MILESTONE 1

Objective:

The objective of this project is to develop a recommendation system that enhances user experience by suggesting relevant products based on historical interactions with products and available product data. The system aims to leverage machine learning techniques to analyse customer behaviour and provide personalised recommendations.

Type of tool:

A recommendation engine will be developed based on the user behaviour, product interactions and pricing trends.

Tech Stack

- **Programming Language:** Python
- **Libraries & Frameworks:** Pandas, NumPy, Scikit-learn, Matplotlib, Seaborn
- **Visualization Tools:** Matplotlib, Seaborn

Data Collection:

The data used is open-source datasets available on Kaggle. The data describes the user's interactions on an ecommerce website/application, providing information regarding the time spent on the website, the products viewed/bought, price, name of the product, etc through the following attributes:

1. **event_time** (timestamp of interaction)
2. **event_type** (type of user interaction)
3. **product_id** (unique identifier for products)
4. **category_id** (unique category identifier)
5. **category_code** (product classification)
6. **brand** (product brand)
7. **price** (product price)
8. **user_id** (unique identifier for users)
9. **user_session** (unique identifier for user sessions)

Three csv files are used, 2 csv files contain information related to user's interactions with an electronic store/website and the other csv file contains information related to users' interactions with a multi category store.

Project Timeline

Milestone 1: Data Collection, Preprocessing, and EDA (Feb 5, 2025 - Feb 21, 2025)

1. Week 1 (Feb 5 - Feb 11): Identify and acquire dataset, verify accessibility, and document dataset details.
2. Week 2 (Feb 12 - Feb 18): Handle missing values, address outliers, and apply feature scaling.
3. Week 3 (Feb 19 - Feb 21): Perform exploratory data analysis (EDA), generate visualizations, and identify patterns.

Milestone 2: Feature Engineering, Feature Selection, and Data Modeling (Feb 21, 2025 - March 21, 2025)

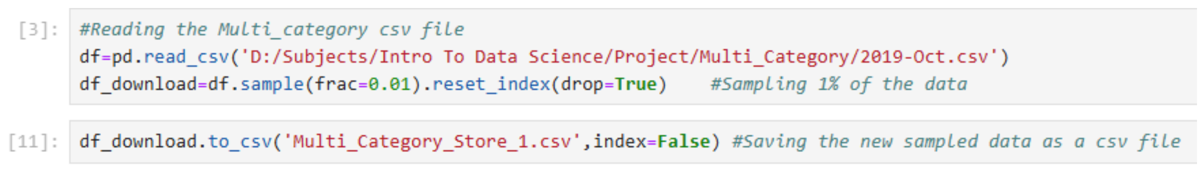
1. Week 4 (Feb 22 - Feb 28**):** Engineer new features and encode categorical variables.
2. Week 5 (March 1 - March 7): Perform feature selection and dimensionality reduction.
3. Week 6 (March 8 - March 14): Split data into training and testing sets, train initial models.
4. Week 7 (March 15 - March 21): Tune hyperparameters and evaluate models using performance metrics.

Milestone 3: Evaluation, Interpretation, Tool Development, and Presentation (March 24, 2025 - April 23, 2025)

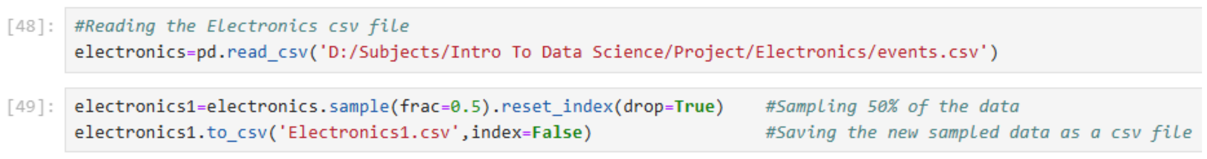
1. Week 8 (March 24 - March 30): Assess model performance and interpret results.
2. Week 9 (April 1 - April 7): Identify biases and refine models if needed.
3. Week 10 (April 8 - April 14): Develop an interactive tool (dashboard, conversational agent, or reporting system).
4. Week 11 (April 15 - April 23): Finalize report, prepare presentation, and deliver findings.

Data Preprocessing:

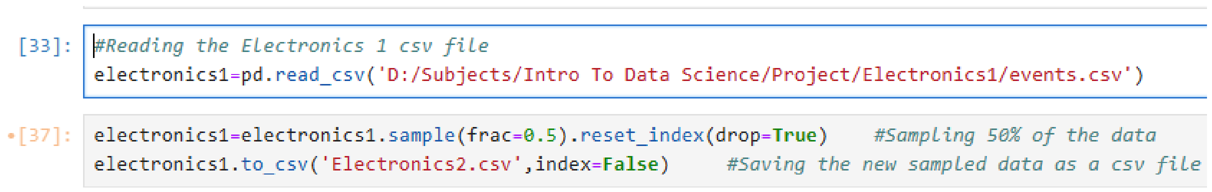
1. Resizing the datasets to a smaller size for memory efficiency:



```
[3]:  #Reading the Multi_category csv file
      df=pd.read_csv('D:/Subjects/Intro To Data Science/Project/Multi_Category/2019-Oct.csv')
      df_download=df.sample(frac=0.01).reset_index(drop=True)    #Sampling 1% of the data

[11]: df_download.to_csv('Multi_Category_Store_1.csv',index=False) #Saving the new sampled data as a csv file
```



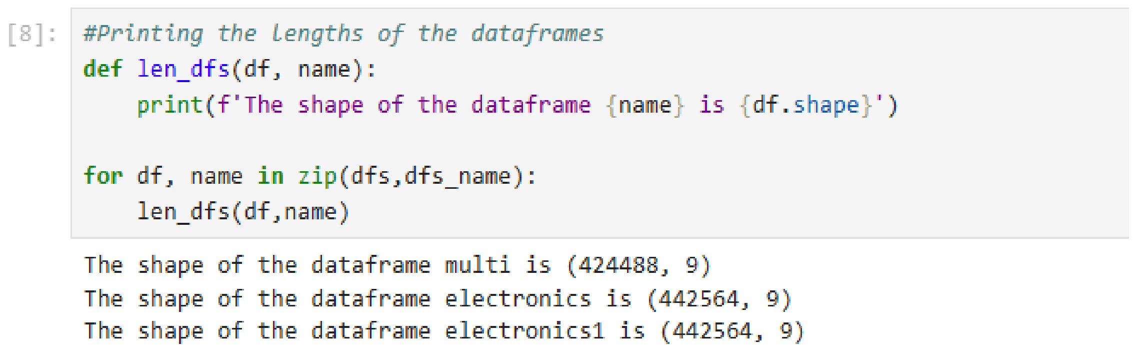
```
[48]: #Reading the Electronics csv file
      electronics=pd.read_csv('D:/Subjects/Intro To Data Science/Project/Electronics/events.csv')

[49]: electronics1=electronics.sample(frac=0.5).reset_index(drop=True)    #Sampling 50% of the data
      electronics1.to_csv('Electronics1.csv',index=False)                #Saving the new sampled data as a csv file
```



```
[33]: #Reading the Electronics 1 csv file
      electronics1=pd.read_csv('D:/Subjects/Intro To Data Science/Project/Electronics1/events.csv')

•[37]: electronics1=electronics1.sample(frac=0.5).reset_index(drop=True)    #Sampling 50% of the data
       electronics1.to_csv('Electronics2.csv',index=False)     #Saving the new sampled data as a csv file
```

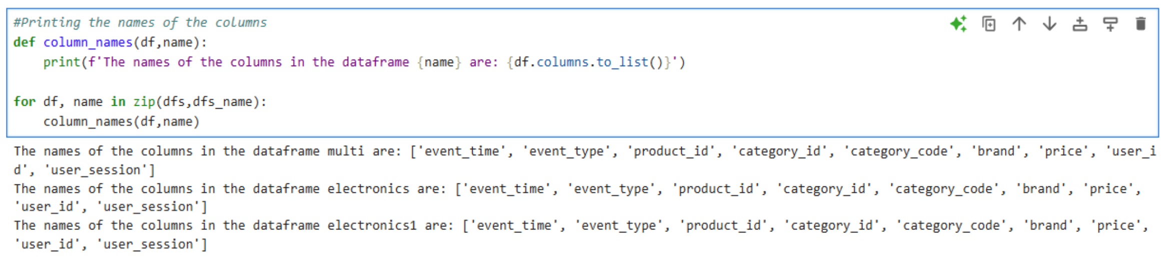
2. Statistical Analysis:

a. Printing the lengths of the dataframes:



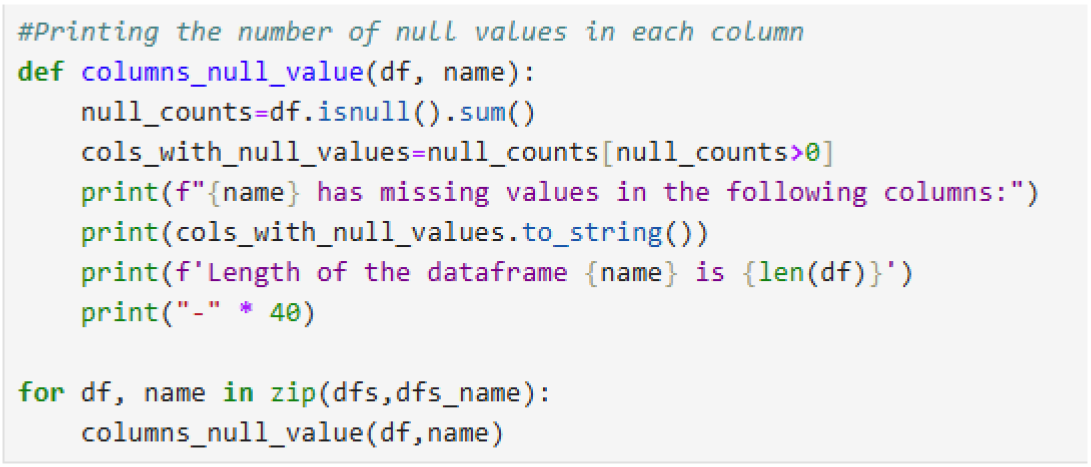
```
[8]: #Printing the lengths of the dataframes
     def len_dfs(df, name):
         print(f'The shape of the dataframe {name} is {df.shape}')

     for df, name in zip(dfs,dfs_name):
         len_dfs(df,name)

     The shape of the dataframe multi is (424488, 9)
     The shape of the dataframe electronics is (442564, 9)
     The shape of the dataframe electronics1 is (442564, 9)
```

b. Printing the names of the columns in the dataframes:



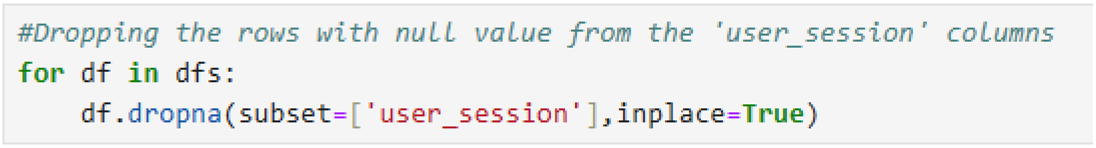
```
#Printing the names of the columns
def column_names(df,name):
    print(f'The names of the columns in the dataframe {name} are: {df.columns.to_list()}')

for df, name in zip(dfs,dfs_name):
    column_names(df,name)

The names of the columns in the dataframe multi are: ['event_time', 'event_type', 'product_id', 'category_id', 'category_code', 'brand', 'price', 'user_i
d', 'user_session']
The names of the columns in the dataframe electronics are: ['event_time', 'event_type', 'product_id', 'category_id', 'category_code', 'brand', 'price',
'user_id', 'user_session']
The names of the columns in the dataframe electronics1 are: ['event_time', 'event_type', 'product_id', 'category_id', 'category_code', 'brand', 'price',
'user_id', 'user_session']
```

c. Printing the number of null values in each column:



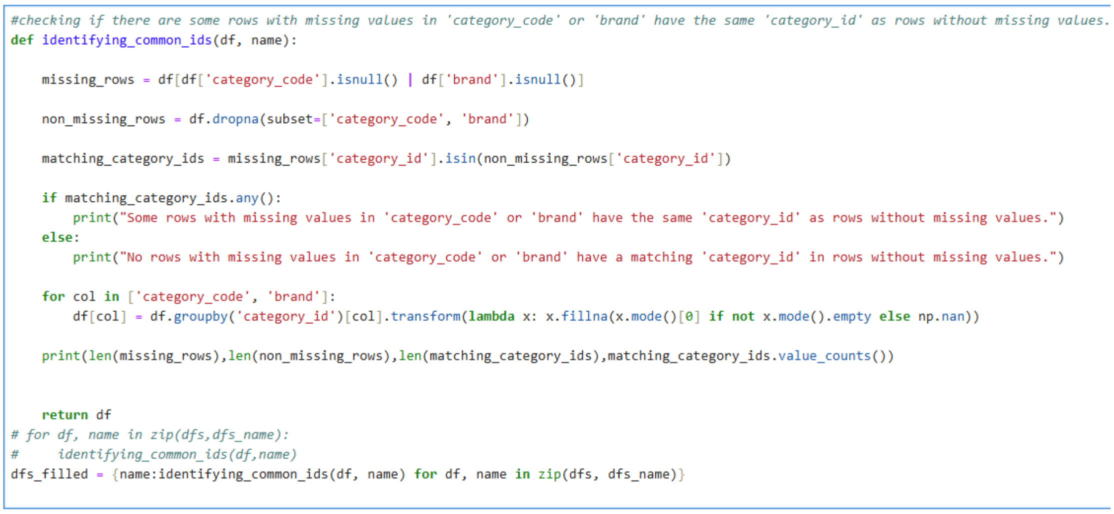
```python
#Printing the number of null values in each column
def columns_null_value(df, name):
    null_counts=df.isnull().sum()
    cols_with_null_values=null_counts[null_counts>0]
    print(f"{name} has missing values in the following columns:")
    print(cols_with_null_values.to_string())
    print(f'Length of the dataframe {name} is {len(df)}')
    print("-" * 40)

for df, name in zip(dfs,dfs_name):
    columns_null_value(df,name)
```

d. Dropping the rows with null values from the ‘user_session’ column:



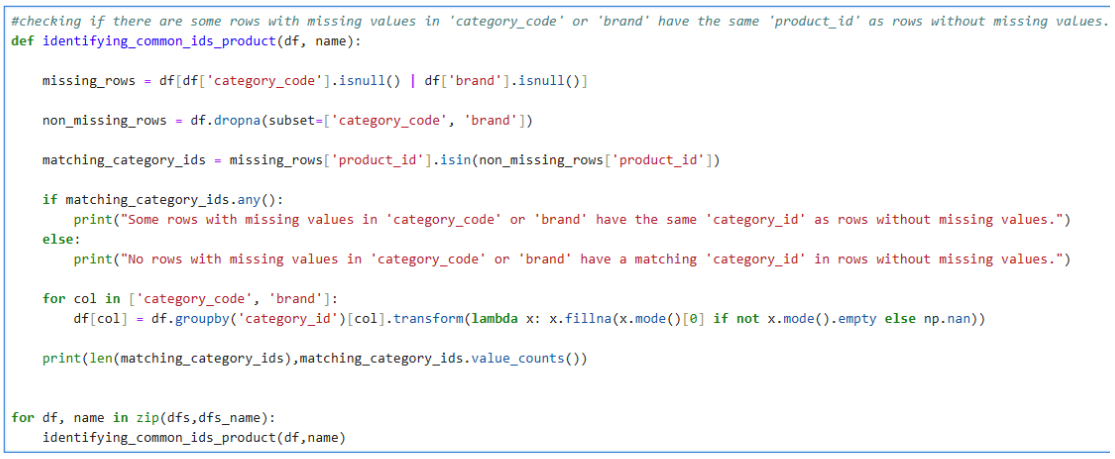
```python
#Dropping the rows with null value from the 'user_session' columns
for df in dfs:
    df.dropna(subset=['user_session'],inplace=True)
```

e. Checking if there are some rows with missing values in 'category_code' or 'brand' have the same 'category_id' as rows without missing values, instead of directly dropping the rows with null values:



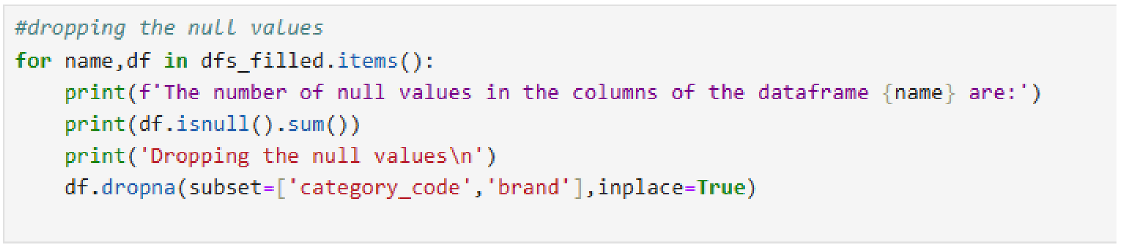
```python
#checking if there are some rows with missing values in 'category_code' or 'brand' have the same 'category_id' as rows without missing values.
def identifying_common_ids(df, name):

    missing_rows = df[df['category_code'].isnull() | df['brand'].isnull()]

    non_missing_rows = df.dropna(subset=['category_code', 'brand'])

    matching_category_ids = missing_rows['category_id'].isin(non_missing_rows['category_id'])

    if matching_category_ids.any():
        print("Some rows with missing values in 'category_code' or 'brand' have the same 'category_id' as rows without missing values.")
    else:
        print("No rows with missing values in 'category_code' or 'brand' have a matching 'category_id' in rows without missing values.")

    for col in ['category_code', 'brand']:
        df[col] = df.groupby('category_id')[col].transform(lambda x: x.fillna(x.mode()[0] if not x.mode().empty else np.nan))

    print(len(missing_rows),len(non_missing_rows),len(matching_category_ids),matching_category_ids.value_counts())


    return df
# for df, name in zip(dfs,dfs_name):
#     identifying_common_ids(df,name)
dfs_filled = {name:identifying_common_ids(df, name) for df, name in zip(dfs, dfs_name)}
```

f. Checking if there are some rows with missing values in 'category_code' or 'brand' have the same 'product_id' as rows without missing values, instead of directly dropping the rows with null values:



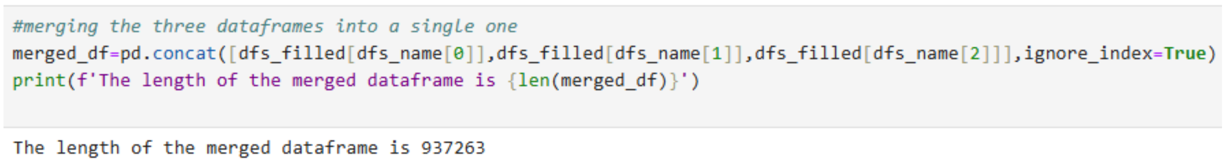
```
#checking if there are some rows with missing values in 'category_code' or 'brand' have the same 'product_id' as rows without missing values.
def identifying_common_ids_product(df, name):

    missing_rows = df[df['category_code'].isnull() | df['brand'].isnull()]

    non_missing_rows = df.dropna(subset=['category_code', 'brand'])

    matching_category_ids = missing_rows['product_id'].isin(non_missing_rows['product_id'])

    if matching_category_ids.any():
        print("Some rows with missing values in 'category_code' or 'brand' have the same 'category_id' as rows without missing values.")
    else:
        print("No rows with missing values in 'category_code' or 'brand' have a matching 'category_id' in rows without missing values.")

    for col in ['category_code', 'brand']:
        df[col] = df.groupby('category_id')[col].transform(lambda x: x.fillna(x.mode()[0] if not x.mode().empty else np.nan))

    print(len(matching_category_ids),matching_category_ids.value_counts())


for df, name in zip(dfs,dfs_name):
    identifying_common_ids_product(df,name)
```

g. Dropping the rows with null values that did not have a common ‘category_id’ or ‘product_id’ with the rows that do not have a null value:



```
#dropping the null values
for name,df in dfs_filled.items():
    print(f'The number of null values in the columns of the dataframe {name} are:')
    print(df.isnull().sum())
    print('Dropping the null values\n')
    df.dropna(subset=['category_code','brand'],inplace=True)
```

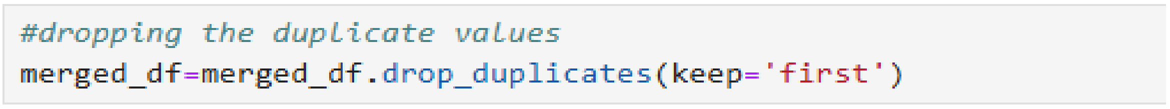
3. Merging the dataframes together:



```
#merging the three dataframes into a single one
merged_df=pd.concat([dfs_filled[dfs_name[0]],dfs_filled[dfs_name[1]],dfs_filled[dfs_name[2]]],ignore_index=True)
print(f'The length of the merged dataframe is {len(merged_df)}')
```

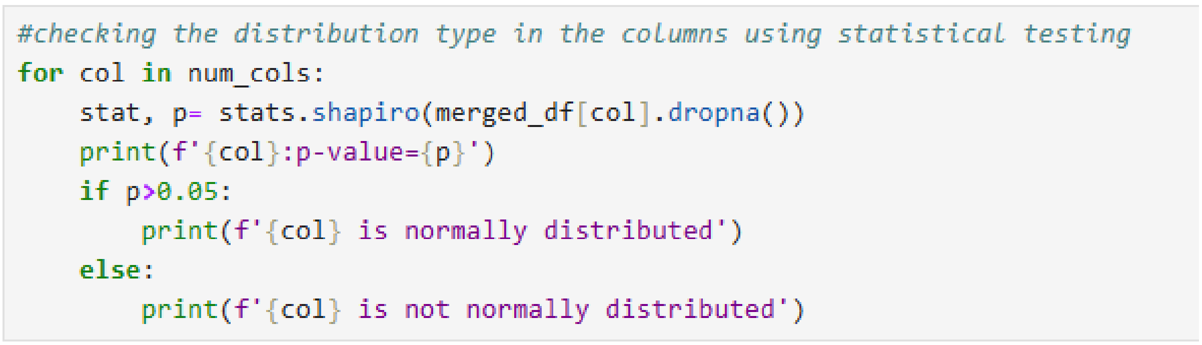
```
The length of the merged dataframe is 937263
```

4. Dropping the duplicate values from the merged dataframe:



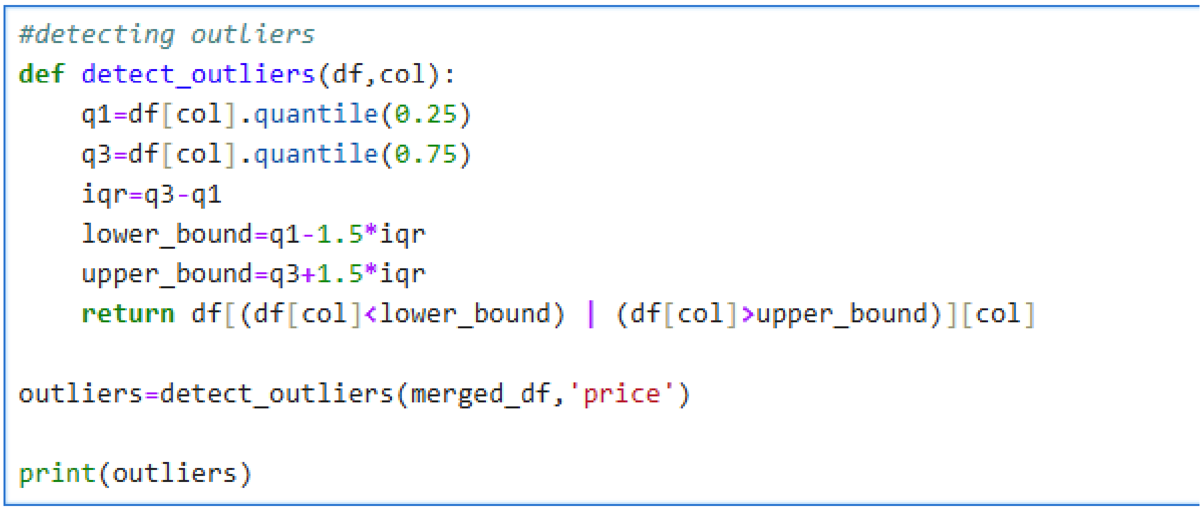
```
#dropping the duplicate values
merged_df=merged_df.drop_duplicates(keep='first')
```

5. **Performing statistical testing on the numerical columns to determine the distribution of the columns (normal or non-normal):**



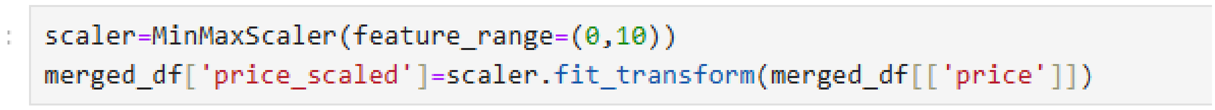
```
#checking the distribution type in the columns using statistical testing
for col in num_cols:
    stat, p= stats.shapiro(merged_df[col].dropna())
    print(f'{col}:p-value={p}')
    if p>0.05:
        print(f'{col} is normally distributed')
    else:
        print(f'{col} is not normally distributed')
```

6. **Detecting the outliers in the 'price' column:**



```
#detecting outliers
def detect_outliers(df,col):
    q1=df[col].quantile(0.25)
    q3=df[col].quantile(0.75)
    iqr=q3-q1
    lower_bound=q1-1.5*iqr
    upper_bound=q3+1.5*iqr
    return df[(df[col]<lower_bound) | (df[col]>upper_bound)][col]

outliers=detect_outliers(merged_df,'price')

print(outliers)
```

7. **Performing scaling operation on the 'price' column to reduce the effect of outliers on the data:**



```
scaler=MinMaxScaler(feature_range=(0,10))
merged_df['price_scaled']=scaler.fit_transform(merged_df[['price']])
```

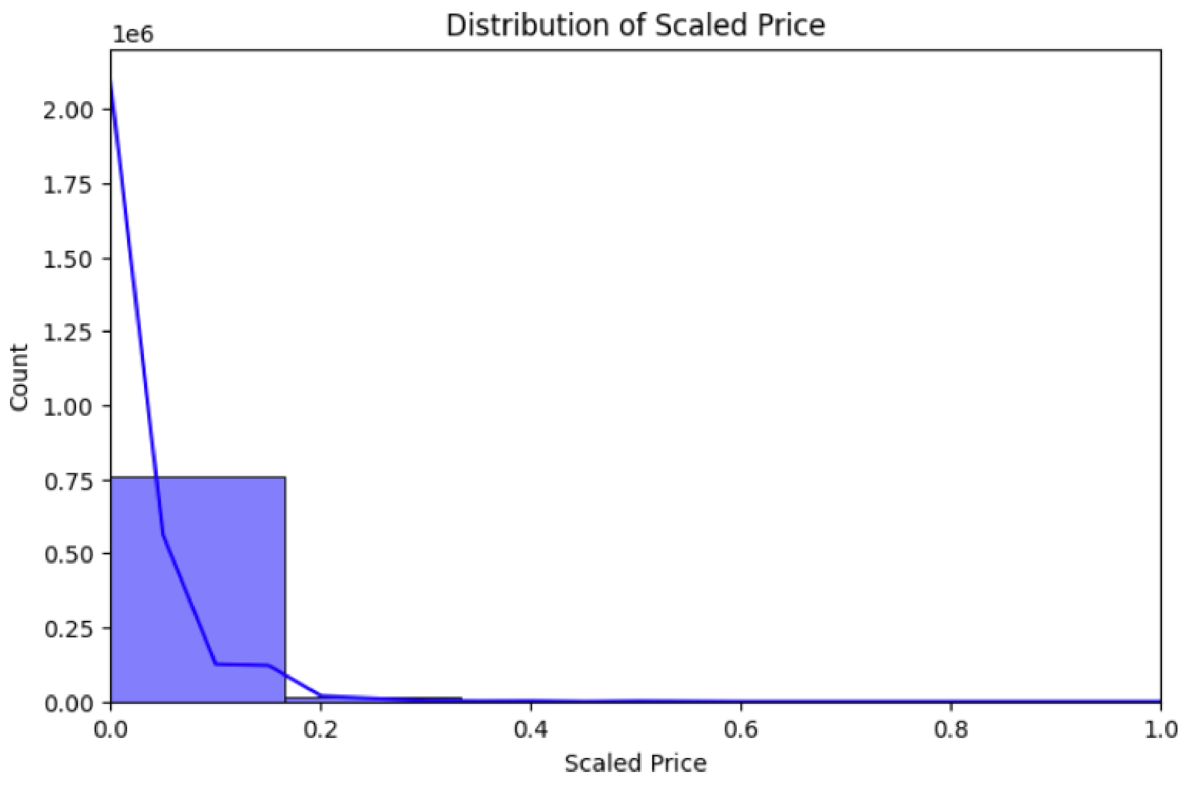
Data Visualization

1. **Exploratory data analysis:**

```
#displaying statistics for numerical columns
merged_df.describe()
```

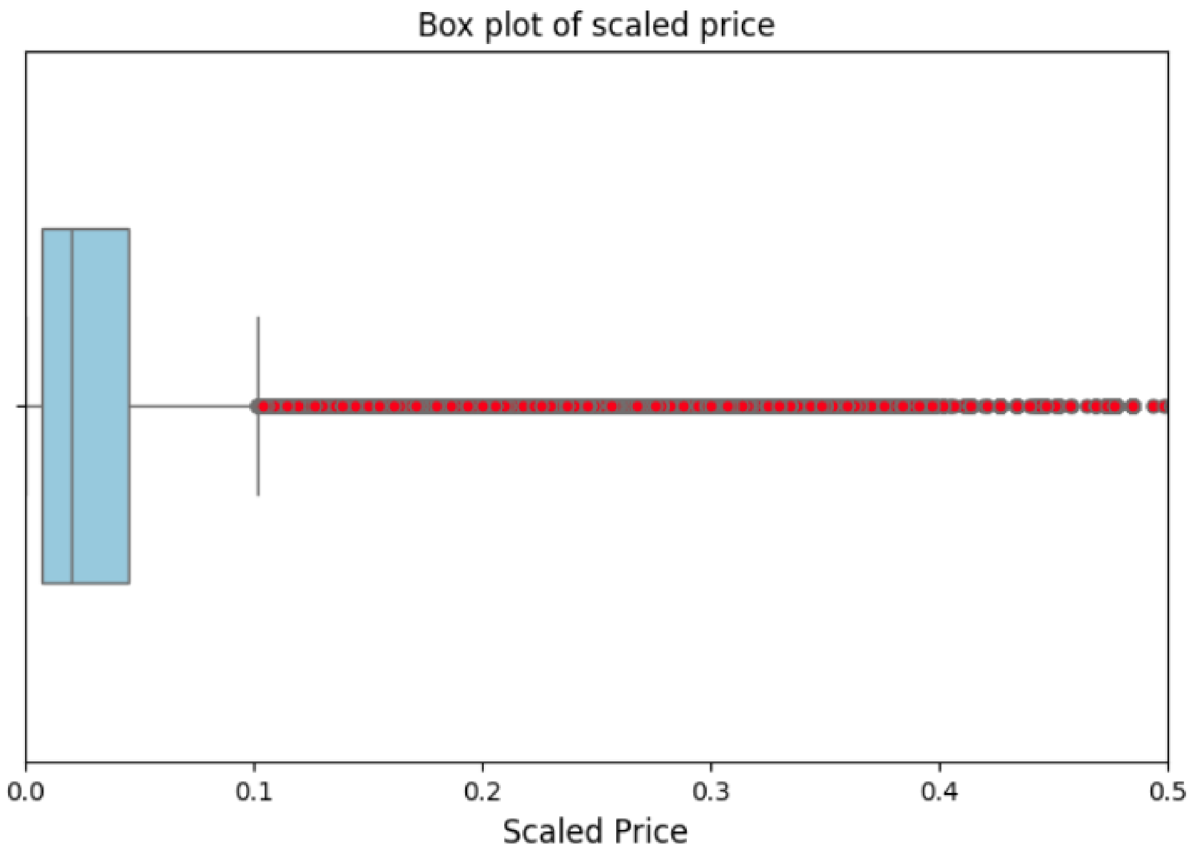
	product_id	category_id	price	user_id	price_scaled
count	7.755380e+05	7.755380e+05	775538.000000	7.755380e+05	775538.000000
mean	3.665912e+06	2.111627e+18	233.445452	9.508108e+17	0.036042
std	6.288530e+06	4.382351e+16	344.099135	7.330133e+17	0.053125
min	8.540000e+02	2.053014e+18	0.000000	2.405221e+08	0.000000
25%	1.004249e+06	2.053014e+18	47.410000	5.458615e+08	0.007320
50%	1.695100e+06	2.144416e+18	128.680000	1.515916e+18	0.019867
75%	3.977213e+06	2.144416e+18	292.830000	1.515916e+18	0.045210
max	6.050001e+07	2.227847e+18	64771.060000	1.515916e+18	10.000000

2. **Histogram to show the distribution of the 'price' column:**



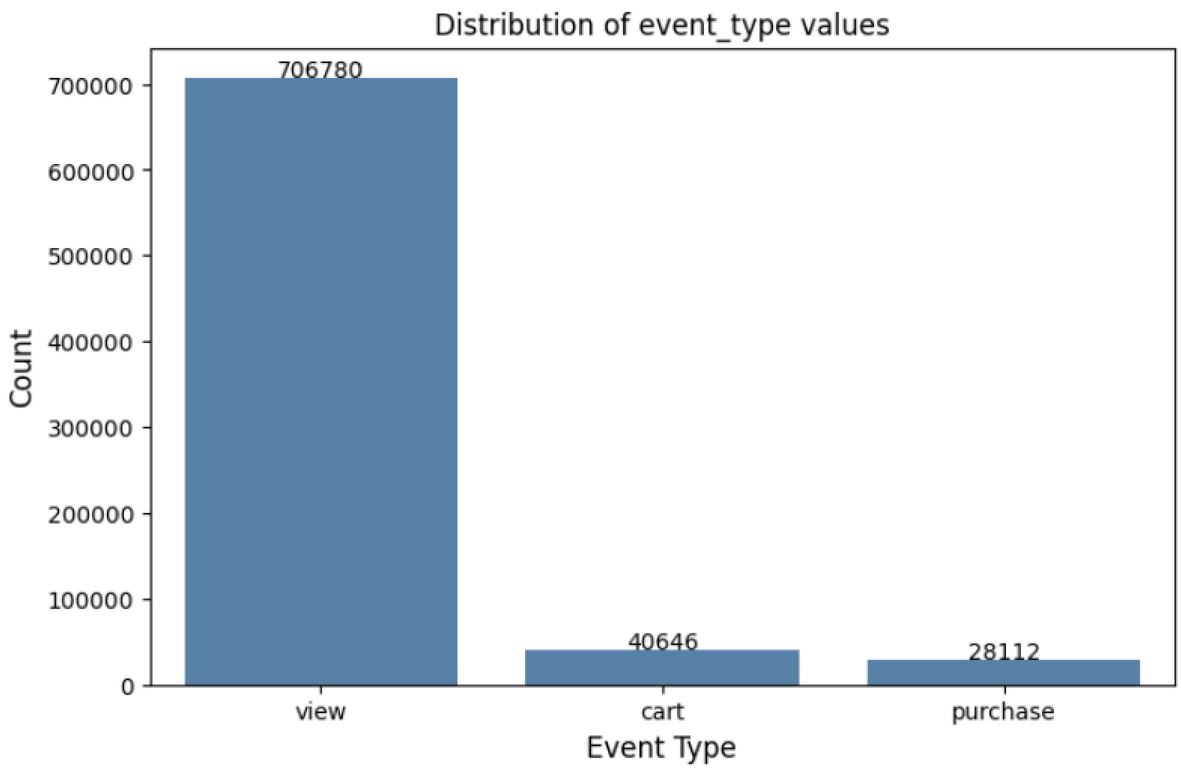
The above graph displays the distribution of the ‘scaled_price’ column, from the graph it is evident that the data follows non-normal distribution which is why IQR method was used to determine outliers in the data.

3. **Box plot to detect outliers in the data:**



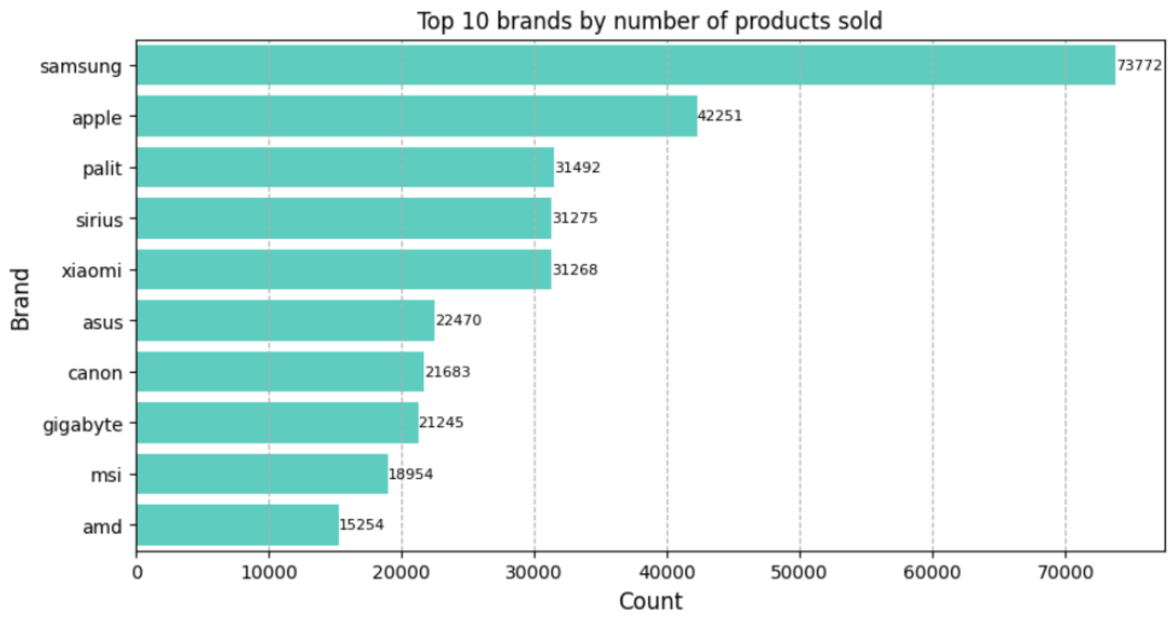
The above box plot displays the range of values in the scaled price column, it’s median and the outliers are denoted by the red dots. From this graph it is visible that the ‘price’ column is highly skewed.

4. **Distribution of values in the 'event_type' column:**



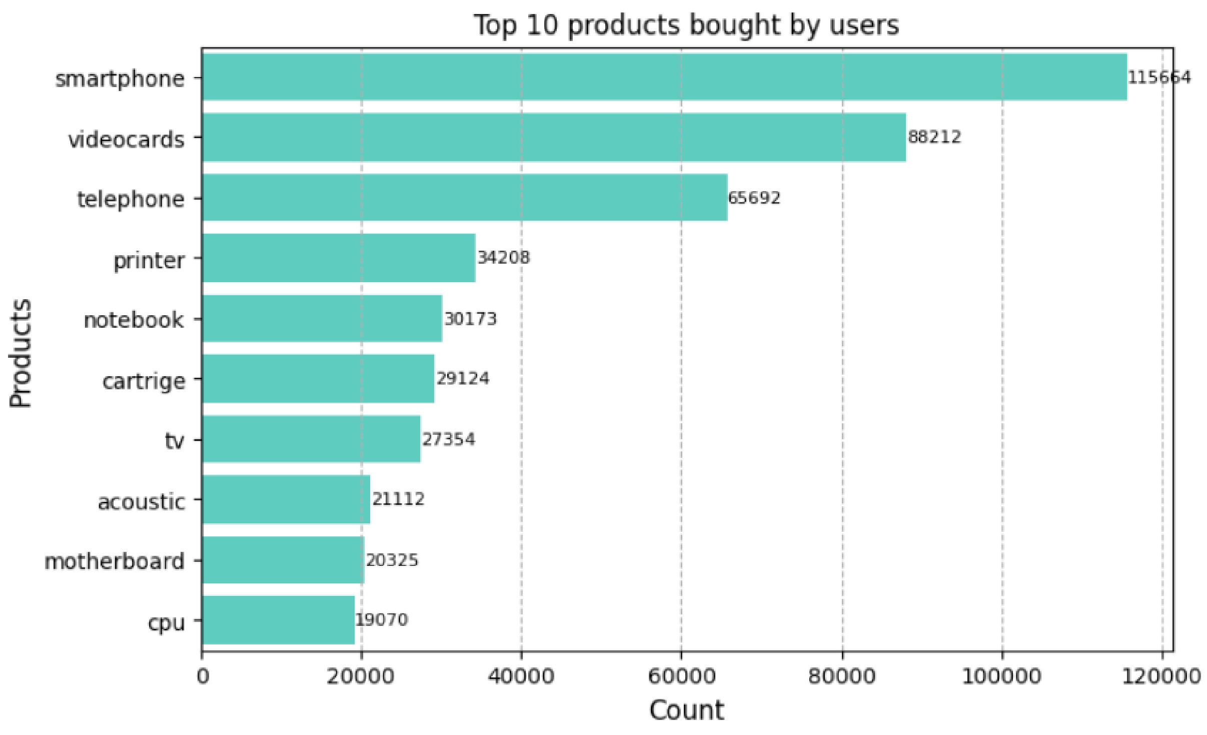
The above bar chart describes the distribution of values in the 'event_type' column. There are three unique values 'view', 'cart' and 'purchase' which describe the interactions the user had with the product on the ecommerce application/website. It is evident from the graph that the data is highly biased towards the 'view' value.

5. **Top 10 brands by number of products sold:**



The above horizontal bar chart denotes the top 10 brands whose products were bought/viewed by the users. From the data, we can conclude that the brand ‘samsung’ product were the most bought/viewed by the users, which will help the recommendation system while providing recommendations to the new users.

6. **Top 10 products bought by users:**



The above horizontal bar chart denotes the top 10 products that users by from the ecommerce website/application.