0.92721907, 0.96289068, 0.90635141,
       0.91971741, 0.8987062 , 0.90057998, 0.92621079, 0.92265659,
       0.94276291, 0.94447287, 0.90382437, 0.89909351, 0.91487985,
       0.94214364, 0.91435371, 0.91906704, 0.95812539, 0.92880992,
       0.91513711, 0.97436676, 0.93449065, 0.92410348, 0.90071152,
       0.93762044, 0.92159849, 0.9386853 , 0.92552842, 0.94433278,
       0.9160445 , 0.97508418, 0.93993138, 0.94579179, 0.90198287,
       0.96650188, 0.9423683 , 0.91785535, 0.94811408, 0.92410348,
       0.92973756, 0.91224914, 0.9035613 , 0.93645513, 0.91224914,
       0.93656816, 0.91435371, 0.91509311, 0.89571316, 0.91671553,
       0.91291893, 0.93380591, 0.91735707, 0.91998778, 0.94602204,
       0.93289774, 0.91946164, 0.92288846, 0.90237748, 0.89460074,
       0.94761056, 1.00277596, 0.90148459, 0.94258959, 0.91067071,
       0.9243446 , 0.92629832, 0.92672079, 0.93037387, 0.91408482,
       0.92011202, 0.94158265, 0.92357734, 0.91251221, 0.94381875,
       0.91751056, 0.9375136 , 0.91566906, 0.91289882, 0.89527898,
       0.91945434, 0.94747902, 0.9230378 , 0.89725201, 0.9265807 ,
       0.92552842, 0.90751386, 0.92724694, 0.96213871, 0.91831592,
```

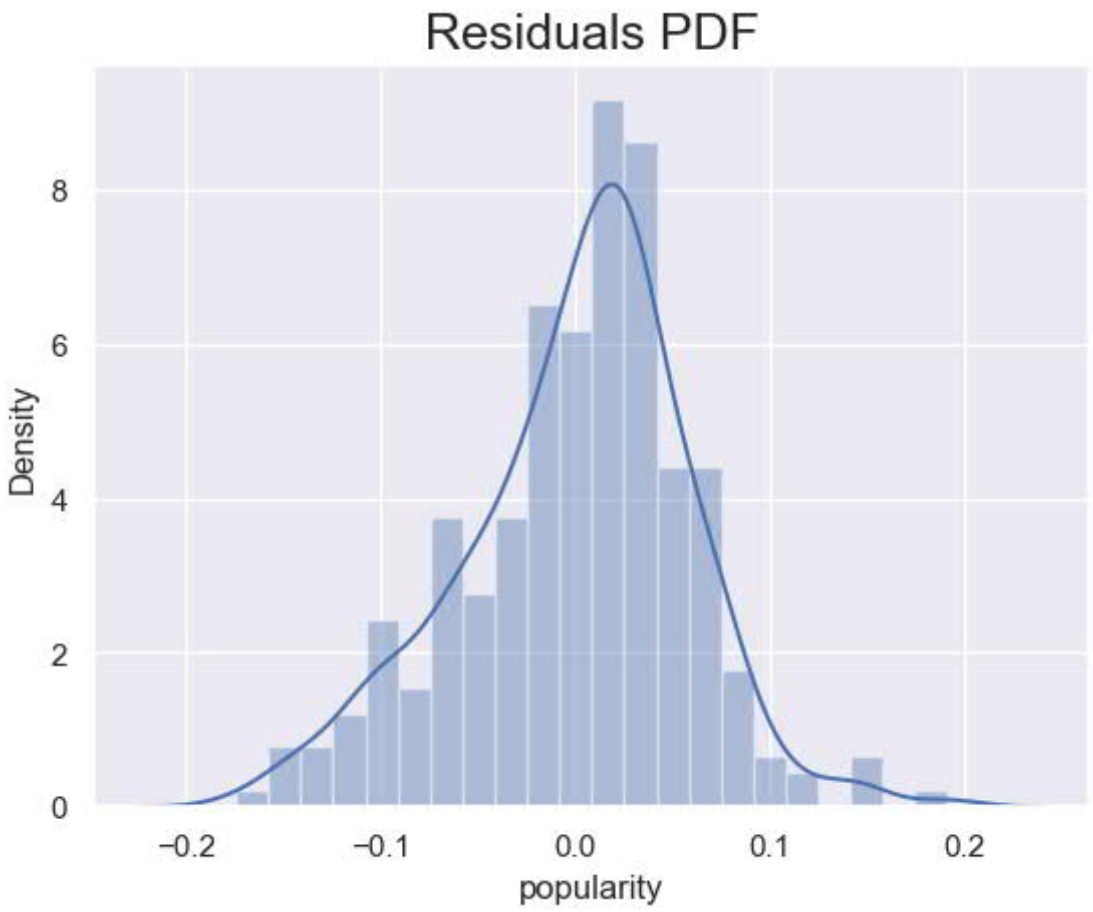
```
0.91856284, 0.94485892, 0.91671553, 0.92952392, 0.92778066,
0.94538506, 0.92747359, 0.93589836, 0.92038779, 0.92265659,
0.91553171, 0.92699145, 0.92589516, 0.9160445 , 0.9161952 ,
0.94579179, 0.89186635, 0.92471134, 0.90790847, 0.90513972,
0.92856229, 0.94631793, 0.90107663, 0.92751759, 0.91932281,
0.96446911, 0.9230512 , 0.90345762, 0.90424684, 0.94356956,
0.90831519, 0.92882536, 0.97384062, 0.90790847, 0.9582708 ,
0.92747359, 0.91435371, 0.94447287, 0.92880992, 0.92710685,
0.92436074, 0.90687755, 0.89712777, 0.9189355 , 0.93195229,
0.88933931, 0.92579274, 0.92921142, 0.92789731, 0.8968574 ,
0.92710685, 0.92011202, 0.94447287, 0.92751279, 0.90608833,
0.92173584, 0.95314091, 0.89674931, 0.94368366, 0.91566325,
0.90464144, 0.95064896, 0.91435371, 0.89923234, 0.91472636,
0.89593665, 0.92603525, 0.89450247, 0.91721824, 0.91905974,
0.90426297, 0.88407789, 0.91527446, 0.92555311, 0.9415098 ,
0.92646531, 0.91382756, 0.90590618, 0.94315751, 0.92646531,
0.91917593, 0.9370943 , 0.90566586, 0.94499901, 0.90790847,
0.91251221, 0.92565307, 0.91566906, 0.91498213, 0.91314794,
0.92484288, 0.9672911 , 0.95209585, 0.93028151, 0.94447287,
0.90198287, 0.94277502, 0.89161943, 0.9098815 , 0.94577967,
0.91671553, 0.91224914, 0.91379831, 0.94244191, 0.92710685,
0.91630479, 0.91998778, 0.91213897, 0.89923234, 0.91590288,
0.91829396, 0.90513972, 0.91566325, 0.90150073, 0.93656816,
0.92147277, 0.91618939, 0.90582526, 0.91456696, 0.94420398,
0.89778545, 0.91621135, 0.89923234, 0.92763424, 0.90186305,
0.91251221, 0.91989434, 0.90635141, 0.94485892, 0.94785599,
0.9294676 , 0.91159146, 0.9370943 , 0.94172274, 0.89554935,
0.90002156, 0.93119371, 0.91528657, 0.91827782, 0.94104713,
0.92552842, 0.94525353, 0.92803615, 0.92326445, 0.91903917,
0.91435371, 0.91577865, 0.92384041, 0.94473013, 0.92921142,
0.90687755, 0.93630508, 0.94495723, 0.94618639, 0.92948304,
0.90148459, 0.91251221, 0.95812539, 0.92593916, 0.89148788,
0.92379059, 0.92409767, 0.90040444, 0.90792983, 0.92828378,
0.90465758, 0.92775764, 0.92593916, 0.93474053, 0.89923234,
0.91224914, 0.91067071, 0.92842346, 0.91946164, 0.92462381,
0.90461358, 0.91840936, 0.93712075, 0.9278682 , 0.92644228,
0.91895745, 0.92357734, 0.91853359, 0.92555311, 0.93195229,
0.92398406, 0.90464144, 0.91800745, 0.98266964, 0.93551587,
0.92460767, 0.91840936, 0.92738606, 0.95212586, 0.91251221,
0.91343296, 0.91251221, 0.92173584, 0.91946164, 0.92868527,
0.91224914, 0.9098815 , 0.94589051, 0.92447392, 0.92489269,
0.90198287, 0.92370306, 0.92829922, 0.91382756, 0.93668757,
0.92617921, 0.90189534, 0.94396416, 0.91487985, 0.94380664,
0.91971741, 0.93710688, 0.93710688, 0.94602204, 0.94800238,
0.93510915, 0.90056384, 0.92042049, 0.91590288, 0.90464144,
0.9160445 ])
```

In [68]:
```
# The simplest way to compare the targets (y_train) and the predictions (y_hat) is

plt.scatter(y_train, y_hat)

# Let's also name the axes
plt.xlabel('Targets (y_train)',size=18)
plt.ylabel('Predictions (y_hat)',size=18)
plt.show()
```



In [69]:
```
# Another useful check of our model is a residual plot
# We can plot the PDF of the residuals and check for anomalies
sns.distplot(y_train - y_hat)

# Include a title
plt.title("Residuals PDF", size=18)

# plot  normally distributed
```

Out[69]:
```
Text(0.5, 1.0, 'Residuals PDF')
```



In [70]:
```
# Find the R-squared of the model
reg.score(x_train,y_train)
```

Out[70]:
```
0.08742440592651268
```

Finding the weights and bias

In [71]:
```
# Obtain the bias (intercept) of the regression
reg.intercept_
```

Out[71]:
```
0.9235583682412195
```

In [72]:
```
# Obtain the weights (coefficients) of the regression
reg.coef_

# Note that they are barely interpretable if at all(because of dummies  and standa
```

Out[72]:
```
array([-3.62809361e-03, -1.90439302e-02, -1.62302925e-02, -3.78247325e-03,
       -2.84857126e-03, -7.96737655e-03, -1.49195276e-02, -1.59607392e-03,
       -9.36429997e-04, -1.26921406e-02, -7.80892555e-03, -8.42448734e-03,
       -2.16840434e-19, -1.36554945e-02, -7.49752727e-03, -7.43359016e-03,
       -1.62539555e-02, -9.16918206e-03, -1.39712025e-02])
```

In [73]:
```
# Create a regression summary where we can compare them with one-another
reg_summary = pd.DataFrame(inputs.columns.values, columns=['Features'])
reg_summary['Weights'] = reg.coef_
reg_summary
```

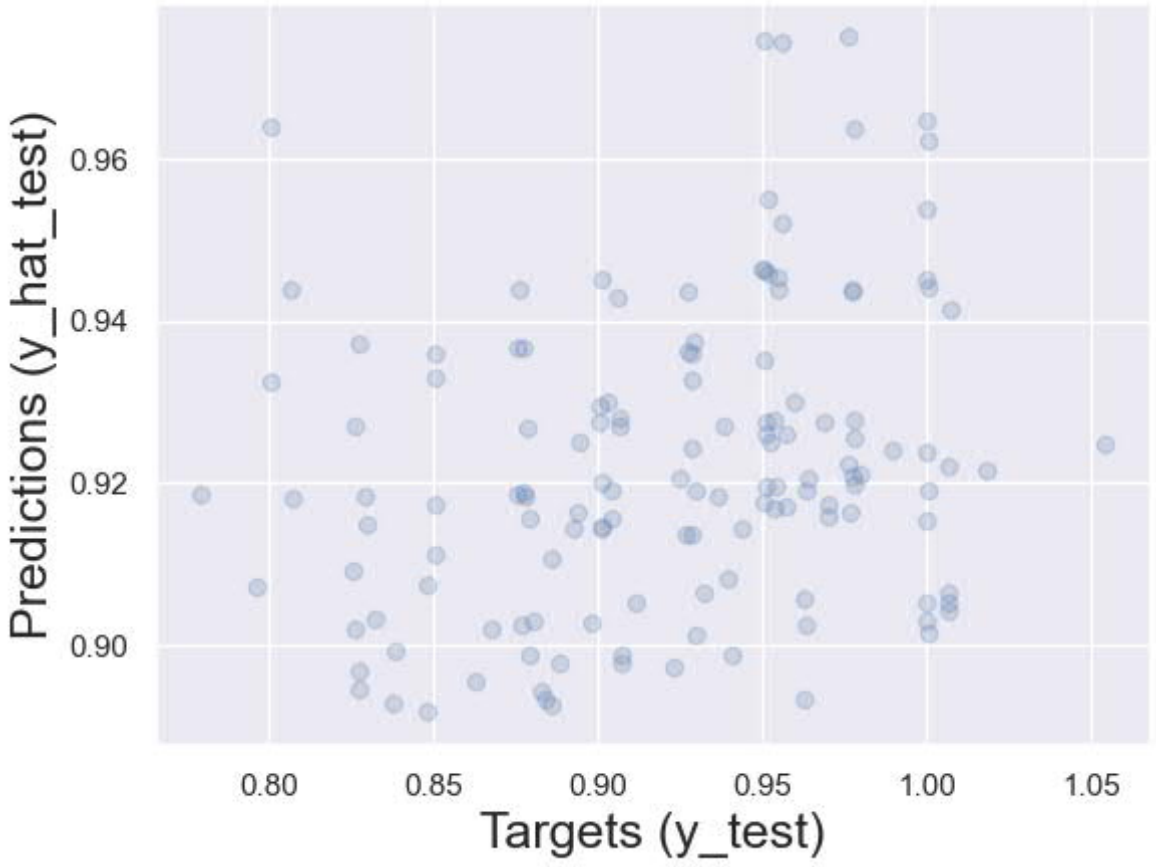
Out[73]:

	Features	Weights
0	selling_price	-3.628094e-03
1	color_Black	-1.904393e-02
2	color_Blue	-1.623029e-02
3	color_Burgundy	-3.782473e-03
4	color_Gold	-2.848571e-03
5	color_Green	-7.967377e-03
6	color_Grey	-1.491953e-02
7	color_Multi	-1.596074e-03
8	color_Orange	-9.364300e-04
9	color_Pink	-1.269214e-02
10	color_Purple	-7.808926e-03
11	color_Red	-8.424487e-03
12	color_Turquoise	-2.168404e-19
13	color_White	-1.365549e-02
14	color_Yellow	-7.497527e-03
15	category_Clothing	-7.433590e-03
16	category_Shoes	-1.625396e-02
17	gender_Men	-9.169182e-03
18	gender_Women	-1.397120e-02

Testing

In [74]:
```
# Once we have trained and fine-tuned our model, we can proceed to testing it
# Testing is done on a dataset that the algorithm has never seen
# If the predictions are far off, we will know that our model overfitted

y_hat_test = reg.predict(x_test)
```

In [75]:
```
# Create a scatter plot with the test targets and the test predictions


plt.scatter(y_test, y_hat_test, alpha=0.2)
plt.xlabel('Targets (y_test)',size=18)
plt.ylabel('Predictions (y_hat_test)',size=18)

plt.show()
```



In [76]:
```
# Finally, let's manually check these predictions

df_pf = pd.DataFrame(y_hat_test, columns=['Prediction'])
df_pf.head()
```

Out[76]:

	Prediction
0	0.919072
1	0.964044
2	0.894102
3	0.974367
4	0.943793

In [77]:
```
# We can also include the test targets in that data frame (so we can manually compa
y_test = y_test.reset_index(drop=True)


df_pf['Target'] = y_test
df_pf
```

Out[77]:

	Prediction	Target
0	0.919072	1.000596
1	0.964044	0.800340
2	0.894102	0.883086
3	0.974367	0.955703
4	0.943793	0.876277
...	...	...
132	0.918316	0.829000
133	0.915518	0.904341
134	0.902668	0.898321
135	0.926721	0.878405
136	0.944204	1.000596

137 rows × 2 columns

In [78]:
```
# Additionally, we can calculate the difference between the targets and the predict
# Note that this is actually the residual (we already plotted the residuals)
df_pf['Residual'] = df_pf['Target'] - df_pf['Prediction']

# Since OLS is basically an algorithm which minimizes the total sum of squared erro
# this comparison makes a lot of sense
```

In [79]:
```
# Finally, it makes sense to see how far off we are from the result percentage-wise
# take the absolute difference in %, so we can easily order the data frame
df_pf['Difference%'] = np.absolute(df_pf['Residual']/df_pf['Target']*100)
df_pf
```

Out[79]:

	Prediction	Target	Residual	Difference%
0	0.919072	1.000596	0.081523	8.147482
1	0.964044	0.800340	-0.163704	20.454300
2	0.894102	0.883086	-0.011017	1.247519
3	0.974367	0.955703	-0.018664	1.952924
4	0.943793	0.876277	-0.067517	7.704933
...	...	...	...	...
132	0.918316	0.829000	-0.089316	10.773889
133	0.915518	0.904341	-0.011178	1.235990
134	0.902668	0.898321	-0.004347	0.483933
135	0.926721	0.878405	-0.048316	5.500455
136	0.944204	1.000596	0.056392	5.635823

137 rows × 4 columns

In [80]:
```
# Exploring the descriptives here gives us additional insights
df_pf.describe()
```

Out[80]:

	Prediction	Target	Residual	Difference%
count	137.000000	137.000000	137.000000	137.000000
mean	0.922182	0.922055	-0.000127	4.977766
std	0.018644	0.058593	0.056858	3.994681
min	0.891619	0.779171	-0.163704	0.136976
25%	0.907267	0.879171	-0.032538	1.952924
50%	0.919462	0.928745	0.003831	3.859775
75%	0.932812	0.963556	0.038627	7.020734
max	0.975024	1.054117	0.129488	20.454300

In [ ]: