Retail Rocket eCommerce

Online Shopping Behavior Modelling
& Recommendation System

Maggie Han

Apr. 14, 2023

CONTENTS

1. Retail Rocket eCommerce Introduction

- Developer of a predictive analytics platform
- Designed to help organizations identify the needs of web shop users by analyzing behavior.
- Leverage big data for determining the needs of visitors
- Provide organizations personalized offers to customers
- Aim to help retailers ultimately increase conversion rate, average order value, retention rate.

Modules

Data Warehouse

Joins customers data in a single storage and quickly creates segments for any marketing needs

AI Personalization Engine

Defines the best timing and communication channel to make a personalized offer to your client

Campaign Management System

Increase your revenue and customer engagement by cross-channel communications in emails, push and sms.

Customer Intelligence Platform

Opens up opportunities for your strategic customer database marketing to boost LTV and revenue

Solutions

Online Merchandising

Email Marketing

Web Push Notifications

2. Dataset and Objective

Original Dataset:

- Contains 3 csv files: Category tree, Events, Item property

	categoryid	parentid
0	1016	213.0
1	809	169.0
2	570	9.0
3	1691	885.0
4	536	1691.0

	timestamp	visitorid	event	itemid	transactionid
0	1433221332117	257597	view	355908	NaN
1	1433224214164	992329	view	248676	NaN
2	1433221999827	111016	view	318965	NaN
3	1433221955914	483717	view	253185	NaN
4	1433221337106	951259	view	367447	NaN

	timestamp	itemid	property	value
0	1435460400000	460429	categoryid	1338
1	1441508400000	206783	888	1116713 960601 n277.200
2	1439089200000	395014	400	n552.000 639502 n720.000 424566
3	1431226800000	59481	790	n15360.000
4	1431831600000	156781	917	828513

- The timestamp is in Unix Epoch format
- Property : category id , availability, while the rest are hashed for confidentiality purposes

Objective:

- Try to build a recommendation system using both collaborative filtering and content based recommender.
- Do thoroughly exploratory data analysis to familiar with the dataset, and try to transform it to a new dataset that works for a classification or regression model.
- To understand, validate, and interpret the model and provide the business insights.

3. Exploratory data analysis

- Unique items: 235061
- Unique visitors: 1407580
- Total visitors: 2756101
- Visitors that made purchase: 11719
- Visitors that only browse: 1395861

Rows: 2705278

	timestamp	visitorid	event	itemid	transactionid	categoryid	parentid
0	2015-06-01 22:02:12	257597	view	355908	NaN	1173	805.0
1	2015-06-01 13:42:45	981382	view	355908	NaN	1173	805.0
2	2015-06-08 21:07:35	979686	view	355908	NaN	1173	805.0
3	2015-06-15 08:31:50	479732	view	355908	NaN	1173	805.0
4	2015-06-14 16:51:34	397425	view	355908	NaN	1173	805.0

- Concat 3 tables:

Concat event and item table using the same itemid

	timestamp	itemid	property	value
0	1435460400000	460429	categoryid	1338
1	1441508400000	206783	888	1116713 960601 n277.200
2	1439089200000	395014	400	n552.000 639502 n720.000 424566
3	1431226800000	59481	790	n15360.000
4	1431831600000	156781	917	828513

	itemid	value
0	460429	1338
140	281245	1277
151	35575	1059
189	8313	1147
197	55102	47

	categoryid	parentid
0	1016	213.0
1	809	169.0
2	570	9.0
3	1691	885.0
4	536	1691.0

```
data.transactionid.isnull().value_counts()
```

```
True     2681915
False      23363
Name: transactionid, dtype: int64
```

```
data.event.value_counts()
```

```
view           2610352
addtocart        71563
transaction      23363
Name: event, dtype: int64
```

4. Recommendation Model

- Logics for item-item based recommendation system:
- Collaborative filtering
  - find the customer who made purchase
  - find their purchased items
  - define a function to return a list of same item in purchase
- Content based recommender
  - add similar products that in the same category or same parentid with top purchase frequency to the list

```
# for custommer who have purchase experience
# first - create a list of visitors who made a purchase
customer_purchased = events[events.transactionid.notnull()].visitorid.unique()

# create a list of purchased items, the values refer to the item_id of every customer
purchased_items = []

for customer in customer_purchased:

    purchased_items.append(list(events.loc[(events.visitorid == customer) & (events.transactionid.notnull())].itemid.values))
```

```
# purchased_items for each unique customer, minimal len of purchased[i] = 1
len(purchased_items)
```

11719

1395861

4. Recommendation Model

```python
# Write a function that would show items that were bought by the sc
def recommend_items(item_id, purchased_items):
    # put the arrays containing that item id in a new list
    recommendation_list =[]
    for x in purchased_items:
        if item_id in x:
            recommendation_list += x
            # print (x)

    #Then merge recommender list and remove the item id and duplica
    recommendation_list = list(set(recommendation_list) - set([item
    # print (set([item_id]))
    return recommendation_list
```

```python
recommend_items(150318, purchased_items)
```

```
[49521]
```

```python
# Step 1: Filter the data DataFrame to get the row corresponding to the given itemid
item_id = 150318
selected_row = freq_table[freq_table['itemid'] == item_id]

# Step 2: Extract the parentid and categoryid values from the selected row
selected_parentid = selected_row['parentid'].iloc[0]
# print(selected_parentid)
selected_categoryid = selected_row['categoryid'].iloc[0]
# print(selected_categoryid)

# Step 3: Filter the data DataFrame to get rows with the same parentid and categoryid
same_parentid_group = freq_table[freq_table['parentid'] == selected_parentid]
same_categoryid_group = freq_table[freq_table['categoryid'] == selected_categoryid]

# Step 4: Filter the rows to exclude the selected itemid
same_parentid_group = same_parentid_group[same_parentid_group['itemid'] != item_id]
same_categoryid_group = same_categoryid_group[same_categoryid_group['itemid'] != item_id]

list_1 = same_categoryid_group['itemid'].tolist()
print(list_1)

# Step 5: Sort the filtered DataFrame by the count of itemid occurrences in descending order
same_parentid_group = same_parentid_group.sort_values('frequency', ascending=False)
list_2 = same_parentid_group['itemid'].head().tolist()
print(list_2)

# Step 6: combine two list into one
top_5_itemids = list_1 + list_2

# Step 7: Print the list of top 10 itemids
print("The similar items:")
print(top_5_itemids)
```

```
[19972, 27216, 34250, 312414, 358598]
[111530, 459475, 403969, 238766, 230911]
The similar items:
[19972, 27216, 34250, 312414, 358598, 111530, 459475, 403969, 238766, 230911]
```

Final RS model

```python
# Write a function that would show items that were bought by the same customer
def recommend_items(item_id, purchased_items):
    recommendation_list =[]
    for x in purchased_items:
        if item_id in x:
            recommendation_list += x

    # Step 1: Filter the data DataFrame to get the row corresponding to the given itemid
    selected_row = freq_table[freq_table['itemid'] == item_id]

    # Step 2: Extract the parentid and categoryid values from the selected row
    selected_parentid = selected_row['parentid'].iloc[0]
    # print(selected_parentid)
    selected_categoryid = selected_row['categoryid'].iloc[0]
    # print(selected_categoryid)

    # Step 3: Filter the data DataFrame to get rows with the same parentid and categoryid
    same_parentid_group = freq_table[freq_table['parentid'] == selected_parentid]
    same_categoryid_group = freq_table[freq_table['categoryid'] == selected_categoryid]

    # Step 4: Filter the rows to exclude the selected itemid
    same_parentid_group = same_parentid_group[same_parentid_group['itemid'] != item_id]
    same_categoryid_group = same_categoryid_group[same_categoryid_group['itemid'] != item_id]

    list_1 = same_categoryid_group['itemid'].tolist()
    # print(list_1)

    # Step 5: Sort the filtered DataFrame by the count of itemid occurrences in descending order
    same_parentid_group = same_parentid_group.sort_values('frequency', ascending=False)
    list_2 = same_parentid_group['itemid'].head().tolist()
    # print(list_2)

    # Step 6: Combine two list into one
    top_5_itemids = list_1 + list_2

    # Step 7: Print the list of top 10 itemids
    # print("The similar items:")
    # print(top_5_itemids)

    #Then merge recommender list and remove the item id
    recommendation_list = list(set(recommendation_list) - set([item_id])) + top_5_itemids
    recommendation_list = list(set(recommendation_list))

    return recommendation_list
```

```python
recommend_items(150318, purchased_items)
```

```
[403969,
 19972,
 358598,
 34250,
 111530,
 238766,
 27216,
 49521,
 459475,
 312414,
 230911]
```

- Build a RS by searching for matches, instead of using similarity.
- Initially create a pivot table for customid as row, itemid as column, and event as the content.

- Need to think about how to build similarity after filtering that event = ‘transaction’
- How to evaluate the RS model without similarity score?

5. Regression Model

- Timestamp:

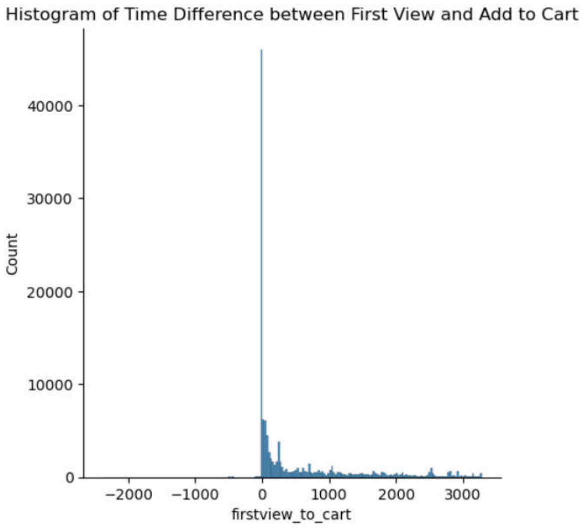
```
# obtain vistor id,item id, and date time of 'tranaction'
item_tra=data[['visitorid','itemid','timestamp']][data['event']=='transaction']
# obtain vistor id,item id, and date time of 'add to cart'
item_atc=data[['visitorid','itemid','timestamp']][data['event']=='addtocart']
# obtain vistor id,item id, and date time of 'view'
item_viw=data[['visitorid','itemid','timestamp']][data['event']=='view']
```

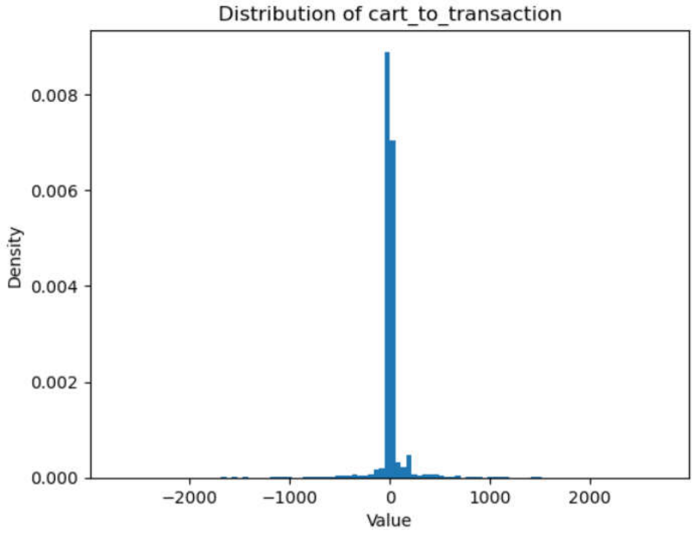
```
# create a dataframe of visitor, itemid found in all three events
time_df=item_tra.merge(item_atc, how='inner', on=['visitorid','itemid'], suffixes=[' (transaction)', ' (add_to_cart)'])
time_df=time_df.merge(item_viw, how='inner', on=['visitorid','itemid'])
time_df=time_df.rename(columns={'timestamp':'timestamp (view)'})
time_df.head()
```

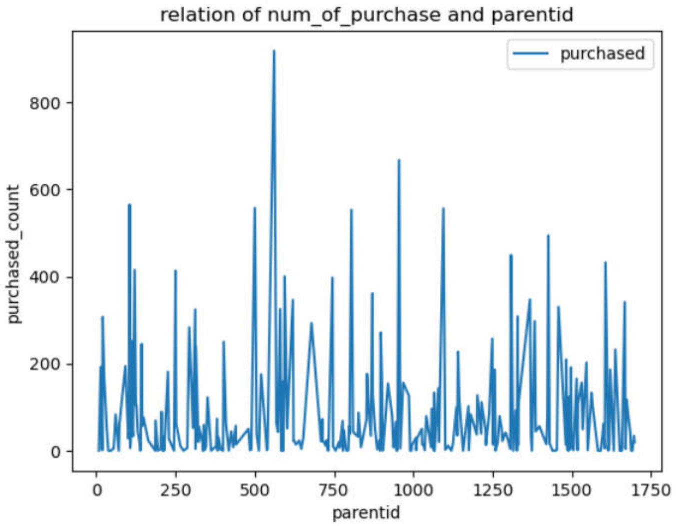
```
# calculate the time differences
time_df['cart_to_transaction'] = (time_df['timestamp (transaction)'] - time_df['timestamp (add_to_cart)']).apply(lambda x: x.tota
time_df['first_view'] = time_df.groupby('itemid')['timestamp (view)'].transform('min')
time_df['firstview_to_cart'] = (time_df['timestamp (add_to_cart)'] - time_df['first_view']).apply(lambda x: x.total_seconds()/360

time_df.head()
```

	visitorid	itemid	timestamp (transaction)	timestamp (add_to_cart)	timestamp (view)	cart_to_transaction	first_view	firstview_to_cart
0	76757	280375	2015-06-10 11:25:41	2015-06-10 11:23:27	2015-06-10 11:23:08	0.037222	2015-05-06 10:00:52	841.376389
1	76757	280375	2015-06-10 11:25:41	2015-06-10 11:23:27	2015-05-06 10:00:52	0.037222	2015-05-06 10:00:52	841.376389
2	76757	280375	2015-06-10 11:25:41	2015-06-10 11:23:27	2015-05-12 10:29:00	0.037222	2015-05-06 10:00:52	841.376389
3	76757	280375	2015-06-10 11:25:41	2015-06-10 11:23:27	2015-05-28 10:50:57	0.037222	2015-05-06 10:00:52	841.376389
4	76757	280375	2015-06-10 11:25:41	2015-05-28 10:51:55	2015-06-10 11:23:08	312.562778	2015-05-06 10:00:52	528.850833

5.1 EDA of the original dataset







Original Dataset:

	timestamp	visitorid	event	itemid	transactionid	categoryid	parentid
0	2015-06-01 22:02:12	257597	view	355908	NaN	1173	805.0
1	2015-06-01 13:42:45	981382	view	355908	NaN	1173	805.0
2	2015-06-08 21:07:35	979686	view	355908	NaN	1173	805.0
3	2015-06-15 08:31:50	479732	view	355908	NaN	1173	805.0
4	2015-06-14 16:51:34	397425	view	355908	NaN	1173	805.0

5.2 Build new dataset

- Goal: predict purchase or not, purchase ability

```python
def create_dataframe(visitor_list):
    df_array = []
    for index in visitor_list:
        # create the base dataframe of each visitor that prepare for following data extraction
        v_df = data[data.visitorid == index]
        m_df = time_df[time_df.visitorid == index]

        temp = []
        # add visitor id
        temp.append(index)

        # num_items_viewed
        temp.append(v_df[v_df.event == 'view'].itemid.unique().size)

        #Add the total number of views regardless of product type
        temp.append(v_df[v_df.event == 'view'].event.count())

        #Add timediff of firstview_to_cart
        temp.append(m_df['firstview_to_cart'].mean())

        #Add timediff of cart_to_transaction
        temp.append(m_df['cart_to_transaction'].mean())

        #Add the total number of purchases
        number_of_items_bought = v_df[v_df.event == 'transaction'].event.count()
        temp.append(number_of_items_bought)

        #Then create binery 0 or 1 for purchased
        if(number_of_items_bought == 0):
            temp.append(0)
        else:
            temp.append(1)

        df_array.append(temp)

    return pd.DataFrame(df_array, columns=['visitorid', 'num_items_viewed', 'view_count', 'firstview_to_cart', 'cart_to_transacti
```

5.2 Build new dataset

```
buying_visitors_df = create_dataframe(customer_purchased)
buying_visitors_df.head()
```

	visitorid	num_items_viewed	view_count	firstview_to_cart	cart_to_transaction	bought_count	purchased
0	599528	2	15	0.012500	0.093056	1	1
1	121688	13	16	-0.145082	0.927173	12	1
2	552148	1	1	0.036667	0.009722	1	1
3	102019	2	6	0.024167	0.259722	2	1
4	189384	7	25	479.378596	0.051974	2	1

```
# This table contains data for all the customer that make a payment, need to concat view data
buying_visitors_df.shape
```

(11719, 7)

```
#Let's shuffle the viewing visitors list for randomness
import random
random.shuffle(customer_browsed)
```

```
# get 23438 samples from the viewing visitors list so that there is a 0.33 split for training
# if select all customer, imbalanced data can be addressed by randomoversampler etc. but the
viewing_visitors_df = create_dataframe(customer_browsed[0:23438])
```

```
data_ml = pd.concat([buying_visitors_df, viewing_visitors_df], ignore_index=True)
```

```
# shuffle main_df first
data_ml = data_ml.sample(frac=1)
```

column item	means
visitorid	unique visitorid
num_items _viewed	How many items one customer viewed
View_count	How many times one customer view online regardless items
firstview_to_cart	Timediff between firstview and addtocart, mean of all the items
cart_to_transaction	Timediff between addtocart and transaction, mean of all the items
Bought_count	Total number of purchase
purchased	0 or 1 (binery)

5.3 Dataset for modelling

```
data_ml = pd.read_csv('data_ml.csv')
data_ml.head()
```

	visitorid	num_items_viewed	view_count	firstview_to_cart	cart_to_transaction	bought_count	purchased
0	299645	1	1	NaN	NaN	0	0
1	901655	1	2	NaN	NaN	0	0
2	964726	4	12	118.104722	0.0425	1	1
3	368247	1	1	NaN	NaN	0	0
4	484387	1	1	NaN	NaN	0	0

view_rate = num_items_viewed / view_count

- total viewed items number / total view number
- lower view_rate refer to that the customer focus on a specific item

purchase_rate = bought_count / view_count

- the total purchase items number / total view number
- The higher purchase_rate, the higher purchase ability

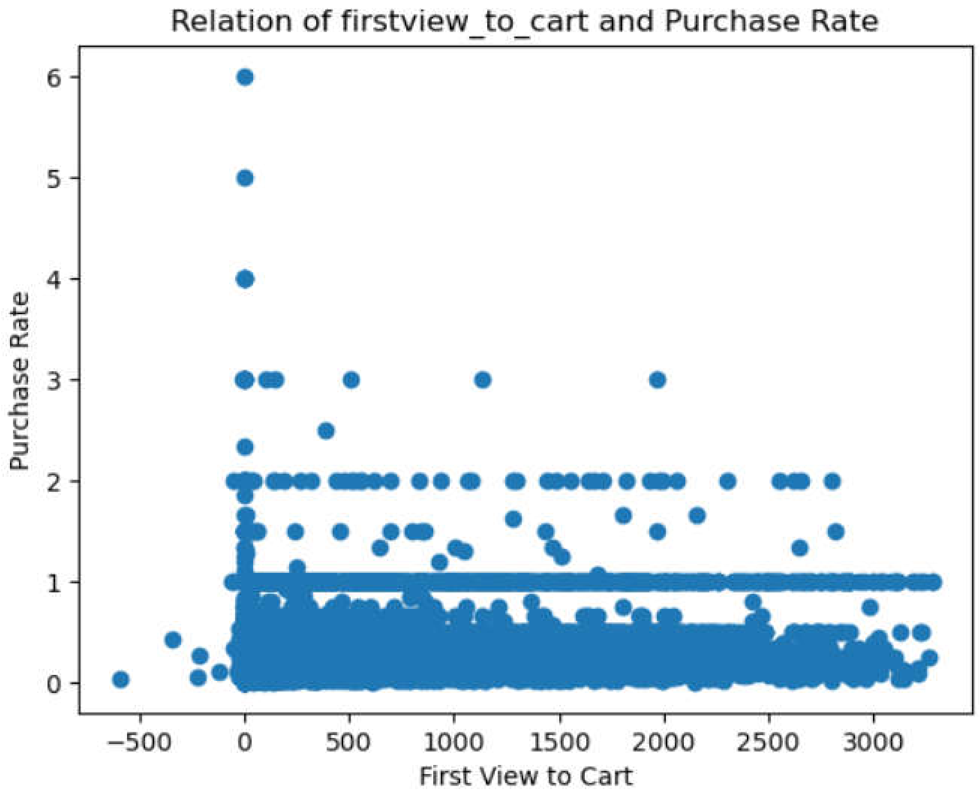
5.3 Dataset for modelling

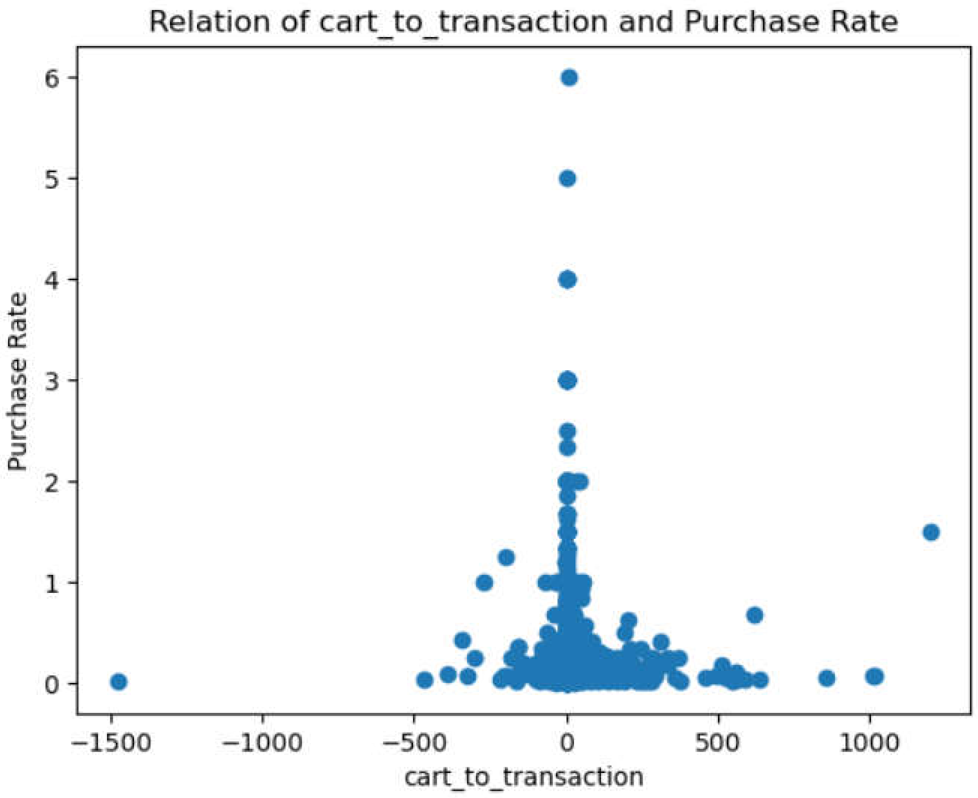
	num_items_viewed	view_count	firstview_to_cart	cart_to_transaction	bought_count	purchased	view_rate	purchase_rate
num_items_viewed	1.000000	0.990989	0.009150	-0.003294	0.856639	0.096755	-0.048262	-0.017260
view_count	0.990989	1.000000	0.008775	0.006017	0.852401	0.108997	-0.082439	-0.018717
firstview_to_cart	0.009150	0.008775	1.000000	0.003346	0.016086	nan	0.012842	0.018534
cart_to_transaction	-0.003294	0.006017	0.003346	1.000000	-0.008059	nan	-0.057414	-0.047450
bought_count	0.856639	0.852401	0.016086	-0.008059	1.000000	0.178362	-0.085311	0.087358
purchased	0.096755	0.108997	nan	nan	0.178362	1.000000	-0.463925	0.660328
view_rate	-0.048262	-0.082439	0.012842	-0.057414	-0.085311	-0.463925	1.000000	-0.043895
purchase_rate	-0.017260	-0.018717	0.018534	-0.047450	0.087358	0.660328	-0.043895	1.000000

For label selection, the columns need to be dropped as follow:

purchased	Purchase_rate
View_count	View_count
Firstview_to_cart (no relation to label)	Purchased
Cart_to_transaction(no relation to label)	
Purchase rate	

5.4 EDA of modelling dataset





How to address the missing value?
Current strategy: Using SimpleImputer to fill a constant value

5.5 Baseline Model

```python
# define features
X = df.drop('purchase_rate', axis=1)
X.head()
```

	num_items_viewed	firstview_to_cart	cart_to_transaction	bought_count	view_rate
0	1	NaN	NaN	0	1.000000
1	1	NaN	NaN	0	0.500000
2	4	118.104722	0.042500	1	0.333333
5	2	-0.093889	0.137222	1	1.000000
6	5	569.576914	0.150772	5	0.625000

```python
# define label
y=df['purchase_rate']
```

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

```python
fvc_pipeline = Pipeline([
    ('Impute_fvc', SimpleImputer(missing_values=np.nan, strategy='constant', fill_value=-1000))
])

cta_pipeline = Pipeline([
    ('Impute_cta', SimpleImputer(missing_values=np.nan, strategy='constant', fill_value=2000))
])

preprocessor = ColumnTransformer(
    transformers=[
        ('fvc_pipe', fvc_pipeline, ['firstview_to_cart']),
        ('cta_pipe', cta_pipeline, ['cart_to_transaction']),

    ], remainder='passthrough'
)

pipe = Pipeline(steps=[('preprocessor', preprocessor),
                       ('scaler', StandardScaler()),
                       ('lr', LinearRegression())])

r2 = cross_val_score(pipe, X, y, cv=5, scoring='r2')
mean_r2 = np.mean(r2)

print(mean_r2)
```

```
0.36412359612556333
```

Baseline model: LinearRegression
Score: r2

MSE is the average squared difference between the predicted values and the actual values.

R2 is a measure of how well the regression model explains the variability in the data, range from 0 to 1.

- 1 indicate that a larger proportion of the variability in the dependent variable is explained by the model,
- 0 indicate that the model explains little of the variability in the data.

5.6 Model selection

Model	R2 score	Performance
Linear Regression	0.36412359612556333	
Polynomial Regression (degree=2)	0.43741591439839744	
RandomForestRegressor	0.9905020649195226	
XGBoostRegressor	0.9936030937168236	
XGBRFRegressor	0.972974771905557	
Neural Network	0.6907307768211305	
Neural Network with hyperparameter tuning	0.8334061193263528	
Stacking of LR, RF, XGBoost	0.9906049860878501	

Non-linearity relationship of features and label:

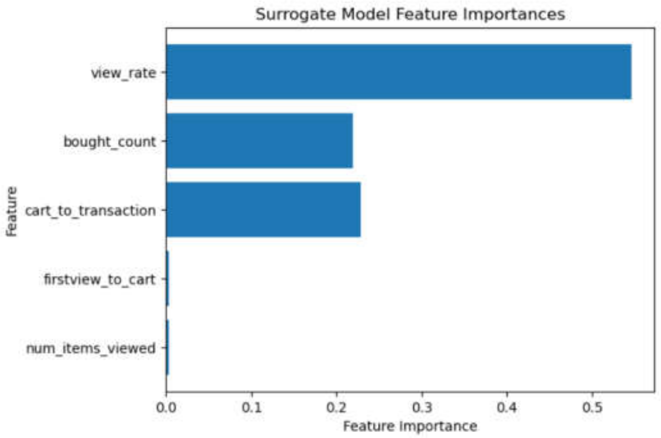
- Linear regression assumes a linear relationship between the dependent and independent variables. If the true relationship between the variables is non-linear, then linear regression may not be able to capture the underlying patterns accurately.
- RandomForest and XGBoost are capable of capturing non-linear relationships between variables.

5.7 Surrogate and Permutation

Surrogate:

Black box model:
- MLPRegressor

Surrogate model:
- Decision Tree



- The feature importance in a surrogate model is typically determined based on the number of times a feature is used for splitting in the surrogate decision tree.
- The more frequently a feature is used for splitting, the higher its importance is considered to be.
- The order of feature importance in the surrogate model may depend on the specific decision tree that was used to create the surrogate, and it may not necessarily reflect the true importance of features in the original complex model.

Permutation:

Weight	Feature
27.1245 ± 0.2626	cart_to_transaction
25.8921 ± 2.2240	bought_count
0.8136 ± 0.0388	view_rate
0.0280 ± 0.0036	firstview_to_cart
0.0061 ± 0.0011	num_items_viewed

- The permutation importance is typically calculated as the decrease in model performance after permuting a feature, and it is ranked based on the magnitude of this decrease.
- The order of feature importance in Permutation Importance is determined based on the calculated decrease in model performance for each feature, and it may reflect the true importance of features.

6. Business Insights and Feature Work

Business Insights

- The feature importance from Permutation shows that the cart_to_transaction and bought_count are more important to predict a customer who has high possibility buying a product online.
- In other words, the behavior of firstview_to_cart and num_items_viewed is implict while cart_to_transaction and bought_count are explict features.
- For business purpose, maintain their regular customer or return customer would be important to keep even increase the order value.

Feature Work

- Improve the recommender system using similarity.
- Check other imputing method for missing values in firstview_to_cart and cart_to_transaction.
- Play with neural network neuron number, layers and hyperparameters to see if it can achieve better results for this small size dataset.

The end.

Thanks for attention.