Advanced features:

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import folium

import datetime

import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
#load the data
data = pd.read_csv("data_with_basic_features.csv")
data.drop("Unnamed: 0",axis=1,inplace=True)
```

In [3]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 113105 entries, 0 to 113104
Data columns (total 49 columns):
 #   Column                         Non-Null Count   Dtype  
---  ------                         --------------   -----  
 0   order_id                       113105 non-null  object 
 1   payment_sequential             113105 non-null  int64  
 2   payment_type                   113105 non-null  object 
 3   payment_installments           113105 non-null  int64  
 4   payment_value                  113105 non-null  float64
 5   customer_id                    113105 non-null  object 
 6   order_status                   113105 non-null  object 
 7   order_purchase_timestamp       113105 non-null  object 
 8   order_approved_at              113105 non-null  object 
 9   order_delivered_carrier_date   113105 non-null  object 
 10  order_delivered_customer_date  113105 non-null  object 
 11  order_estimated_delivery_date  113105 non-null  object 
 12  review_score                   113105 non-null  int64  
 13  customer_unique_id             113105 non-null  object 
 14  zip_code_prefix_customer       113105 non-null  int64  
 15  lat_customer                   113105 non-null  float64
 16  lng_customer                   113105 non-null  float64
 17  customer_city                  113105 non-null  object 
 18  customer_state                 113105 non-null  object 
 19  product_id                     113105 non-null  object 
 20  product_name_lenght            113105 non-null  float64
 21  product_description_lenght     113105 non-null  float64
 22  product_photos_qty             113105 non-null  float64
 23  product_weight_g               113105 non-null  float64
 24  product_length_cm              113105 non-null  float64
 25  product_height_cm              113105 non-null  float64
 26  product_width_cm               113105 non-null  float64
 27  order_item_id                  113105 non-null  int64  
 28  seller_id                      113105 non-null  object 
 29  shipping_limit_date            113105 non-null  object 
 30  price                          113105 non-null  float64
 31  freight_value                  113105 non-null  float64
 32  zip_code_prefix_seller         113105 non-null  int64  
 33  lat_seller                     113105 non-null  float64
 34  lng_seller                     113105 non-null  float64
 35  seller_city                    113105 non-null  object 
 36  seller_state                   113105 non-null  object 
 37  product_category_name          113105 non-null  object 
 38  estimated_time                 113105 non-null  float64
 39  actual_time                    113105 non-null  float64
 40  diff_actual_estimated          113105 non-null  float64
 41  diff_purchased_approved        113105 non-null  float64
 42  diff_purchased_courrier        113105 non-null  float64
 43  distance                       113105 non-null  float64
 44  speed                          113105 non-null  float64
 45  same_state                     113105 non-null  int64  
 46  same_city                      113105 non-null  int64  
 47  late_shipping                  113105 non-null  int64  
 48  high_freight                   113105 non-null  int64  
dtypes: float64(21), int64(10), object(18)
memory usage: 42.3+ MB
```

I created seller order_item_id share and customer order_id share features. using this features I want to create similarity between user and seller

This feature is inspired by the research paper [https://www.kdd.org/kdd2016/papers/files/adf0160-liuA.pdf (https://www.kdd.org/kdd2016/papers/files/adf0160-liuA.pdf)](https://www.kdd.org/kdd2016/papers/files/adf0160-liuA.pdf)

Let NMB be the number of purchases of the brand from the merchant, NM be the total number of purchases from the merchant, and NB be the number of purchases of the brand from all the merchants. Similarly, we define UMB as the number of users buying the brand from the merchant, UM the total number of buyers of the merchant, and UB the number of buyers of the brand from all the merchants. The following four features are then generated:

- 1) merchant’s market share on the brand = NMB /NB
- 2) merchant’s user share on the brand = UMB/UB

- 3) brand's market share within the merchant = NMB /NM
- 4) brand's user share within the merchant = UMB/UM

In [4]:
```
#groupby order item_id
order_seller = data.groupby("order_item_id")["seller_id"].value_counts().unstack()
order_seller.fillna(0,inplace=True)

total_order_id = np.sum(order_seller,axis=1).to_dict()
total_seller_id = np.sum(order_seller,axis=0).to_dict()
```

In [5]:
```
#creating feature
seller_share = []
bs_share = []
for i in range(len(data)):

     seller_share.append((order_seller.loc[(data["order_item_id"][i],data["seller_id"][i])]/total_order_id

     bs_share.append((order_seller.loc[(data["order_item_id"][i],data["seller_id"][i])]/total_seller_id[da


data["seller_share"] = seller_share
data["bs_share"] = bs_share
```

In [ ]:
```
```

In [6]:
```
user_order = data.groupby("order_item_id")["customer_unique_id"].value_counts().unstack()
user_order.fillna(0,inplace=True)

user_total = np.sum(user_order,axis=0).to_dict()
order_total = np.sum(user_order,axis=1).to_dict()
```

In [7]:
```
cust_share = []
bu_share = []
for i in range(len(data)):

    cust_share.append((user_order.loc[(data["order_item_id"][i],data["customer_unique_id"][i])]/order_tota

    bu_share.append((user_order.loc[(data["order_item_id"][i],data["customer_unique_id"][i])]/user_total[d


data["cust_share"] = cust_share
data["bu_share"] = bu_share
```

In [8]:
```
#calculating similarity
similarity = []
for i in range(len(data)):
     similarity.append((np.dot([data["seller_share"][i],data["bs_share"][i]] , [data["cust_share"][i],data

data["similarity"] = similarity
```

In [ ]:
```
```

Using product category name

In [9]:
```
order_seller = data.groupby("product_category_name")["seller_id"].value_counts().unstack()
order_seller.fillna(0,inplace=True)

total_order_id = np.sum(order_seller,axis=1).to_dict()
total_seller_id = np.sum(order_seller,axis=0).to_dict()


seller_share = []
bs_share = []
for i in range(len(data)):

     seller_share.append((order_seller.loc[(data["product_category_name"][i],data["seller_id"][i])]/total_

     bs_share.append((order_seller.loc[(data["product_category_name"][i],data["seller_id"][i])]/total_sell


data["seller_category_share"] = seller_share
data["cat_seller_share"] = bs_share




user_order = data.groupby("product_category_name")["customer_unique_id"].value_counts().unstack()
user_order.fillna(0,inplace=True)

user_total = np.sum(user_order,axis=0).to_dict()
order_total = np.sum(user_order,axis=1).to_dict()


cust_share = []
bu_share = []
for i in range(len(data)):

    cust_share.append((user_order.loc[(data["product_category_name"][i],data["customer_unique_id"][i])]/or

    bu_share.append((user_order.loc[(data["product_category_name"][i],data["customer_unique_id"][i])]/user


data["cust_category_share"] = cust_share
data["cat_cust_share"] = bu_share



#calculating similarity
similarity = []
for i in range(len(data)):
     similarity.append((np.dot([data["seller_category_share"][i],data["cat_seller_share"][i]] , [data["cus

data["similarity_using_cat"] = similarity
```

In [ ]:

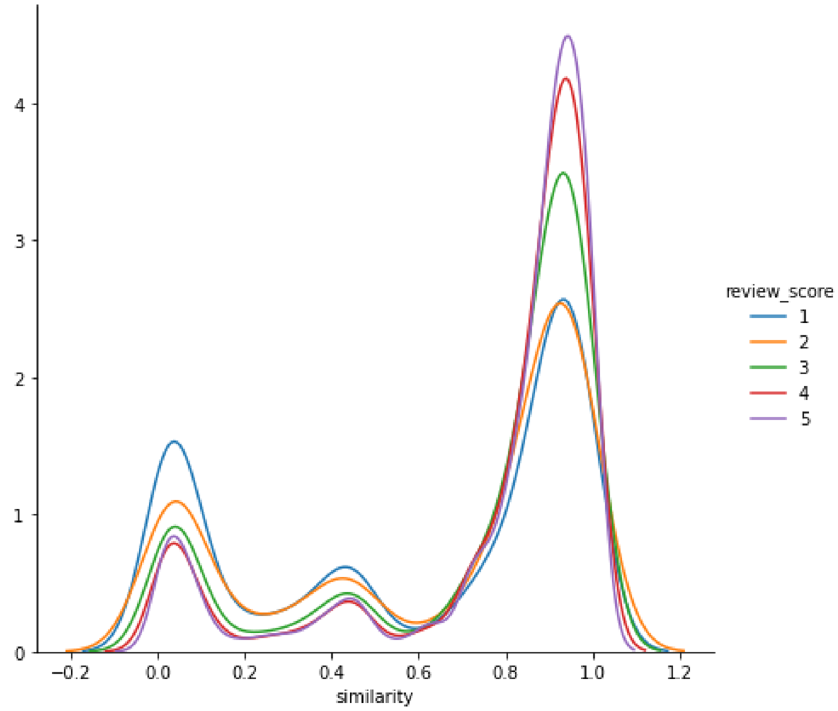
Analysis of new features:

In [11]:
```
data.corr()[["similarity_using_cat","similarity"]]
```

Out[11]:

	similarity_using_cat	similarity
payment_sequential	-0.005006	-0.015276
payment_installments	0.025927	-0.086158
payment_value	0.041945	-0.195735
review_score	0.001009	0.185706
zip_code_prefix_customer	0.016477	0.014063
lat_customer	0.012423	0.040204
lng_customer	0.005352	0.025916
product_name_lenght	-0.024414	0.015939
product_description_lenght	-0.012010	0.051073
product_photos_qty	-0.149814	0.104617
product_weight_g	0.029666	-0.007632
product_length_cm	0.053158	-0.030486
product_height_cm	-0.000675	-0.041785
product_width_cm	0.003577	-0.017591
order_item_id	0.028239	-0.633682
price	0.031639	0.128410
freight_value	0.033814	0.047077
zip_code_prefix_seller	0.060413	0.036971
lat_seller	0.070375	0.002797
lng_seller	-0.065087	0.031118
estimated_time	0.028796	-0.021506
actual_time	0.053080	0.024462
diff_actual_estimated	0.024288	0.041439
diff_purchased_approved	0.007281	-0.039264
diff_purchased_courrier	0.054406	-0.065888
distance	0.043977	0.042091
speed	0.012425	0.009733
same_state	-0.034635	-0.024945
same_city	-0.021512	-0.011947
late_shipping	0.040229	-0.040275
high_freight	-0.010508	-0.066841
seller_share	-0.003175	-0.162745
bs_share	-0.037874	0.915310
cust_share	0.004706	-0.070455
bu_share	-0.026346	0.937759
similarity	-0.034206	1.000000
seller_category_share	0.126259	-0.050253
cat_seller_share	0.967572	-0.064842
cust_category_share	-0.072668	-0.087874
cat_cust_share	0.257722	0.137571
similarity_using_cat	1.000000	-0.034206

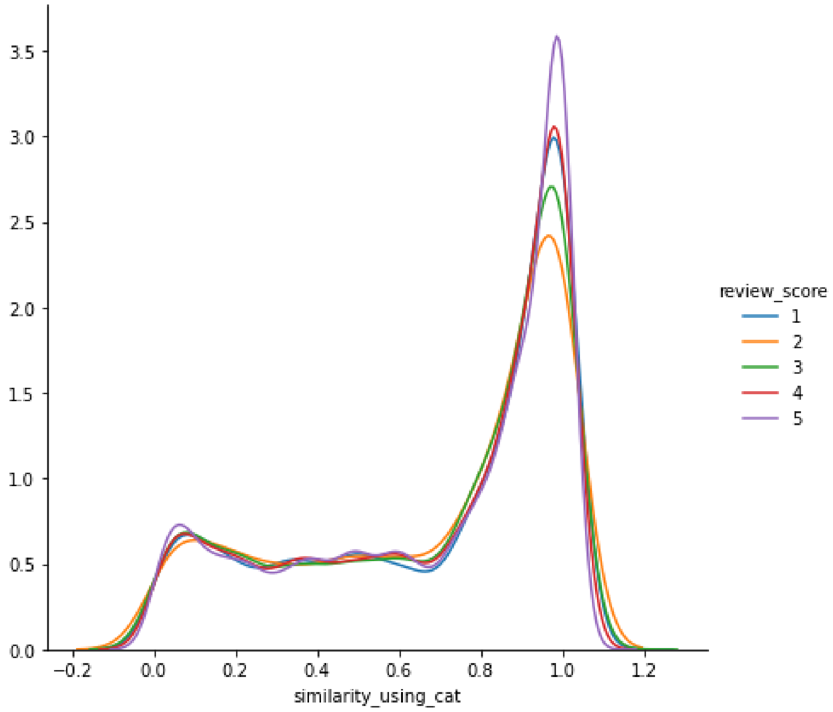
In [12]:
```
#similarity
sns.FacetGrid(data,hue="review_score",height=6)\
    .map(sns.kdeplot,"similarity")\
    .add_legend()
plt.show()
```



- This looks good. At low similarity values that is near zero, density of review_score 1 is high.
- At high similarity density of review_score 5 and 4 is high.

In [ ]:

In [13]:
```
#similarity_using_cat
sns.FacetGrid(data,hue="review_score",height=6)\
    .map(sns.kdeplot,"similarity_using_cat")\
    .add_legend()
plt.show()
```



- Although review_scores are not clearly separable, At higher similarity values, review_score 5 has high density and review_score 2 has less density.

In [ ]:

In [10]:
```
#saving all the created features with data.
data.to_csv("data_with_advanced_features.csv")
```

In [ ]:

In [ ]:

In [ ]: