SI 671 Project: E-commerce User Behavior Analysis and Modeling

Xinyu Zhang (xyuzhang, 42192372)

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
%matplotlib inline
from matplotlib.patches import Shadow
from matplotlib.lines import Line2D
```

In [2]:
```
import warnings
warnings.filterwarnings("ignore")
```

1. Data Preprocessing

In [3]:
```
from sklearn.model_selection import train_test_split
```

1.1 Data Sampling

In [4]:
```
column_names = ["User_ID", "Product_ID", "Category_ID", "Behavior", "Timestamp"]
df_ori = pd.read_csv("UserBehavior.csv", names=column_names)
```

In [5]:
```
print("There are in total " + str(len(df_ori)) + " records in the original dataset.")
print("There are in total " + str(df_ori['User_ID'].nunique()) + " unique users in the original dataset.")
print("There are in total " + str(df_ori['Category_ID'].nunique()) + " categories in the original dataset.")
print("There are in total " + str(df_ori['Product_ID'].nunique()) + " products in the original dataset.")
```

```
There are in total 100150807 records in the original dataset.
There are in total 987994 unique users in the original dataset.
There are in total 9439 categories in the original dataset.
There are in total 4162024 products in the original dataset.
```

In [6]:
```
# sample users
users = df_ori['User_ID'].unique()
_, users_sampled = train_test_split(users, test_size=0.05, random_state=42)
df_5M = df_ori[df_ori['User_ID'].isin(users_sampled)]
df_5M.head()
```

Out[6]:

	User_ID	Product_ID	Category_ID	Behavior	Timestamp
1747	1000060	2178355	4962280	pv	1511642462
1748	1000060	857323	1320293	pv	1511822512
1749	1000060	4983347	4069500	pv	1511822555
1750	1000060	2952829	4145813	cart	1511823929
1751	1000060	2290719	4756105	pv	1511823982

In [7]:
```
print("There are in total " + str(len(df_5M)) + " records in the sampled dataset.")
print("There are in total " + str(df_5M['User_ID'].nunique()) + " unique users in the sampled dataset.")
print("There are in total " + str(df_5M['Category_ID'].nunique()) + " categories in the sampled dataset.")
print("There are in total " + str(df_5M['Product_ID'].nunique()) + " products in the sampled dataset.")
```

```
There are in total 5015266 records in the sampled dataset.
There are in total 49400 unique users in the sampled dataset.
There are in total 7484 categories in the sampled dataset.
There are in total 1106939 products in the sampled dataset.
```

In [8]:
```
df_5M.to_csv("UserBehavior_5M.csv")
```

In [9]:
```
# after the first time initialization, to reproduce the result, use the code
# df_5M = pd.read_csv("UserBehavior_5M.csv")
```

1.2 Data Processing

- Data Cleaning

In [10]:
```
# check if there are null values
df_5M.isnull().sum()
```

Out[10]:
```
User_ID        0
Product_ID     0
Category_ID    0
Behavior       0
Timestamp      0
dtype: int64
```

In [11]:
```
# check if there are duplicated records
df_5M.duplicated().sum()
```

Out[11]:
```
3
```

In [12]:
```
df_5M.drop_duplicates(inplace=True)
df_5M.duplicated().sum()
```

Out[12]:
```
0
```

In [13]:
```
# rename the "Behavior" column
behavior_mapping = {'pv': 'PageView', 'buy': 'Buy', 'cart': 'AddToCart', 'fav': 'Favorite'}
df_5M['Behavior'] = df_5M['Behavior'].replace(behavior_mapping)
df_5M.head()
```

Out[13]:

	User_ID	Product_ID	Category_ID	Behavior	Timestamp
1747	1000060	2178355	4962280	PageView	1511642462
1748	1000060	857323	1320293	PageView	1511822512
1749	1000060	4983347	4069500	PageView	1511822555
1750	1000060	2952829	4145813	AddToCart	1511823929
1751	1000060	2290719	4756105	PageView	1511823982

- Format Transformation

In [14]:
```
# transform `timestamp` to datetime
df_5M['Datetime'] = pd.to_datetime(df_5M['Timestamp'], unit='s')
df_5M.head()
```

Out[14]:

	User_ID	Product_ID	Category_ID	Behavior	Timestamp	Datetime
1747	1000060	2178355	4962280	PageView	1511642462	2017-11-25 20:41:02
1748	1000060	857323	1320293	PageView	1511822512	2017-11-27 22:41:52
1749	1000060	4983347	4069500	PageView	1511822555	2017-11-27 22:42:35
1750	1000060	2952829	4145813	AddToCart	1511823929	2017-11-27 23:05:29
1751	1000060	2290719	4756105	PageView	1511823982	2017-11-27 23:06:22

Filter out the valid year range

In [15]:
```
df_5M['Datetime'].dt.year.value_counts()
```

Out[15]:
```
Datetime
2017    5015227
2019         11
1970         10
2018          6
2001          6
2025          2
1929          1
Name: count, dtype: int64
```

In [16]:
```
df_5M = df_5M[df_5M['Datetime'].dt.year == 2017]
df_5M['Datetime'].dt.year.unique()
```

Out[16]:
```
array([2017], dtype=int32)
```

In [17]:
```
df_5M['Datetime'].dt.date.value_counts()[:15]
```

Out[17]:
```
Datetime
2017-12-02    700591
2017-12-03    600014
2017-12-01    561311
2017-11-25    531806
2017-11-26    529949
2017-11-30    526781
2017-11-29    509690
2017-11-27    502362
2017-11-28    490128
2017-11-24     61618
2017-10-02       336
2017-11-23       281
2017-11-22        83
2017-11-21        39
2017-12-05        32
Name: count, dtype: int64
```

In [18]:
```
df_5M = df_5M[(df_5M['Datetime'] >= '2017-11-25 00:00:00') & (df_5M['Datetime'] <= '2017-12-03 23:59:59')]
```

In [19]:
```
df_5M['Day_of_Week'] = df_5M['Datetime'].dt.day_name()
df_5M['Hour'] = df_5M['Datetime'].dt.hour
df_5M['Date'] = df_5M['Datetime'].dt.date
df_5M.head()
```

Out[19]:

	User_ID	Product_ID	Category_ID	Behavior	Timestamp	Datetime	Day_of_Week	Hour	Date
1747	1000060	2178355	4962280	PageView	1511642462	2017-11-25 20:41:02	Saturday	20	2017-11-25
1748	1000060	857323	1320293	PageView	1511822512	2017-11-27 22:41:52	Monday	22	2017-11-27
1749	1000060	4983347	4069500	PageView	1511822555	2017-11-27 22:42:35	Monday	22	2017-11-27
1750	1000060	2952829	4145813	AddToCart	1511823929	2017-11-27 23:05:29	Monday	23	2017-11-27
1751	1000060	2290719	4756105	PageView	1511823982	2017-11-27 23:06:22	Monday	23	2017-11-27

In [20]:
```
df_5M['User_ID'].nunique()
```

Out[20]:
```
49400
```

In [21]:
```
len(df_5M)
```

Out[21]:
```
4952632
```

In [22]:
```
# df_5M.to_csv("UserBehavior_5M_cleaned.csv")
```

2. Exploratory Data Analysis

In [79]:
```
# The top 5 most popular categories
df_5M['Category_ID'].value_counts()[:5]
```

Out[79]:
```
Category_ID
4756105    237964
4145813    168878
2355072    164428
3607361    155187
982926     150914
Name: count, dtype: int64
```

In [80]:
```
# Daily Count of "Buy" Behavior over Time
buy = df_5M[df_5M.Behavior == 'Buy']
buy_count = buy.groupby('Date').size().reset_index(name='count')
buy_count = buy_count.sort_values('Date')
```

In [81]:
```
# Day_of_week count of "Buy" Behavior over Time
days_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
buy_day_of_week = buy.groupby('Day_of_Week').size().reset_index(name='count')
# used for ordinal data where the order matters
buy_day_of_week['Day_of_Week'] = pd.Categorical(buy_day_of_week['Day_of_Week'], categories=days_order, ordered=True)
```

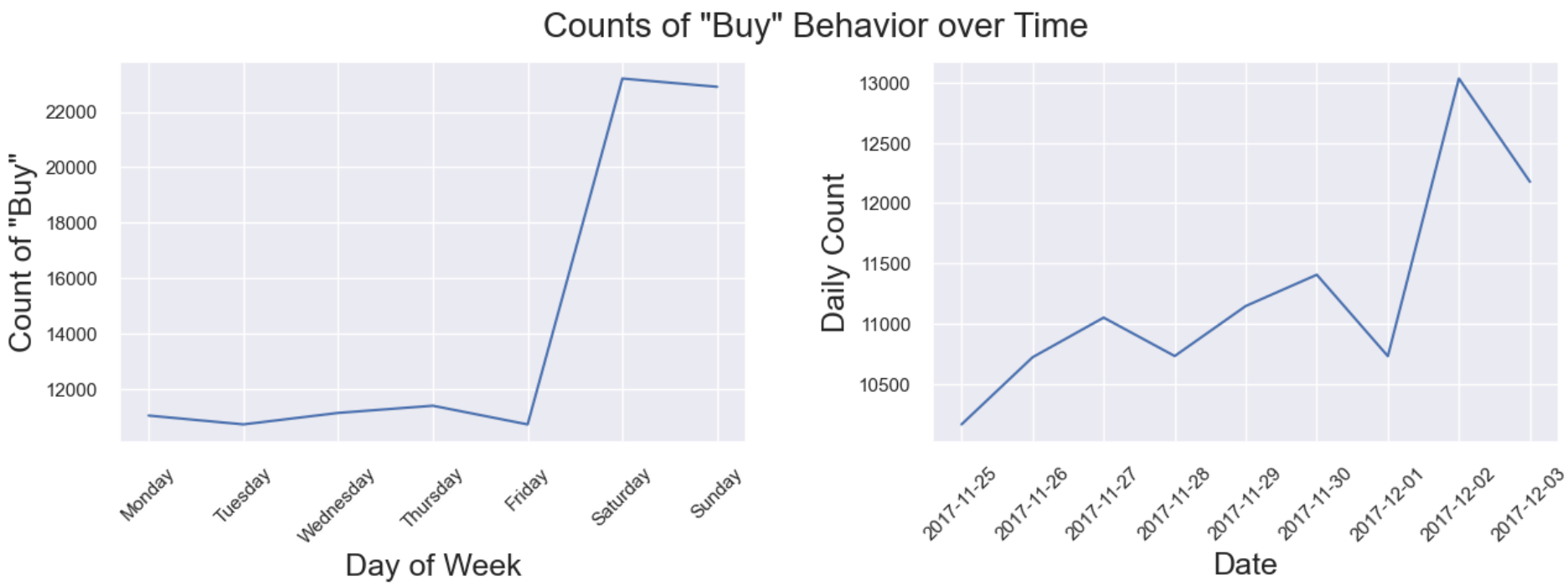
```
buy_day_of_week = buy_day_of_week.sort_values('Day_of_Week')
buy_day_of_week
```

Out[81]:

	Day_of_Week	count
1	Monday	11053
5	Tuesday	10735
6	Wednesday	11149
4	Thursday	11408
0	Friday	10734
2	Saturday	23201
3	Sunday	22902

In [230…

```
f, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 4))
ax2.plot(buy_count['Date'], buy_count['count'])
ax2.set_xlabel('Date', fontsize=18)
ax2.set_ylabel('Daily Count', fontsize=18)
ax2.tick_params(axis='x', rotation=45)
ax1.plot(buy_day_of_week['Day_of_Week'], buy_day_of_week['count'])
ax1.set_xlabel('Day of Week', fontsize=18)
ax1.set_ylabel('Count of "Buy"', fontsize=18)
ax1.tick_params(axis='x', rotation=45)
f.suptitle('Counts of "Buy" Behavior over Time', fontsize=20)
plt.subplots_adjust(wspace=0.3)
plt.savefig('Buy_Count.png', format='png', bbox_inches='tight')
```



In [82]:
```
df_5M['Day_of_Week'] = pd.Categorical(df_5M['Day_of_Week'], categories=days_order, ordered=True)
df_5M = df_5M.sort_values('Day_of_Week')
df_5M
```

Out[82]:

	User_ID	Product_ID	Category_ID	Behavior	Timestamp	Datetime	Day_of_Week	Hour	Date
61926662	17266	5006106	4181361	Buy	1511747369	2017-11-27 01:49:29	Monday	1	2017-11-27
87608099	423084	4715663	4801426	PageView	1511751060	2017-11-27 02:51:00	Monday	2	2017-11-27
87608098	423084	1757201	1320293	PageView	1511751038	2017-11-27 02:50:38	Monday	2	2017-11-27
87608097	423084	879245	3702593	PageView	1511751024	2017-11-27 02:50:24	Monday	2	2017-11-27
87608096	423084	4976579	4801426	PageView	1511750963	2017-11-27 02:49:23	Monday	2	2017-11-27
...	...	...	...	...	...	...	...	...	...
35822558	809612	3496543	3524510	PageView	1511667625	2017-11-26 03:40:25	Sunday	3	2017-11-26
35822559	809612	3182688	411153	PageView	1511668617	2017-11-26 03:56:57	Sunday	3	2017-11-26
35822560	809612	2022824	3323163	PageView	1511668669	2017-11-26 03:57:49	Sunday	3	2017-11-26
35822552	809612	1519068	3717634	PageView	1511667382	2017-11-26 03:36:22	Sunday	3	2017-11-26
100150754	999997	3572913	1051370	PageView	1512276510	2017-12-03 04:48:30	Sunday	4	2017-12-03

4952632 rows × 9 columns

In [98]:
```
hourly_behavior = df_5M.groupby(['Hour', 'Behavior']).size().reset_index(name='Behavior_Count')
hourly_behavior_count = hourly_behavior.pivot_table(index='Hour', columns='Behavior', values='Behavior_Count', fill_val
hourly_behavior_count.reset_index(inplace=True)
hourly_behavior_count.head()
```

Out[98]:

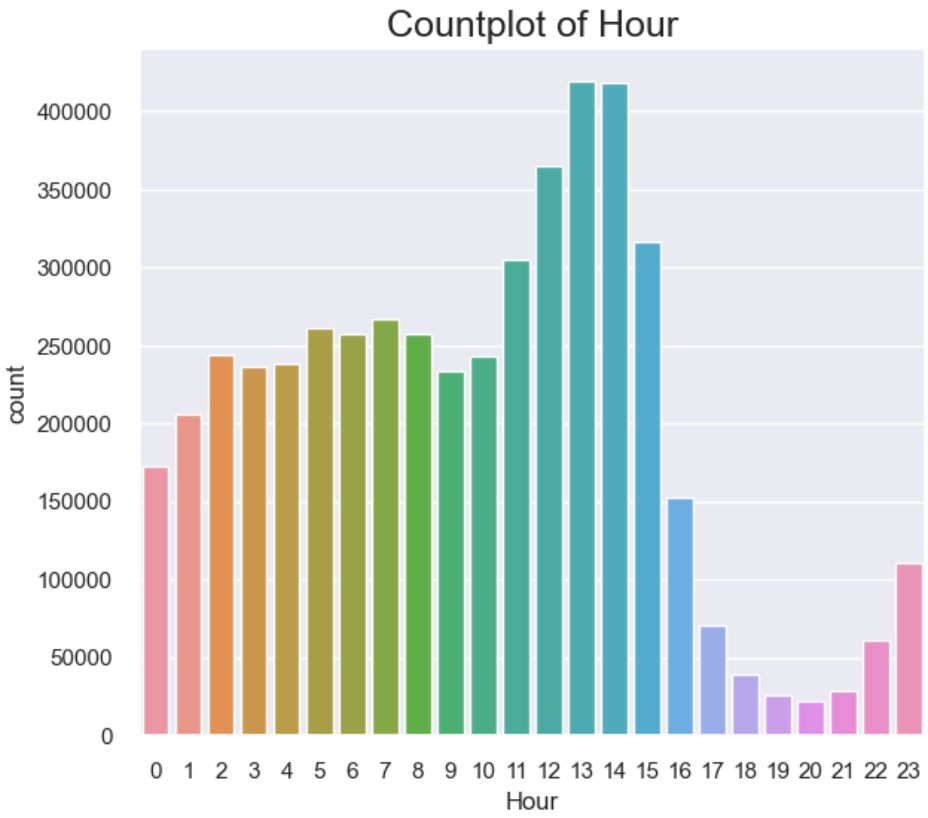
Behavior	Hour	AddToCart	Buy	Favorite	PageView
0	0	9631	3178	5430	154049
1	1	11362	4785	6462	183832
2	2	13051	6492	7492	216719
3	3	12923	6325	7081	210573
4	4	12746	5890	6940	212588

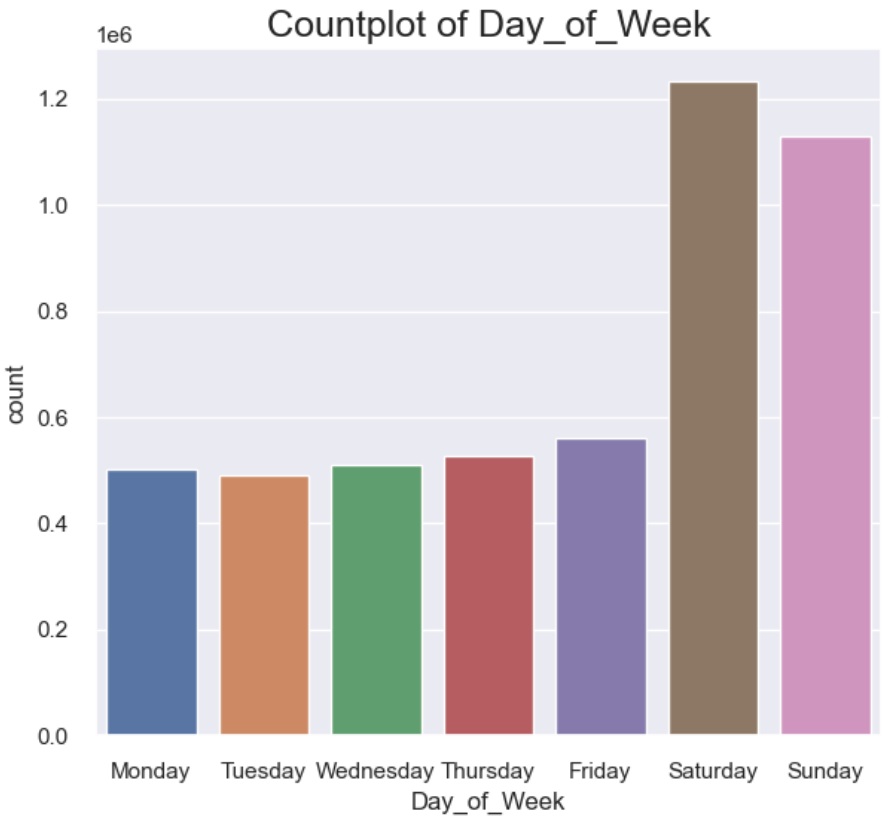
In [ ]:
```
f, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 4))
ax1.plot(buy_count['Date'], buy_count['count'])
ax1.set_xlabel('Date', fontsize=18)
ax1.set_ylabel('Daily Count', fontsize=18)
ax1.tick_params(axis='x', rotation=45)
ax2.plot(buy_day_of_week['Day_of_Week'], buy_day_of_week['count'])
```

```
ax2.set_xlabel('Day of Week', fontsize=18)
ax2.set_ylabel('Count of "Buy"', fontsize=18)
ax2.tick_params(axis='x', rotation=45)
f.suptitle('Counts of "Buy" Behavior over Time', fontsize=20)
plt.subplots_adjust(wspace=0.3)
plt.savefig('Buy_Count.png', format='png', bbox_inches='tight')
```

In [145…

```
plt.figure(1, figsize=(15, 6))
n = 0
for x in ['Hour', 'Day_of_Week']:
    n += 1
    plt.subplot(1, 2, n)
    plt.subplots_adjust(hspace=0.5, wspace=0.2)
    sns.countplot(data=df_5M, x=x)
    plt.title('Countplot of {}'.format(x), fontsize=18)

plt.savefig('Countplot of Hour and Day_of_Week.png', format='png', bbox_inches='tight')
plt.show()
```





In [235…

```
df_5M.groupby('Hour')['Behavior'].count().sum()
```

Out[235… 4952632

In [186…
```
# Calculate conversion rate for each category
conversion = df_5M[df_5M['Behavior'].isin(['Buy', 'PageView'])]
conversion_pb = conversion.groupby('Category_ID')['Behavior'].value_counts().unstack().fillna(0)
conversion_pb = conversion_pb[conversion_pb['PageView'] >= conversion_pb['Buy']]
conversion_pb['Conversion_Rate'] = conversion_pb['Buy'] / (conversion_pb['Buy'] + conversion_pb['PageView'])
conversion_pb.head()
```

Out[186…

Behavior	Buy	PageView	Conversion_Rate
Category_ID			
2171	6.0	48.0	0.111111
2410	2.0	31.0	0.060606
3579	0.0	5.0	0.000000
4907	2.0	18.0	0.100000
5064	18.0	1152.0	0.015385

In [272…
```
# the conversion rate of the top 10 sale categories
conversion_10 = conversion_pb.sort_values(by='Buy', ascending=False)[:10]
conversion_10
```

Out[272…

Category_ID	Behavior	Buy	PageView	Conversion_Rate
2735466		1793.0	57013.0	0.030490
1464116		1722.0	34721.0	0.047252
4145813		1578.0	153557.0	0.010172
2885642		1546.0	48395.0	0.030957
4756105		1378.0	219062.0	0.006251
4801426		1306.0	91667.0	0.014047
982926		1193.0	138090.0	0.008565
2640118		965.0	37297.0	0.025221
4159072		925.0	9417.0	0.089441
1320293		920.0	88614.0	0.010275

In [265…

```
top10_pv_categories = df_5M[df_5M['Behavior'] == 'PageView'].groupby('Category_ID').size().sort_values(ascending=False)
top10_buy_categories = df_5M[df_5M['Behavior'] == 'Buy'].groupby('Category_ID').size().sort_values(ascending=False)[:10
top10_pv_categories
```

Out[265…

	Category_ID	PageView Count
0	4756105	219062
1	2355072	154040
2	4145813	153557
3	3607361	145799
4	982926	138090
5	2520377	100726
6	4801426	91667
7	1320293	88614
8	2465336	71762
9	3002561	69887

In [266…

```
top10_buy_categories
```

Out[266…

	Category_ID	Buy Count
0	2735466	1793
1	1464116	1722
2	4145813	1578
3	2885642	1546
4	4756105	1378
5	4801426	1306
6	982926	1193
7	2640118	965
8	4159072	925
9	1320293	920

3. Customer Segmentation

3.1 RFM Analysis

In [273…

```
# 1. Calculate recency
recency = df_5M[df_5M['Behavior'] == 'Buy'].groupby(by='User_ID', as_index=False)['Date'].max()
recency.columns = ['User_ID', 'LastPurshaceDate']

current_date = df_5M[df_5M['Behavior'] == 'Buy']['Date'].max()
recency['Recency'] = recency['LastPurshaceDate'].apply(lambda x: (current_date - x).days)
recency.head()
```

Out[273…

	User_ID	LastPurshaceDate	Recency
0	20	2017-12-01	2
1	50	2017-12-03	0
2	66	2017-12-03	0
3	76	2017-11-30	3
4	101	2017-12-03	0

In [274…
```
# 2. Calculate Frequency
frequency = df_5M[df_5M['Behavior'] == 'Buy'].groupby('User_ID')['Behavior'].count().reset_index()
frequency.columns = ['User_ID', 'Frequency']
frequency.head()
```

Out[274…

	User_ID	Frequency
0	20	1
1	50	19
2	66	4
3	76	1
4	101	7

In [283…
```
# 3. Create RFM table
rfm = recency.merge(frequency, on='User_ID')
rfm.drop(columns=['LastPurshaceDate'], inplace=True)
rfm.head()
```

Out[283…

	User_ID	Recency	Frequency
0	20	2	1
1	50	0	19
2	66	0	4
3	76	3	1
4	101	0	7

In [284…
```
# 4. Assign R, F quartile values
quantiles = rfm.quantile(q=[0.25, 0.5, 0.75])
quantiles = quantiles.to_dict()

def r_score(x):
    if x <= quantiles['Recency'][0.25]:
        return 4
    elif x <= quantiles['Recency'][0.5]:
        return 3
    elif x <= quantiles['Recency'][0.75]:
        return 2
    else:
        return 1
```

```
def f_score(x):
    if x <= quantiles['Frequency'][0.25]:
        return 1
    elif x <= quantiles['Frequency'][0.5]:
        return 2
    elif x <= quantiles['Frequency'][0.75]:
        return 3
    else:
        return 4

rfm['R_score'] = rfm['Recency'].apply(r_score)
rfm['F_score'] = rfm['Frequency'].apply(f_score)
# rfm['RFM_Class'] = rfm['R_score'].astype(str) + rfm['F_score'].astype(str)
rfm.head()
```

Out[284…

	User_ID	Recency	Frequency	R_score	F_score
0	20	2	1	3	1
1	50	0	19	4	4
2	66	0	4	4	3
3	76	3	1	2	1
4	101	0	7	4	4

3. K-means Clustering-based Customer Segmentation

In [279…

```
from sklearn.preprocessing import StandardScaler
from yellowbrick.cluster import KElbowVisualizer
from sklearn.cluster import KMeans
```

In [280…

```
rfm_val = rfm[['Recency', 'Frequency']]
scaler = StandardScaler()
rfm_scaled = scaler.fit(rfm_val)
rfm_scaled = scaler.fit_transform(rfm_val)
rfm_scaled
```

Out[280…

```
array([[-0.25092592, -0.66796957],
       [-1.07656108,  5.31883678],
       [-1.07656108,  0.32983149],
       ...,
       [-0.25092592,  0.99503219],
       [ 0.57470923,  0.32983149],
       [ 1.81316196, -0.66796957]])
```

In [281…
```
rfm_val
```

Out[281…

	Recency	Frequency
0	2	1
1	0	19
2	0	4
3	3	1
4	0	7
…	…	…
33629	0	7
33630	7	5
33631	2	6
33632	4	4
33633	7	1

33634 rows × 2 columns

In [143…
```
# determine the number of clusters using Elbow method
plt.figure(figsize=(6, 4))
model = KMeans()
visualizer = KElbowVisualizer(model, k=(1,12), timings=False)
visualizer.fit(rfm_scaled)
plt.savefig('Elbow_Plot.png')
visualizer.show()
```

Out[143… <Axes: title={'center': 'Distortion Score Elbow for KMeans Clustering'}, xlabel='k', ylabel='distortion score'>

The optimal cluster value (K) for Elbow was found to be 3.

In [282…
```
kmeans_scaled = KMeans(3)
kmeans_scaled.fit(rfm_scaled)
identified_clusters = kmeans_scaled.fit_predict(rfm_val)

rfm['Cluster'] = kmeans_scaled.fit_predict(rfm_scaled)
rfm.head()
```

Out[282…

	User_ID	Recency	Frequency	R_score	F_score	Cluster
0	20	2	1	3	1	0
1	50	0	19	4	4	2
2	66	0	4	4	3	0
3	76	3	1	2	1	0
4	101	0	7	4	4	2

In [289…
```
# Filter the original dataset based on cluster labels
cluster_0 = clusters_scaled[clusters_scaled['cluster_pred'] == 0]
cluster_1 = clusters_scaled[clusters_scaled['cluster_pred'] == 1]
cluster_2 = clusters_scaled[clusters_scaled['cluster_pred'] == 2]
```

In [286…
```
cluster_0.describe()
```

Out[286…

	Recency	Frequency	cluster_pred
count	10546.000000	10546.000000	10546.0
mean	5.698464	1.810449	0.0
std	1.399015	1.132397	0.0
min	4.000000	1.000000	0.0
25%	4.000000	1.000000	0.0
50%	6.000000	1.000000	0.0
75%	7.000000	2.000000	0.0
max	8.000000	8.000000	0.0

In [287…
```
cluster_1.describe()
```

Out[287…

	Recency	Frequency	cluster_pred
count	19076.000000	19076.000000	19076.0
mean	1.218180	2.444800	1.0
std	1.090888	1.272697	0.0
min	0.000000	1.000000	1.0
25%	0.000000	1.000000	1.0
50%	1.000000	2.000000	1.0
75%	2.000000	3.000000	1.0
max	3.000000	5.000000	1.0

In [288…
```
cluster_2.describe()
```

Out[288…

	Recency	Frequency	cluster_pred
count	4012.000000	4012.000000	4012.0
mean	1.091226	8.836491	2.0
std	1.341716	5.046647	0.0
min	0.000000	6.000000	2.0
25%	0.000000	6.000000	2.0
50%	1.000000	7.000000	2.0
75%	2.000000	10.000000	2.0
max	7.000000	175.000000	2.0

In [130…

```
def categorize_customers(row):
    if row['Cluster'] == 0:
        return 'Churn Risk Customers'
    elif row['Cluster'] == 1:
        return 'Potential Customers'
    elif row['Cluster'] == 2:
        return 'High-Value Customers'

rfm['class'] = rfm.apply(categorize_customers, axis=1)
```

In [131…

```
rfm.head()
```

Out[131…

	User_ID	Recency	Frequency	R_score	F_score	RFM_Class	Cluster	class
0	20	2	1	3	1	31	1	Potential Customers
1	50	0	19	4	4	44	0	Churn Risk Customers
2	66	0	4	4	3	43	1	Potential Customers
3	76	3	1	2	1	21	1	Potential Customers
4	101	0	7	4	4	44	0	Churn Risk Customers

In [139…

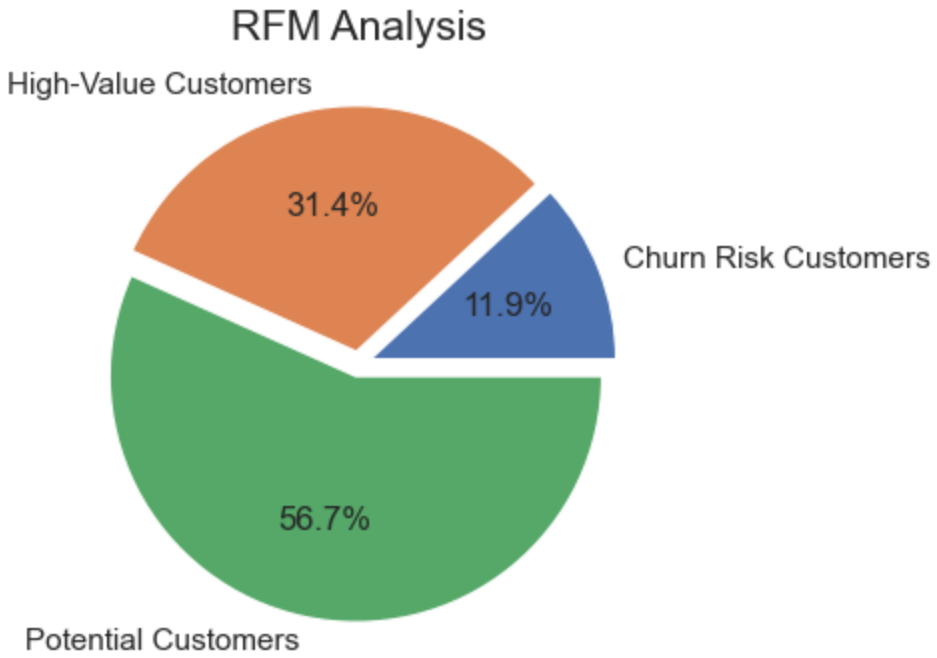
```
customer_class = rfm.groupby('class')['User_ID'].count().reset_index()
customer_class.columns = ['Customer Class', 'Counts']
customer_class.head()
```

Out[139…

	Customer Class	Counts
0	Churn Risk Customers	4012
1	High-Value Customers	10546
2	Potential Customers	19076

In [147…

```
plt.figure(figsize=(4, 4))
plt.pie(customer_class['Counts'], labels=customer_class['Customer Class'], autopct='%1.1f%%', explode=[0.05]*3)
plt.title('RFM Analysis', fontsize=15)

plt.savefig('RFM Analysis.png', format='png', bbox_inches='tight')
plt.show()
```



4. Predictive Behavior Modeling

In [30]:

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split
from torch.nn.utils.rnn import pack_padded_sequence, pad_packed_sequence
```

```
from tqdm import tqdm
from sklearn.metrics import accuracy_score
```

In [24]:
```
df_5M_time = df_5M.sort_values(by=['User_ID', 'Datetime'])
behavior_count = df_5M_time.groupby(['Product_ID', 'User_ID'])['Behavior'].size().reset_index(name='Behavior_Count')
filtered_bc = behavior_count[behavior_count['Behavior_Count'] >= 5]
df_5M_filtered = df_5M_time.merge(filtered_bc, on=['Product_ID', 'User_ID'])
df_5M_filtered.head()
```

Out[24]:

	User_ID	Product_ID	Category_ID	Behavior	Timestamp	Datetime	Day_of_Week	Hour	Date	Behavior_Count
0	20	78457	4671427	AddToCart	1511853310	2017-11-28 07:15:10	Tuesday	7	2017-11-28	6
1	20	78457	4671427	PageView	1511853498	2017-11-28 07:18:18	Tuesday	7	2017-11-28	6
2	20	78457	4671427	PageView	1511853565	2017-11-28 07:19:25	Tuesday	7	2017-11-28	6
3	20	78457	4671427	PageView	1511853640	2017-11-28 07:20:40	Tuesday	7	2017-11-28	6
4	20	78457	4671427	PageView	1512048422	2017-11-30 13:27:02	Thursday	13	2017-11-30	6

In [25]:
```
sequence = df_5M_filtered.groupby(['Product_ID', 'User_ID'])['Behavior'].apply(list).reset_index(name='Sequence')
sequence.drop(columns=['Product_ID'], inplace=True)
sequence.head()
```

Out[25]:

	User_ID	Sequence
0	491602	[PageView, Favorite, PageView, PageView, PageV...
1	834428	[AddToCart, PageView, PageView, PageView, Page...
2	503736	[PageView, Favorite, PageView, AddToCart, Page...
3	522151	[PageView, PageView, PageView, PageView, PageV...
4	520895	[PageView, PageView, PageView, PageView, PageV...

In [26]:
```
sequence_train, sequence_test = train_test_split(sequence, test_size=0.2, random_state=42)
sequence_val, sequence_test = train_test_split(sequence_test, test_size=0.5, random_state=42)
```

In [27]:
```
sequence_train.head()
```

Out[27]:

	User_ID	Sequence
3504	917237	[AddToCart, PageView, PageView, PageView, Page...
38671	280254	[PageView, PageView, PageView, PageView, PageV...
61227	719781	[PageView, PageView, PageView, PageView, PageV...
29353	983515	[AddToCart, Buy, PageView, PageView, Favorite]
40033	623583	[PageView, PageView, Buy, PageView, PageView, ...

In [28]:

```
# LSTM
PADDING_IDX = 4
class UserBehaviorDataset(Dataset):
    def __init__(self, dataframe):
        self.dataframe = dataframe
        self.token_to_idx = {'PageView': 0, 'AddToCart': 1, 'Buy': 2, 'Favorite': 3,  'Padding': PADDING_IDX}

    def __len__(self):
        return len(self.dataframe)

    def __getitem__(self, idx):
        sequence = self.dataframe.iloc[idx, 1]
        sequence_idx = [self.token_to_idx[action] for action in sequence]
        return torch.tensor(sequence_idx), torch.tensor(len(sequence_idx))

def pad_collate(batch):
    (xx, yy) = zip(*batch)
    x_lens = torch.tensor(yy)
    xx_pad = torch.nn.utils.rnn.pad_sequence(xx, batch_first=True, padding_value=0)
    return xx_pad, x_lens

train_dataset = UserBehaviorDataset(sequence_train)
val_dataset = UserBehaviorDataset(sequence_val)
device = torch.device("cuda:6" if torch.cuda.is_available() else "cpu")
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True, collate_fn=pad_collate)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False, collate_fn=pad_collate)

class LSTMBehaviorModel(nn.Module):
    def __init__(self, num_tokens, hidden_size=128, num_layers=2):
        super(LSTMBehaviorModel, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.embed = nn.Embedding(num_tokens, hidden_size)
        self.lstm = nn.LSTM(hidden_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, num_tokens)
```

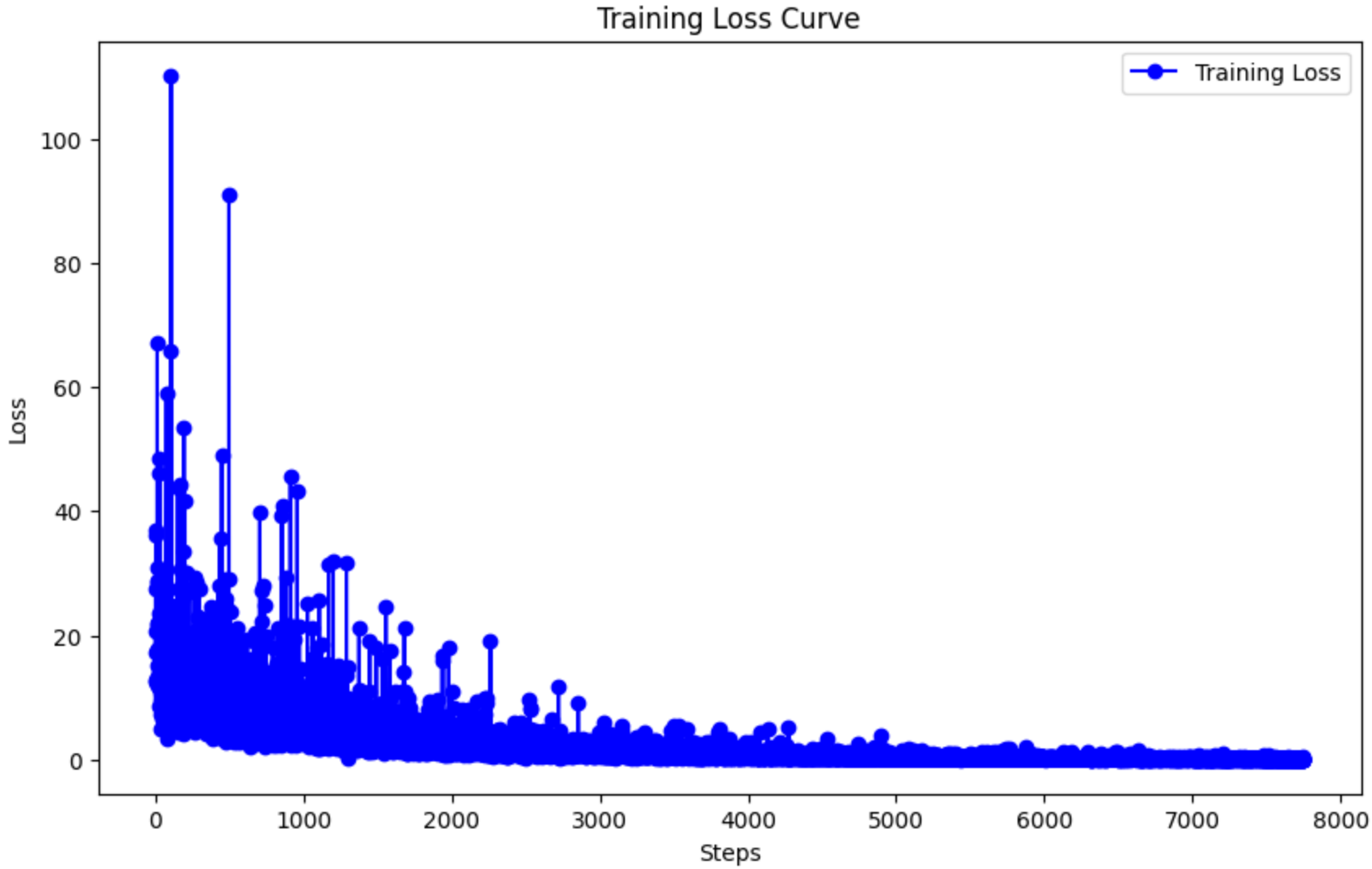
```
    def forward(self, x, x_lens):
        x = self.embed(x)
        packed_x = pack_padded_sequence(x, x_lens.cpu(), batch_first=True, enforce_sorted=False)
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(x.device)
        c0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(x.device)
        packed_out, _ = self.lstm(packed_x, (h0, c0))
        out, _ = pad_packed_sequence(packed_out, batch_first=True)
        out = self.fc(out)
        return out

# Hyper-parameters
model = LSTMBehaviorModel(num_tokens=5).to(device)
criterion = nn.CrossEntropyLoss(ignore_index=PADDING_IDX, reduction='none').to(device)
optimizer = optim.Adam(model.parameters(), lr=0.001)

best_val_loss = float('inf')
num_epochs = 5
train_loss_list = []
val_loss_list = []
for epoch in range(num_epochs):
    model.train()
    for x_padded, x_lens in tqdm(train_loader):
        x_padded, x_lens = x_padded.to(device), x_lens.to(device)
        optimizer.zero_grad()
        output = model(x_padded, x_lens)
        loss = criterion(output.view(-1, 5), x_padded.view(-1))
        loss = loss.view(x_padded.size(0), -1).sum(dim=1).mean()
        train_loss_list.append(loss.item())
        loss.backward()
        optimizer.step()

    model.eval()
    total_val_loss = 0
    with torch.no_grad():
        for x_padded, x_lens in val_loader:
            x_padded, x_lens = x_padded.to(device), x_lens.to(device)
            output = model(x_padded, x_lens)
            val_loss = criterion(output.view(-1, 5), x_padded.view(-1))
            #val_loss = val_loss / x_padded.size(0)
            val_loss = val_loss.view(x_padded.size(0), -1).sum(dim=1).mean()
            total_val_loss += val_loss.item()

    avg_val_loss = total_val_loss / len(val_loader)
    print(f"Epoch {epoch}: Avg. Validation Loss: {avg_val_loss}")

    if avg_val_loss < best_val_loss:
```

```
        best_val_loss = avg_val_loss
        torch.save(model.state_dict(), 'best_model.pth')
```

```
100%|██████████| 1550/1550 [00:17<00:00, 86.47it/s]
Epoch 0: Avg. Validation Loss: 3.955136271174421
100%|██████████| 1550/1550 [00:17<00:00, 90.67it/s]
Epoch 1: Avg. Validation Loss: 1.2421085392076945
100%|██████████| 1550/1550 [00:17<00:00, 91.10it/s]
Epoch 2: Avg. Validation Loss: 0.48849425549359665
100%|██████████| 1550/1550 [00:17<00:00, 89.28it/s]
Epoch 3: Avg. Validation Loss: 0.21568399404665245
100%|██████████| 1550/1550 [00:17<00:00, 89.16it/s]
Epoch 4: Avg. Validation Loss: 0.10051983695707678
```

In [29]:

```
# plot training loss curve
plt.figure(figsize=(10, 6))
plt.plot(list(range(0, len(train_loss_list))), train_loss_list, marker='o', color='b', label='Training Loss')
plt.title('Training Loss Curve')
plt.xlabel('Steps')
plt.ylabel('Loss')
plt.legend()
plt.savefig("loss.png", dpi=600)
plt.show()
```



In [32]:
```
test_df = sequence_test
test_dataset = UserBehaviorDataset(test_df)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False, collate_fn=pad_collate)

model.eval()

all_predictions = []
all_labels = []
with torch.no_grad():
    for x_padded, x_lens in test_loader:
        x_padded, x_lens = x_padded.to(device), x_lens.to(device)
        output = model(x_padded, x_lens)
        preds = output.argmax(dim=-1)
        for i, length in enumerate(x_lens):
```

```
            all_predictions.append(preds[i, length - 1].item())
            all_labels.append(x_padded[i, length - 1].item())

accuracy = accuracy_score(all_labels, all_predictions)
print(f"Test Accuracy: {accuracy}")
```

```
Test Accuracy: 0.9517741935483871
```

In [40]:

```
# rnn
device = torch.device("cuda:6" if torch.cuda.is_available() else "cpu")

class RNNBehaviorModel(nn.Module):
    def __init__(self, num_tokens, hidden_size=128, num_layers=1):
        super(RNNBehaviorModel, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.embed = nn.Embedding(num_tokens, hidden_size)
        self.rnn = nn.RNN(hidden_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, num_tokens)

    def forward(self, x, x_lens):
        x = self.embed(x)
        packed_x = pack_padded_sequence(x, x_lens.cpu(), batch_first=True, enforce_sorted=False)
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(x.device)
        packed_out, _ = self.rnn(packed_x, h0)
        out, _ = pad_packed_sequence(packed_out, batch_first=True)
        out = self.fc(out)
        return out


model = RNNBehaviorModel(num_tokens=5).to(device)
criterion = nn.CrossEntropyLoss(ignore_index=PADDING_IDX, reduction='none').to(device)
optimizer = optim.Adam(model.parameters(), lr=0.001)

best_val_loss = float('inf')
num_epochs = 5

for epoch in range(num_epochs):
    model.train()
    for x_padded, x_lens in tqdm(train_loader):
        x_padded, x_lens = x_padded.to(device), x_lens.to(device)
        optimizer.zero_grad()
        output = model(x_padded, x_lens)
        loss = criterion(output.view(-1, 5), x_padded.view(-1))
        #loss = loss / x_padded.size(0)
        loss = loss.view(x_padded.size(0), -1).sum(dim=1).mean()
        loss.backward()
        optimizer.step()
```

```
    model.eval()
    total_val_loss = 0
    with torch.no_grad():
        for x_padded, x_lens in val_loader:
            x_padded, x_lens = x_padded.to(device), x_lens.to(device)
            output = model(x_padded, x_lens)
            val_loss = criterion(output.view(-1, 5), x_padded.view(-1))
            val_loss = val_loss.view(x_padded.size(0), -1).sum(dim=1).mean()
            total_val_loss += val_loss.item()

    avg_val_loss = total_val_loss / len(val_loader)
    print(f"Epoch {epoch}: Avg. Validation Loss: {avg_val_loss}")

    if avg_val_loss < best_val_loss:
        best_val_loss = avg_val_loss
        torch.save(model.state_dict(), 'best_model.pth')
```

```
  0%|          | 0/1550 [00:00<?, ?it/s]100%|██████████| 1550/1550 [00:14<00:00, 105.02it/s]
Epoch 0: Avg. Validation Loss: 3.9904904024502668
100%|██████████| 1550/1550 [00:14<00:00, 107.76it/s]
Epoch 1: Avg. Validation Loss: 1.2342553794691242
100%|██████████| 1550/1550 [00:14<00:00, 108.08it/s]
Epoch 2: Avg. Validation Loss: 0.49265703601167377
100%|██████████| 1550/1550 [00:14<00:00, 106.65it/s]
Epoch 3: Avg. Validation Loss: 0.2187645414210472
100%|██████████| 1550/1550 [00:14<00:00, 106.48it/s]
Epoch 4: Avg. Validation Loss: 0.10188945355949942
```

In [42]:

```
model = RNNBehaviorModel(num_tokens=5).to(device)
model.eval()

all_predictions = []
all_labels = []

with torch.no_grad():
    for x_padded, x_lens in test_loader:
        x_padded, x_lens = x_padded.to(device), x_lens.to(device)
        output = model(x_padded, x_lens)
        preds = output.argmax(dim=-1)
        for i, length in enumerate(x_lens):
            all_predictions.append(preds[i, length - 1].item())
            all_labels.append(x_padded[i, length - 1].item())

accuracy = accuracy_score(all_labels, all_predictions)
```

```
print(f"Test Accuracy: {accuracy}")
```

```
Test Accuracy: 0.7324193548387097
```

In [ ]: