In [1]:
```
# This Python 3 environment comes with many helpful analytics librarie:
 installed
# It is defined by the kaggle/python docker image: https://github.com/ka
ggle/docker-python
# For example, here's several helpful packages to load in

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) wi
ll list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

sns.set(rc={"font.style":"normal",
            "axes.grid":False,
            'figure.figsize':(10.0,10.0)})

import nltk
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from wordcloud import WordCloud

stop_words = set(stopwords.words('english'))

from sklearn.cluster import KMeans
# Any results you write to the current directory are saved as output.
```

```
/kaggle/input/online-retail-customer-clustering/OnlineRetail.csv
```

EDA

In [2]:
```
retail_df = pd.read_csv('/kaggle/input/online-retail-customer-clusterin
g/OnlineRetail.csv',encoding='ISO_8859-1')
retail_df.head()
```

Out[2]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	01-12-2010 08:26	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	01-12-2010 08:26	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	01-12-2010 08:26	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	01-12-2010 08:26	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	01-12-2010 08:26	3.39	17850.0	United Kingdom

In [3]:
```
retail_df.CustomerID = retail_df.CustomerID.astype(object)
retail_df.InvoiceDate = pd.to_datetime(retail_df.InvoiceDate,format='%d-
%m-%Y %H:%M')
```

In [4]:
```
retail_df.shape
# 541909 rows, 8 columns
```

Out[4]:
```
(541909, 8)
```

In [5]:
```
retail_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 541909 entries, 0 to 541908
Data columns (total 8 columns):
InvoiceNo      541909 non-null object
StockCode      541909 non-null object
Description    540455 non-null object
Quantity       541909 non-null int64
InvoiceDate    541909 non-null datetime64[ns]
UnitPrice      541909 non-null float64
CustomerID     406829 non-null object
Country        541909 non-null object
dtypes: datetime64[ns](1), float64(1), int64(1), object(5)
memory usage: 33.1+ MB
```

In [6]:
```
# Drop rows containing missing values
retail_df = retail_df.dropna()

#Drop rows containing negative values
idx_qty = retail_df[retail_df.Quantity < 0].index
retail_df.drop(idx_qty,axis=0,inplace=True)

idx_price = retail_df[retail_df.UnitPrice < 0].index
retail_df.drop(idx_price,axis=0,inplace=True)

# We do have a few outliers, so we can drop them safely for now.
idx_outliers = retail_df[retail_df['Quantity'] > 5000].index
retail_df.drop(idx_outliers,axis=0,inplace=True)
```

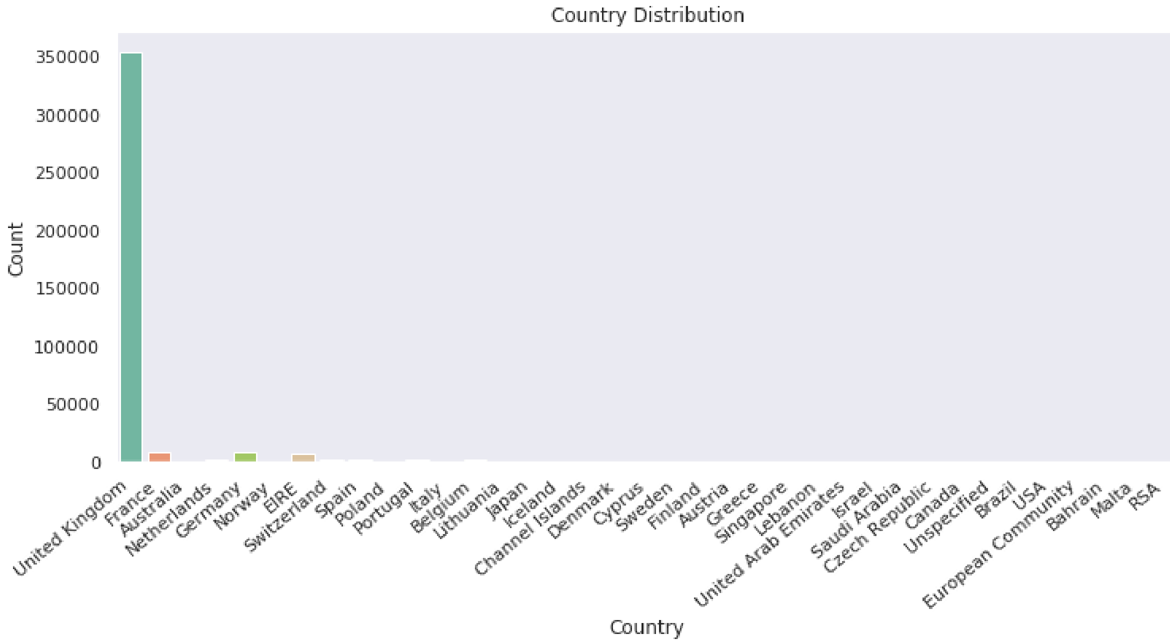
In [7]:
```
retail_df.describe()
```

Out[7]:

	Quantity	UnitPrice
count	397921.000000	397921.00000
mean	12.600355	3.11619
std	42.889024	22.09687
min	1.000000	0.00000
25%	2.000000	1.25000
50%	6.000000	1.95000
75%	12.000000	3.75000
max	4800.000000	8142.75000

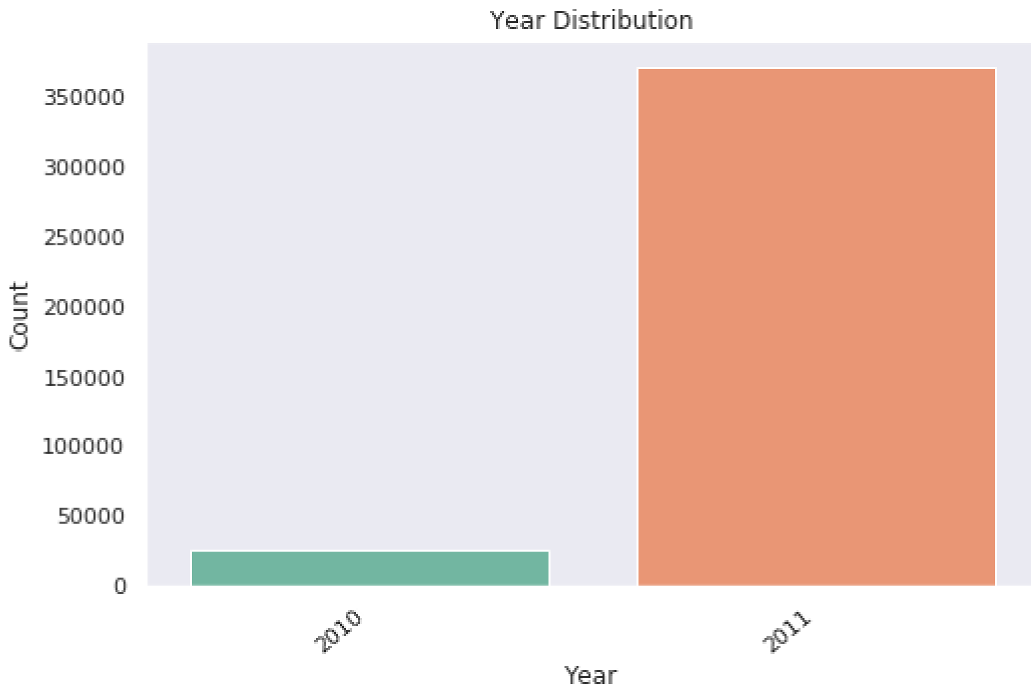
Visualization

In [8]:
```
plt.figure(figsize=(12,5))
sns.countplot(retail_df['Country'],palette= 'Set2')
plt.xticks(rotation=40,ha='right')
plt.title("Country Distribution")
plt.xlabel('Country')
plt.ylabel('Count');
```



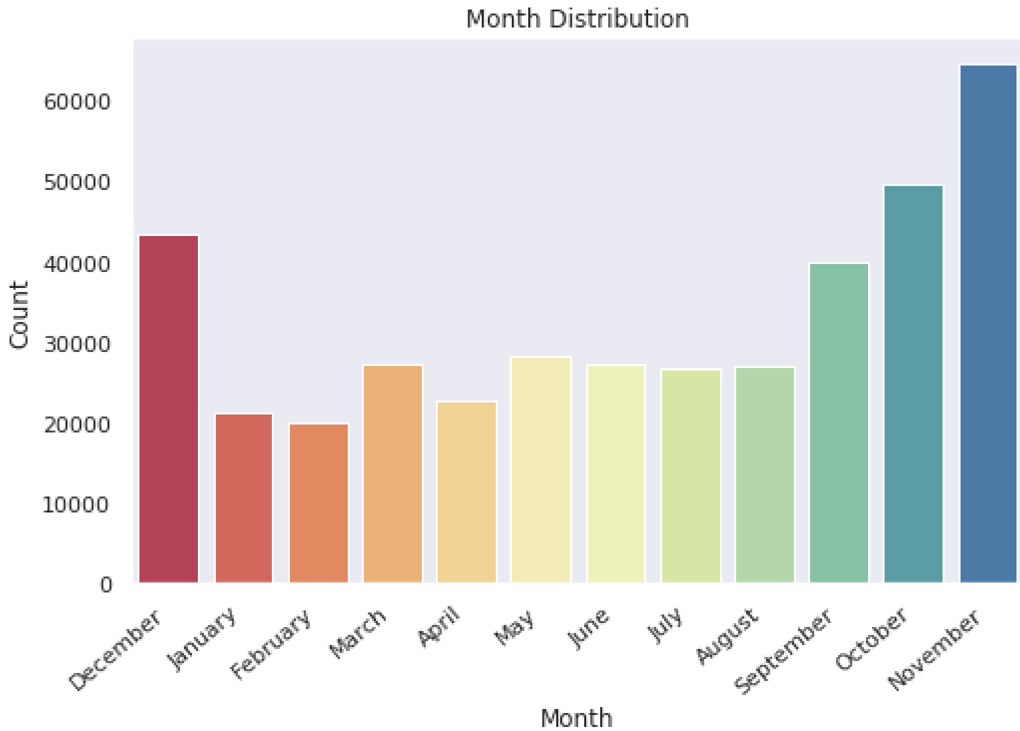
From the above bar chart, we can clearly see that UK has the highest number of customers.

In [9]:
```
plt.figure(figsize=(8,5))
sns.countplot(retail_df['InvoiceDate'].dt.year,palette= 'Set2')
plt.xticks(rotation=40,ha='right')
plt.title("Year Distribution")
plt.xlabel('Year')
plt.ylabel('Count');
```



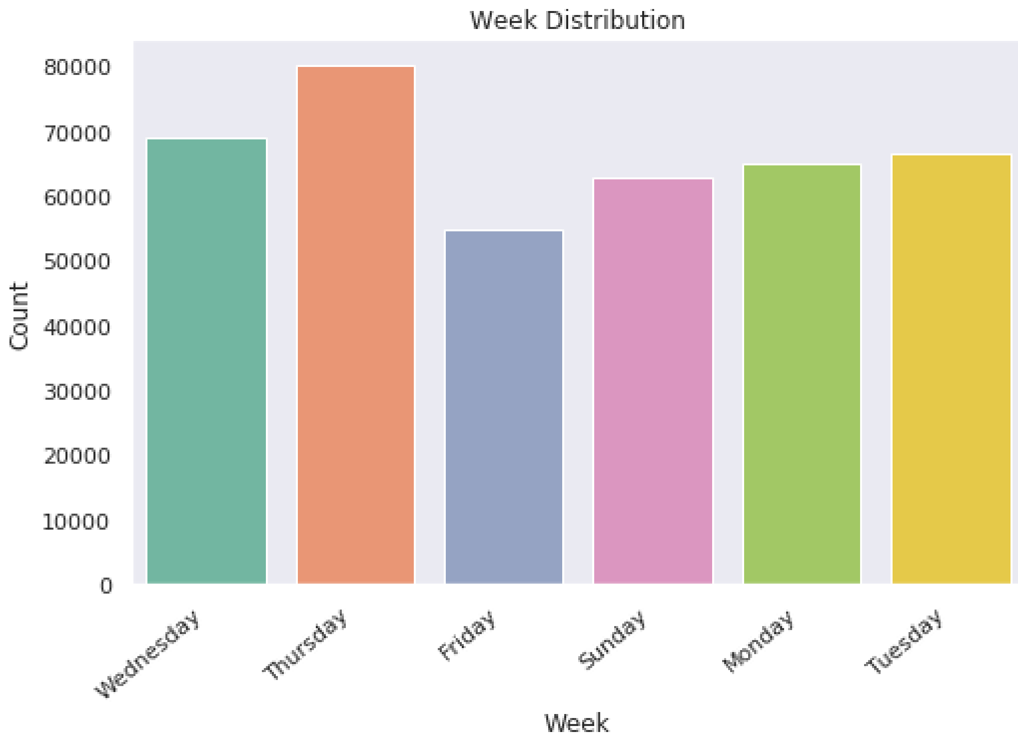
The above bar plot shows that most of the transactions occurred in 2011.

In [10]:
```
plt.figure(figsize=(8,5))
sns.countplot(retail_df['InvoiceDate'].dt.month_name(),palette= 'Spectra
l')
plt.xticks(rotation=40,ha='right')
plt.title("Month Distribution")
plt.xlabel('Month')
plt.ylabel('Count');
```



The above bar plot shows that transaction volume increase towards the end of a year, peaking in November (early Thanksgiving / Christmas purchases probably).

In [11]:
```
plt.figure(figsize=(8,5))
sns.countplot(retail_df['InvoiceDate'].dt.day_name(),palette= 'Set2')
plt.xticks(rotation=40,ha='right')
plt.title("Week Distribution")
plt.xlabel('Week')
plt.ylabel('Count');
```



The above bar plot indicates higher transaction volumes on Thursdays and the lowest on Fridays.

Saturday is not shown as no transaction occured.

Next, I'm creating a word cloud to demonstrate the most frequently purchased items and the least purchased.

wordcloud() generates the Word Cloud using seaborn library.

tokenize() validates each word in the text using validate_word() and word_tokenize() and returns them.

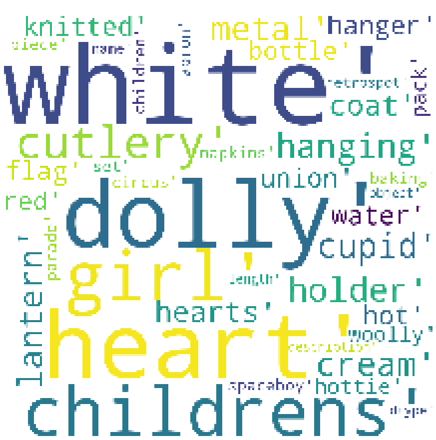
In [12]:
```
def wordcloud(text,my_mask=None):
    wc = WordCloud(width=1000,height=1000,max_words=1000,collocations=False,
    min_font_size=10,contour_width=2, mask=my_mask,background_color='white').generate(text)
    plt.imshow(wc)
    plt.axis('off')
    plt.show()

def tokenize(text):
    token = word_tokenize(text)
    word_token = []
    for word in token:
        if validate_word(word, stop_words) :
            word_token.append(str(word))
    return(str(word_token))

def validate_word(word, stop_words):
    if word not in stop_words and word.isalpha():
        return True
    return False
```

In [13]:
```
text = tokenize(str(retail_df['Description']).lower())

wordcloud(text)
```



From the above word cloud, we can see that `white` is the most frequently used descriptor and `object` is the least frequently used descriptor.

In [14]:
```
# Temporary dataframes that will be merged.
temp_df1 = pd.DataFrame()
temp_df2 = pd.DataFrame()
temp_df3 = pd.DataFrame()
```

The `Total_Amount` is the total spent by the customer.

The `Transaction_Count` is the number of transactions made by the customer.

The `Latest_Transaction` is used to store how recently the customer made the last transaction (in terms of days).

In [15]:
```
# First, we calculate the transaction amount.
retail_df['Transaction_Amount'] = retail_df['Quantity'] * retail_df['UnitPrice']

# Also we store the sum of amounts for all transactions conducted by each customer.
temp_df1['Transaction_Amount'] = retail_df.groupby('CustomerID')['Transaction_Amount'].sum()
```

In [16]:
```
# Second, we count the number of invoices generated for each customer and rename the column to Transaction_Count.
temp_df2 = retail_df.groupby('CustomerID')['InvoiceNo'].count()
temp_df2 = temp_df2.reset_index()
temp_df2.columns = ['CustomerID','Transaction_Count']
temp_df2.head()
```

Out[16]:

	CustomerID	Transaction_Count
0	12347.0	182
1	12348.0	31
2	12349.0	73
3	12350.0	17
4	12352.0	85

In [17]:
```
# Merge the above to DFs on CustomerID.
df = pd.merge(temp_df1, temp_df2, on='CustomerID', how='inner')
df.head()
```

Out[17]:

	CustomerID	Transaction_Amount	Transaction_Count
0	12347.0	4310.00	182
1	12348.0	1797.24	31
2	12349.0	1757.55	73
3	12350.0	334.40	17
4	12352.0	2506.04	85

In [18]:
```
# Then, we retrieve the latest invoice date and calculate the number o:
 days since then to the present day.

# Most recent invoice
latest_transaction = retail_df['InvoiceDate'].max()
# Updating retail_df
retail_df['Latest_Transaction'] = latest_transaction - retail_df['Invoic
eDate']

temp_df3 = retail_df.groupby('CustomerID')['Latest_Transaction'].min()
temp_df3 = temp_df3.reset_index()
temp_df3['Latest_Transaction'] = temp_df3['Latest_Transaction'].dt.days
```

In [19]:
```
# Merge the above DFs based on CustomerID again.
df = pd.merge(df, temp_df3, on='CustomerID', how='inner')
df.columns = ['CustomerID', 'Total_Amount', 'Transaction_Count', 'Latest
_Transaction']
df.head()
```

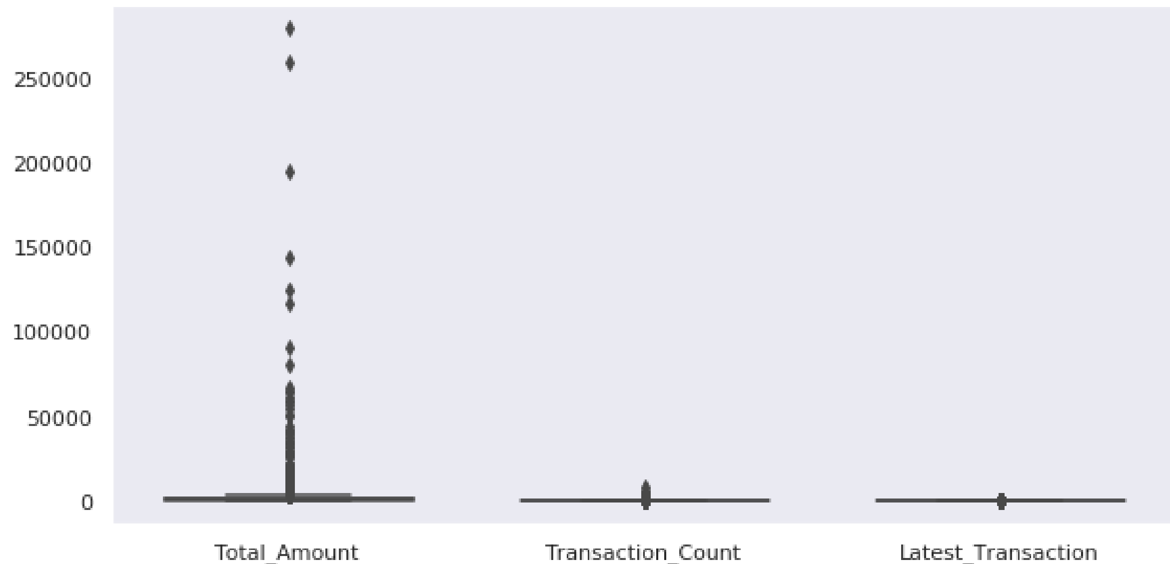
Out[19]:

	CustomerID	Total_Amount	Transaction_Count	Latest_Transaction
0	12347.0	4310.00	182	1
1	12348.0	1797.24	31	74
2	12349.0	1757.55	73	18
3	12350.0	334.40	17	309
4	12352.0	2506.04	85	35

The box plot below shows us that the `Total_Amount` values for the customers are widely spread. On the other hand, `Transaction_Count` (transaction volume) and `Latest_Transaction` (Days since most recent purchase date) values are extremely close.

In [20]:
```
plt.figure(figsize=(10,5))
sns.boxplot(data = df[['Total_Amount','Transaction_Count','Latest_Transa
ction']],orient="v", palette="Set1" ,whis=1.5,saturation=1, width=0.7)
```

Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x7f46b9f6b828>



In [21]:
```
# Just getting a peek into the details here.
df.describe()
```

Out[21]:

	CustomerID	Total_Amount	Transaction_Count	Latest_Transaction
count	4337.000000	4337.000000	4337.000000	4337.000000
mean	15301.089232	1998.098848	91.750288	91.529859
std	1721.422291	8551.826038	228.837406	99.968030
min	12347.000000	2.900000	1.000000	0.000000
25%	13814.000000	307.090000	17.000000	17.000000
50%	15300.000000	673.260000	41.000000	50.000000
75%	16779.000000	1661.330000	100.000000	141.000000
max	18287.000000	280206.020000	7847.000000	373.000000

In [22]:
```
Q1 = df.Total_Amount.quantile(0.05)
Q3 = df.Total_Amount.quantile(0.95)

IQR = Q3 - Q1
df = df[ (df['Total_Amount']  >= Q1 - 1.5 * IQR) & (df['Total_Amount'] <
= Q3 + 1.5 * IQR)]

Q1 = df.Transaction_Count.quantile(0.05)
Q3 = df.Transaction_Count.quantile(0.95)

IQR = Q3 - Q1
df = df[ (df['Transaction_Count']  >= Q1 - 1.5 * IQR) & (df['Transaction
_Count'] <= Q3 + 1.5 * IQR)]

Q1 = df.Latest_Transaction.quantile(0.05)
Q3 = df.Latest_Transaction.quantile(0.95)

IQR = Q3 - Q1
df = df[ (df['Latest_Transaction']  >= Q1 - 1.5 * IQR) & (df['Latest_Tra
nsaction'] <= Q3 + 1.5 * IQR)]

df
```

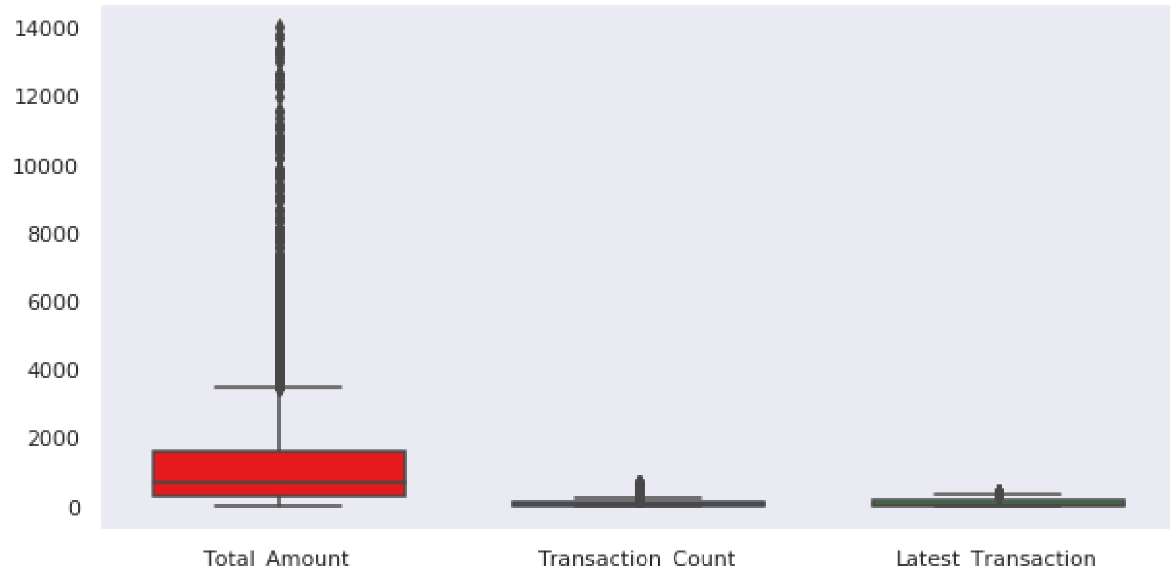
Out[22]:

	CustomerID	Total_Amount	Transaction_Count	Latest_Transaction
0	12347.0	4310.00	182	1
1	12348.0	1797.24	31	74
2	12349.0	1757.55	73	18
3	12350.0	334.40	17	309
4	12352.0	2506.04	85	35
...	...	...	...	...
4331	18278.0	173.90	9	73
4332	18280.0	180.60	10	277
4333	18281.0	80.82	7	180
4334	18282.0	178.05	12	7
4336	18287.0	1837.28	70	42

4256 rows × 4 columns

In [23]:
```
plt.figure(figsize=(10,5))
sns.boxplot(data = df[['Total_Amount','Transaction_Count','Latest_Transa
ction']],orient="v", palette="Set1" ,whis=1.5,saturation=1, width=0.7)
```

Out[23]: <matplotlib.axes._subplots.AxesSubplot at 0x7f46b9ca5fd0>



Preprocessing & Clustering using K-Means algorithm

In [24]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaled = scaler.fit_transform(df[['Total_Amount','Transaction_Count','La
test_Transaction']])
df_scaled = pd.DataFrame(scaled,columns=['Total_Amount','Transaction_Cou
nt','Latest_Transaction'])
df_scaled.head()
```

Out[24]:

	Total_Amount	Transaction_Count	Latest_Transaction
0	1.657337	1.067766	-0.918417
1	0.265403	-0.460499	-0.189896
2	0.243417	-0.035419	-0.748762
3	-0.544932	-0.602193	2.155342
4	0.658040	0.086033	-0.579106

In [25]:
```
# Running K-Means for multiple number of clusters to see which gives us
 the least error.
SSE = []

for cluster in range(2,8):
    kmeans = KMeans(n_clusters=cluster,random_state=42)
    kmeans.fit(df_scaled)
    centroids = kmeans.cluster_centers_
    pred_clusters = kmeans.predict(df_scaled)
    SSE.append(kmeans.inertia_)

frame = pd.DataFrame({'Cluster':range(2,8) , 'SSE':SSE})
frame
```

Out[25]:

	Cluster	SSE
0	2	7622.643433
1	3	4427.719029
2	4	3363.937855
3	5	2789.316136
4	6	2391.580656
5	7	2040.103609

Here, we plot the results:

In [26]:
```
plt.figure(figsize=(5,5))
plt.plot(frame['Cluster'],frame['SSE'],marker='o')
plt.title('Custers Vs SSE')
plt.xlabel('No of Clusters')
plt.ylabel('Intertia')
plt.show()
```

As we can see, the number of clusters can be anything betwen 3 and 10 as the optimal cluster value.

I'm choosing 3.

Training & predicting using K-Means:

In [27]:
```
kmeans = KMeans(n_clusters=3,random_state=42)
kmeans.fit(df_scaled)
pred = kmeans.predict(df_scaled)
```

In [28]:
```
df['Cluster'] = kmeans.labels_
df['Cluster'].value_counts()
```

Out[28]:
```
2    2708
0    1053
1     495
Name: Cluster, dtype: int64
```

In [29]:
```
df.head()
```

Out[29]:

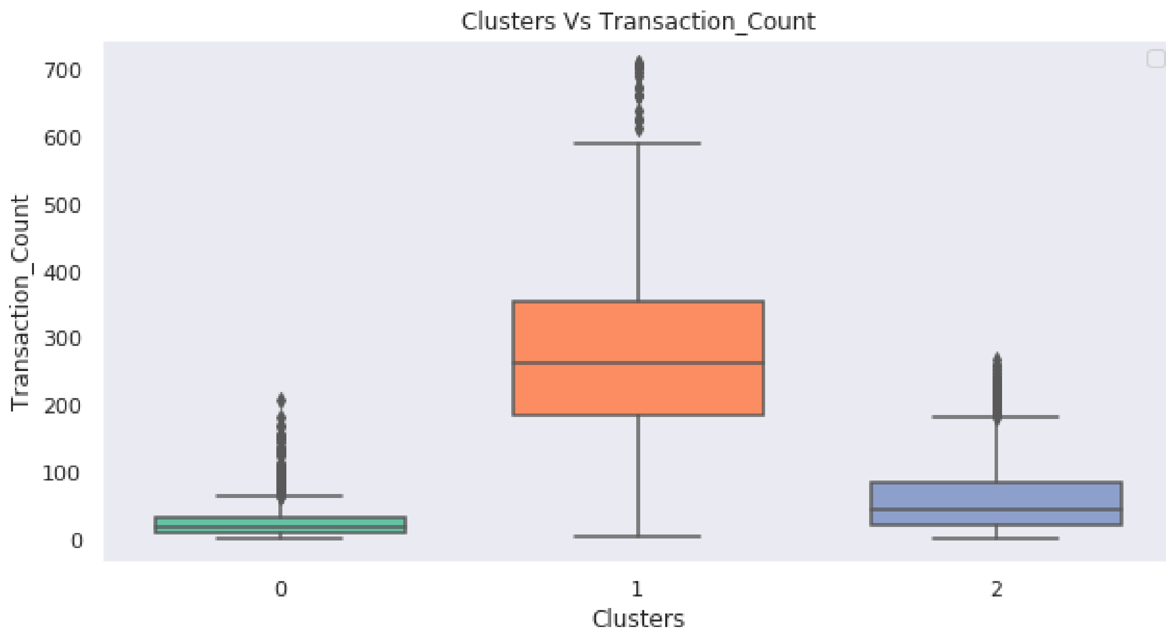
	CustomerID	Total_Amount	Transaction_Count	Latest_Transaction	Cluster
0	12347.0	4310.00	182	1	1
1	12348.0	1797.24	31	74	2
2	12349.0	1757.55	73	18	2
3	12350.0	334.40	17	309	0
4	12352.0	2506.04	85	35	2

The below boxplot shows that customers in Cluster 1 spend more money followed by customers in Cluster 2, followed by customers in Cluster 0.

In [30]:
```
plt.figure(figsize=(10,5))
sns.boxplot(x = df['Cluster'] ,y = df['Total_Amount'],orient="v", palett
e="Set2" ,whis=1.5,saturation=1, width=0.7)
plt.title("Clusters Vs Total_Amount")
plt.xlabel("Clusters")
plt.ylabel("Total_Amount")
plt.legend();
```

The below boxplot shows that customers in Cluster 1 purchase more frequently followed by customers in Cluster 2, followed by customers in Cluster 0.

In [31]:
```
plt.figure(figsize=(10,5))
sns.boxplot(x = df['Cluster'] ,y = df['Transaction_Count'],orient="v", p
alette="Set2" ,whis=1.5,saturation=1, width=0.7)
plt.title("Clusters Vs Transaction_Count")
plt.xlabel("Clusters")
plt.ylabel("Transaction_Count")
plt.legend();
```

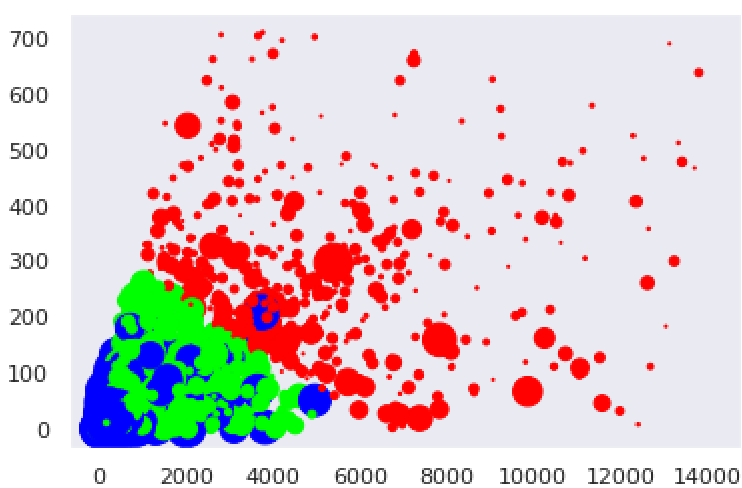


The below boxplot shows that customers in Cluster 1 had made recent transactions followed by customers in Cluster 2, followed by customers in Cluster 0.

In [32]:
```
plt.figure(figsize=(10,5))
sns.boxplot(x = df['Cluster'] ,y = df['Latest_Transaction'],orient="v",
palette="Set2" ,whis=1.5,saturation=1, width=0.7)
plt.title("Clusters Vs Latest_Transaction")
plt.xlabel("Clusters")
plt.ylabel("Latest_Transaction")
plt.legend();
```

Thus, we can conclude that Cluster 1 customers have recently had higher transaction volume and also spent higher amount.

In [33]:
```
plt.scatter(df['Total_Amount'],df['Transaction_Count'],df['Latest_Trans
ction'],
                    c=kmeans.labels_, cmap='brg');
```



In [ ]: