

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 model to predict which combination 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 [2]: #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 math
from PIL import ImageColor

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[2]:

	product ID	identifier	selling_price	original_price	currency	availability	color	category	breadcrumbs
0	100	FJ5089	40	NaN	euro	InStock	Black	Clothing	Women
1	101	BC0770	150	NaN	euro	InStock	Grey	Shoes	Woman
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	Men

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 [4]:

```
#get some general statistic first glance
raw_data.describe()
```

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

raw_data.head()
```

	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	Woman
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	Men

```
In [6]: #checking the data in new column, how much of each value we have
raw_data['gender'].value_counts()
```

```
Out[6]: 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 [7]: #also take a look at the number of each value
raw_data['color'].value_counts()
```

```
Out[7]: 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 [8]: #as soon as we have Multicolor in different way of typing - make the common name for it
raw_data.loc[raw_data['color']=='Multicolor', ['color']]='Multi'
raw_data['color'].value_counts()
```

```
Out[8]: 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 [9]: raw_data
```

Out[9]:

	product ID	identifier	selling_price	original_price	currency	availability	color	category	b
0	100	FJ5089	40	NaN	euro	InStock	Black	Clothing	Won
1	101	BC0770	150	NaN	euro	InStock	Grey	Shoes	W
2	102	GC7946	70	NaN	euro	InStock	White	Clothing	K
3	103	FV4744	160	NaN	euro	InStock	Black	Shoes	Fin
4	104	GM0239	65	NaN	euro	InStock	Blue	Clothing	N
...
840	940	FX2858	72	120.0	euro	InStock	White	Shoes	W
841	941	H00667	70	100.0	euro	InStock	White	Shoes	W
842	942	GZ7705	35	50.0	euro	InStock	Black	Shoes	
843	943	GZ7706	40	50.0	euro	InStock	Pink	Shoes	
844	944	FY6503	70	100.0	euro	InStock	Black	Shoes	W

845 rows × 15 columns

Union Colors

We see that 'White','Black','Blue','Grey','Pink','Green' , 'Purple','Red','Yellow','Orange' and 'Multi' has more data inputs than 'Beige','Gold','Turquoise','Silver','Brown'

So next - we gonna transform our colors to rgb Make our own dictionary for colors(from that which have more items) And find the closest color from our dictionary for the list ['Beige','Gold','Turquoise','Silver','Brown'], and replace it

In order to find the closest color - we will find euclidean distance between two RGB vectors(calculate absolute distance between colors)

```
In [10]: #For some reason, that color - 'Burgundy' - did not appear in ImageColor.getrgb,
# 'Burgundy'-->'Purple'
```

```
In [11]: import math
from PIL import ImageColor

#So once again - colors_to_leave = ['White', 'Black', 'Blue', 'Grey', 'Pink', 'Green' ,
colors_check=['Beige', 'Gold', 'Turquoise', 'Silver', 'Brown']

#find euclidean distance between two RGB vectors
def distance(c1, c2):
    (r1,g1,b1) = c1
    (r2,g2,b2) = c2
    return math.sqrt((r1 - r2)**2 + (g1 - g2) ** 2 + (b1 - b2) **2)

#Make our own dictionary of colors
rgb_code_dictionary={ImageColor.getrgb('White') : 'White',
```

```

        ImageColor.getrgb('Black'): 'Black',
        ImageColor.getrgb('Blue'): 'Blue',
        ImageColor.getrgb('Grey'): 'Grey',
        ImageColor.getrgb('Pink'): 'Pink',
        ImageColor.getrgb('Green'): 'Green',
        ImageColor.getrgb('Purple'): 'Purple',
        ImageColor.getrgb('Red'): 'Red',
        ImageColor.getrgb('Yellow'): 'Yellow',
        ImageColor.getrgb('Orange'): 'Orange'
    }

    colors = list(rgb_code_dictionary.keys())

    colors_replace = []
    for color_check in colors_check:
        closest_colors = sorted(colors, key=lambda color: distance(color, ImageColor.getrgb(color)))
        closest_color = closest_colors[0]
        code = rgb_code_dictionary[closest_color]
        colors_replace.append(code)

#So, colors to replace is
colors_replace

```

Out[11]: ['White', 'Yellow', 'Grey', 'Pink', 'Purple']

In [12]: #So add manually our Burgundy, and replace it in our table
raw_data['color'] = raw_data['color'].replace(['Burgundy', 'Beige', 'Gold', 'Turquoise',
'Purple', 'White', 'Yellow', 'Grey'])
raw_data

Out[12]:

	product ID	identifier	selling_price	original_price	currency	availability	color	category	b
0	100	FJ5089	40	NaN	euro	InStock	Black	Clothing	Woman
1	101	BC0770	150	NaN	euro	InStock	Grey	Shoes	Woman
2	102	GC7946	70	NaN	euro	InStock	White	Clothing	Kid
3	103	FV4744	160	NaN	euro	InStock	Black	Shoes	Female
4	104	GM0239	65	NaN	euro	InStock	Blue	Clothing	Men
...
840	940	FX2858	72	120.0	euro	InStock	White	Shoes	Woman
841	941	H00667	70	100.0	euro	InStock	White	Shoes	Woman
842	942	GZ7705	35	50.0	euro	InStock	Black	Shoes	
843	943	GZ7706	40	50.0	euro	InStock	Pink	Shoes	
844	944	FY6503	70	100.0	euro	InStock	Black	Shoes	Woman

845 rows × 15 columns

In [13]: #Lets check the value of color column(and count it)
raw_data['color'].value_counts()

```
Out[13]:
```

White	228
Black	187
Blue	104
Grey	83
Pink	63
Green	59
Purple	41
Red	25
Multi	24
Yellow	20
Orange	11

Name: color, dtype: int64

```
In [14]:
```

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

```
Out[14]:
```

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

```
In [15]:
```

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

```
In [16]:
```

```
#Checking if our new column, extracted from breadcrumbs is the same as category column
(raw_data['category']==raw_data['clothesType']).unique()
```

```
Out[16]:
```

```
array([ True])
```

```
In [17]:
```

```
#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 can drop them
# significant_columns_data = raw_data.drop(['currency', 'clothesType', 'country', 'language'])
# significant_columns_data.head()
```

```
Out[17]:
```

	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 [18]:
```

```
#dropping all values from ageGroup which is not adultn or children
```

```
significant_columns_data_clean = significant_columns_data.loc[~((significant_columns_data['original_price'].isna() & significant_columns_data['average_rating'].isna()) | (significant_columns_data['reviews_count'].isna() & significant_columns_data['color'].isna()))]
#and resetting the index
significant_columns_data_clean.reset_index().drop(['index'], axis=1).head()
```

Out[18]:

	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 [19]:

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

Out[19]:

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

In [20]:

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

Out[20]:

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

In [21]:

```
#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
```

Missing values

In [22]:

```
#we still have missing values. So we need to fill it, cause dropping can be cruel
```

In [23]:

```
#As soon as missing values is in original_price column, we will use columns selling_price
#for prediction of missing values
to_predict_nulls_array = np.array([significant_columns_data_clean['selling_price'],
                                    significant_columns_data_clean['original_price'],
                                    significant_columns_data_clean['average_rating'],
                                    significant_columns_data_clean['reviews_count'],
                                    significant_columns_data_clean['category'],
                                    significant_columns_data_clean['color'],
                                    significant_columns_data_clean['breadcrumbs'],
                                    significant_columns_data_clean['ageGroup'],
                                    significant_columns_data_clean['gender'],
                                    significant_columns_data_clean['availability'],
                                    significant_columns_data_clean['identifier'],
                                    significant_columns_data_clean['product ID']])
```

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

```
Out[23]: 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 [24]: # The KNNImputer class provides imputation for filling in missing values using the
#Each sample's missing values are imputed using the mean value from n_neighbors nearest
#Two samples are close if the features that neither is missing are close.

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

```
In [25]: predicted_nulls_array = imputer.fit_transform(to_predict_nulls_array)
predicted_nulls_array
```

```
Out[25]: 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 [26]: #adding values to our table
significant_columns_data_clean.loc[:, ('original_price')]=predicted_nulls_array[:,
```

```
In [27]: significant_columns_data_clean.head(10)
```

	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 [28]: 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 [29]: `#lets check the corelation between average_rating and reviews_count
np.corrcoef(significant_columns_data_clean['average_rating'], significant_columns_`

Out[29]: `array([[1. , 0.01708148],
 [0.01708148, 1.]])`

In [30]: `#we see that correlation is not null`

In [31]: `significant_columns_data_clean.describe()`

Out[31]:

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

In [33]: `no_duplicates_data.reset_index().drop(['index'], axis=1).head()`

Out[33]:

	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

Introducing 'Popularity' column

```
In [34]: #it can be logically assumed that the popularity of a particular color is made up of
#but also of the number of reviews for the product. Therefore, we introduce a new column
#To do this, we need to standardize these two columns, since the data is too different
#Locking our two columns
no_duplicates_data_check=no_duplicates_data.loc[:, ['reviews_count','average_rating']]
no_duplicates_data_check
```

Out[34]:

	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

We can use a method inspired by Bayesian probability. The hint of the approach is to have an initial belief about the true rating of an item, and use users' ratings to update our belief.

This approach requires two parameters:

What do we think is the true "default" rating of an item, if you have no ratings at all for the item? Lets call this number `initial_value`.

How much weight do we give to the `initial_value`, compared to the user ratings? Call this `weigh_for_initial`.

With the parameters `initial_value` and `weigh_for_initial`, computing the new rating or popularity is simple: assume we have `weigh_for_initial` ratings of value `initial_value` along with any user ratings, and compute the average.

For example, if `initial_value = 2` and `weigh_for_initial = 3`, we compute the final score for various scenarios below:

100 (user) ratings of 4: $(32 + 1004) / (3 + 100) = 3.94$ 1 rating of 5: $(32 + 15) / (3 + 1) = 2.75$
 No user ratings: $(3*2 + 0) / (3 + 0) = 2$

In []:

```
In [35]: #So lets take our initial_value for the item in first quantile - like first meanie

initial_value = no_duplicates_data_check['average_rating'].quantile(0.25)
initial_value
```

Out[35]: 4.5

```
In [36]: #The choice of weigh_for_initial should depend on how many ratings a typical item has
weigh_for_initial = (no_duplicates_data_check['reviews_count'].median()) / 4
weigh_for_initial
```

Out[36]: 16.75

```
In [37]: no_duplicates_data_check['product_popularity'] = (weigh_for_initial * initial_value +
no_duplicates_data_check['reviews_count'] * no_duplicates_data_check['average_rating'])

no_duplicates_data_check
```

Out[37]:

	reviews_count	average_rating	product_popularity
0	35	4.5	4.500000
1	4	4.8	4.557831
2	42	4.9	4.785957
4	11	4.7	4.579279
5	30	4.9	4.756684
...
840	151	4.3	4.319970
841	135	4.7	4.677924
842	190	4.7	4.683797
843	190	4.7	4.683797
844	135	4.7	4.677924

733 rows × 3 columns

In [38]: *#Lets normalise the popularity*

```
X_train = np.array(no_duplicates_data_check['product_popularity']).reshape(-1, 1)
X_train
```

```
Out[38]: array([[4.5      ],
 [4.55783133],
 [4.78595745],
 [4.57927928],
 [4.75668449],
 [4.5      ],
 [4.5      ],
 [4.5      ],
 [4.73927492],
 [4.43918129],
 [4.38505747],
 [4.55333333],
 [4.66688742],
 [4.76573427],
 [4.76214689],
 [4.76214689],
 [4.71992032],
 [4.76214689],
 [4.23125972],
 [4.5      ],
 [4.58896211],
 [4.59512727],
 [4.76214689],
 [4.68691589],
 [4.64904943],
 [4.32729124],
 [4.47052632],
 [4.7781457 ],
 [4.47052632],
 [4.5823219 ],
 [4.5      ],
 [4.67475728],
 [4.59105691],
 [4.44666667],
 [4.44666667],
 [4.66326531],
 [4.67225673],
 [4.83603819],
 [4.72897527],
 [4.36813187],
 [4.58896211],
 [4.7781457 ],
 [4.65518395],
 [4.7469657 ],
 [4.5969697 ],
 [4.68322904],
 [4.52816901],
 [4.66161616],
 [4.44666667],
 [4.40229008],
 [4.71258741],
 [4.40229008],
 [4.65518395],
 [4.57912088],
 [4.54552846],
 [4.57728814],
 [4.68322904],
 [4.37836257],
 [4.7469657 ],
 [4.58970814],
 [4.59574603],
 [4.56883721],
 [4.5      ],
 [4.65518395],
```

```
[4.65518395],  
[4.2536    ],  
[4.52816901],  
[4.52816901],  
[4.2536    ],  
[4.2536    ],  
[4.69608073],  
[4.78626111],  
[4.2536    ],  
[4.21370143],  
[4.43535354],  
[4.78626111],  
[4.21370143],  
[4.5953569 ],  
[4.78755418],  
[4.77822319],  
[4.68918483],  
[4.41554524],  
[4.73452769],  
[4.66801909],  
[4.64745098],  
[4.77534884],  
[4.46962025],  
[4.69826851],  
[4.69826851],  
[4.57048458],  
[3.96666667],  
[4.69912504],  
[4.69912504],  
[4.65951662],  
[4.69912504],  
[4.64485597],  
[4.6352657 ],  
[4.68461538],  
[3.96666667],  
[4.58      ],  
[4.5      ],  
[4.63186813],  
[4.68461538],  
[4.5823219 ],  
[4.5823219 ],  
[4.73927492],  
[4.72583026],  
[4.5823219 ],  
[4.64485597],  
[4.73927492],  
[4.72583026],  
[4.69912504],  
[4.65265018],  
[4.75668449],  
[4.58      ],  
[4.68461538],  
[4.58      ],  
[4.69912504],  
[4.69912504],  
[4.75668449],  
[4.5823219 ],  
[4.69016393],  
[4.58      ],  
[4.73927492],  
[4.5      ],  
[4.42757202],  
[4.59574603],  
[4.59574603],
```

```
[4.74806202],  
[4.40274253],  
[4.42757202],  
[4.40274253],  
[4.42757202],  
[4.40274253],  
[4.2536    ],  
[4.5      ],  
[4.69071379],  
[4.40748603],  
[4.40274253],  
[4.40748603],  
[4.69071379],  
[4.2536    ],  
[4.40748603],  
[4.40748603],  
[4.69071379],  
[4.68322904],  
[4.68322904],  
[4.37657143],  
[4.68322904],  
[4.72795699],  
[4.61976048],  
[4.3466899 ],  
[4.61976048],  
[4.40748603],  
[4.3466899 ],  
[4.68322904],  
[4.30307692],  
[4.53232323],  
[4.71258741],  
[4.40992593],  
[4.37657143],  
[4.40992593],  
[4.30307692],  
[4.52298851],  
[4.40992593],  
[4.71446809],  
[4.37657143],  
[4.5      ],  
[4.59731893],  
[4.72117647],  
[4.43103448],  
[4.75248227],  
[4.79157939],  
[4.5      ],  
[4.54552846],  
[4.56338798],  
[4.06462585],  
[4.7705314 ],  
[4.63186813],  
[4.40355815],  
[4.67495327],  
[4.40355815],  
[4.5      ],  
[4.40355815],  
[4.40355815],  
[4.73366337],  
[4.65539568],  
[4.59350145],  
[4.5      ],  
[4.40355815],  
[4.58970814],
```

```
[4.58970814],  
[4.58970814],  
[4.5      ],  
[4.68897638],  
[4.5      ],  
[4.5      ],  
[4.58970814],  
[4.74962406],  
[4.6947644 ],  
[4.5      ],  
[4.74962406],  
[4.58970814],  
[4.03094463],  
[4.5      ],  
[4.74962406],  
[4.06462585],  
[4.03094463],  
[4.74962406],  
[4.74962406],  
[4.43103448],  
[4.43103448],  
[4.64199134],  
[4.86076135],  
[4.78035191],  
[4.52816901],  
[4.30100503],  
[4.52947368],  
[4.69954158],  
[4.75012407],  
[4.64656489],  
[4.69608073],  
[4.58      ],  
[4.5      ],  
[4.69608073],  
[4.5      ],  
[4.5823219 ],  
[4.59736532],  
[4.57594937],  
[4.79105474],  
[4.7855914 ],  
[4.59842981],  
[4.7855914 ],  
[4.5992688 ],  
[4.78952579],  
[4.5992688 ],  
[4.79105474],  
[4.77996012],  
[4.5992688 ],  
[4.64574899],  
[4.5      ],  
[4.8277628 ],  
[4.77057101],  
[4.5      ],  
[4.79269357],  
[4.74070796],  
[4.74070796],  
[4.79269357],  
[4.5      ],  
[4.5      ],  
[4.86313618],  
[4.5      ],  
[4.5      ],  
[4.69698673],  
[4.86313618],
```

```
[4.5      ],
[4.43103448],
[4.5      ],
[4.5      ],
[4.43103448],
[4.69698673],
[4.75899281],
[4.5      ],
[4.5      ],
[4.69311762],
[4.74070796],
[4.70148148],
[4.56896552],
[4.55333333],
[4.6997153 ],
[4.34      ],
[4.6997153 ],
[4.47183099],
[4.69755251],
[4.69755251],
[4.43103448],
[4.66093294],
[4.78019704],
[4.5      ],
[4.69086571],
[4.43103448],
[4.6924507 ],
[4.66093294],
[4.6924507 ],
[4.6924507 ],
[4.40722762],
[4.71258741],
[4.5969697 ],
[4.46261682],
[4.46261682],
[4.46261682],
[4.5969697 ],
[4.46504854],
[4.5      ],
[4.5      ],
[4.46504854],
[4.63186813],
[4.65111111],
[4.63186813],
[4.68965251],
[4.77534884],
[4.30458015],
[4.71589958],
[4.5      ],
[4.76834646],
[4.54885496],
[4.84451346],
[4.5      ],
[4.5      ],
[4.68965251],
[4.77534884],
[4.63399015],
[4.61572327],
[4.61572327],
[4.67377691],
[4.71589958],
[4.5      ],
[4.70719424],
[4.67032258],
```

```
[4.65944056],  
[4.43009709],  
[4.67475728],  
[4.55783133],  
[4.78595745],  
[4.77762557],  
[4.65944056],  
[4.55783133],  
[4.73622047],  
[4.69692308],  
[4.65539568],  
[4.55333333],  
[4.55333333],  
[4.55783133],  
[4.55333333],  
[4.69692308],  
[4.65799373],  
[4.55333333],  
[4.7705314 ],  
[4.5815427 ],  
[4.59909837],  
[4.69965295],  
[4.69965295],  
[4.59909837],  
[4.59909837],  
[4.5         ],  
[4.37848101],  
[4.59909837],  
[4.6928837 ],  
[4.69868974],  
[4.6928837 ],  
[4.69868974],  
[4.6928837 ],  
[4.5         ],  
[4.37848101],  
[4.2005988 ],  
[4.77235213],  
[4.5         ],  
[4.69868974],  
[4.5         ],  
[4.69792666],  
[4.69792666],  
[4.6691954 ],  
[4.66975169],  
[4.6928837 ],  
[4.69792666],  
[4.6928837 ],  
[4.59805176],  
[4.6928837 ],  
[4.5         ],  
[4.5         ],  
[4.69868974],  
[4.62984293],  
[4.59683215],  
[4.5         ],  
[4.5         ],  
[4.77235213],  
[4.62984293],  
[4.69016393],  
[4.5         ],  
[4.73952096],  
[4.59683215],  
[4.75061425],  
[4.59729293],
```

```
[4.5      ],
[4.12502334],
[3.91889986],
[3.91889986],
[3.91889986],
[4.59729293],
[4.2536    ],
[4.2536    ],
[4.13388116],
[4.5      ],
[4.79733103],
[3.91889986],
[4.5      ],
[4.12502334],
[4.12502334],
[4.12502334],
[4.21370143],
[4.70719424],
[4.12502334],
[4.59683215],
[4.69521941],
[4.69521941],
[4.59683215],
[4.69521941],
[4.12502334],
[4.2536    ],
[4.59957235],
[4.79733103],
[4.5      ],
[4.2536    ],
[4.12502334],
[4.12502334],
[4.5      ],
[4.69398294],
[4.59868188],
[3.91889986],
[4.59868188],
[4.59868188],
[4.59868188],
[4.79733103],
[4.5      ],
[4.78967643],
[4.59957235],
[4.69398294],
[4.59868188],
[4.21072   ],
[4.59957235],
[4.21072   ],
[4.59641903],
[4.59572431],
[4.21072   ],
[4.59622535],
[4.59622535],
[4.21072   ],
[4.33198091],
[4.69781723],
[4.21072   ],
[4.40273805],
[4.5      ],
[4.59622535],
[4.67225673],
[4.21072   ],
[4.21072   ],
[4.5      ],
```

```
[4.5      ],
[4.59622535],
[4.59423904],
[4.69781723],
[4.29178082],
[4.21496649],
[4.69781723],
[4.56883721],
[4.21496649],
[4.29178082],
[4.40273805],
[4.59499627],
[4.40273805],
[4.5      ],
[4.21496649],
[4.67225673],
[4.5      ],
[4.59574603],
[4.5      ],
[4.59574603],
[4.5      ],
[4.5      ],
[4.5      ],
[4.59499627],
[4.29178082],
[4.74806202],
[4.74806202],
[4.7469657 ],
[4.595932 ],
[4.74806202],
[4.59574603],
[4.5      ],
[4.60074074],
[4.60074074],
[4.59574603],
[4.56256983],
[4.58896211],
[4.7469657 ],
[4.42757202],
[4.60359712],
[4.59574603],
[4.5      ],
[4.42757202],
[4.60074074],
[4.5      ],
[4.68211382],
[4.53232323],
[4.6533101 ],
[4.6724846 ],
[4.46962025],
[4.69016393],
[4.59638554],
[4.59448819],
[4.75248227],
[4.60359712],
[4.5      ],
[4.55333333],
[4.64199134],
[4.57594937],
[4.52816901],
[4.59448819],
[4.58699029],
[4.52816901],
[4.86431425],
```

```
[4.40551181],  
[4.70719424],  
[4.46962025],  
[4.3018018 ],  
[4.5      ],  
[4.70986547],  
[4.52816901],  
[4.60485437],  
[4.63109244],  
[4.5819407 ],  
[4.76325411],  
[4.52816901],  
[4.52816901],  
[4.39514563],  
[4.52816901],  
[4.52816901],  
[4.52816901],  
[4.52816901],  
[4.64199134],  
[4.65055351],  
[4.65055351],  
[4.31788618],  
[4.52816901],  
[4.65518395],  
[4.65518395],  
[4.5      ],  
[4.24843373],  
[4.68514286],  
[4.20037453],  
[4.61891892],  
[4.56990291],  
[4.31788618],  
[4.55783133],  
[4.53037975],  
[4.52947368],  
[4.56633166],  
[4.54552846],  
[4.44666667],  
[4.56633166],  
[4.5      ],  
[4.5      ],  
[4.5      ],  
[4.52816901],  
[4.54552846],  
[4.43300493],  
[4.44666667],  
[4.42757202],  
[4.59448819],  
[4.43507853],  
[4.62521739],  
[4.39514563],  
[4.46261682],  
[4.57912088],  
[4.55333333],  
[4.54552846],  
[4.74751958],  
[4.67475728],  
[4.43535354],  
[4.64656489],  
[4.57330677],  
[4.67792422],  
[4.31997019],  
[4.5      ],
```

```
[4.5      ],
[4.54885496],
[4.5      ],
[4.66707617],
[4.63186813],
[4.54885496],
[4.74685714],
[4.59638554],
[4.45114504],
[4.43535354],
[4.67792422],
[4.55562914],
[4.5      ],
[4.63186813],
[4.5      ],
[4.5      ],
[4.35173745],
[4.59638554],
[4.3028169 ],
[4.75336427],
[4.70098522],
[4.47052632],
[4.64656489],
[4.40361446],
[4.5      ],
[4.36019417],
[4.56896552],
[4.57594937],
[3.93673469],
[4.58337469],
[4.55562914],
[4.58337469],
[4.5      ],
[4.58337469],
[4.91927711],
[4.58337469],
[4.42405063],
[4.58337469],
[4.42087912],
[4.57728814],
[4.57728814],
[4.69251337],
[4.69251337],
[4.53738318],
[4.77987988],
[4.58970814],
[4.58970814],
[4.70986547],
[4.58069164],
[4.69251337],
[4.35173745],
[4.61789474],
[4.77107914],
[4.5      ],
[4.5      ],
[4.63980583],
[4.57563636],
[4.20868251],
[4.65055351],
[4.20868251],
[4.20868251],
[4.57413127],
[4.77107914],
[4.58069164],
```

```
[4.63881279],  
[4.20868251],  
[4.67762938],  
[4.68739417],  
[4.70986547],  
[4.66161616],  
[4.82450704],  
[4.55783133],  
[4.5      ],  
[4.73452769],  
[4.67105832],  
[4.72690909],  
[4.47701149],  
[4.69692308],  
[4.73277592],  
[4.59638554],  
[4.57287449],  
[4.53037975],  
[4.59638554],  
[4.73452769],  
[4.5      ],  
[4.65265018],  
[4.41260504],  
[4.46962025],  
[4.73277592],  
[4.41260504],  
[4.73277592],  
[4.5      ],  
[4.65265018],  
[4.7647986 ],  
[4.58133705],  
[4.58133705],  
[4.57912088],  
[4.46504854],  
[4.52816901],  
[4.55783133],  
[4.47183099],  
[4.5      ],  
[4.63399015],  
[4.77895288],  
[4.77895288],  
[4.67178947],  
[4.79265082],  
[4.67178947],  
[4.78398268],  
[4.77895288],  
[4.5      ],  
[4.55333333],  
[4.5      ],  
[4.77895288],  
[4.61494253],  
[4.77895288],  
[4.55783133],  
[4.55333333],  
[4.8277628 ],  
[4.5      ],  
[4.54724409],  
[4.5      ],  
[4.70148148],  
[4.52816901],  
[4.62513966],  
[4.59448819],  
[4.56896552],  
[4.5      ],
```

```
[4.64199134],  
[4.66695652],  
[4.78398268],  
[4.5      ],  
[4.73952096],  
[4.57925697],  
[4.61789474],  
[4.74327485],  
[4.5      ],  
[4.553333333],  
[4.63109244],  
[4.6708061  ],  
[4.63109244],  
[4.63109244],  
[4.61494253],  
[4.67358491],  
[4.66161616],  
[4.31997019],  
[4.71589958],  
[4.68270968],  
[4.68270968],  
[4.66047198],  
[4.31997019],  
[4.40355815],  
[4.31997019],  
[4.67792422],  
[4.68379686],  
[4.68379686],  
[4.67792422]])
```

```
In [39]: #going to standardise 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[39]: array([[0.580881  ],
 [0.63869052],
 [0.86673061],
 [0.66013039],
 [0.8374687 ],
 [0.580881  ],
 [0.580881  ],
 [0.580881  ],
 [0.8200657 ],
 [0.52008523],
 [0.46598182],
 [0.63419422],
 [0.74770549],
 [0.84651506],
 [0.84292904],
 [0.84292904],
 [0.80071839],
 [0.84292904],
 [0.31224207],
 [0.580881  ],
 [0.66980956],
 [0.6759724 ],
 [0.84292904],
 [0.7677264 ],
 [0.72987423],
 [0.40823738],
 [0.55141843],
 [0.85892181],
 [0.55141843],
 [0.66317186],
 [0.580881  ],
 [0.75557238],
 [0.67190358],
 [0.52756778],
 [0.52756778],
 [0.74408474],
 [0.75307277],
 [0.91679247],
 [0.80976992],
 [0.4490626 ],
 [0.66980956],
 [0.85892181],
 [0.73600643],
 [0.82775357],
 [0.67781413],
 [0.76404094],
 [0.6090394 ],
 [0.74243622],
 [0.52756778],
 [0.48320793],
 [0.79338825],
 [0.48320793],
 [0.73600643],
 [0.65997205],
 [0.62639229],
 [0.65813999],
 [0.76404094],
 [0.45928945],
 [0.82775357],
 [0.67055532],
 [0.67659093],
 [0.64969225],
 [0.580881  ],
 [0.73600643],
```

```
[0.73600643],  
[0.33457392],  
[0.6090394 ],  
[0.6090394 ],  
[0.33457392],  
[0.33457392],  
[0.77688779],  
[0.86703416],  
[0.33457392],  
[0.2946904 ],  
[0.51625892],  
[0.86703416],  
[0.2946904 ],  
[0.67620194],  
[0.86832674],  
[0.85899927],  
[0.76999449],  
[0.4964581 ],  
[0.81532025],  
[0.74883674],  
[0.72827638],  
[0.85612601],  
[0.55051271],  
[0.77907475],  
[0.77907475],  
[0.65133901],  
[0.04774879],  
[0.77993095],  
[0.77993095],  
[0.74033747],  
[0.77993095],  
[0.72568234],  
[0.71609569],  
[0.76542677],  
[0.04774879],  
[0.66085084],  
[0.580881 ],  
[0.71269941],  
[0.76542677],  
[0.66317186],  
[0.66317186],  
[0.8200657 ],  
[0.8066261 ],  
[0.66317186],  
[0.72568234],  
[0.8200657 ],  
[0.8066261 ],  
[0.77993095],  
[0.73347362],  
[0.8374687 ],  
[0.66085084],  
[0.76542677],  
[0.66085084],  
[0.77993095],  
[0.77993095],  
[0.8374687 ],  
[0.66317186],  
[0.77097323],  
[0.66085084],  
[0.8200657 ],  
[0.580881 ],  
[0.50848033],  
[0.67659093],  
[0.67659093],
```

```
[0.82884947],  
[0.48366021],  
[0.50848033],  
[0.48366021],  
[0.50848033],  
[0.48366021],  
[0.33457392],  
[0.580881 ],  
[0.77152288],  
[0.48840193],  
[0.48366021],  
[0.48840193],  
[0.77152288],  
[0.33457392],  
[0.48840193],  
[0.48840193],  
[0.77152288],  
[0.76404094],  
[0.76404094],  
[0.45749898],  
[0.76404094],  
[0.80875203],  
[0.70059632],  
[0.42762871],  
[0.70059632],  
[0.48840193],  
[0.42762871],  
[0.76404094],  
[0.38403219],  
[0.61319205],  
[0.79338825],  
[0.4908409 ],  
[0.45749898],  
[0.4908409 ],  
[0.38403219],  
[0.60386084],  
[0.4908409 ],  
[0.79526821],  
[0.45749898],  
[0.580881 ],  
[0.67816323],  
[0.80197407],  
[0.51194149],  
[0.83326806],  
[0.87235044],  
[0.580881 ],  
[0.62639229],  
[0.64424508],  
[0.14567104],  
[0.85131039],  
[0.71269941],  
[0.48447552],  
[0.7557683 ],  
[0.48447552],  
[0.580881 ],  
[0.48447552],  
[0.48447552],  
[0.81445625],  
[0.73621809],  
[0.6743472 ],  
[0.580881 ],  
[0.48447552],  
[0.67055532],
```

```
[0.67055532],  
[0.67055532],  
[0.580881 ],  
[0.76978612],  
[0.580881 ],  
[0.580881 ],  
[0.67055532],  
[0.83041093],  
[0.77557195],  
[0.580881 ],  
[0.83041093],  
[0.67055532],  
[0.11200251],  
[0.580881 ],  
[0.83041093],  
[0.14567104],  
[0.11200251],  
[0.83041093],  
[0.83041093],  
[0.51194149],  
[0.51194149],  
[0.7228188 ],  
[0.94150631],  
[0.86112719],  
[0.6090394 ],  
[0.38196107],  
[0.61034357],  
[0.78034734],  
[0.83091075],  
[0.72739062],  
[0.77688779],  
[0.66085084],  
[0.580881 ],  
[0.77688779],  
[0.580881 ],  
[0.66317186],  
[0.6782096 ],  
[0.65680173],  
[0.87182598],  
[0.8663647 ],  
[0.6792737 ],  
[0.8663647 ],  
[0.68011237],  
[0.87029762],  
[0.68011237],  
[0.87182598],  
[0.86073555],  
[0.68011237],  
[0.72657503],  
[0.580881 ],  
[0.90852021],  
[0.85134998],  
[0.580881 ],  
[0.87346419],  
[0.8214982 ],  
[0.8214982 ],  
[0.87346419],  
[0.580881 ],  
[0.580881 ],  
[0.94388024],  
[0.580881 ],  
[0.580881 ],  
[0.77779345],  
[0.94388024],
```

```
[0.580881 ],  
[0.51194149],  
[0.580881 ],  
[0.580881 ],  
[0.51194149],  
[0.77779345],  
[0.83977614],  
[0.580881 ],  
[0.580881 ],  
[0.77392579],  
[0.8214982 ],  
[0.78228651],  
[0.64982051],  
[0.63419422],  
[0.78052099],  
[0.42094134],  
[0.78052099],  
[0.55272261],  
[0.77835902],  
[0.77835902],  
[0.51194149],  
[0.74175326],  
[0.86097238],  
[0.580881 ],  
[0.77167474],  
[0.51194149],  
[0.77325913],  
[0.74175326],  
[0.77325913],  
[0.77325913],  
[0.48814361],  
[0.79338825],  
[0.67781413],  
[0.54351192],  
[0.54351192],  
[0.54351192],  
[0.67781413],  
[0.54594273],  
[0.580881 ],  
[0.580881 ],  
[0.54594273],  
[0.71269941],  
[0.73193513],  
[0.71269941],  
[0.77046199],  
[0.85612601],  
[0.38553485],  
[0.79669917],  
[0.580881 ],  
[0.84912627],  
[0.62971754],  
[0.92526454],  
[0.580881 ],  
[0.580881 ],  
[0.77046199],  
[0.85612601],  
[0.71482062],  
[0.69656063],  
[0.69656063],  
[0.75459238],  
[0.79669917],  
[0.580881 ],  
[0.78799711],  
[0.75113935],
```

```
[0.74026144],  
[0.51100445],  
[0.75557238],  
[0.63869052],  
[0.86673061],  
[0.85840188],  
[0.74026144],  
[0.63869052],  
[0.8170124 ],  
[0.77772982],  
[0.73621809],  
[0.63419422],  
[0.63419422],  
[0.63869052],  
[0.63419422],  
[0.77772982],  
[0.73881515],  
[0.63419422],  
[0.85131039],  
[0.66239295],  
[0.679942 ],  
[0.78045866],  
[0.78045866],  
[0.679942 ],  
[0.679942 ],  
[0.580881 ],  
[0.45940784],  
[0.679942 ],  
[0.77369196],  
[0.77949582],  
[0.77369196],  
[0.77949582],  
[0.77369196],  
[0.580881 ],  
[0.45940784],  
[0.28159271],  
[0.85313043],  
[0.580881 ],  
[0.77949582],  
[0.580881 ],  
[0.77873302],  
[0.77873302],  
[0.7500126 ],  
[0.75056868],  
[0.77369196],  
[0.77873302],  
[0.77369196],  
[0.67889579],  
[0.77369196],  
[0.580881 ],  
[0.580881 ],  
[0.77949582],  
[0.71067497],  
[0.67767664],  
[0.580881 ],  
[0.580881 ],  
[0.85313043],  
[0.71067497],  
[0.77097323],  
[0.580881 ],  
[0.82031164],  
[0.67767664],  
[0.83140075],  
[0.67813724],
```

```
[0.580881 ],  
[0.20604575],  
[0.          ],  
[0.          ],  
[0.          ],  
[0.67813724],  
[0.33457392],  
[0.33457392],  
[0.21490023],  
[0.580881 ],  
[0.87809991],  
[0.          ],  
[0.580881 ],  
[0.20604575],  
[0.20604575],  
[0.20604575],  
[0.20604575],  
[0.2946904 ],  
[0.78799711],  
[0.20604575],  
[0.67767664],  
[0.77602679],  
[0.77602679],  
[0.67767664],  
[0.77602679],  
[0.20604575],  
[0.33457392],  
[0.6804158 ],  
[0.87809991],  
[0.580881 ],  
[0.33457392],  
[0.20604575],  
[0.20604575],  
[0.580881 ],  
[0.77479079],  
[0.67952567],  
[0.          ],  
[0.67952567],  
[0.67952567],  
[0.67952567],  
[0.87809991],  
[0.580881 ],  
[0.87044819],  
[0.6804158 ],  
[0.77479079],  
[0.67952567],  
[0.29171009],  
[0.6804158 ],  
[0.29171009],  
[0.67726367],  
[0.67656922],  
[0.29171009],  
[0.67707007],  
[0.67707007],  
[0.29171009],  
[0.41292527],  
[0.77862364],  
[0.29171009],  
[0.48365573],  
[0.580881 ],  
[0.67707007],  
[0.75307277],  
[0.29171009],  
[0.29171009],  
[0.580881 ],
```

```
[0.580881 ],  
[0.67707007],  
[0.6750845 ],  
[0.77862364],  
[0.37274035],  
[0.29595499],  
[0.77862364],  
[0.64969225],  
[0.29595499],  
[0.37274035],  
[0.48365573],  
[0.67584145],  
[0.48365573],  
[0.580881 ],  
[0.29595499],  
[0.75307277],  
[0.580881 ],  
[0.67659093],  
[0.580881 ],  
[0.67659093],  
[0.580881 ],  
[0.580881 ],  
[0.580881 ],  
[0.67584145],  
[0.37274035],  
[0.82884947],  
[0.82884947],  
[0.82775357],  
[0.67677682],  
[0.82884947],  
[0.67659093],  
[0.580881 ],  
[0.68158375],  
[0.68158375],  
[0.67659093],  
[0.64342724],  
[0.66980956],  
[0.82775357],  
[0.50848033],  
[0.68443906],  
[0.67659093],  
[0.580881 ],  
[0.50848033],  
[0.68158375],  
[0.580881 ],  
[0.76292615],  
[0.61319205],  
[0.73413329],  
[0.75330056],  
[0.55051271],  
[0.77097323],  
[0.6772302 ],  
[0.67533356],  
[0.83326806],  
[0.68443906],  
[0.580881 ],  
[0.63419422],  
[0.7228188 ],  
[0.65680173],  
[0.6090394 ],  
[0.67533356],  
[0.66783849],  
[0.6090394 ],  
[0.94505787],
```

```
[0.48642845],  
[0.78799711],  
[0.55051271],  
[0.38275755],  
[0.580881 ],  
[0.79066733],  
[0.6090394 ],  
[0.68569583],  
[0.71192401],  
[0.6627908 ],  
[0.84403584],  
[0.6090394 ],  
[0.6090394 ],  
[0.47606618],  
[0.6090394 ],  
[0.6090394 ],  
[0.6090394 ],  
[0.6090394 ],  
[0.7228188 ],  
[0.73137773],  
[0.73137773],  
[0.39883586],  
[0.6090394 ],  
[0.73600643],  
[0.73600643],  
[0.580881 ],  
[0.32940961],  
[0.76595404],  
[0.28136853],  
[0.69975508],  
[0.65075756],  
[0.39883586],  
[0.63869052],  
[0.61124929],  
[0.61034357],  
[0.64718765],  
[0.62639229],  
[0.52756778],  
[0.64718765],  
[0.580881 ],  
[0.580881 ],  
[0.580881 ],  
[0.6090394 ],  
[0.62639229],  
[0.51391119],  
[0.52756778],  
[0.50848033],  
[0.67533356],  
[0.51598402],  
[0.70605117],  
[0.47606618],  
[0.54351192],  
[0.65997205],  
[0.63419422],  
[0.62639229],  
[0.82830724],  
[0.75557238],  
[0.51625892],  
[0.72739062],  
[0.65416013],  
[0.75873812],  
[0.40091909],  
[0.580881 ],
```

```
[0.580881 ],  
[0.62971754],  
[0.580881 ],  
[0.74789417],  
[0.71269941],  
[0.62971754],  
[0.82764506],  
[0.6772302 ],  
[0.53204447],  
[0.51625892],  
[0.75873812],  
[0.63648916],  
[0.580881 ],  
[0.71269941],  
[0.580881 ],  
[0.580881 ],  
[0.43267437],  
[0.6772302 ],  
[0.38377226],  
[0.83414973],  
[0.78179043],  
[0.55141843],  
[0.72739062],  
[0.48453181],  
[0.580881 ],  
[0.4411279 ],  
[0.64982051],  
[0.65680173],  
[0.01782811],  
[0.66422425],  
[0.63648916],  
[0.66422425],  
[0.580881 ],  
[0.66422425],  
[1. ],  
[0.66422425],  
[0.50496028],  
[0.66422425],  
[0.50178996],  
[0.65813999],  
[0.65813999],  
[0.77332177],  
[0.77332177],  
[0.61825008],  
[0.86065534],  
[0.67055532],  
[0.67055532],  
[0.79066733],  
[0.66154222],  
[0.77332177],  
[0.43267437],  
[0.69873128],  
[0.85185791],  
[0.580881 ],  
[0.580881 ],  
[0.72063411],  
[0.65648884],  
[0.28967337],  
[0.73137773],  
[0.28967337],  
[0.28967337],  
[0.65498432],  
[0.85185791],  
[0.66154222],
```

```
[0.71964144],  
[0.28967337],  
[0.7584434 ],  
[0.7682045 ],  
[0.79066733],  
[0.74243622],  
[0.90526567],  
[0.63869052],  
[0.580881 ],  
[0.81532025],  
[0.75187481],  
[0.80770453],  
[0.55790117],  
[0.77772982],  
[0.81356914],  
[0.6772302 ],  
[0.65372802],  
[0.61124929],  
[0.6772302 ],  
[0.81532025],  
[0.580881 ],  
[0.73347362],  
[0.493519 ],  
[0.55051271],  
[0.81356914],  
[0.493519 ],  
[0.81356914],  
[0.580881 ],  
[0.73347362],  
[0.84557975],  
[0.66218738],  
[0.66218738],  
[0.65997205],  
[0.54594273],  
[0.6090394 ],  
[0.63869052],  
[0.55272261],  
[0.580881 ],  
[0.71482062],  
[0.85972869],  
[0.85972869],  
[0.75260569],  
[0.87342147],  
[0.75260569],  
[0.8647566 ],  
[0.85972869],  
[0.580881 ],  
[0.63419422],  
[0.580881 ],  
[0.85972869],  
[0.69578019],  
[0.85972869],  
[0.63869052],  
[0.63419422],  
[0.90852021],  
[0.580881 ],  
[0.62810728],  
[0.580881 ],  
[0.78228651],  
[0.6090394 ],  
[0.70597348],  
[0.67533356],  
[0.64982051],  
[0.580881 ],
```

```
[0.7228188 ],
[0.74777457],
[0.8647566 ],
[0.580881 ],
[0.82031164],
[0.66010808],
[0.69873128],
[0.82406412],
[0.580881 ],
[0.63419422],
[0.71192401],
[0.75162269],
[0.71192401],
[0.71192401],
[0.69578019],
[0.75440045],
[0.74243622],
[0.40091909],
[0.79669917],
[0.76352178],
[0.76352178],
[0.74129247],
[0.40091909],
[0.48447552],
[0.40091909],
[0.75873812],
[0.76460855],
[0.76460855],
[0.75873812])]
```

In [40]: no_duplicates_data_check['popularity_normalised'] = X_train_minmax
no_duplicates_data_check

Out[40]:

	reviews_count	average_rating	product_popularity	popularity_normalised
0	35	4.5	4.500000	0.580881
1	4	4.8	4.557831	0.638691
2	42	4.9	4.785957	0.866731
4	11	4.7	4.579279	0.660130
5	30	4.9	4.756684	0.837469
...
840	151	4.3	4.319970	0.400919
841	135	4.7	4.677924	0.758738
842	190	4.7	4.683797	0.764609
843	190	4.7	4.683797	0.764609
844	135	4.7	4.677924	0.758738

733 rows × 4 columns

In [41]: no_duplicates_data

Out[41]:

	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
...
840	940	FX2858	72	120	InStock	White	Shoes	Women/Shoes
841	941	H00667	70	100	InStock	White	Shoes	Women/Shoes
842	942	GZ7705	35	50	InStock	Black	Shoes	Kids/Shoes
843	943	GZ7706	40	50	InStock	Pink	Shoes	Kids/Shoes
844	944	FY6503	70	100	InStock	Black	Shoes	Women/Shoes

733 rows × 12 columns

In [42]: no_duplicates_data['popularity'] = no_duplicates_data_check['popularity_normalised
no_duplicates_data

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
4	104	GM0239	65	87	InStock	Blue	Clothing	Men/Clothing
5	105	FX7449	110	143	InStock	Grey	Shoes	Women/Shoes
...
840	940	FX2858	72	120	InStock	White	Shoes	Women/Shoes
841	941	H00667	70	100	InStock	White	Shoes	Women/Shoes
842	942	GZ7705	35	50	InStock	Black	Shoes	Kids/Shoes
843	943	GZ7706	40	50	InStock	Pink	Shoes	Kids/Shoes
844	944	FY6503	70	100	InStock	Black	Shoes	Women/Shoes

733 rows × 13 columns

In [43]: no_duplicates_data = no_duplicates_data.reset_index().drop(['index'], axis=1)
no_duplicates_data

Out[43]:

	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
...
728	940	FX2858	72	120	InStock	White	Shoes	Women/Shoes
729	941	H00667	70	100	InStock	White	Shoes	Women/Shoes
730	942	GZ7705	35	50	InStock	Black	Shoes	Kids/Shoes
731	943	GZ7706	40	50	InStock	Pink	Shoes	Kids/Shoes
732	944	FY6503	70	100	InStock	Black	Shoes	Women/Shoes

733 rows × 13 columns



In [44]: no_duplicates_data.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 733 entries, 0 to 732
Data columns (total 13 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  
 12  popularity       733 non-null    float64 
dtypes: float64(2), int32(1), int64(3), object(7)
memory usage: 71.7+ KB
```

In [45]: #Now we need to plot the box_plot to find the outliers and delete it

no_duplicates_data_scaled = no_duplicates_data

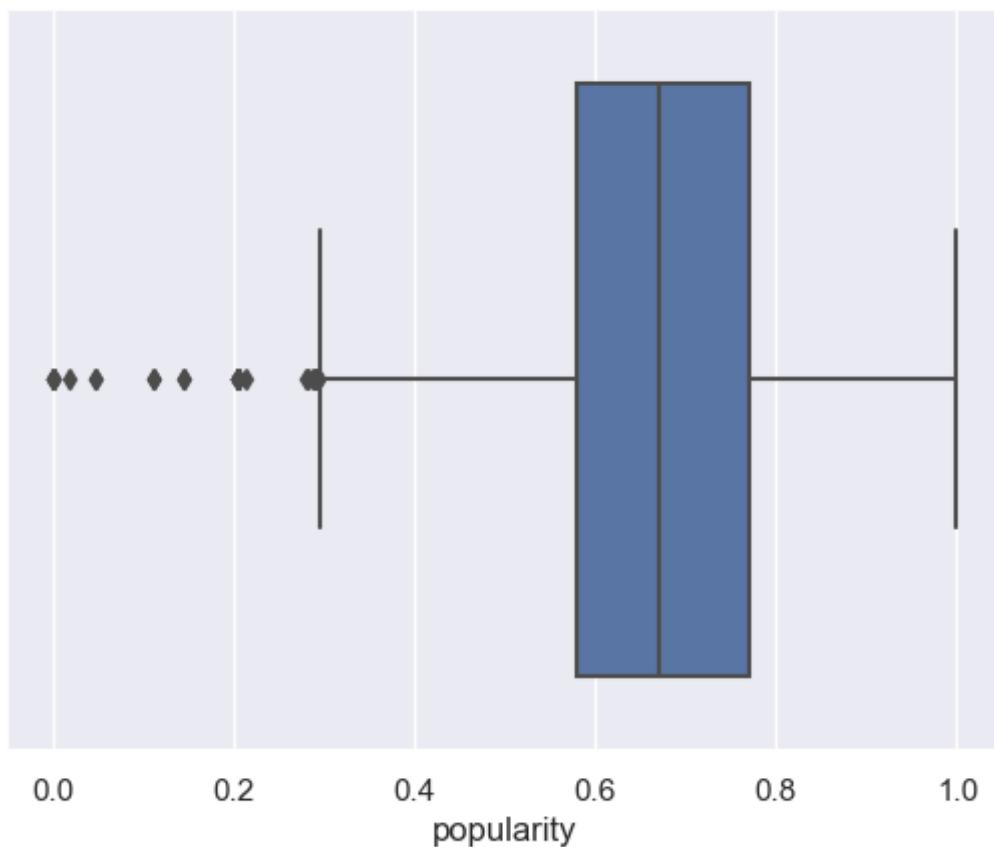
In [46]: no_duplicates_data_scaled.head(10)

Out[46]:

	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
5	106	GM5505	80	100	InStock	Black	Clothing	Men/Clothing
6	107	GP4975	60	75	InStock	Black	Clothing	Kids/Clothing
7	108	FS3923	40	47	InStock	Black	Clothing	Women/Clothing
8	109	GP4967	65	85	InStock	Black	Clothing	Men/Clothing
9	110	GL1127	60	72	InStock	Black	Clothing	Women/Clothing

In [47]: `sns.boxplot(no_duplicates_data_scaled['popularity'])`

Out[47]: <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: $Q1 = \text{quantile}(0.25)$ $Q3 = \text{quantile}(0.75)$ $IQR = Q3 - Q1$

In [48]: `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.

$$\text{Lower_Fence} = Q1 - (1.5 * \text{IQR})$$

$$\text{Upper_Fence} = Q3 + (1.5 * \text{IQR})$$

```
In [49]: 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= 0.5808810039885035
q3_1= 0.7709732262232256
iqr_1= 0.1900922222347221
lower_fence_1= 0.29574267063642035
upper_fence_1= 1.0561115595753088
```

```
In [50]: #outliers_data are all the data above the upper fence and below the lower fence
outliers_data_popularity = no_duplicates_data_scaled.loc[(no_duplicates_data_scaled
                                                               (no_duplicates_data_scaled['popularity'] > upper_fence_1) | (no_duplicates_data_scaled['popularity'] < lower_fence_1))]
```

	product ID	identifier	selling_price	original_price	availability	color	category	breadcrumbs
73	180	H04498	56	80	InStock	White	Shoes	Women/Shoes
76	183	H02975	56	80	InStock	Yellow	Shoes	Women/Shoes
90	201	GT0140	36	45	InStock	Blue	Clothing	Women/Clothing
98	209	GL3961	36	45	InStock	Black	Clothing	Women/Clothing
177	292	GR9652	36	45	InStock	Black	Clothing	Women/Clothing

```
In [51]: #examine outliers_data
outliers_data_popularity.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 37 entries, 73 to 641
Data columns (total 13 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   product ID      37 non-null     int64  
 1   identifier       37 non-null     object  
 2   selling_price    37 non-null     int64  
 3   original_price   37 non-null     int32  
 4   availability     37 non-null     object  
 5   color            37 non-null     object  
 6   category         37 non-null     object  
 7   breadcrumbs      37 non-null     object  
 8   average_rating   37 non-null     float64 
 9   reviews_count    37 non-null     int64  
 10  gender           37 non-null     object  
 11  ageGroup         37 non-null     object  
 12  popularity       37 non-null     float64 
dtypes: float64(2), int32(1), int64(3), object(7)
memory usage: 3.9+ KB
```

In [52]: *#We see that we have 37 outliers, due to the amount of rows we can just delete it*
`non_outliers_data_popularity = no_duplicates_data_scaled.loc[~((no_duplicates_data_scaled['popularity'] < 10) | (no_duplicates_data_scaled['popularity'] > 100))]`
#and resetting indexes
`non_outliers_data_popularity = non_outliers_data_popularity.reset_index().drop(['index'], axis=1)`
`non_outliers_data_popularity`

Out[52]:

	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
...
691	940	FX2858	72	120	InStock	White	Shoes	Women/Shoes
692	941	H00667	70	100	InStock	White	Shoes	Women/Shoes
693	942	GZ7705	35	50	InStock	Black	Shoes	Kids/Shoes
694	943	GZ7706	40	50	InStock	Pink	Shoes	Kids/Shoes
695	944	FY6503	70	100	InStock	Black	Shoes	Women/Shoes

696 rows × 13 columns

In [53]: `non_outliers_data_popularity.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 696 entries, 0 to 695
Data columns (total 13 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   product ID      696 non-null    int64  
 1   identifier       696 non-null    object  
 2   selling_price    696 non-null    int64  
 3   original_price   696 non-null    int32  
 4   availability     696 non-null    object  
 5   color            696 non-null    object  
 6   category         696 non-null    object  
 7   breadcrumbs      696 non-null    object  
 8   average_rating   696 non-null    float64 
 9   reviews_count    696 non-null    int64  
 10  gender           696 non-null    object  
 11  ageGroup         696 non-null    object  
 12  popularity       696 non-null    float64 
dtypes: float64(2), int32(1), int64(3), object(7)
memory usage: 68.1+ KB
```

In [54]: `non_outliers_data_popularity['color'].value_counts()`

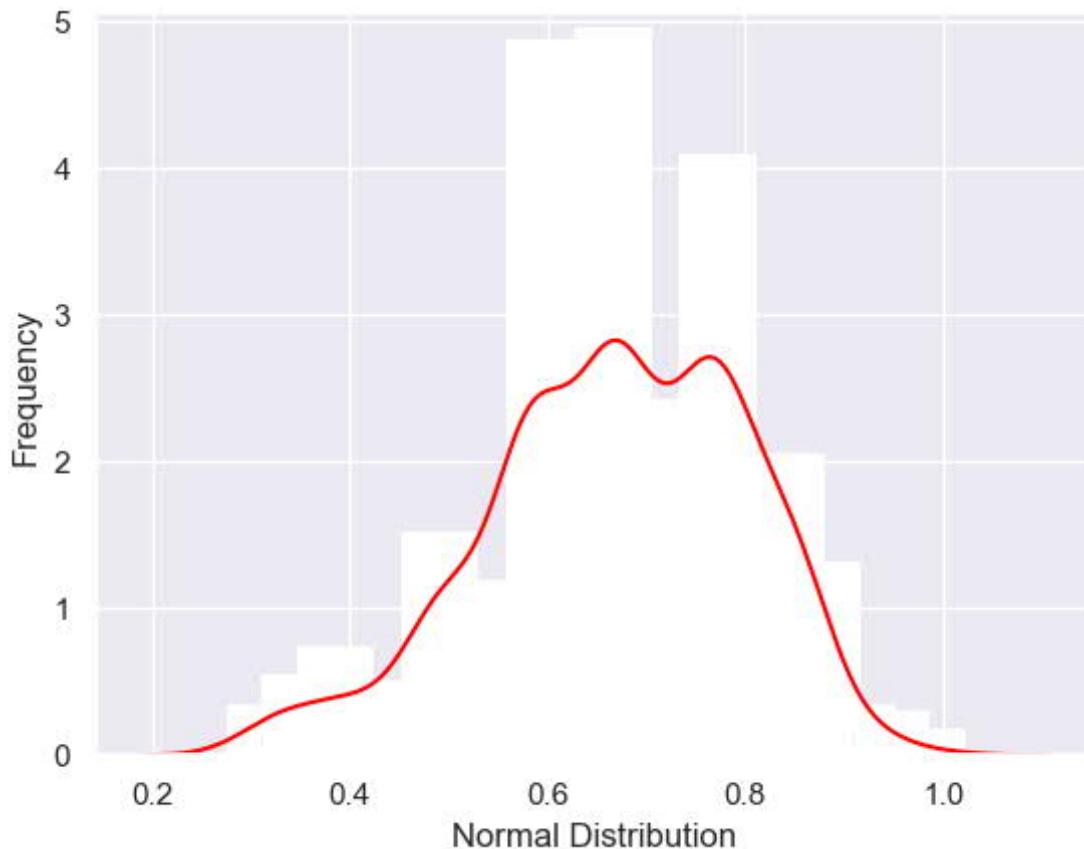
```
Out[54]:
```

White	187
Black	143
Blue	85
Grey	69
Pink	54
Green	51
Purple	40
Red	23
Multi	22
Yellow	13
Orange	9

Name: color, dtype: int64

In [55]: `#Lets check if we can say that distribution is normal
ax = sns.distplot(non_outliers_data_popularity['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(bell shaped)`

Out[55]: [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)
- $H_1 : \text{At least one of the means differ}$

```
In [56]: #Lets separate the three samples (one for age and gender category) into a variable
Women = non_outliers_data_popularity[non_outliers_data_popularity['gender'] == 'Woman']
Men = non_outliers_data_popularity[non_outliers_data_popularity['gender'] == 'Men']
Kids = non_outliers_data_popularity[non_outliers_data_popularity['gender'] == 'Kid']
```

```
In [57]: Women
```

```
Out[57]: 0    0.580881
1    0.638691
4    0.837469
7    0.580881
9    0.520085
...
689   0.400919
690   0.484476
691   0.400919
692   0.758738
695   0.758738
Name: popularity, Length: 327, dtype: float64
```

```
In [58]: Men
```

```
Out[58]: 3      0.660130
          5      0.580881
          8      0.820066
         10     0.465982
         12     0.747705
          ...
        668    0.747775
        669    0.864757
        673    0.698731
        676    0.634194
        682    0.754400
Name: popularity, Length: 268, dtype: float64
```

In [59]: Kids

```
Out[59]: 2      0.866731
          6      0.580881
         11     0.634194
         16     0.800718
         31     0.755572
          ...
        686    0.763522
        687    0.763522
        688    0.741292
        693    0.764609
        694    0.764609
Name: popularity, Length: 101, dtype: float64
```

```
In [60]: #Now, run a one-way ANOVA.
f_statistic, p_value = scipy.stats.f_oneway(Men, Kids, Women, )
print("F_Statistic: {0}, P-Value: {1}".format(f_statistic,p_value))
```

```
F_Statistic: 26.6432832902332, P-Value: 7.116508862295456e-12
```

In []:

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 [61]: color1=pd.DataFrame(non_outliers_data_popularity[non_outliers_data_popularity['gender']=='Men'])
color2=pd.DataFrame(non_outliers_data_popularity[non_outliers_data_popularity['gender']=='Women'])
color3=pd.DataFrame(non_outliers_data_popularity[non_outliers_data_popularity['gender']=='Kids'])
```

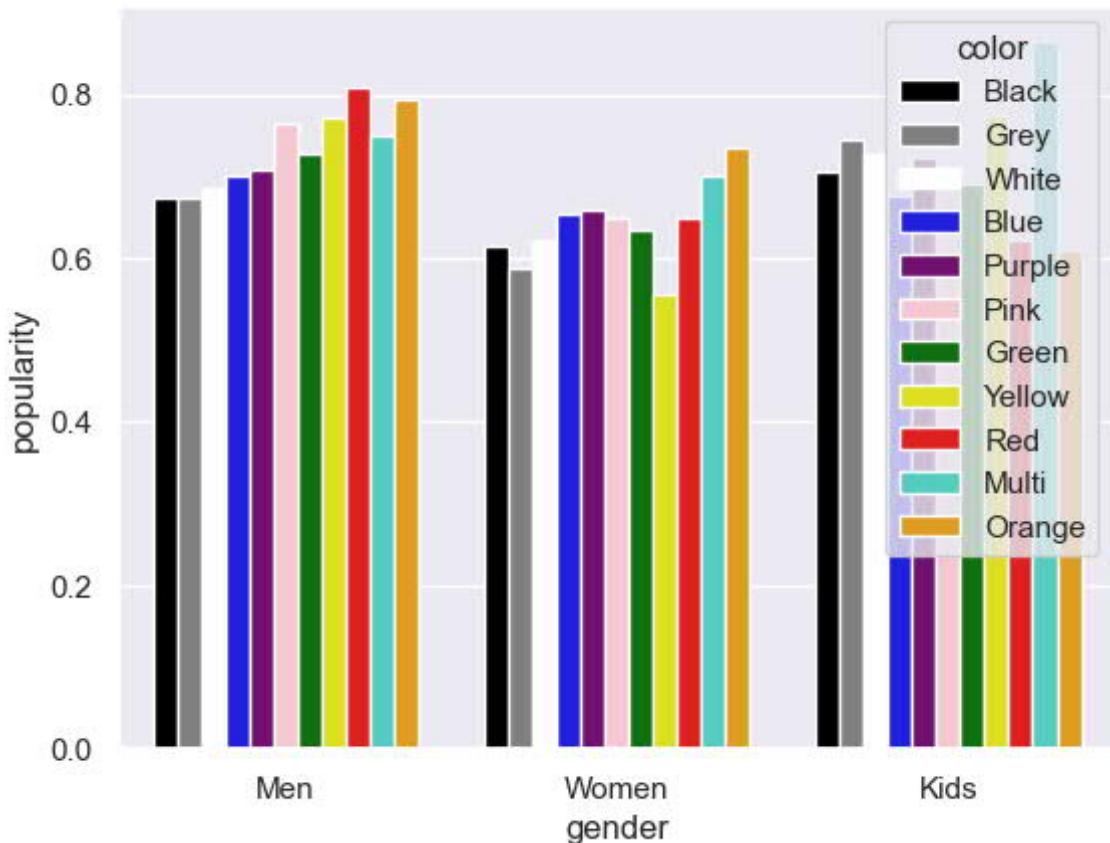
```
In [62]: #Lets examine Popularity by gender Group mean
```

```
genderGroupMeanPopularity = non_outliers_data_popularity.groupby(['gender', 'color']).mean()
#genderGroupMeanPopularity
```

```
In [63]: #Lets make visualisation on the popularity of colors amoung Women, Men, Kids
```

```
colors = ["black", "grey", "white", "blue", "purple", "pink", "green", "yellow", "red"]
sns.barplot(
    x="gender",
    y="popularity",
    hue="color",
    ci=None,
    order=["Men", "Women", "Kids"],
    palette=colors,
    data=non_outliers_data_popularity
)
```

```
Out[63]: <AxesSubplot:xlabel='gender', ylabel='popularity'>
```

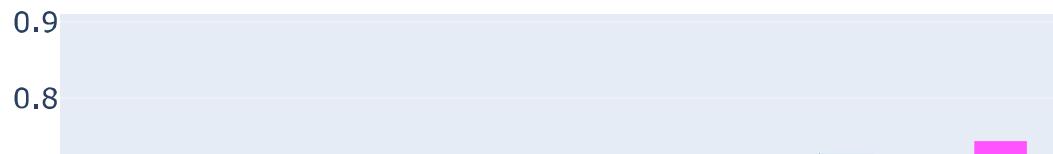


In [64]: #One more chart Popularity of color for gender
`import plotly.graph_objects as go`

```
fig = go.Figure()
fig.add_trace(go.Bar(x=color1[0],
                     y=genderGroupMeanPopularity['Kids'],
                     name='Kids',
                     marker_color='rgb(255, 83, 255)',
                     hovertemplate="  
".join([
                     "color: %{x}",
                     "popularity: %{y}",
                     "gender: Kids",]))
))
fig.add_trace(go.Bar(x=color2[0],
                     y=genderGroupMeanPopularity['Women'],
                     name='Women',
                     marker_color='rgb(26, 255, 30)',
                     hovertemplate="  
".join([
                     "color: %{x}",
                     "popularity: %{y}",
                     "gender: Women",]))
))
fig.add_trace(go.Bar(x=color3[0],
                     y=genderGroupMeanPopularity['Men'],
                     name='Men',
                     marker_color='rgb(70, 200, 255)',
                     hovertemplate="  
".join([
                     "color: %{x}",
                     "popularity: %{y}",
                     "gender: Men",]))
))
fig.update_layout(
    title='Popularity of color for gender',
    xaxis_tickfont_size=14,
```

```
yaxis=dict(
    title='Popularity',
    titlefont_size=16,
    tickfont_size=14,
),
legend=dict(
    x=1,
    y=1.0,
    bgcolor='rgba(255, 255, 255, 0)',
    bordercolor='rgba(255, 255, 255, 0)'
),
barmode='group',
bargap=0.3, # gap between bars of adjacent Location coordinates.
bargroupgap=0.2 # gap between bars of the same Location coordinate.
)
fig.show()
```

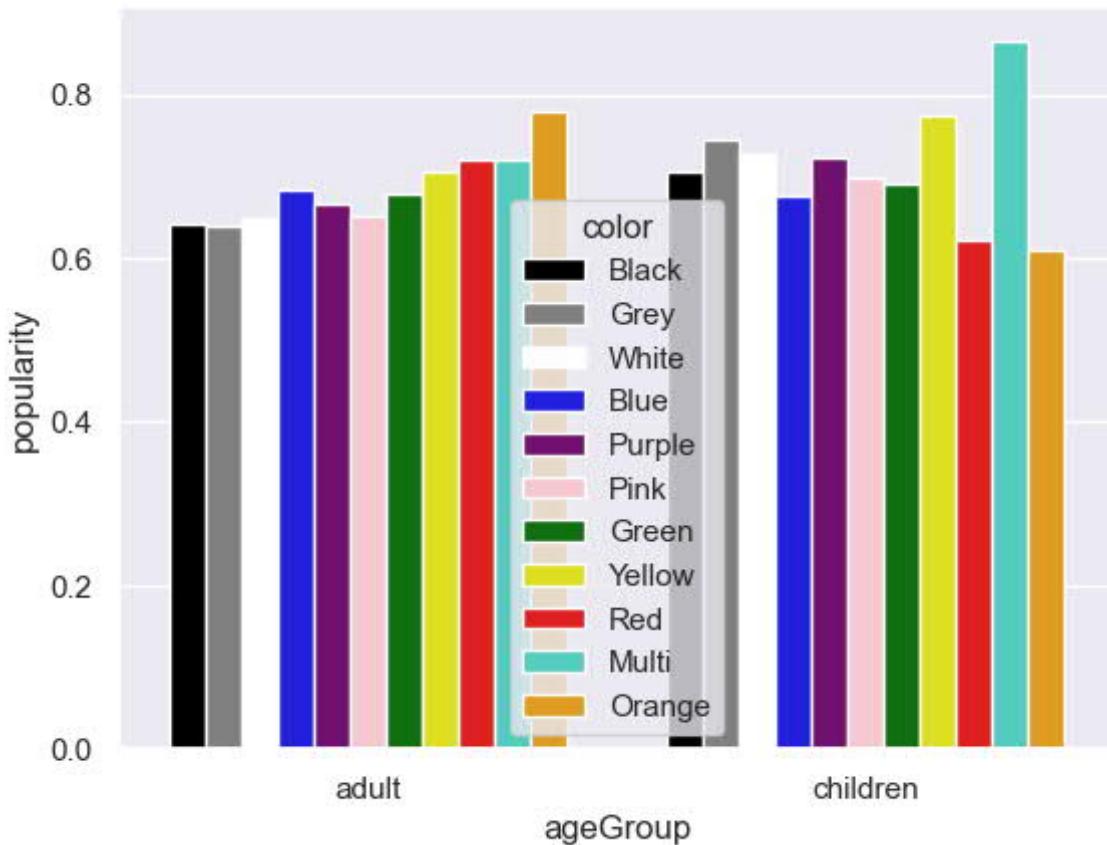
Popularity of color for gender



In [65]:

```
sns.barplot(
    x="ageGroup",
    y="popularity",
    hue="color",
    ci=None,
    palette=colors,
    data=non_outliers_data_popularity
)
```

Out[65]: <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 men - high popularity is at red, orange and Yellow option. For women - orange and multicolor. while for kids high popularity is for Multicolor color (second place Yellow)

We also see, that orange is also popular for adult

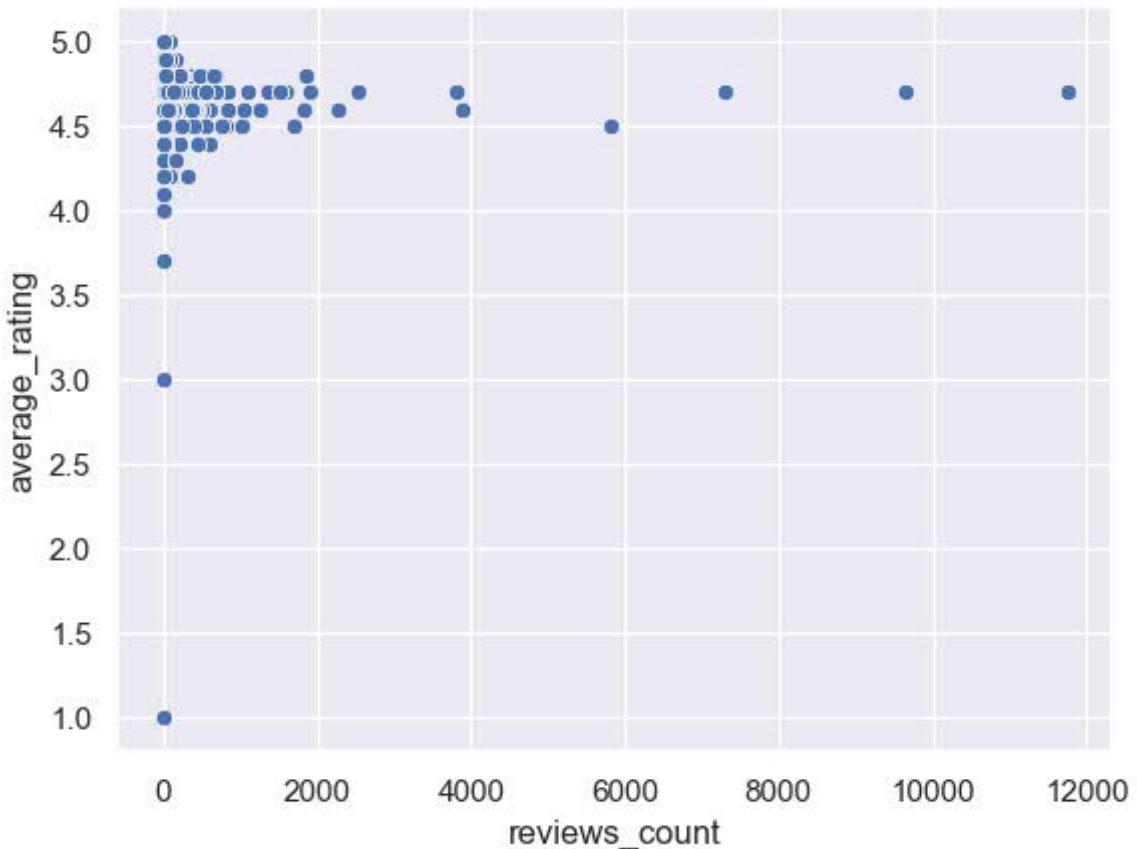
Task 2: Please analyze the correlation between ratings and reviews to determine which factors are most important to customers, in the end please create a dashboard to show your results.

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 [66]: #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_outliers_data_
```

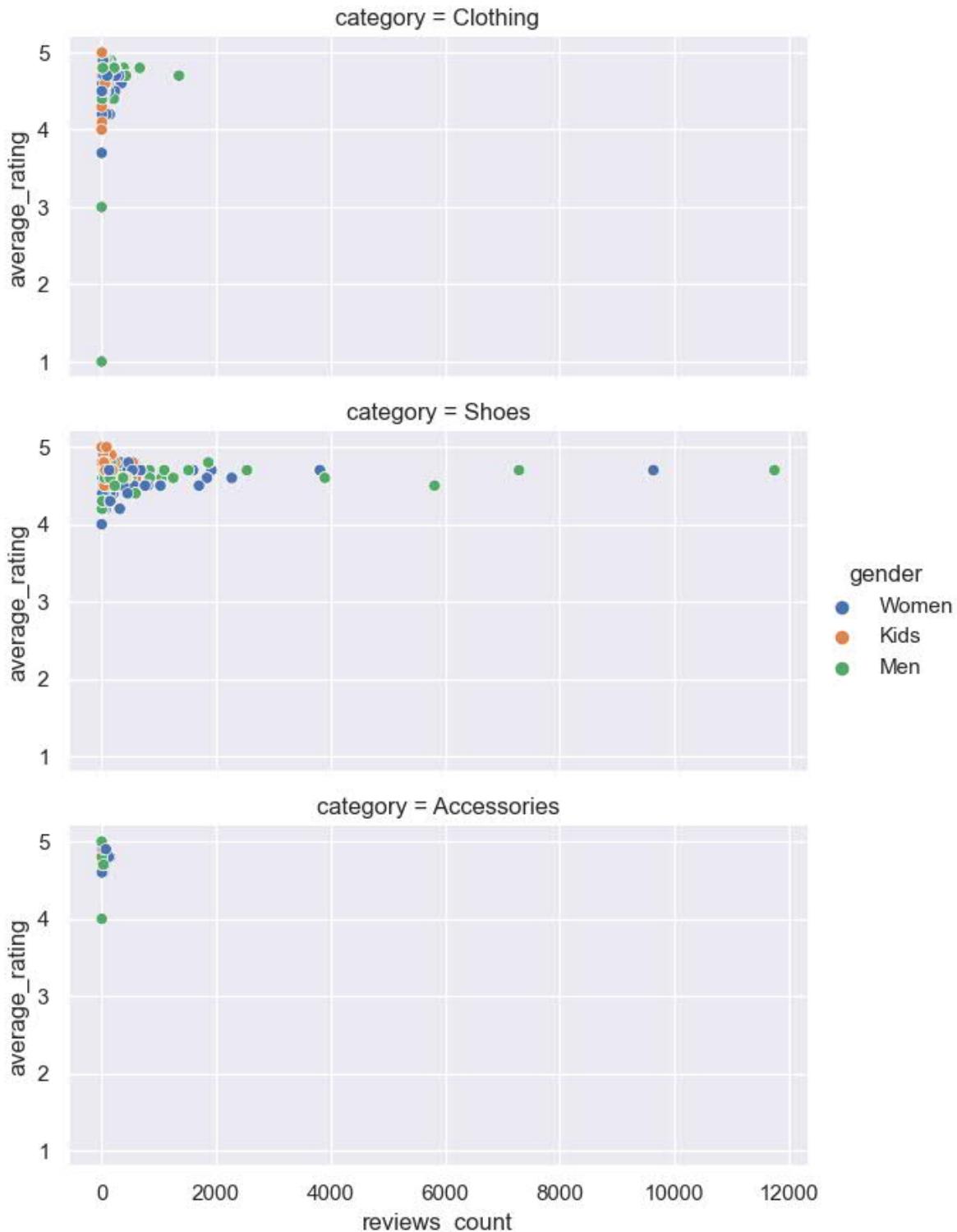


```
In [67]: #Lets see the pearson correlation  
scipy.stats.pearsonr(non_outliers_data_popularity['reviews_count'], non_outliers_d
```

```
Out[67]: PearsonRResult(statistic=0.013479366966699588, pvalue=0.7225992887221286)
```

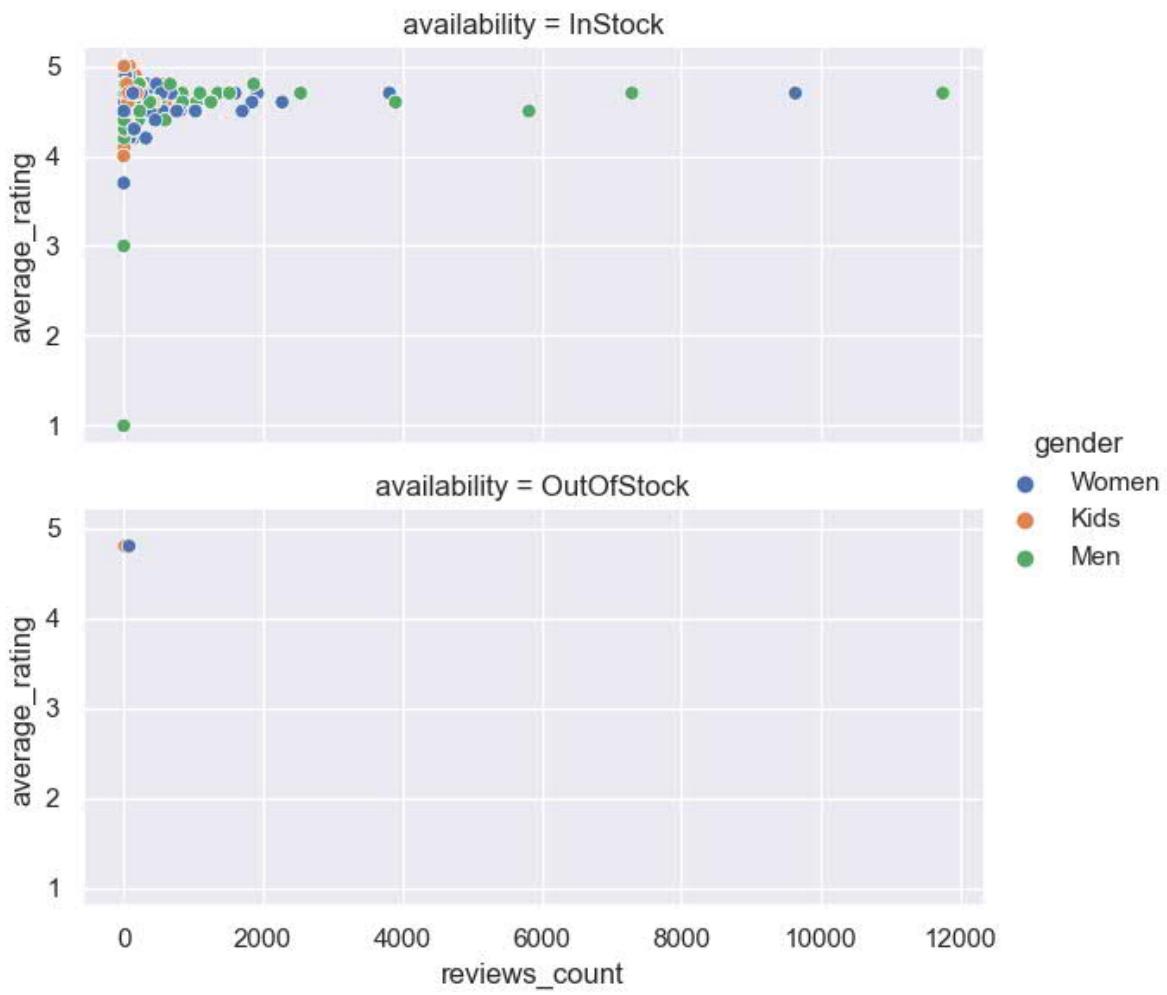
```
In [68]: sns.relplot(x="reviews_count", y="average_rating", hue="gender",  
row="category",  
data=non_outliers_data_popularity, height = 3, aspect = 2)
```

```
Out[68]: <seaborn.axisgrid.FacetGrid at 0x21aa0e5f190>
```



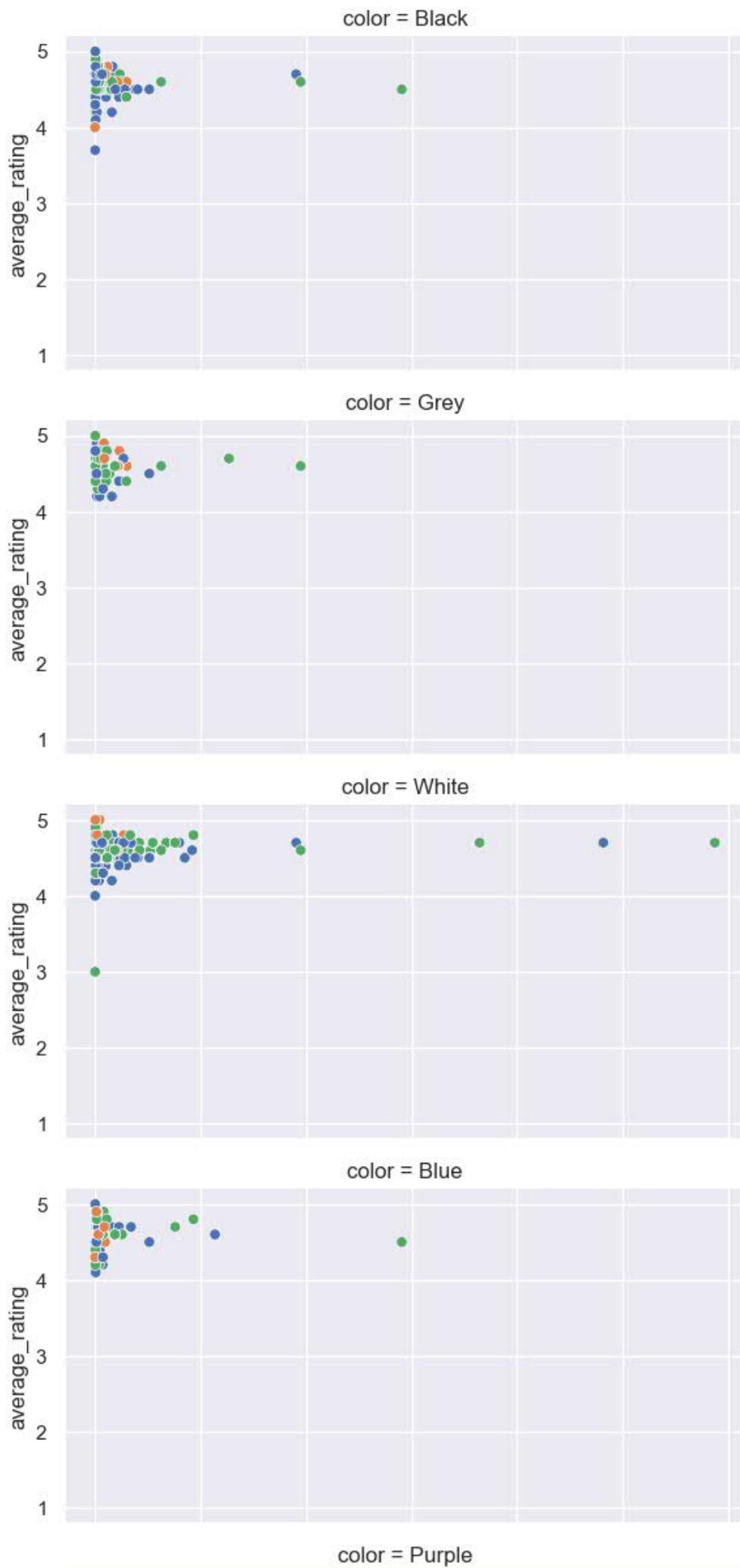
```
In [69]: sns.relplot(x="reviews_count", y="average_rating", hue="gender",
                   row="availability",
                   data=non_outliers_data_popularity, height = 3, aspect = 2)
```

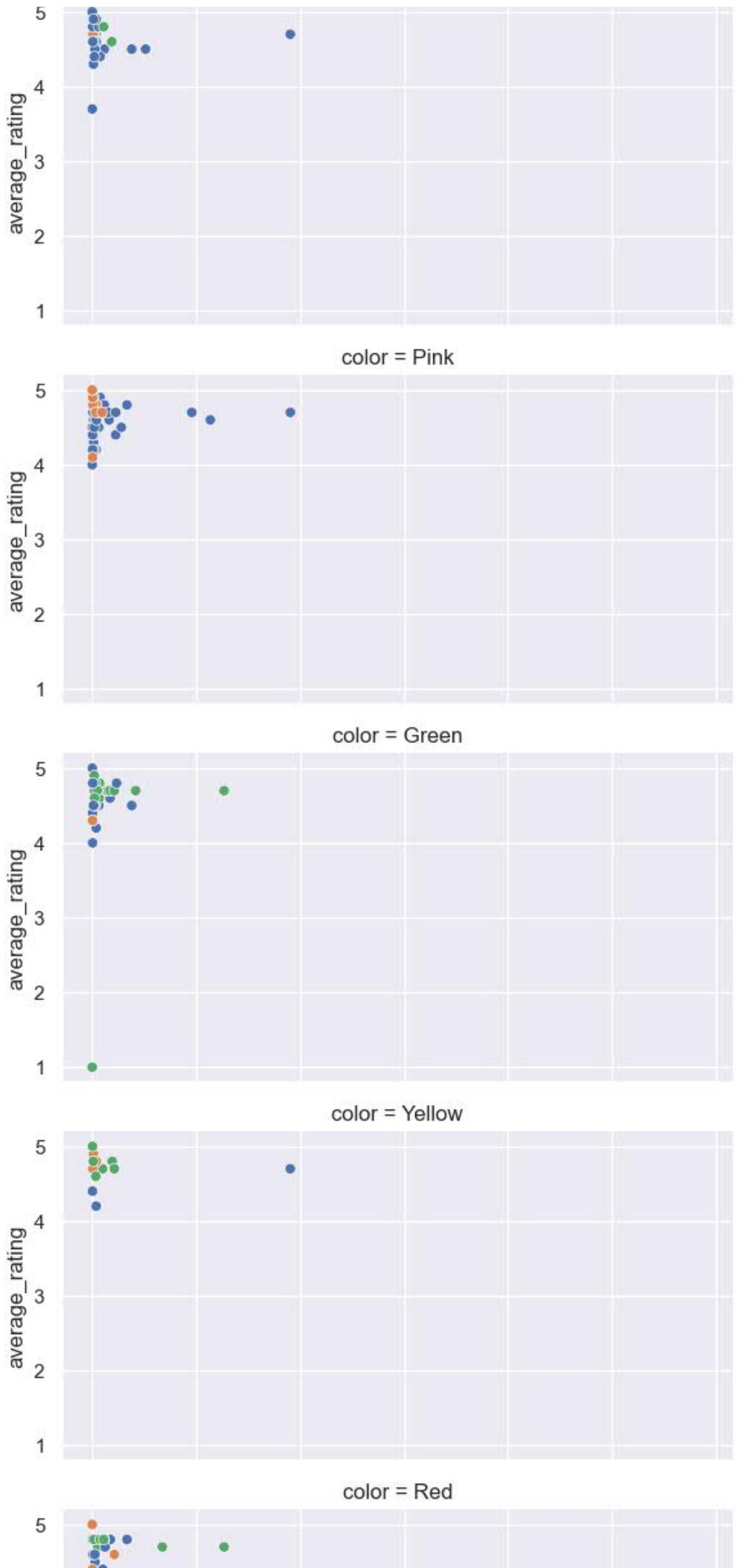
```
Out[69]: <seaborn.axisgrid.FacetGrid at 0x21a9fe8bdc0>
```

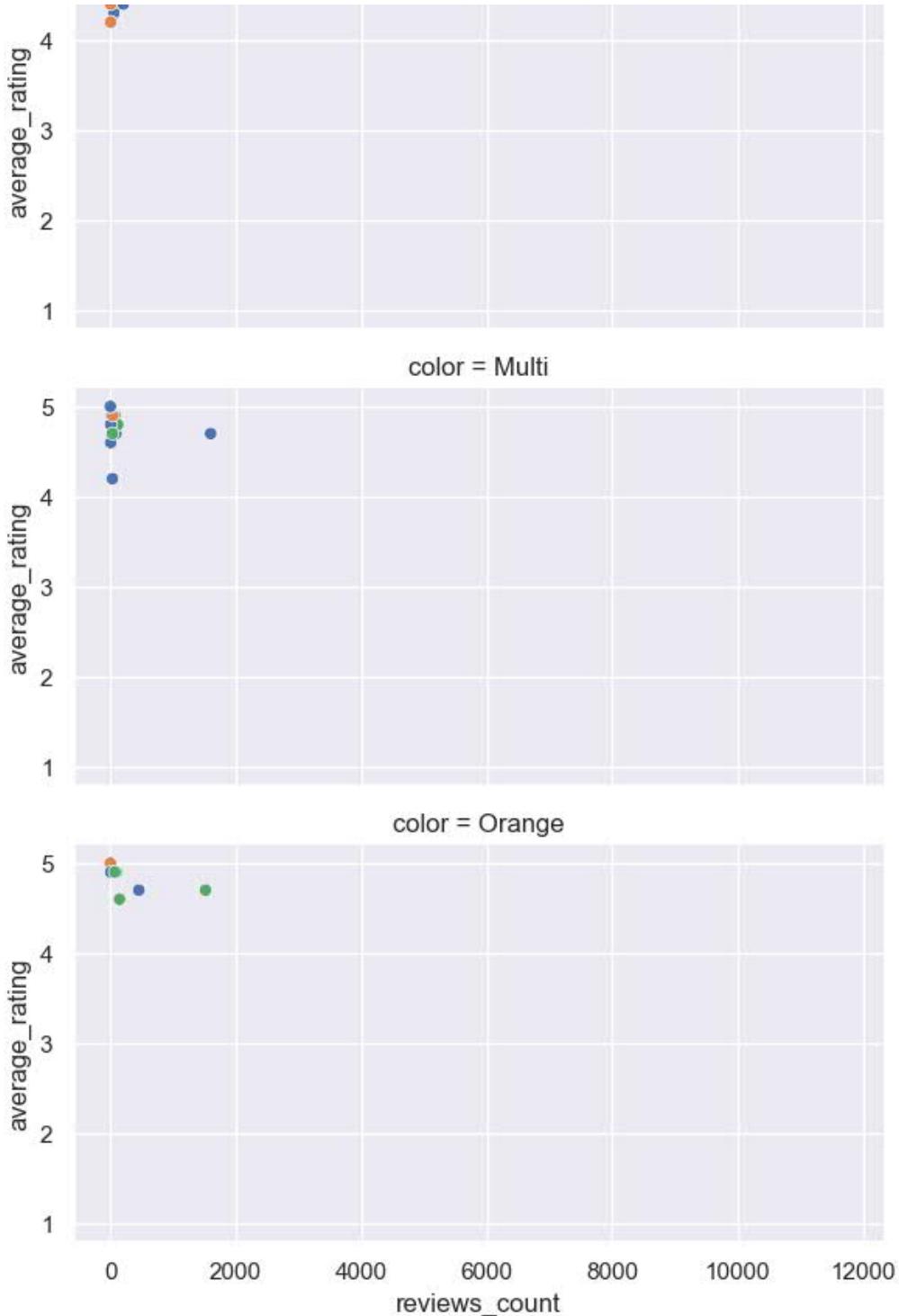


```
In [70]: sns.relplot(x="reviews_count", y="average_rating", hue="gender",
                     row="color",
                     data=non_outliers_data_popularity, height = 3, aspect = 2)
```

```
Out[70]: <seaborn.axisgrid.FacetGrid at 0x21aa1127ee0>
```







Conclusion on Task 2: correlation between ratings and reviews

Conclusion: Since the p-value > 0.05, we can not reject the Null hypothesis and conclude that there is no significant relationship between reviews_count and average_rating score.

Due to the visualisation, we can see it more clearly

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 [71]: # Descriptive statistics are very useful for initial exploration of the variables
non_outliers_data_popularity.describe(include='all')

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

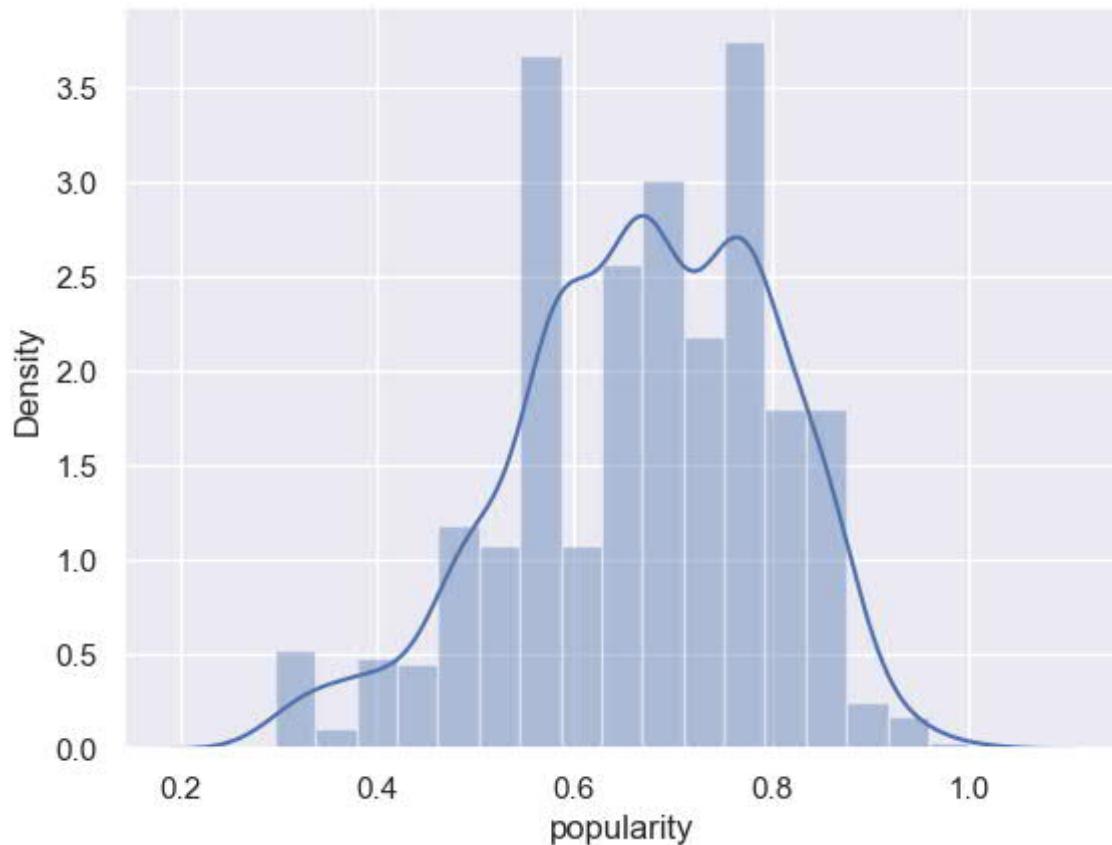
	product ID	identifier	selling_price	original_price	availability	color	category	bread
count	696.000000	696	696.000000	696.000000	696	696	696	696
unique	NaN	696	NaN	NaN	2	11	3	
top	NaN	FJ5089	NaN	NaN	InStock	White	Shoes	Women/
freq	NaN	1	NaN	NaN	694	187	336	
mean	521.050287	NaN	51.021552	66.741379	NaN	NaN	NaN	NaN
std	250.670403	NaN	27.151551	36.569505	NaN	NaN	NaN	NaN
min	100.000000	NaN	10.000000	14.000000	NaN	NaN	NaN	NaN
25%	293.750000	NaN	28.000000	35.000000	NaN	NaN	NaN	NaN
50%	517.500000	NaN	48.000000	65.000000	NaN	NaN	NaN	NaN
75%	744.250000	NaN	64.000000	85.000000	NaN	NaN	NaN	NaN
max	944.000000	NaN	240.000000	300.000000	NaN	NaN	NaN	NaN

Determining the variables of interest

```
In [72]: #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_outliers_data_popularity['popularity'])
```

```
Out[72]: <AxesSubplot:xlabel='popularity', ylabel='Density'>
```

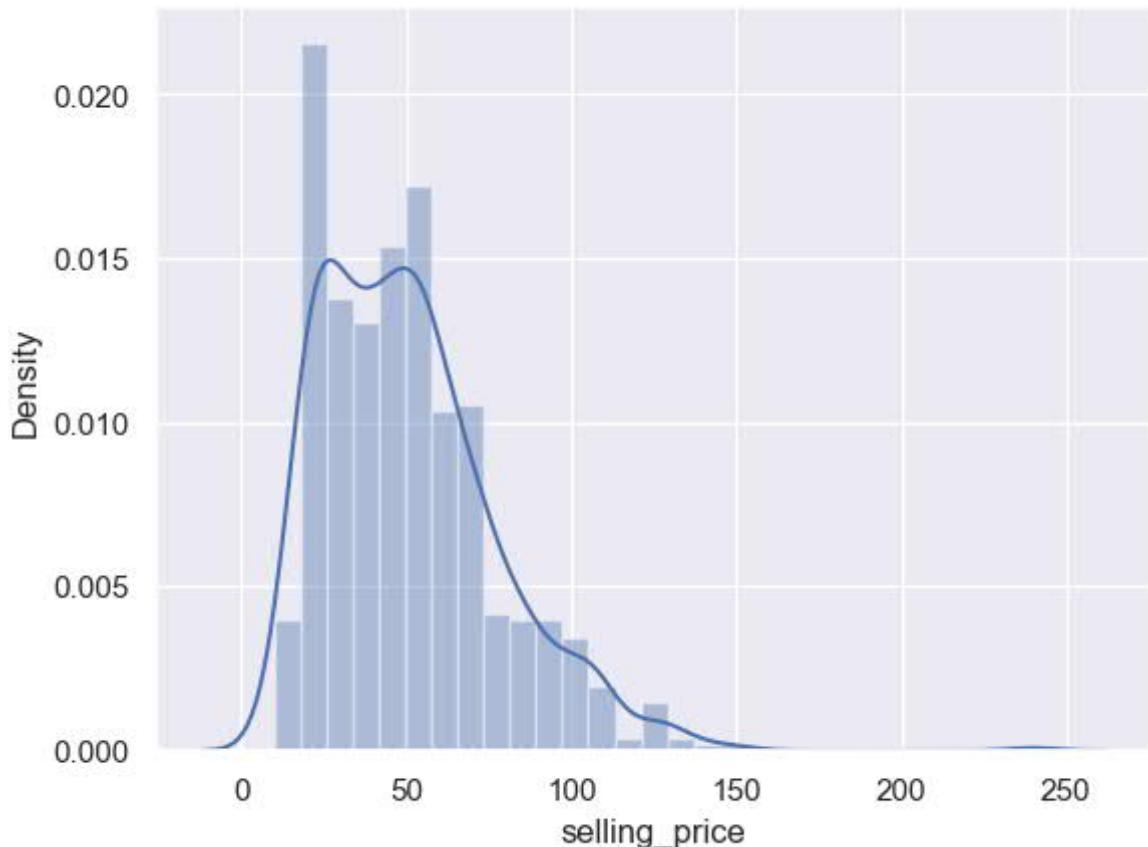


```
In [73]: non_outliers_data_popularity['popularity'].describe()
```

```
Out[73]: count    696.000000
mean      0.669200
std       0.130836
min      0.295955
25%      0.580881
50%      0.676591
75%      0.773322
max      1.000000
Name: popularity, dtype: float64
```

```
In [74]: #Lets also check selling_price
sns.distplot(non_outliers_data_popularity['selling_price'])
```

```
Out[74]: <AxesSubplot:xlabel='selling_price', ylabel='Density'>
```



```
In [75]: non_outliers_data_popularity['selling_price'].describe()
```

```
Out[75]: count    696.000000
mean      51.021552
std       27.151551
min      10.000000
25%     28.000000
50%     48.000000
75%     64.000000
max     240.000000
Name: selling_price, dtype: float64
```

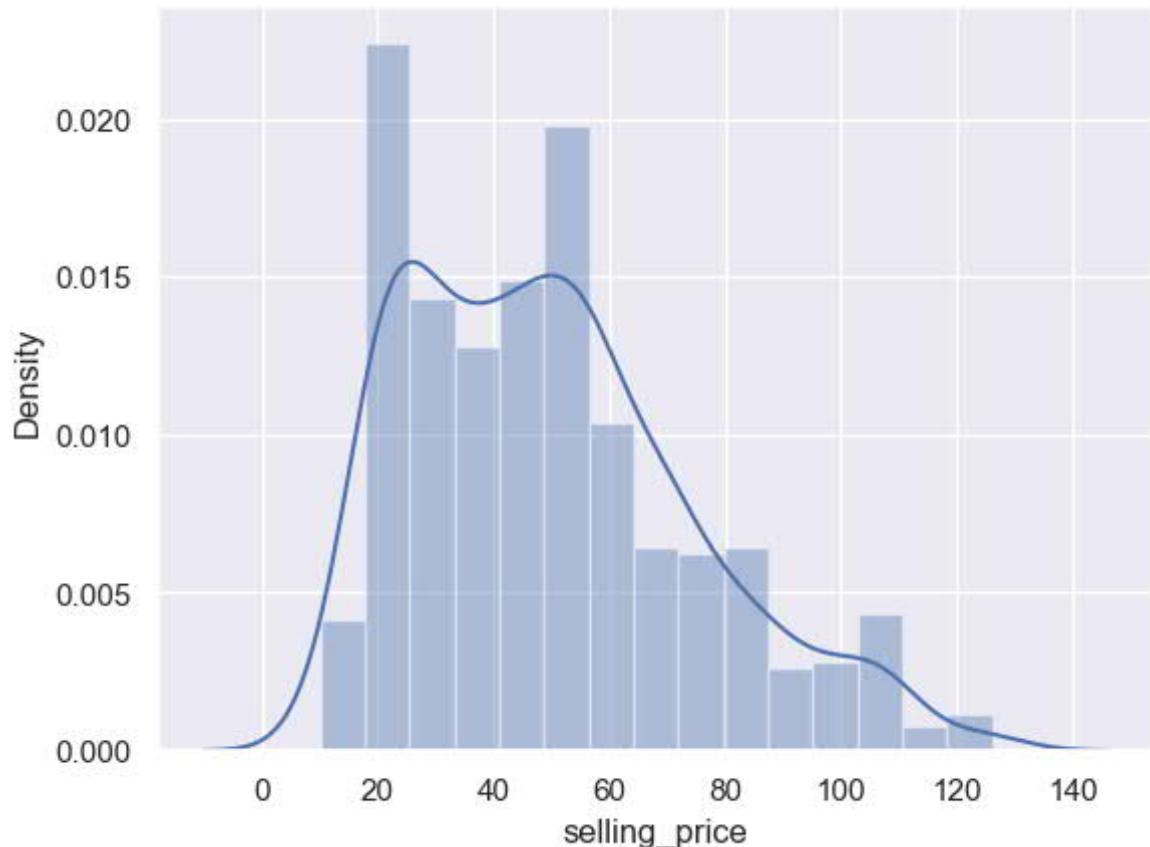
```
In [76]: # Obviously there are some outliers present in selling_price
```

```
# we can deal with the problem easily by removing 0.5%, or 1% of the problematic so
# Here, the outliers are situated around the higher prices (right side of the graph)

# Let's declare a variable that will be equal to the 99th percentile of the 'Price'
q = non_outliers_data_popularity['selling_price'].quantile(0.99)
# Then we can create a new DataFrame, with the condition that all prices must be be
data_1 = non_outliers_data_popularity[non_outliers_data_popularity['selling_price' < q]
# In this way we have essentially removed the top 1% of the data about 'Price'

sns.distplot(data_1['selling_price'])
```

```
Out[76]: <AxesSubplot:xlabel='selling_price', ylabel='Density'>
```



```
In [77]: non_outliers_data_popularity = data_1.reset_index().drop(['index'], axis=1)
```

```
In [78]: non_outliers_data_popularity
```

Out[78]:

	product ID	identifier	selling_price	original_price	availability	color	category	breadcrumbs
0	100	FJ5089	40	47	InStock	Black	Clothing	Women/Clothin
1	102	GC7946	70	87	InStock	White	Clothing	Kids/Clothin
2	104	GM0239	65	87	InStock	Blue	Clothing	Men/Clothin
3	105	FX7449	110	143	InStock	Grey	Shoes	Women/Shoe
4	106	GM5505	80	100	InStock	Black	Clothing	Men/Clothin
...
682	940	FX2858	72	120	InStock	White	Shoes	Women/Shoe
683	941	H00667	70	100	InStock	White	Shoes	Women/Shoe
684	942	GZ7705	35	50	InStock	Black	Shoes	Kids/Shoe
685	943	GZ7706	40	50	InStock	Pink	Shoes	Kids/Shoe
686	944	FY6503	70	100	InStock	Black	Shoes	Women/Shoe

687 rows × 13 columns

Checking the OLS assumptions

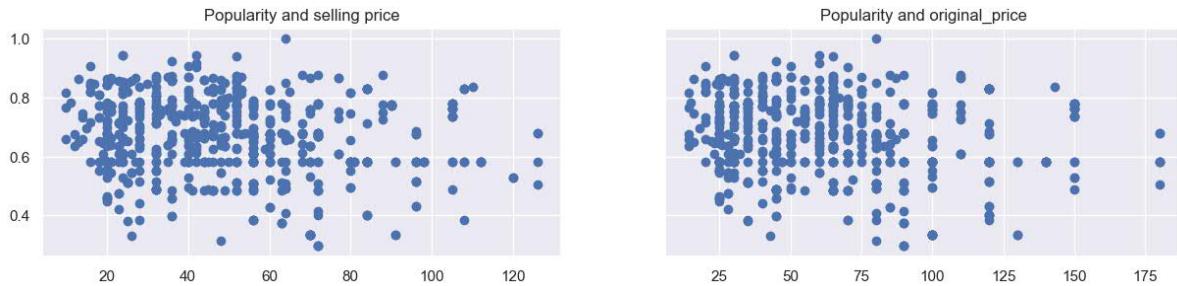
```
In [79]: f, (ax1, ax2) = plt.subplots(1, 2, sharey=True, figsize=(15,3))
ax1.scatter(non_outliers_data_popularity['selling_price'],non_outliers_data_popula
```

```

ax1.set_title('Popularity and selling price')
ax2.scatter(non_outliers_data_popularity['original_price'],non_outliers_data_popularity['popularity'])
ax2.set_title('Popularity and original_price')

plt.show()

```



In [80]: `regressionAnalys = non_outliers_data_popularity.loc[:, ['selling_price', 'color',`

In [81]: `regressionAnalys.columns.values`

Out[81]: `array(['selling_price', 'color', 'category', 'gender', 'popularity'],
 dtype=object)`

Create dummy variables

In [82]: `# 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[82]:

	selling_price	popularity	color_Blue	color_Green	color_Grey	color_Multi	color_Orange	color_Pink
0	40	0.580881	0	0	0	0	0	0
1	70	0.866731	0	0	0	0	0	0
2	65	0.660130	1	0	0	0	0	0
3	110	0.837469	0	0	1	0	0	0
4	80	0.580881	0	0	0	0	0	0

In [83]: `# To make our data frame more organized rearrange a bit
data_with_dummies.columns.values`

Out[83]: `array(['selling_price', 'popularity', 'color_Blue', 'color_Green',
 'color_Grey', 'color_Multi', 'color_Orange', 'color_Pink',
 'color_Purple', 'color_Red', 'color_White', 'color_Yellow',
 'category_Clothing', 'category_Shoes', 'gender_Men',
 'gender_Women'], dtype=object)`

In [84]: `# To make the code a bit more parametrized, declare a new variable that will contain the column names`

```

cols = ['popularity', 'selling_price', 'color_Blue', 'color_Green',
        'color_Grey', 'color_Multi', 'color_Orange', 'color_Pink',
        'color_Purple', 'color_Red', 'color_White', 'color_Yellow',
        'category_Clothing', 'category_Shoes', 'gender_Men',
        'gender_Women']

```

In [85]: `# 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[85]:

	popularity	selling_price	color_Blue	color_Green	color_Grey	color_Multi	color_Orange	color
0	0.580881	40	0	0	0	0	0	0
1	0.866731	70	0	0	0	0	0	0
2	0.660130	65	1	0	0	0	0	0
3	0.837469	110	0	0	1	0	0	0
4	0.580881	80	0	0	0	0	0	0

```
In [ ]: '''price_notnorm = np.array(data_preprocessed['selling_price']).reshape(-1, 1)
#going to standardise selling_price to the range of (0,1)
min_max_scaler = preprocessing.MinMaxScaler()
price_minmax = min_max_scaler.fit_transform(price_notnorm)
#price_minmax
data_preprocessed['selling_price']=price_minmax
data_preprocessed
'''
```

In []:

Linear regression model

In [87]:

```
#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 [88]:

```
#Scale the data

# Create a scaler object
scaler = StandardScaler()

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

Out[88]:

StandardScaler()

In [89]:

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

Train Test Split

In [90]:

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

Out[91]: LinearRegression()

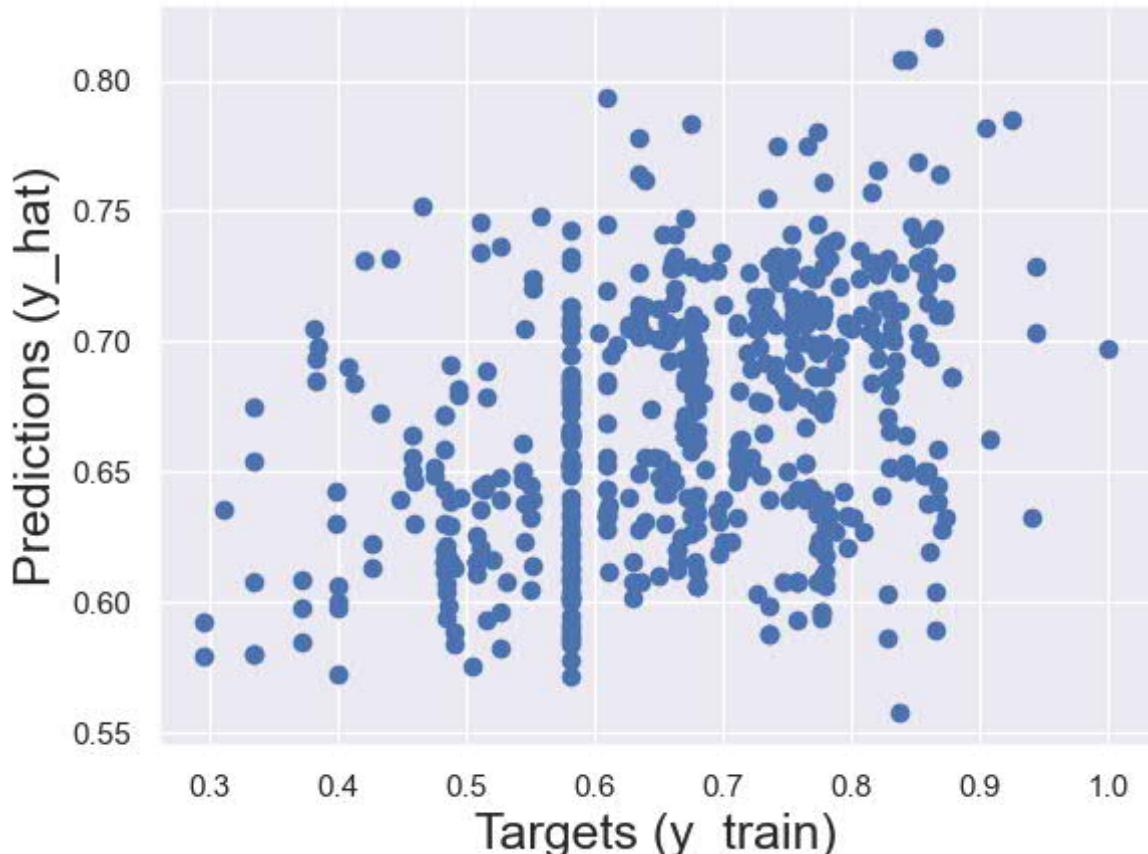
```
In [92]: print('Slope of the lines are', reg.coef_)
print('Intercept value is', reg.intercept_)
```

Slope of the lines are [-0.01417606 0.00394619 0.00642956 -0.00395302 0.0120543
0.00860214
0.00298324 0.00372991 0.00419102 0.00630229 0.00825903 -0.00943913
-0.0180204 -0.00302035 -0.04279511]
Intercept value is 0.6703170007564828

```
In [93]: # Let's check the outputs of the regression
# store them in y_hat
y_hat = reg.predict(x_train)
#y_hat
```

```
In [94]: # 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 [95]: # 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
```

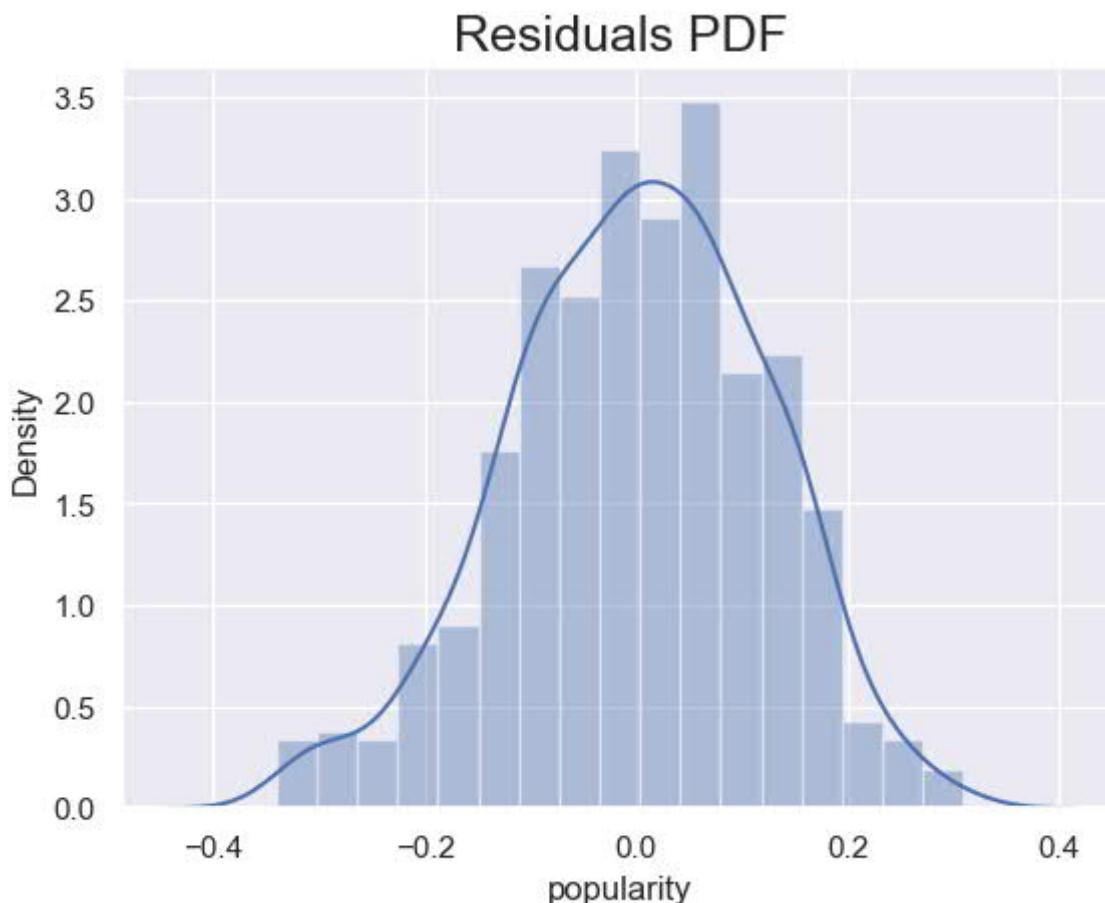
```

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

# plot normally distributed

```

Out[95]: Text(0.5, 1.0, 'Residuals PDF')



In [96]:

```

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error, r2_score

# Find the R-squared of the model
reg.score(x_train,y_train)

```

Out[96]: 0.1379492309675374

In []:

It is important to note that the linear model will not be able to handle the non-linear relationship between data and target since linear models assume the relationship between data and target to be linear.

Lets check several more options:

Polynomial

In [97]:

```

from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import PolynomialFeatures

```

```

polynomial_regression = make_pipeline(
    PolynomialFeatures(degree=3, include_bias=False),
    LinearRegression(),
)
polynomial_regression.fit(x_train, y_train)
target_predicted_poli = polynomial_regression.predict(x_train)
mse2 = mean_squared_error(y_train, target_predicted_poli)
r2_poli = r2_score(y_train, target_predicted_poli)
#print(polynomial_regression.score(x_train,y_train))==r2

#Mean square error (MSE) is the average of the square of the errors. The Larger the
#Error in this case means the difference between the observed values y1, y2, y3, ...
print(mse2)
print(r2_poli)

```

0.011379937673354367
0.3294444118863855

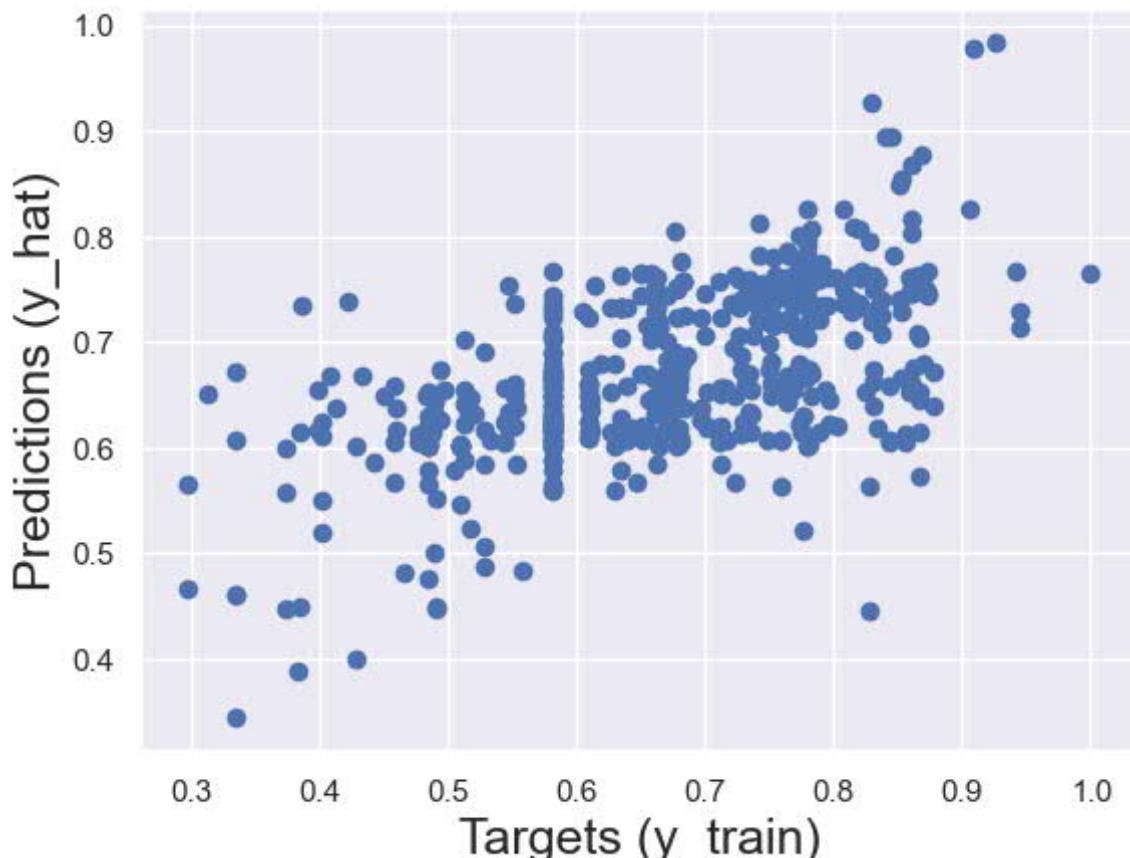
In [98]: # The simplest way to compare the targets (y_train) and the predictions (y_hat) is

```

plt.scatter(y_train, target_predicted_poli)

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

```



In []:

Decision tree

In [99]: `from sklearn.tree import DecisionTreeRegressor`

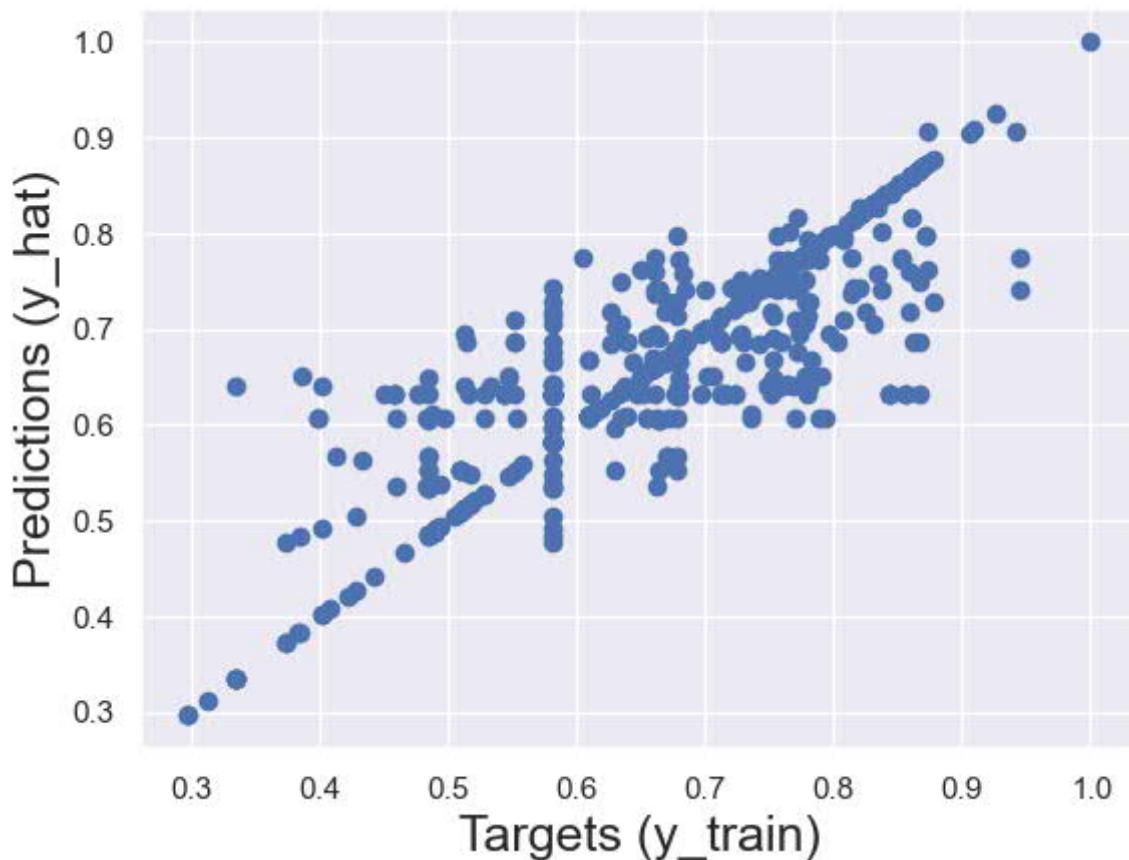
```
tree = DecisionTreeRegressor(max_depth=15).fit(x_train, y_train)
target_predicted = tree.predict(x_train)
mse3 = mean_squared_error(y_train, target_predicted)
r2_destree = r2_score(y_train, target_predicted)
print(mse3)
print(r2_destree)
```

```
0.004794460469959167
0.7174894667790126
```

In [100...]:

```
# The simplest way to compare the targets (y_train) and the predictions (y_hat) is
plt.scatter(y_train, target_predicted)

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



In []:

Support Vector Machines

The last possibility is to make a linear model more expressive is to use a "kernel". Instead of learning a weight per feature as we previously emphasized, a weight will be assigned to each sample. However, not all samples will be used. This is the base of the support vector machine algorithm.

support vector machines with non-linear kernel

rbf: $\exp(-\gamma |x-x'|^2)$, where γ is specified by parameter gamma, must be greater than 0.

In [105...]:

```
from sklearn.svm import SVR
```

```

svr = SVR(kernel='rbf', degree=5)
svr.fit(x_train, y_train)
target_predicted_svr = svr.predict(x_train)
mse_svr = mean_squared_error(y_train, target_predicted_svr)
r2_svr = r2_score(y_train, target_predicted_svr)
print(mse_svr)
print(r2_svr)

```

0.004794460469959167
0.7174894667790126

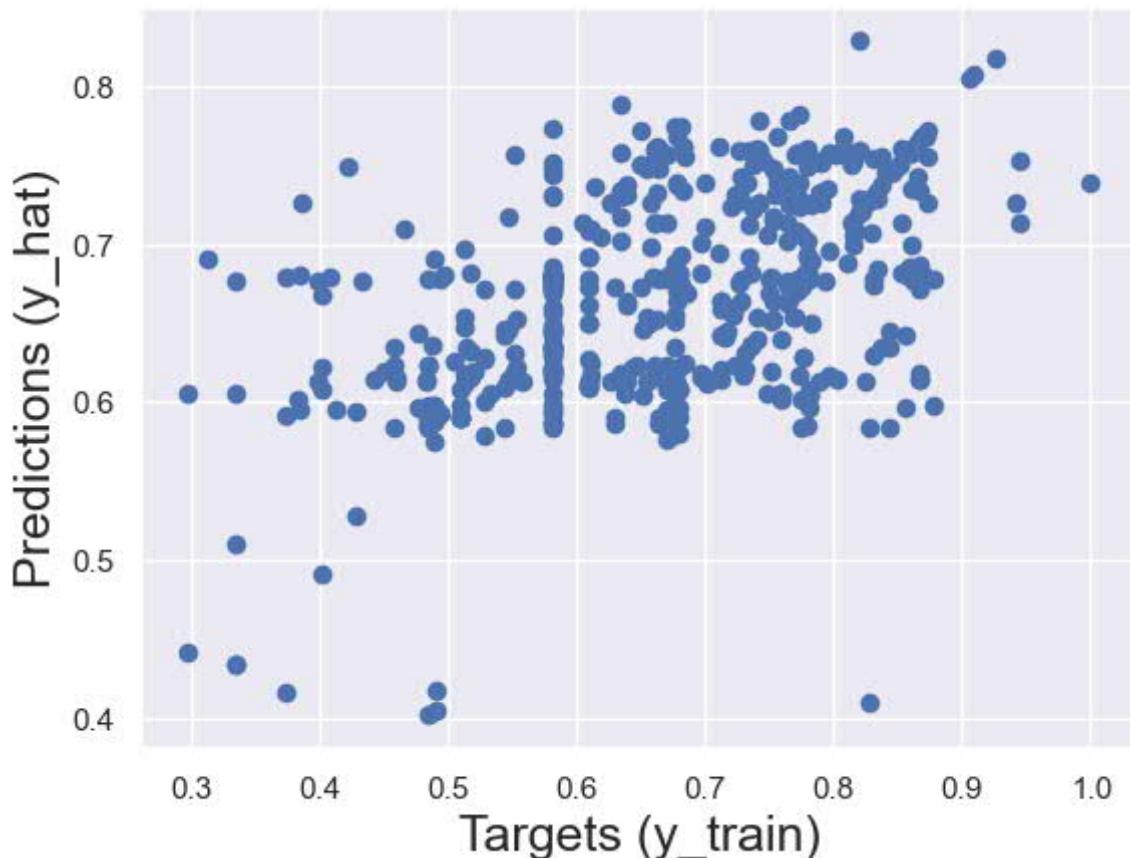
In [106...]:

```

# The simplest way to compare the targets (y_train) and the predictions (y_hat) is
plt.scatter(y_train, target_predicted_svr)

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

```



In []:

Bin continuous data into intervals.

In [107...]:

```

from sklearn.preprocessing import KBinsDiscretizer

binned_regression = make_pipeline(
    KBinsDiscretizer(n_bins=15), LinearRegression(),
)
binned_regression.fit(x_train, y_train)
target_predicted_kb = binned_regression.predict(x_train)
mse_kb = mean_squared_error(y_train, target_predicted_kb)
r2_kb = reg.score(x_train, y_train)

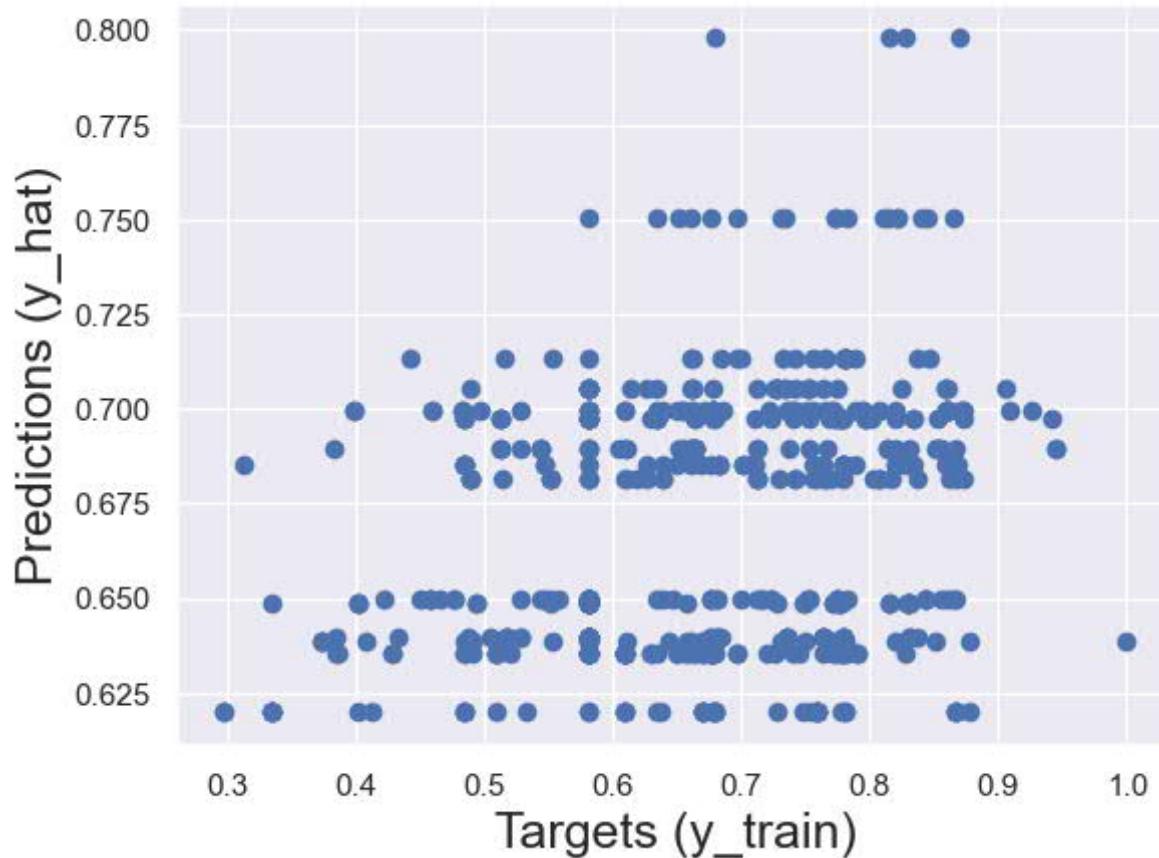
```

```
print(mse_kb)
print(r2_kb)
```

0.01576735140277855
0.13794923096753742

```
In [108]: # The simplest way to compare the targets (y_train) and the predictions (y_hat) is
plt.scatter(y_train, target_predicted_kb)

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



In []:

Finding the weights and bias

```
In [109]: # Obtain the bias (intercept) of the regression
reg.intercept_
```

Out[109]: 0.6703170007564828

```
In [110]: # Obtain the weights (coefficients) of the regression
reg.coef_
```

Note that they are barely interpretable if at all (because of dummies and standardization)

Out[110]: array([-0.01417606, 0.00394619, 0.00642956, -0.00395302, 0.0120543 ,
 0.00860214, 0.00298324, 0.00372991, 0.00419102, 0.00630229,
 0.00825903, -0.00943913, -0.0180204 , -0.00302035, -0.04279511])

```
In [111]: # 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[111]:

	Features	Weights
0	selling_price	-0.014176
1	color_Blue	0.003946
2	color_Green	0.006430
3	color_Grey	-0.003953
4	color_Multi	0.012054
5	color_Orange	0.008602
6	color_Pink	0.002983
7	color_Purple	0.003730
8	color_Red	0.004191
9	color_White	0.006302
10	color_Yellow	0.008259
11	category_Clothing	-0.009439
12	category_Shoes	-0.018020
13	gender_Men	-0.003020
14	gender_Women	-0.042795

In []:

Testing

In [112...]

```
# 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)
y_hat_poly= polynomial_regression.predict(x_test)
y_hat_tree = tree.predict(x_test)
y_hat_svr=svr.predict(x_test)
y_hat_kb=binned_regression.predict(x_test)
```

model evaluation

Lets check mse ans R2 on test data(not traned one)

```
mse = mean_squared_error(y_test, y_hat_test)
```

```
r2 = r2_score(y_test, y_hat_test)
```

In [113...]

```
#Linear Regression
mse1 = mean_squared_error(y_test, y_hat_test)
r2_1 = r2_score(y_test, y_hat_test)
```

```
#Polynomial
mse2 = mean_squared_error(y_test, y_hat_poly)
r2_2 = r2_score(y_test, y_hat_poly)

#Desicion Tree
mse3 = mean_squared_error(y_test, y_hat_tree)
r2_3 = r2_score(y_test, y_hat_tree)

#SVM
mse4 = mean_squared_error(y_test, y_hat_svr)
r2_4 = r2_score(y_test, y_hat_svr)

#KB
mse5 = mean_squared_error(y_test, y_hat_kb)
r2_5 = r2_score(y_test, y_hat_kb)
```

In [114...]:

#Linear Regression

```
print(mse1)
print(r2_1)
```

```
0.01689498891094623
0.06644062331836509
```

In [115...]:

#Polynomial

```
print(mse2)
print(r2_2)
```

```
9.235022496810081e+17
-5.102957978372556e+19
```

In [116...]:

#Desicion Tree

```
print(mse3)
print(r2_3)
```

```
0.02124440931230697
-0.1738934911473724
```

In [117...]:

#SVM

```
print(mse4)
print(r2_4)
```

```
0.01897066427983898
-0.04825410739681524
```

In [118...]:

#KB

```
print(mse5)
print(r2_5)
```

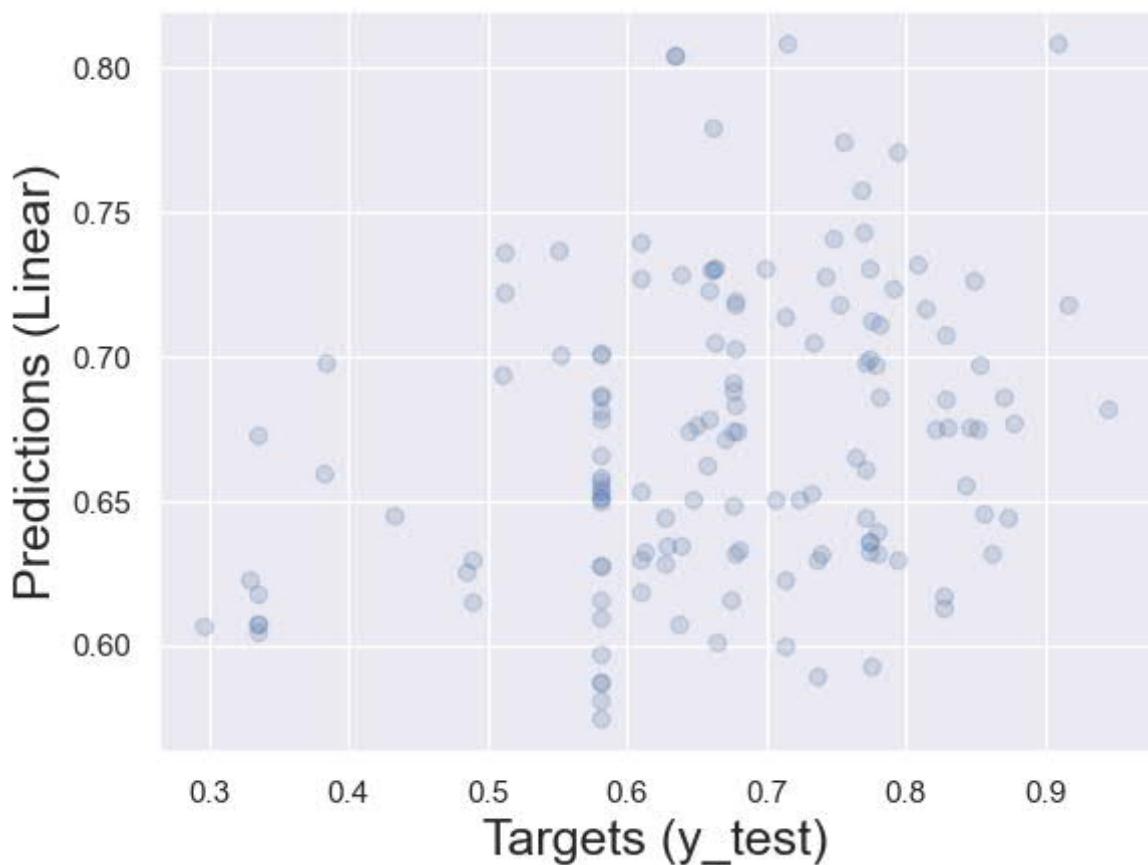
```
0.01620223106155557
0.10472005572742937
```

In [129...]:

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 (Linear)', size=18)

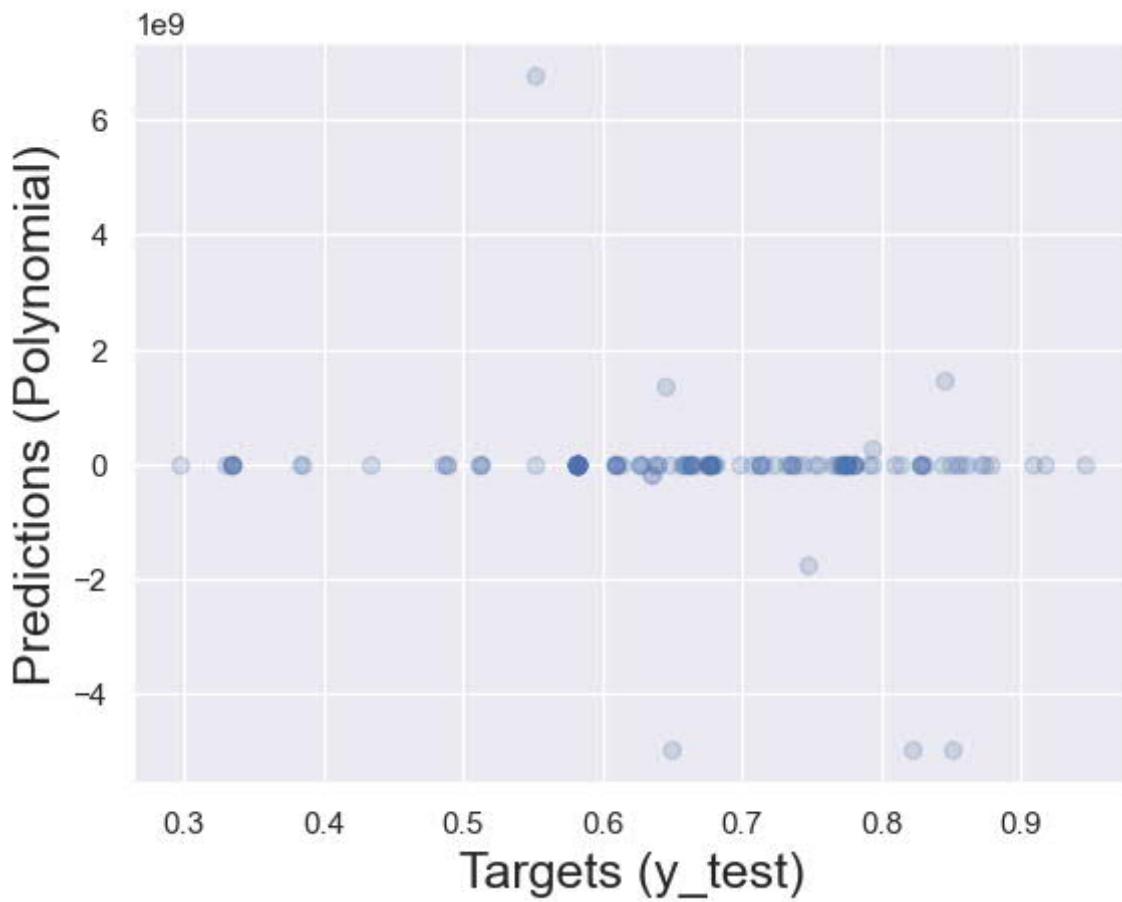
plt.show()
```



```
In [130]: # Create a scatter plot with the test targets and the test predictions
```

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

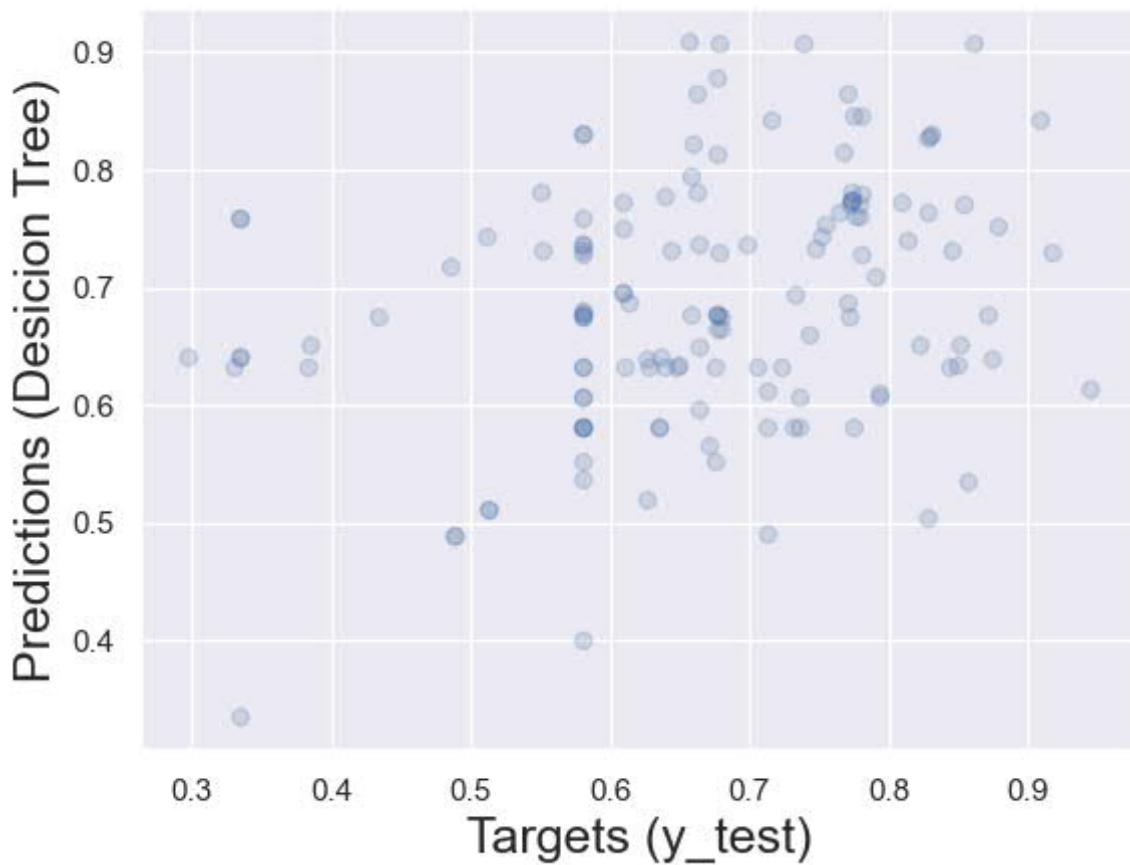
plt.show()
```



```
In [131]: # Create a scatter plot with the test targets and the test predictions
```

```
plt.scatter(y_test, y_hat_tree, alpha=0.2)
plt.xlabel('Targets (y_test)', size=18)
plt.ylabel('Predictions (Desicion Tree)', size=18)

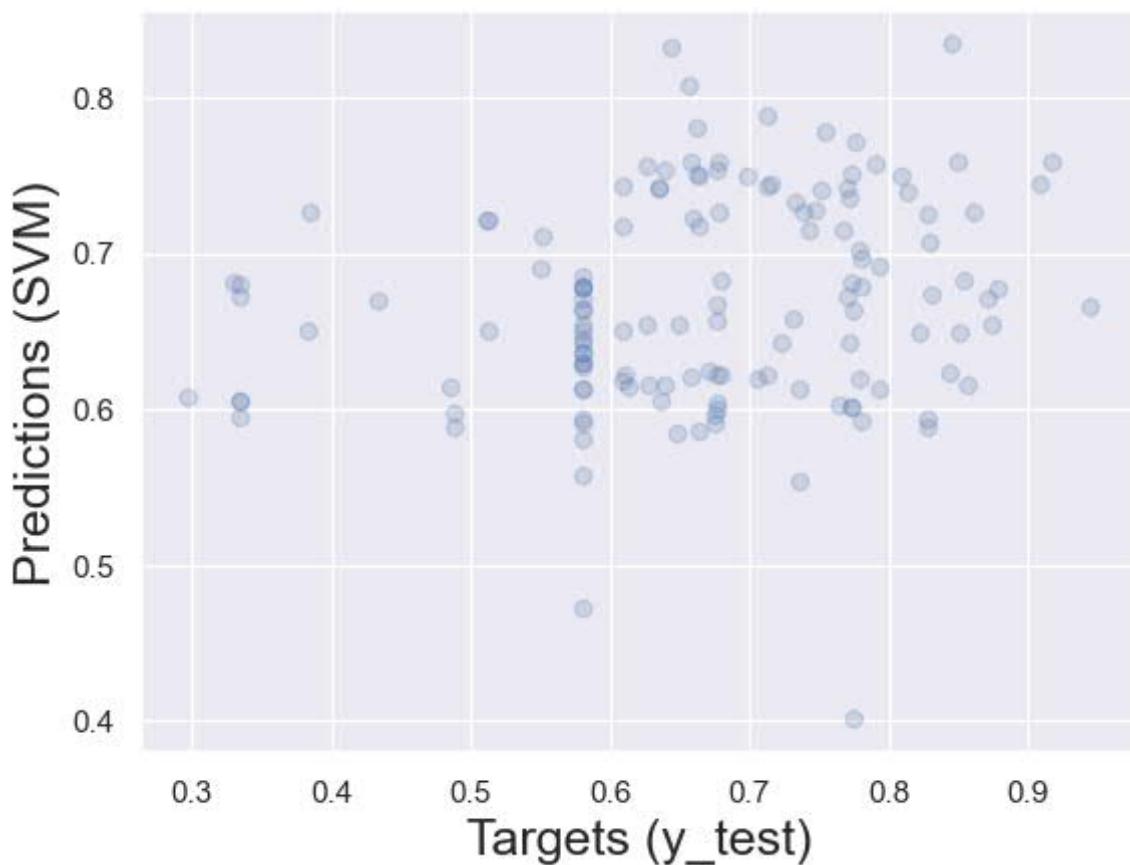
plt.show()
```



```
In [132]: # Create a scatter plot with the test targets and the test predictions
```

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

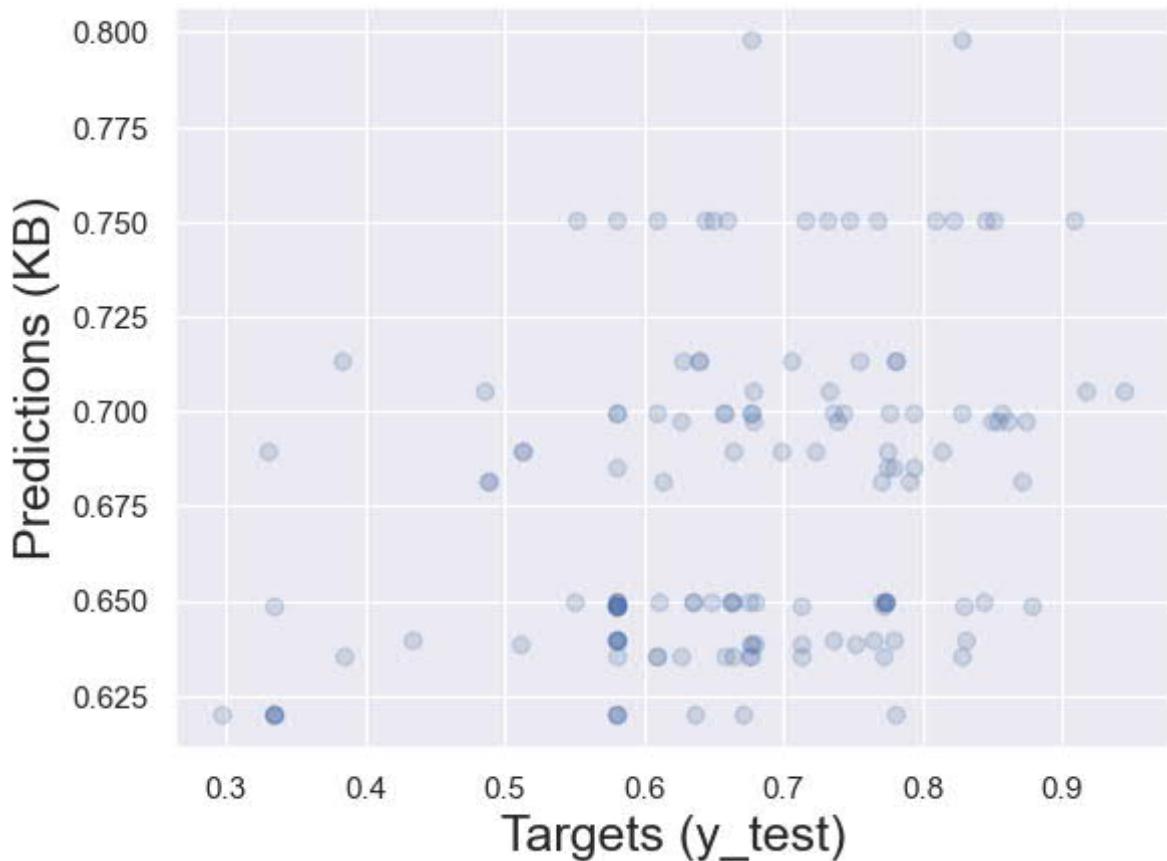
plt.show()
```



```
In [133]: # Create a scatter plot with the test targets and the test predictions
```

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

plt.show()
```



```
In [124]: # Finally, Let's manually check these predictions for Linear regression
```

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

	Prediction
0	0.726805
1	0.726908
2	0.651572
3	0.736271
4	0.627704

```
In [125]: # We can also include the test targets in that data frame (so we can manually compare)
y_test = y_test.reset_index(drop=True)
```

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

Out[125]:

	Prediction	Target
0	0.726805	0.849126
1	0.726908	0.609039
2	0.651572	0.580881
3	0.736271	0.511941
4	0.627704	0.580881
...
133	0.675788	0.830411
134	0.697573	0.778624
135	0.685902	0.870298
136	0.673989	0.679526
137	0.718179	0.677814

138 rows × 2 columns

In [126]:

```
# Additionally, we can calculate the difference between the targets and the predictions
# 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 errors
# this comparison makes a lot of sense
```

In [127]:

```
# 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[127]:

	Prediction	Target	Residual	Difference%
0	0.726805	0.849126	0.122321	14.405543
1	0.726908	0.609039	-0.117869	19.353225
2	0.651572	0.580881	-0.070691	12.169589
3	0.736271	0.511941	-0.224330	43.819414
4	0.627704	0.580881	-0.046823	8.060681
...
133	0.675788	0.830411	0.154623	18.620036
134	0.697573	0.778624	0.081050	10.409413
135	0.685902	0.870298	0.184396	21.187702
136	0.673989	0.679526	0.005536	0.814718
137	0.718179	0.677814	-0.040365	5.955187

138 rows × 4 columns

In [128]:

```
# Exploring the descriptives here gives us additional insights
df_pf.describe()
```

Out[128]:

	Prediction	Target	Residual	Difference%
count	138.000000	138.000000	138.000000	138.000000
mean	0.672174	0.668587	-0.003588	17.569618
std	0.051180	0.135017	0.130405	20.758914
min	0.574352	0.295955	-0.338326	0.068246
25%	0.631999	0.580881	-0.073678	4.601519
50%	0.672297	0.675209	-0.004455	12.327963
75%	0.706818	0.773584	0.093416	20.637497
max	0.808362	0.945058	0.263173	104.899888

In []: