Importing Necessary Libraries

In [46]:
```
import warnings
warnings.filterwarnings('ignore')

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import plotly.graph_objects as go
from matplotlib.colors import LinearSegmentedColormap
from matplotlib import colors as mcolors
from scipy.stats import linregress
from sklearn.ensemble import IsolationForest
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from yellowbrick.cluster import KElbowVisualizer, SilhouetteVisualizer
from sklearn.metrics import silhouette_score, calinski_harabasz_score, davie
from sklearn.cluster import KMeans
from tabulate import tabulate
from collections import Counter

%matplotlib inline
```

In [ ]:
```
from plotly.offline import init_notebook_mode
init_notebook_mode(connected=True)
```

In [ ]:
```
sns.set(rc={'axes.facecolor': '#fcf0dc'}, style='darkgrid')
```

Loading the Dataset

In [ ]:
```
df = pd.read_csv('/content/archive (8).zip', encoding="ISO-8859-1")
```

Dataset Overview

In [ ]:
```
df.head(10)
```

Out[ ]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	Custo
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/1/2010 8:26	2.55	1
1	536365	71053	WHITE METAL LANTERN	6	12/1/2010 8:26	3.39	1
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	12/1/2010 8:26	2.75	1
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	12/1/2010 8:26	3.39	1
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	12/1/2010 8:26	3.39	1
5	536365	22752	SET 7 BABUSHKA NESTING BOXES	2	12/1/2010 8:26	7.65	1
6	536365	21730	GLASS STAR FROSTED T-LIGHT HOLDER	6	12/1/2010 8:26	4.25	1
7	536366	22633	HAND WARMER UNION JACK	6	12/1/2010 8:28	1.85	1
8	536366	22632	HAND WARMER RED POLKA DOT	6	12/1/2010 8:28	1.85	1
9	536367	84879	ASSORTED COLOUR BIRD ORNAMENT	32	12/1/2010 8:34	1.69	1

In [ ]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 541909 entries, 0 to 541908
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype  
---  ------       --------------   -----  
 0   InvoiceNo    541909 non-null  object 
 1   StockCode    541909 non-null  object 
 2   Description  540455 non-null  object 
 3   Quantity     541909 non-null  int64  
 4   InvoiceDate  541909 non-null  object 
 5   UnitPrice    541909 non-null  float64
 6   CustomerID   406829 non-null  float64
 7   Country      541909 non-null  object 
dtypes: float64(2), int64(1), object(5)
memory usage: 33.1+ MB
```

Summary Statistics

In [ ]:
```
df.describe().T
```

Out[ ]:

	count	mean	std	min	25%	50%
Quantity	541909.0	9.552250	218.081158	-80995.00	1.00	3.0
UnitPrice	541909.0	4.611114	96.759853	-11062.06	1.25	2.0
CustomerID	406829.0	15287.690570	1713.600303	12346.00	13953.00	15152.0

In [ ]:
```
df.describe(include='object').T
```

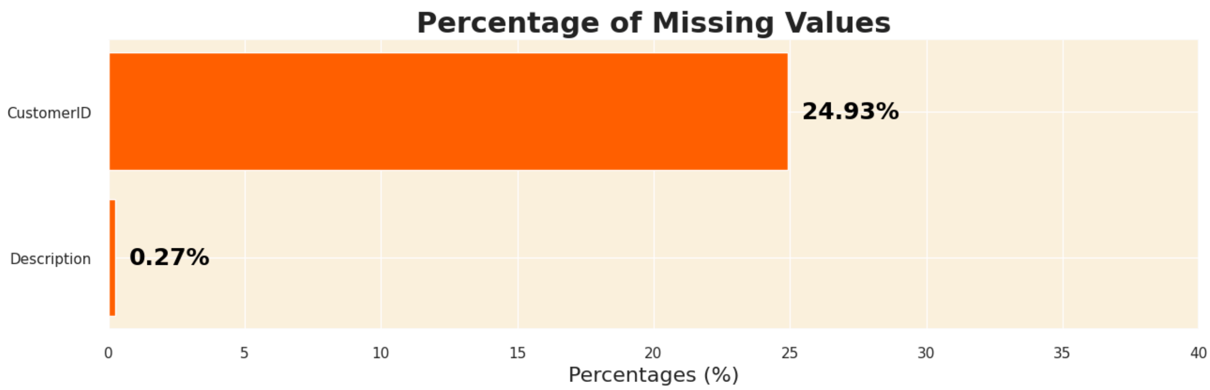
Out[ ]:

	count	unique	top	freq
InvoiceNo	541909	25900	573585	1114
StockCode	541909	4070	85123A	2313
Description	540455	4223	WHITE HANGING HEART T-LIGHT HOLDER	2369
InvoiceDate	541909	23260	10/31/2011 14:41	1114
Country	541909	38	United Kingdom	495478

Handling Missing Values

In [ ]:
```
missing_data = df.isnull().sum()
missing_percentage = (missing_data[missing_data > 0] / df.shape[0]) * 100

# Prepare values
missing_percentage.sort_values(ascending=True, inplace=True)

# Plot the barh chart
fig, ax = plt.subplots(figsize=(15, 4))
ax.barh(missing_percentage.index, missing_percentage, color='#ff6200')
```

```python
# Annotate the values and indexes
for i, (value, name) in enumerate(zip(missing_percentage, missing_percentage
    ax.text(value+0.5, i, f"{value:.2f}%", ha='left', va='center', fontweigh

# Set x-axis limit
ax.set_xlim([0, 40])

# Add title and xlabel
plt.title("Percentage of Missing Values", fontweight='bold', fontsize=22)
plt.xlabel('Percentages (%)', fontsize=16)
plt.show()
```



In [ ]:
```python
df[df['CustomerID'].isnull() | df['Description'].isnull()].head()
```

Out[ ]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	Cu
622	536414	22139	NaN	56	12/1/2010 11:52	0.00	
1443	536544	21773	DECORATIVE ROSE BATHROOM BOTTLE	1	12/1/2010 14:32	2.51	
1444	536544	21774	DECORATIVE CATS BATHROOM BOTTLE	2	12/1/2010 14:32	2.51	
1445	536544	21786	POLKADOT RAIN HAT	4	12/1/2010 14:32	0.85	
1446	536544	21787	RAIN PONCHO RETROSPOT	2	12/1/2010 14:32	1.66	

In [ ]:
```python
df = df.dropna(subset=['CustomerID', 'Description'])
```

In [ ]:
```python
df.isnull().sum().sum()
```

Out[ ]: 0

Handling Duplicates

In [ ]:
```python
duplicate_rows = df[df.duplicated(keep=False)]

# Sorting the data by certain columns to see the duplicate rows next to each
duplicate_rows_sorted = duplicate_rows.sort_values(by=['InvoiceNo', 'StockCo

# Displaying the first 10 records
duplicate_rows_sorted.head(10)
```

Out[ ]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	Cu:
494	536409	21866	UNION JACK FLAG LUGGAGE TAG	1	12/1/2010 11:45	1.25	
517	536409	21866	UNION JACK FLAG LUGGAGE TAG	1	12/1/2010 11:45	1.25	
485	536409	22111	SCOTTIE DOG HOT WATER BOTTLE	1	12/1/2010 11:45	4.95	
539	536409	22111	SCOTTIE DOG HOT WATER BOTTLE	1	12/1/2010 11:45	4.95	
489	536409	22866	HAND WARMER SCOTTY DOG DESIGN	1	12/1/2010 11:45	2.10	
527	536409	22866	HAND WARMER SCOTTY DOG DESIGN	1	12/1/2010 11:45	2.10	
521	536409	22900	SET 2 TEA TOWELS I LOVE LONDON	1	12/1/2010 11:45	2.95	
537	536409	22900	SET 2 TEA TOWELS I LOVE LONDON	1	12/1/2010 11:45	2.95	
578	536412	21448	12 DAISY PEGS IN WOOD BOX	1	12/1/2010 11:49	1.65	
598	536412	21448	12 DAISY PEGS IN WOOD BOX	1	12/1/2010 11:49	1.65	

In [ ]:
```
print(f"The dataset contains {df.duplicated().sum()} duplicate rows that nee

# Removing duplicate rows
df.drop_duplicates(inplace=True)
```

The dataset contains 5225 duplicate rows that need to be removed.

In [ ]:
```
df.shape[0]
```

Out[ ]: 401604

Treating Cancelled Transactions

In [ ]:
```
df['Transaction_Status'] = np.where(df['InvoiceNo'].astype(str).str.startswi

# Analyze the characteristics of these rows (considering the new column)
cancelled_transactions = df[df['Transaction_Status'] == 'Cancelled']
cancelled_transactions.describe().drop('CustomerID', axis=1)
```

Out[ ]:

	Quantity	UnitPrice
count	8872.000000	8872.000000
mean	-30.774910	18.899512
std	1172.249902	445.190864
min	-80995.000000	0.010000
25%	-6.000000	1.450000
50%	-2.000000	2.950000
75%	-1.000000	4.950000
max	-1.000000	38970.000000

In [ ]:
```
cancelled_percentage = (cancelled_transactions.shape[0] / df.shape[0]) * 100

# Printing the percentage of cancelled transactions
print(f"The percentage of cancelled transactions in the dataset is: {cancell
```

The percentage of cancelled transactions in the dataset is: 2.21%

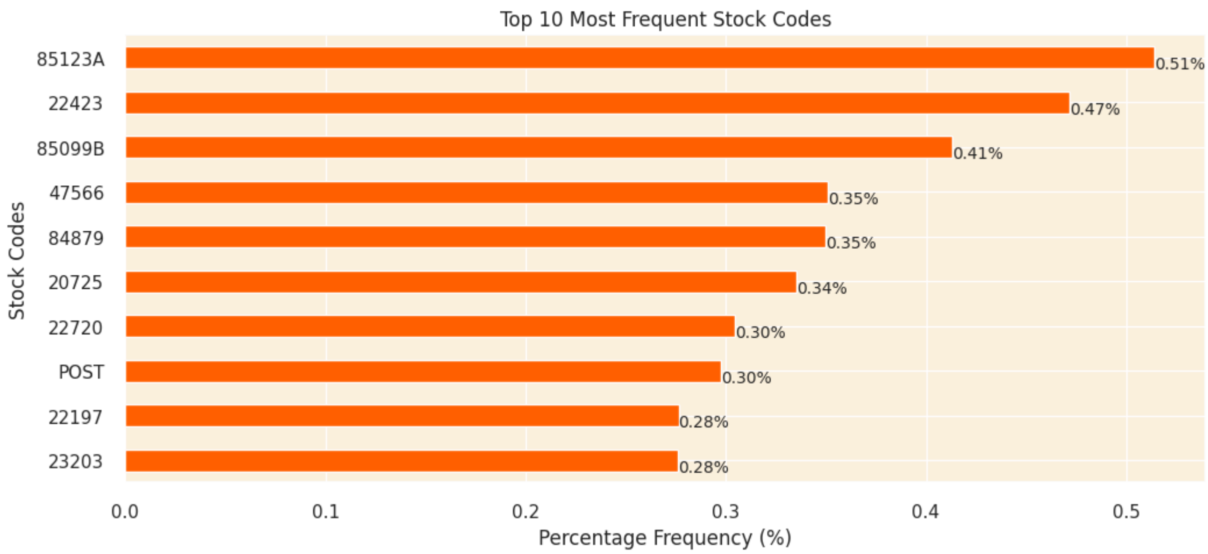
Correcting StockCode Anomalies

In [ ]:
```
unique_stock_codes = df['StockCode'].nunique()

# Printing the number of unique stock codes
print(f"The number of unique stock codes in the dataset is: {unique_stock_cc
```

The number of unique stock codes in the dataset is: 3684

In [ ]:
```
top_10_stock_codes = df['StockCode'].value_counts(normalize=True).head(10) *

# Plotting the top 10 most frequent stock codes
plt.figure(figsize=(12, 5))
```

```
top_10_stock_codes.plot(kind='barh', color='#ff6200')

# Adding the percentage frequency on the bars
for index, value in enumerate(top_10_stock_codes):
    plt.text(value, index+0.25, f'{value:.2f}%', fontsize=10)

plt.title('Top 10 Most Frequent Stock Codes')
plt.xlabel('Percentage Frequency (%)')
plt.ylabel('Stock Codes')
plt.gca().invert_yaxis()
plt.show()
```



In [ ]:
```
unique_stock_codes = df['StockCode'].unique()
numeric_char_counts_in_unique_codes = pd.Series(unique_stock_codes).apply(la

# Printing the value counts for unique stock codes
print("Value counts of numeric character frequencies in unique stock codes:"
print("-"*70)
print(numeric_char_counts_in_unique_codes)
```

```
Value counts of numeric character frequencies in unique stock codes:
----------------------------------------------------------------------
5    3676
0       7
1       1
Name: count, dtype: int64
```

In [ ]:
```
anomalous_stock_codes = [code for code in unique_stock_codes if sum(c.isdigi

# Printing each stock code on a new line
print("Anomalous stock codes:")
print("-"*22)
for code in anomalous_stock_codes:
    print(code)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
Anomalous stock codes:
----------------------
POST
D
C2
M
BANK CHARGES
PADS
DOT
CRUK
```

In [ ]:
```python
percentage_anomalous = (df['StockCode'].isin(anomalous_stock_codes).sum() /

# Printing the percentage
print(f"The percentage of records with anomalous stock codes in the dataset
```

```
The percentage of records with anomalous stock codes in the dataset is: 0.4
8%
```

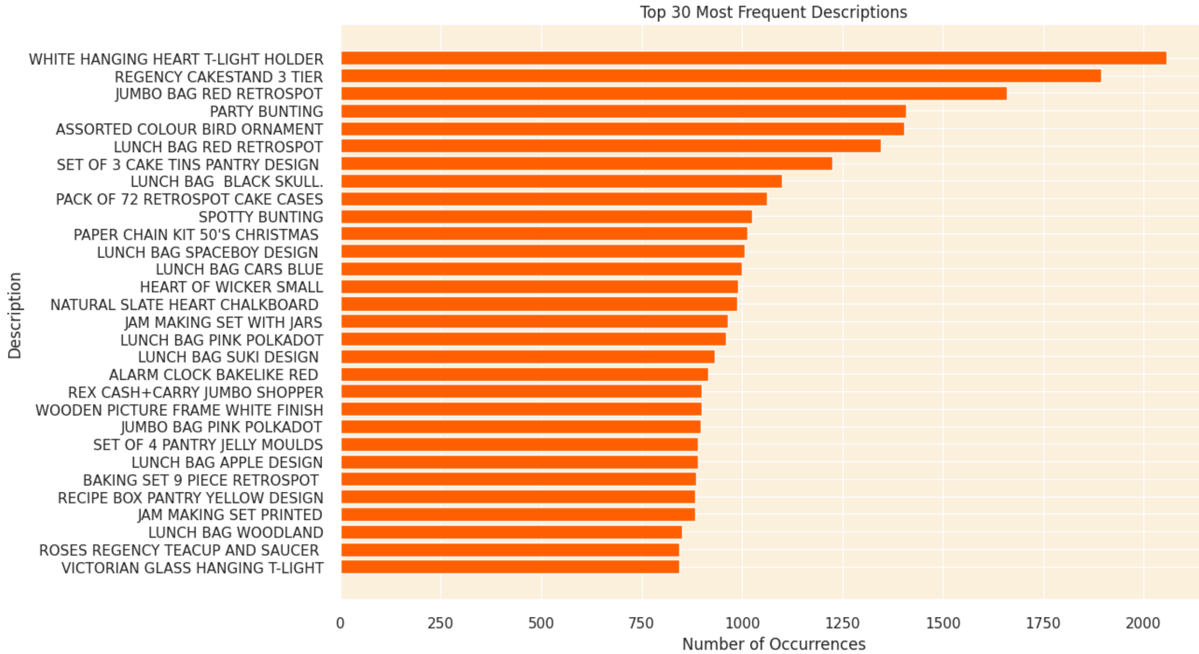
In [ ]:
```python
df = df[~df['StockCode'].isin(anomalous_stock_codes)]
```

In [ ]:
```python
df.shape[0]
```

Out[ ]: 399689

Cleaning Description Column

In [ ]:
```python
description_counts = df['Description'].value_counts()

# Get the top 30 descriptions
top_30_descriptions = description_counts[:30]

# Plotting
plt.figure(figsize=(12,8))
plt.barh(top_30_descriptions.index[::-1], top_30_descriptions.values[::-1],

# Adding labels and title
plt.xlabel('Number of Occurrences')
plt.ylabel('Description')
plt.title('Top 30 Most Frequent Descriptions')

# Show the plot
plt.show()
```



In [ ]:

```
lowercase_descriptions = df['Description'].unique()
lowercase_descriptions = [desc for desc in lowercase_descriptions if any(cha

# Print the unique descriptions containing lowercase characters
print("The unique descriptions containing lowercase characters are:")
print("-"*60)
for desc in lowercase_descriptions:
    print(desc)
```

```
The unique descriptions containing lowercase characters are:
------------------------------------------------------------
BAG 500g SWIRLY MARBLES
POLYESTER FILLER PAD 45x45cm
POLYESTER FILLER PAD 45x30cm
POLYESTER FILLER PAD 40x40cm
FRENCH BLUE METAL DOOR SIGN No
BAG 250g SWIRLY MARBLES
BAG 125g SWIRLY MARBLES
3 TRADITIONAl BISCUIT CUTTERS  SET
NUMBER TILE COTTAGE GARDEN No
FOLK ART GREETING CARD,pack/12
ESSENTIAL BALM 3.5g TIN IN ENVELOPE
POLYESTER FILLER PAD 65CMx65CM
NUMBER TILE VINTAGE FONT No
POLYESTER FILLER PAD 30CMx30CM
POLYESTER FILLER PAD 60x40cm
FLOWERS HANDBAG blue and orange
Next Day Carriage
THE KING GIFT BAG 25x24x12cm
High Resolution Image
```

In [ ]:

```
service_related_descriptions = ["Next Day Carriage", "High Resolution Image"

# Calculate the percentage of records with service-related descriptions
service_related_percentage = df[df['Description'].isin(service_related_descr
```

```python
# Print the percentage of records with service-related descriptions
print(f"The percentage of records with service-related descriptions in the c

# Remove rows with service-related information in the description
df = df[~df['Description'].isin(service_related_descriptions)]

# Standardize the text to uppercase to maintain uniformity across the datase
df['Description'] = df['Description'].str.upper()
```

```
The percentage of records with service-related descriptions in the dataset i
s: 0.02%
```

In [ ]:
```python
df.shape[0]
```

Out[ ]:
```
399606
```

Treating Zero Unit Prices

In [ ]:
```python
df['UnitPrice'].describe()
```

Out[ ]:

	UnitPrice
count	399606.000000
mean	2.904957
std	4.448796
min	0.000000
25%	1.250000
50%	1.950000
75%	3.750000
max	649.500000

dtype: float64

In [ ]:
```python
df[df['UnitPrice']==0].describe()[['Quantity']]
```

Out[ ]:

	Quantity
count	33.000000
mean	420.515152
std	2176.713608
min	1.000000
25%	2.000000
50%	11.000000
75%	36.000000
max	12540.000000

In [ ]:
```
df = df[df['UnitPrice'] > 0]
```

Outlier Treatment

In [ ]:
```
# Resetting the index of the cleaned dataset
df.reset_index(drop=True, inplace=True)
```

In [ ]:
```
df.shape[0]
```

Out[ ]: 399573

Recency (R)

In [ ]:
```
df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'])

# Convert InvoiceDate to datetime and extract only the date
df['InvoiceDay'] = df['InvoiceDate'].dt.date

# Find the most recent purchase date for each customer
customer_data = df.groupby('CustomerID')['InvoiceDay'].max().reset_index()

# Find the most recent date in the entire dataset
most_recent_date = df['InvoiceDay'].max()

# Convert InvoiceDay to datetime type before subtraction
customer_data['InvoiceDay'] = pd.to_datetime(customer_data['InvoiceDay'])
most_recent_date = pd.to_datetime(most_recent_date)

# Calculate the number of days since the last purchase for each customer
customer_data['Days_Since_Last_Purchase'] = (most_recent_date - customer_dat

# Remove the InvoiceDay column
customer_data.drop(columns=['InvoiceDay'], inplace=True)
```

In [ ]:
```
customer_data.head()
```

Out[ ]:

	CustomerID	Days_Since_Last_Purchase
0	12346.0	325
1	12347.0	2
2	12348.0	75
3	12349.0	18
4	12350.0	310

Frequency (F)

In [ ]:

```
# Calculate the total number of transactions made by each customer
total_transactions = df.groupby('CustomerID')['InvoiceNo'].nunique().reset_i
total_transactions.rename(columns={'InvoiceNo': 'Total_Transactions'}, inpla

# Calculate the total number of products purchased by each customer
total_products_purchased = df.groupby('CustomerID')['Quantity'].sum().reset_
total_products_purchased.rename(columns={'Quantity': 'Total_Products_Purchas

# Merge the new features into the customer_data dataframe
customer_data = pd.merge(customer_data, total_transactions, on='CustomerID')
customer_data = pd.merge(customer_data, total_products_purchased, on='Custom

# Display the first few rows of the customer_data dataframe
customer_data.head()
```

Out[ ]:

	CustomerID	Days_Since_Last_Purchase	Total_Transactions	Total_Products_I
0	12346.0	325	2	
1	12347.0	2	7	
2	12348.0	75	4	
3	12349.0	18	1	
4	12350.0	310	1	

Monetary (M)

In [ ]:

```
# Calculate the total spend by each customer
df['Total_Spend'] = df['UnitPrice'] * df['Quantity']
total_spend = df.groupby('CustomerID')['Total_Spend'].sum().reset_index()

# Calculate the average transaction value for each customer
average_transaction_value = total_spend.merge(total_transactions, on='Custom
average_transaction_value['Average_Transaction_Value'] = average_transactior

# Merge the new features into the customer_data dataframe
customer_data = pd.merge(customer_data, total_spend, on='CustomerID')
customer_data = pd.merge(customer_data, average_transaction_value[['Customer
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
# Display the first few rows of the customer_data dataframe
customer_data.head()
```

Out[ ]:

	CustomerID	Days_Since_Last_Purchase	Total_Transactions	Total_Products_I
0	12346.0	325	2	
1	12347.0	2	7	
2	12348.0	75	4	
3	12349.0	18	1	
4	12350.0	310	1	

Product Diversity

In [ ]:

```
# Calculate the number of unique products purchased by each customer
unique_products_purchased = df.groupby('CustomerID')['StockCode'].nunique().
unique_products_purchased.rename(columns={'StockCode': 'Unique_Products_Purc

# Merge the new feature into the customer_data dataframe
customer_data = pd.merge(customer_data, unique_products_purchased, on='Custo

# Display the first few rows of the customer_data dataframe
customer_data.head()
```

Out[ ]:

	CustomerID	Days_Since_Last_Purchase	Total_Transactions	Total_Products_I
0	12346.0	325	2	
1	12347.0	2	7	
2	12348.0	75	4	
3	12349.0	18	1	
4	12350.0	310	1	

Behavioral Features

In [ ]:

```
# Extract day of week and hour from InvoiceDate
df['Day_Of_Week'] = df['InvoiceDate'].dt.dayofweek
df['Hour'] = df['InvoiceDate'].dt.hour

# Calculate the average number of days between consecutive purchases
days_between_purchases = df.groupby('CustomerID')['InvoiceDay'].apply(lambda
average_days_between_purchases = days_between_purchases.groupby('CustomerID'
average_days_between_purchases.rename(columns={'InvoiceDay': 'Average_Days_B

# Find the favorite shopping day of the week
favorite_shopping_day = df.groupby(['CustomerID', 'Day_Of_Week']).size().res
favorite_shopping_day = favorite_shopping_day.loc[favorite_shopping_day.grou

# Find the favorite shopping hour of the day
f.groupby(['CustomerID', 'Hour']).size().reset_inc
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
favorite_shopping_hour = favorite_shopping_hour.loc[favorite_shopping_hour.g

# Merge the new features into the customer_data dataframe
customer_data = pd.merge(customer_data, average_days_between_purchases, on='
customer_data = pd.merge(customer_data, favorite_shopping_day, on='CustomerI
customer_data = pd.merge(customer_data, favorite_shopping_hour, on='Customer

# Display the first few rows of the customer_data dataframe
customer_data.head()
```

Out[ ]:

	CustomerID	Days_Since_Last_Purchase	Total_Transactions	Total_Products_I
0	12346.0	325	2	
1	12347.0	2	7	
2	12348.0	75	4	
3	12349.0	18	1	
4	12350.0	310	1	

Geographic Features

In [47]:

```
df['Country'].value_counts(normalize=True).head()
```

Out[47]:

Country	proportion
United Kingdom	0.890971
Germany	0.022722
France	0.020402
EIRE	0.018440
Spain	0.006162

dtype: float64

In [ ]:

```
# Group by CustomerID and Country to get the number of transactions per cour
customer_country = df.groupby(['CustomerID', 'Country']).size().reset_index(

# Get the country with the maximum number of transactions for each customer
customer_main_country = customer_country.sort_values('Number_of_Transactions

# Create a binary column indicating whether the customer is from the UK or r
customer_main_country['Is_UK'] = customer_main_country['Country'].apply(lamb

# Merge this data with our customer_data dataframe
customer_data = pd.merge(customer_data, customer_main_country[['CustomerID',

# Display the first few rows of the customer_data dataframe
customer_data.head()
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Out[ ]:

	CustomerID	Days_Since_Last_Purchase	Total_Transactions	Total_Products_I
0	12346.0	325	2	
1	12347.0	2	7	
2	12348.0	75	4	
3	12349.0	18	1	
4	12350.0	310	1	

In [ ]:

```
# Display feature distribution
customer_data['Is_UK'].value_counts()
```

Out[ ]:

Is_UK	count
1	3866
0	416

dtype: int64

Cancellation Insights

In [49]:

```
# Calculate the total number of transactions made by each customer
total_transactions = df.groupby('CustomerID')['InvoiceNo'].nunique().reset_i

# Calculate the number of cancelled transactions for each customer
cancelled_transactions = df[df['Transaction_Status'] == 'Cancelled']
cancellation_frequency = cancelled_transactions.groupby('CustomerID')['Invoi
cancellation_frequency.rename(columns={'InvoiceNo': 'Cancellation_Frequency'

# Merge the Cancellation Frequency data into the customer_data dataframe
customer_data = pd.merge(customer_data, cancellation_frequency, on='Customer

# Replace NaN values with 0 (for customers who have not cancelled any transa
customer_data['Cancellation_Frequency'].fillna(0, inplace=True)

# Calculate the Cancellation Rate
customer_data['Cancellation_Rate'] = customer_data['Cancellation_Frequency']

# Display the first few rows of the customer_data dataframe
customer_data.head()
```

Out[49]:

	CustomerID	Days_Since_Last_Purchase	Total_Transactions	Total_Products_I
0	12346.0	325	2	
1	12347.0	2	7	
2	12348.0	75	4	
3	12349.0	18	1	
4	12350.0	310	1	

Seasonality & Trends

In [50]:

```
# Extract month and year from InvoiceDate
df['Year'] = df['InvoiceDate'].dt.year
df['Month'] = df['InvoiceDate'].dt.month

# Calculate monthly spending for each customer
monthly_spending = df.groupby(['CustomerID', 'Year', 'Month'])['Total_Spend'

# Calculate Seasonal Buying Patterns: We are using monthly frequency as a pr
seasonal_buying_patterns = monthly_spending.groupby('CustomerID')['Total_Spe
seasonal_buying_patterns.rename(columns={'mean': 'Monthly_Spending_Mean', 's

# Replace NaN values in Monthly_Spending_Std with 0, implying no variability
seasonal_buying_patterns['Monthly_Spending_Std'].fillna(0, inplace=True)

# Calculate Trends in Spending
# We are using the slope of the linear trend line fitted to the customer's s
def calculate_trend(spend_data):
    # If there are more than one data points, we calculate the trend using l
    if len(spend_data) > 1:
        x = np.arange(len(spend_data))
        slope, _, _, _, _ = linregress(x, spend_data)
        return slope
    # If there is only one data point, no trend can be calculated, hence we
    else:
        return 0

# Apply the calculate_trend function to find the spending trend for each cus
spending_trends = monthly_spending.groupby('CustomerID')['Total_Spend'].appl
spending_trends.rename(columns={'Total_Spend': 'Spending_Trend'}, inplace=Tr

# Merge the new features into the customer_data dataframe
customer_data = pd.merge(customer_data, seasonal_buying_patterns, on='Custom
customer_data = pd.merge(customer_data, spending_trends, on='CustomerID')

# Display the first few rows of the customer_data dataframe
customer_data.head()
```

Out[50]:

	CustomerID	Days_Since_Last_Purchase	Total_Transactions	Total_Products_I
0	12346.0	325	2	
1	12347.0	2	7	
2	12348.0	75	4	
3	12349.0	18	1	
4	12350.0	310	1	

In [51]:
```
# Changing the data type of 'CustomerID' to string as it is a unique identi
customer_data['CustomerID'] = customer_data['CustomerID'].astype(str)

# Convert data types of columns to optimal types
customer_data = customer_data.convert_dtypes()
```

In [52]:
```
customer_data.head(10)
```

Out[52]:

	CustomerID	Days_Since_Last_Purchase	Total_Transactions	Total_Products_I
0	12346.0	325	2	
1	12347.0	2	7	
2	12348.0	75	4	
3	12349.0	18	1	
4	12350.0	310	1	
5	12352.0	36	8	
6	12353.0	204	1	
7	12354.0	232	1	
8	12355.0	214	1	
9	12356.0	22	3	

In [53]:
```
customer_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4282 entries, 0 to 4281
Data columns (total 18 columns):
 #   Column                           Non-Null Count  Dtype  
---  ------                           --------------  -----  
 0   CustomerID                       4282 non-null   string 
 1   Days_Since_Last_Purchase         4282 non-null   Int64  
 2   Total_Transactions               4282 non-null   Int64  
 3   Total_Products_Purchased         4282 non-null   Int64  
 4   Total_Spend                      4282 non-null   Float64
 5   Average_Transaction_Value        4282 non-null   Float64
 6   Unique_Products_Purchased        4282 non-null   Int64  
 7   Average_Days_Between_Purchases   4282 non-null   Float64
 8   Day_Of_Week                      4282 non-null   Int32  
 9   Hour                             4282 non-null   Int32  
 10  Is_UK                            4282 non-null   Int64  
 11  Cancellation_Frequency_x         4282 non-null   Int64  
 12  Cancellation_Rate                4282 non-null   Float64
 13  Cancellation_Frequency_y         1523 non-null   Int64  
 14  Cancellation_Frequency           4282 non-null   Int64  
 15  Monthly_Spending_Mean            4282 non-null   Float64
 16  Monthly_Spending_Std             4282 non-null   Float64
 17  Spending_Trend                   4282 non-null   Float64
dtypes: Float64(7), Int32(2), Int64(8), string(1)
memory usage: 639.9 KB
```

Outlier Detection and Treatment

In [58]:

```
import pandas as pd
import numpy as np
from sklearn.ensemble import IsolationForest

# Assuming customer_data is already loaded as a DataFrame
# Example customer_data initialization (if needed):
# customer_data = pd.read_csv('customer_data.csv')

# Initialize the IsolationForest model with a contamination parameter of 0.0
model = IsolationForest(contamination=0.05, random_state=0)

# Select only numeric columns from the dataset for fitting the model
numeric_cols = customer_data.select_dtypes(include=[np.number]).columns

# Fill missing values: for integer columns, fill with the median (to ensure 
for col in numeric_cols:
    if pd.api.types.is_integer_dtype(customer_data[col]):
        # Fill missing integer columns with the median and convert back to i
        customer_data[col] = customer_data[col].fillna(customer_data[col].me
    else:
        # Fill other numeric columns (e.g., floats) with the mean
        customer_data[col] = customer_data[col].fillna(customer_data[col].me

# Fit the model on the filled dataset and predict outlier scores
customer_data_filled = customer_data[numeric_cols]
customer_data['Outlier_Scores'] = model.fit_predict(customer_data_filled.to_

dentify outliers (1 for inliers and 0 for outliers
```

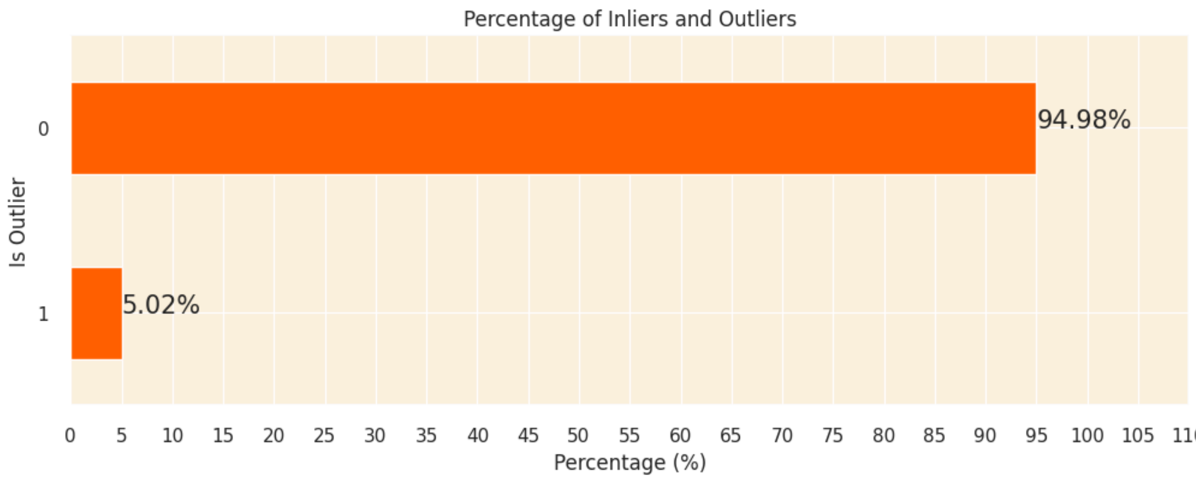
```
customer_data['Is_Outlier'] = [1 if x == -1 else 0 for x in customer_data['O

# Display the first few rows of the customer_data dataframe
print(customer_data.head())
```

```
   CustomerID  Days_Since_Last_Purchase  Total_Transactions  \
0     12346.0                       325                   2
1     12347.0                         2                   7
2     12348.0                        75                   4
3     12349.0                        18                   1
4     12350.0                       310                   1

   Total_Products_Purchased  Total_Spend  Average_Transaction_Value  \
0                         0          0.0                        0.0
1                      2458       4310.0                 615.714286
2                      2332      1437.24                     359.31
3                       630      1457.55                    1457.55
4                       196        294.4                      294.4

   Unique_Products_Purchased  Average_Days_Between_Purchases  Day_Of_Week  \
0                          1                             0.0            1
1                        103                        2.016575            1
2                         21                       10.884615            3
3                         72                             0.0            0
4                         16                             0.0            2

   Hour  Is_UK  Cancellation_Frequency_x  Cancellation_Rate  \
0    10      1                         1                0.5
1    14      0                         0                0.0
2    19      0                         0                0.0
3     9      0                         0                0.0
4    16      0                         0                0.0

   Cancellation_Frequency_y  Cancellation_Frequency  Monthly_Spending_Mean
\
0                         1                       1                    0.0
1                         1                       0             615.714286
2                         1                       0                 359.31
3                         1                       0                1457.55
4                         1                       0                  294.4

   Monthly_Spending_Std  Spending_Trend  Outlier_Scores  Is_Outlier
0                   0.0             0.0               1           0
1            341.070789        4.486071               1           0
2            203.875689        -100.884               1           0
3                   0.0             0.0               1           0
4                   0.0             0.0               1           0
```

In [59]:

```
# Calculate the percentage of inliers and outliers
outlier_percentage = customer_data['Is_Outlier'].value_counts(normalize=True

# Plotting the percentage of inliers and outliers
plt.figure(figsize=(12, 4))
outlier_percentage.plot(kind='barh', color='#ff6200')

# Adding the percentage labels on the bars
```

```python
for index, value in enumerate(outlier_percentage):
    plt.text(value, index, f'{value:.2f}%', fontsize=15)

plt.title('Percentage of Inliers and Outliers')
plt.xticks(ticks=np.arange(0, 115, 5))
plt.xlabel('Percentage (%)')
plt.ylabel('Is Outlier')
plt.gca().invert_yaxis()
plt.show()
```



In [60]:
```python
# Separate the outliers for analysis
outliers_data = customer_data[customer_data['Is_Outlier'] == 1]

# Remove the outliers from the main dataset
customer_data_cleaned = customer_data[customer_data['Is_Outlier'] == 0]

# Drop the 'Outlier_Scores' and 'Is_Outlier' columns
customer_data_cleaned = customer_data_cleaned.drop(columns=['Outlier_Scores'

# Reset the index of the cleaned data
customer_data_cleaned.reset_index(drop=True, inplace=True)
```

In [61]:
```python
# Getting the number of rows in the cleaned customer dataset
customer_data_cleaned.shape[0]
```

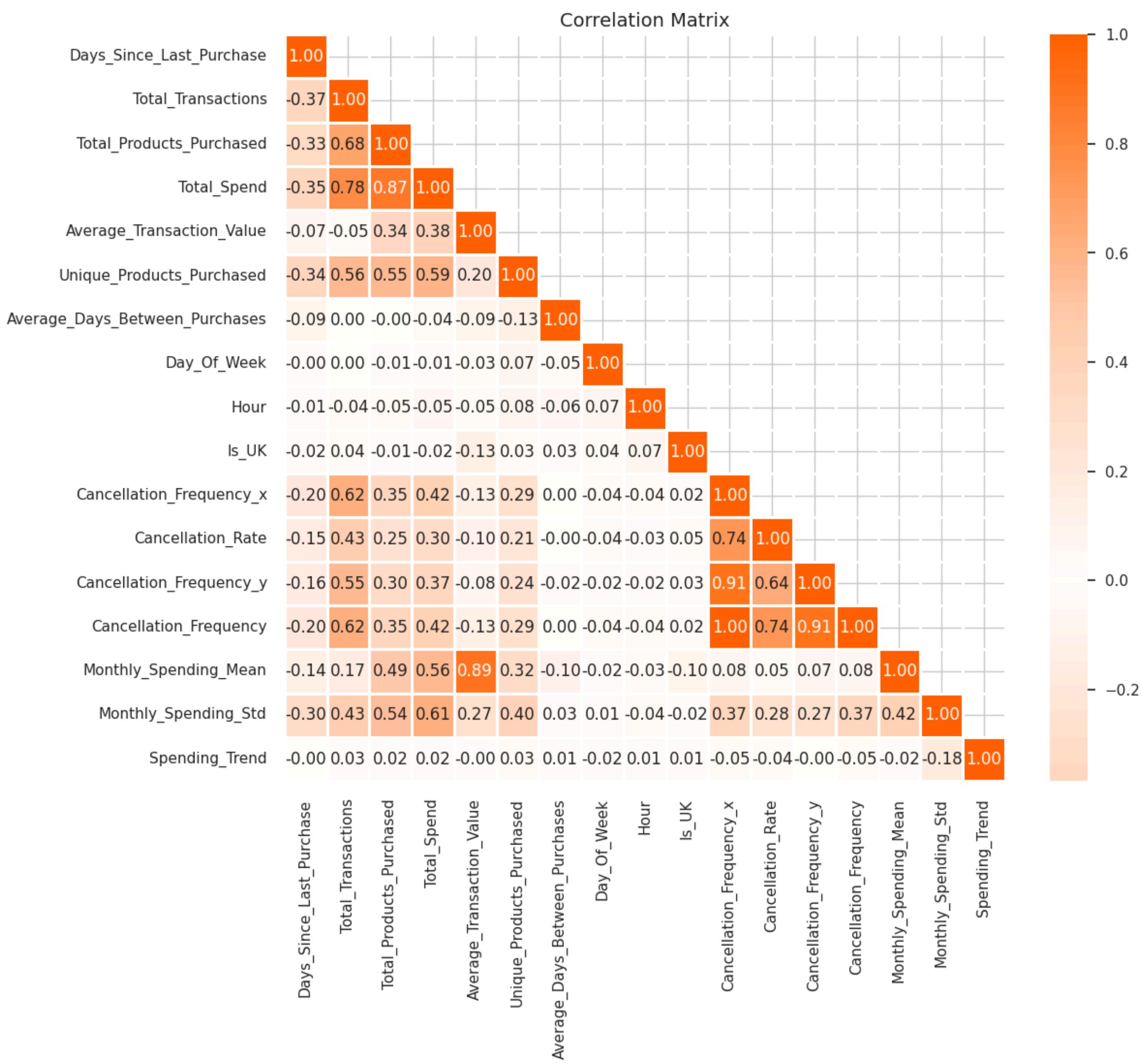
Out[61]: 4067

Correlation Analysis

In [62]:
```python
# Reset background style
sns.set_style('whitegrid')

# Calculate the correlation matrix excluding the 'CustomerID' column
corr = customer_data_cleaned.drop(columns=['CustomerID']).corr()

# Define a custom colormap
colors = ['#ff6200', '#ffcaa8', 'white', '#ffcaa8', '#ff6200']
my_cmap = LinearSegmentedColormap.from_list('custom_map', colors, N=256)

# C                        ow the lower triangle of the matrix (since it's mi
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
# top-left to bottom-right diagonal)
mask = np.zeros_like(corr)
mask[np.triu_indices_from(mask, k=1)] = True

# Plot the heatmap
plt.figure(figsize=(12, 10))
sns.heatmap(corr, mask=mask, cmap=my_cmap, annot=True, center=0, fmt='.2f',
plt.title('Correlation Matrix', fontsize=14)
plt.show()
```



Feature Scaling

In [63]:
```
# Initialize the StandardScaler
scaler = StandardScaler()

# List of columns that don't need to be scaled
columns_to_exclude = ['CustomerID', 'Is_UK', 'Day_Of_Week']

# List of columns that need to be scaled
columns_to_scale = customer_data_cleaned.columns.difference(columns_to_exclu
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
customer_data_scaled = customer_data_cleaned.copy()

# Applying the scaler to the necessary columns in the dataset
customer_data_scaled[columns_to_scale] = scaler.fit_transform(customer_data_

# Display the first few rows of the scaled data
customer_data_scaled.head()
```

Out[63]:

	CustomerID	Days_Since_Last_Purchase	Total_Transactions	Total_Products_I
0	12346.0	2.337854	-0.468031	
1	12347.0	-0.906207	0.713523	
2	12348.0	-0.173029	0.004590	
3	12349.0	-0.745511	-0.704342	
4	12350.0	2.187201	-0.704342	

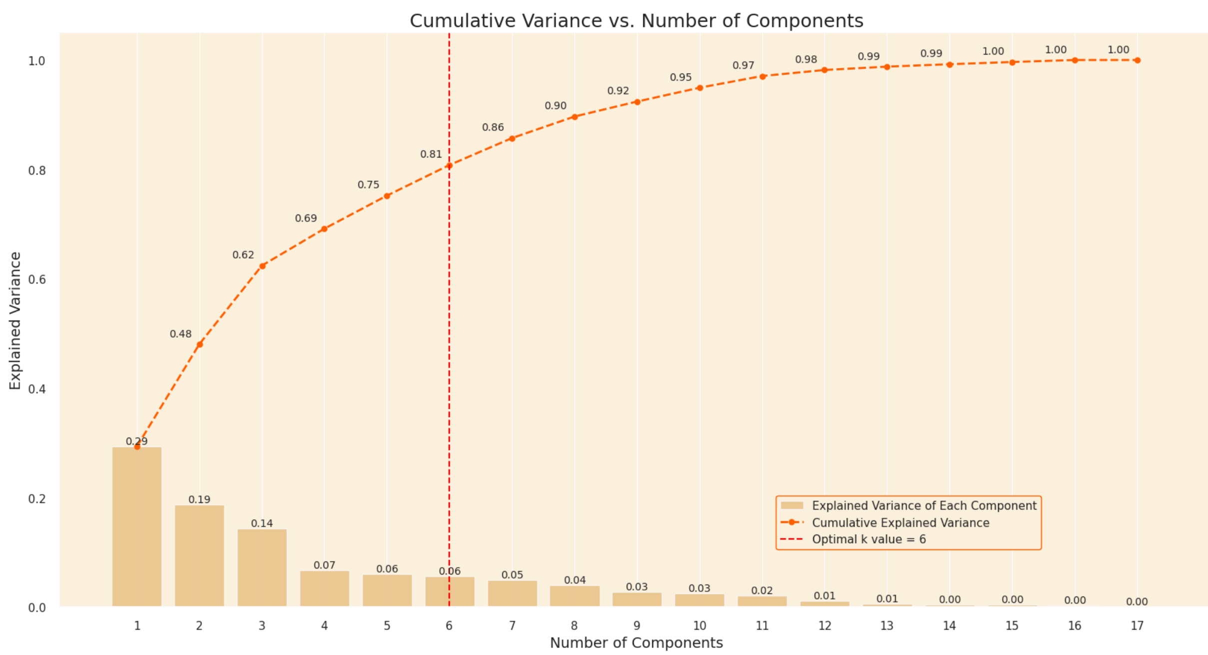
Dimensionality Reduction

In [64]:

```
# Setting CustomerID as the index column
customer_data_scaled.set_index('CustomerID', inplace=True)

# Apply PCA
pca = PCA().fit(customer_data_scaled)

# Calculate the Cumulative Sum of the Explained Variance
explained_variance_ratio = pca.explained_variance_ratio_
cumulative_explained_variance = np.cumsum(explained_variance_ratio)

# Set the optimal k value (based on our analysis, we can choose 6)
optimal_k = 6

# Set seaborn plot style
sns.set(rc={'axes.facecolor': '#fcf0dc'}, style='darkgrid')

# Plot the cumulative explained variance against the number of components
plt.figure(figsize=(20, 10))

# Bar chart for the explained variance of each component
barplot = sns.barplot(x=list(range(1, len(cumulative_explained_variance) + 1
                      y=explained_variance_ratio,
                      color='#fcc36d',
                      alpha=0.8)

# Line plot for the cumulative explained variance
lineplot, = plt.plot(range(0, len(cumulative_explained_variance)), cumulativ
                     marker='o', linestyle='--', color='#ff6200', linewidth=

# Plot optimal k value line
optimal_k_line = plt.axvline(optimal_k - 1, color='red', linestyle='--', lab

# Set labels and title
plt.xlabel('Number of Components', fontsize=14)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
plt.ylabel('Explained Variance', fontsize=14)
plt.title('Cumulative Variance vs. Number of Components', fontsize=18)

# Customize ticks and legend
plt.xticks(range(0, len(cumulative_explained_variance)))
plt.legend(handles=[barplot.patches[0], lineplot, optimal_k_line],
           labels=['Explained Variance of Each Component', 'Cumulative Expla
           loc=(0.62, 0.1),
           frameon=True,
           framealpha=1.0,
           edgecolor='#ff6200')

# Display the variance values for both graphs on the plots
x_offset = -0.3
y_offset = 0.01
for i, (ev_ratio, cum_ev_ratio) in enumerate(zip(explained_variance_ratio, c
    plt.text(i, ev_ratio, f"{ev_ratio:.2f}", ha="center", va="bottom", fonts
    if i > 0:
        plt.text(i + x_offset, cum_ev_ratio + y_offset, f"{cum_ev_ratio:.2f}

plt.grid(axis='both')
plt.show()
```



In [65]:
```
# Creating a PCA object with 6 components
pca = PCA(n_components=6)

# Fitting and transforming the original data to the new PCA dataframe
customer_data_pca = pca.fit_transform(customer_data_scaled)

# Creating a new dataframe from the PCA dataframe, with columns labeled PC1,
customer_data_pca = pd.DataFrame(customer_data_pca, columns=['PC'+str(i+1) f

# Adding the CustomerID index back to the new PCA dataframe
customer_data_pca.index = customer_data_scaled.index
```

In [66]:
```
# Displaying the resulting dataframe based on the PCs
customer_data_pca.head()
```

Out[66]:

CustomerID	PC1	PC2	PC3	PC4	PC5	PC6
12346.0	-1.502393	-1.828999	2.237205	-0.948595	-0.078333	-0.685397
12347.0	2.231916	-1.198779	-3.314165	1.006539	-0.277953	0.464954
12348.0	0.067095	0.642905	-1.148545	0.854184	-0.426443	1.629822
12349.0	0.566591	-2.488291	-5.236476	-3.102705	0.291673	-1.717575
12350.0	-2.055544	-0.580509	0.142653	-0.948095	-1.310880	0.422936

In [67]:
```
# Define a function to highlight the top 3 absolute values in each column of
def highlight_top3(column):
    top3 = column.abs().nlargest(3).index
    return ['background-color:  #ffeacc' if i in top3 else '' for i in colum

# Create the PCA component DataFrame and apply the highlighting function
pc_df = pd.DataFrame(pca.components_.T, columns=['PC{}'.format(i+1) for i in
                     index=customer_data_scaled.columns)

pc_df.style.apply(highlight_top3, axis=0)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Out[67]:

	PC1	PC2	PC3	PC4
Days_Since_Last_Purchase	-0.179972	-0.020901	0.082395	-0.434640
Total_Transactions	0.357081	0.022778	0.036871	0.288405
Total_Products_Purchased	0.323768	0.023863	-0.247652	0.179571
Total_Spend	0.356101	0.028221	-0.244201	0.129155
Average_Transaction_Value	0.088691	0.002459	-0.486716	-0.391746
Unique_Products_Purchased	0.264113	0.075044	-0.171313	0.267046
Average_Days_Between_Purchases	-0.011452	-0.041186	0.061802	0.291487
Day_Of_Week	-0.031866	0.991987	0.067786	-0.045505
Hour	-0.020778	0.056684	0.012573	0.124389
Is_UK	0.000678	0.006265	0.017539	0.025116
Cancellation_Frequency_x	0.348049	-0.022566	0.312531	-0.174064
Cancellation_Rate	0.274900	-0.025002	0.277162	-0.199080
Cancellation_Frequency_y	0.316984	-0.017297	0.303159	-0.201910
Cancellation_Frequency	0.348049	-0.022566	0.312531	-0.174064
Monthly_Spending_Mean	0.183435	0.015342	-0.450435	-0.363380
Monthly_Spending_Std	0.277632	0.027740	-0.169394	-0.021686
Spending_Trend	-0.014711	-0.010317	-0.010158	0.278485

K-Means Clustering

Determining the Optimal Number of Clusters

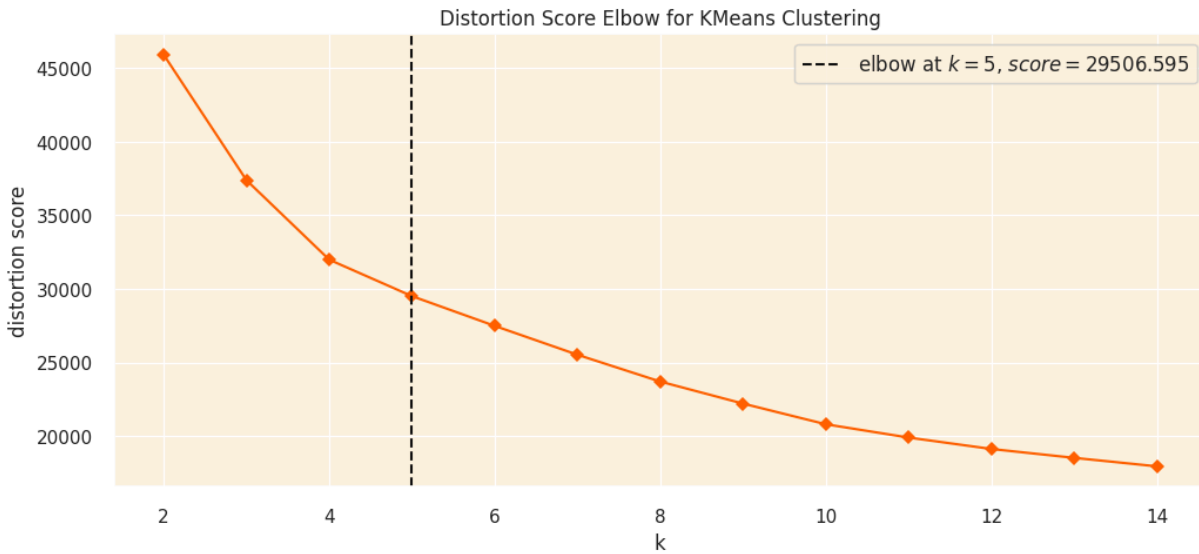
Elbow Method

In [68]:

```
# Set plot style, and background color
sns.set(style='darkgrid', rc={'axes.facecolor': '#fcf0dc'})

# Set the color palette for the plot
sns.set_palette(['#ff6200'])

# Instantiate the clustering model with the specified parameters
km = KMeans(init='k-means++', n_init=10, max_iter=100, random_state=0)

# Create a figure and axis with the desired size
fig, ax = plt.subplots(figsize=(12, 5))

# Instantiate the KElbowVisualizer with the model and range of k values, and
visualizer = KElbowVisualizer(km, k=(2, 15), timings=False, ax=ax)

# Fit the data to the visualizer
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
visualizer.fit(customer_data_pca)

# Finalize and render the figure
visualizer.show();
```



Silhouette Method

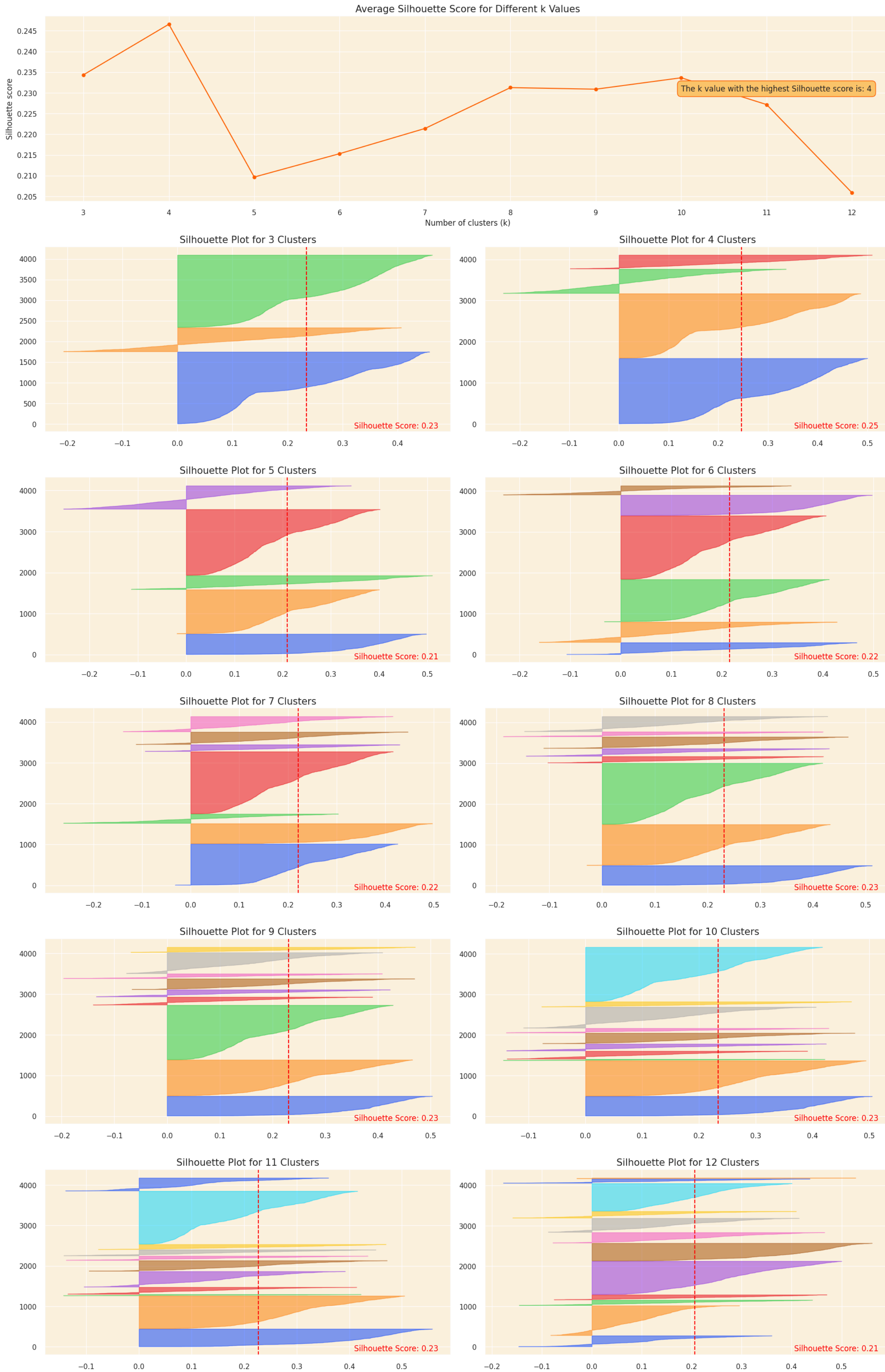
In [69]:
```
def silhouette_analysis(df, start_k, stop_k, figsize=(15, 16)):
    """
    Perform Silhouette analysis for a range of k values and visualize the re
    """

    # Set the size of the figure
    plt.figure(figsize=figsize)

    # Create a grid with (stop_k - start_k + 1) rows and 2 columns
    grid = gridspec.GridSpec(stop_k - start_k + 1, 2)

    # Assign the first plot to the first row and both columns
    first_plot = plt.subplot(grid[0, :])

    # First plot: Silhouette scores for different k values
    sns.set_palette(['darkorange'])

    silhouette_scores = []

    # Iterate through the range of k values
    for k in range(start_k, stop_k + 1):
        km = KMeans(n_clusters=k, init='k-means++', n_init=10, max_iter=100,
        km.fit(df)
        labels = km.predict(df)
        score = silhouette_score(df, labels)
        silhouette_scores.append(score)

    best_k = start_k + silhouette_scores.index(max(silhouette_scores))

    plt.plot(range(start_k, stop_k + 1), silhouette_scores, marker='o')
    plt.xticks(range(start_k, stop_k + 1))
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
    plt.xlabel('Number of clusters (k)')
    plt.ylabel('Silhouette score')
    plt.title('Average Silhouette Score for Different k Values', fontsize=15

    # Add the optimal k value text to the plot
    optimal_k_text = f'The k value with the highest Silhouette score is: {be
    plt.text(10, 0.23, optimal_k_text, fontsize=12, verticalalignment='botto
             horizontalalignment='left', bbox=dict(facecolor='#fcc36d', edge


    # Second plot (subplot): Silhouette plots for each k value
    colors = sns.color_palette("bright")

    for i in range(start_k, stop_k + 1):
        km = KMeans(n_clusters=i, init='k-means++', n_init=10, max_iter=100,
        row_idx, col_idx = divmod(i - start_k, 2)

        # Assign the plots to the second, third, and fourth rows
        ax = plt.subplot(grid[row_idx + 1, col_idx])

        visualizer = SilhouetteVisualizer(km, colors=colors, ax=ax)
        visualizer.fit(df)

        # Add the Silhouette score text to the plot
        score = silhouette_score(df, km.labels_)
        ax.text(0.97, 0.02, f'Silhouette Score: {score:.2f}', fontsize=12, \
                ha='right', transform=ax.transAxes, color='red')

        ax.set_title(f'Silhouette Plot for {i} Clusters', fontsize=15)

    plt.tight_layout()
    plt.show()
```

In [70]:

```
silhouette_analysis(customer_data_pca, 3, 12, figsize=(20, 50))
```



Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Clustering Model - K-means

In [71]:
```
# Apply KMeans clustering using the optimal k
kmeans = KMeans(n_clusters=3, init='k-means++', n_init=10, max_iter=100, rar
kmeans.fit(customer_data_pca)

# Get the frequency of each cluster
cluster_frequencies = Counter(kmeans.labels_)

# Create a mapping from old labels to new labels based on frequency
label_mapping = {label: new_label for new_label, (label, _) in
                 enumerate(cluster_frequencies.most_common())}

# Reverse the mapping to assign labels as per your criteria
label_mapping = {v: k for k, v in {2: 1, 1: 0, 0: 2}.items()}

# Apply the mapping to get the new labels
new_labels = np.array([label_mapping[label] for label in kmeans.labels_])

# Append the new cluster labels back to the original dataset
customer_data_cleaned['cluster'] = new_labels

# Append the new cluster labels to the PCA version of the dataset
customer_data_pca['cluster'] = new_labels
```

In [72]:
```
# Display the first few rows of the original dataframe
customer_data_cleaned.head()
```

Out[72]:

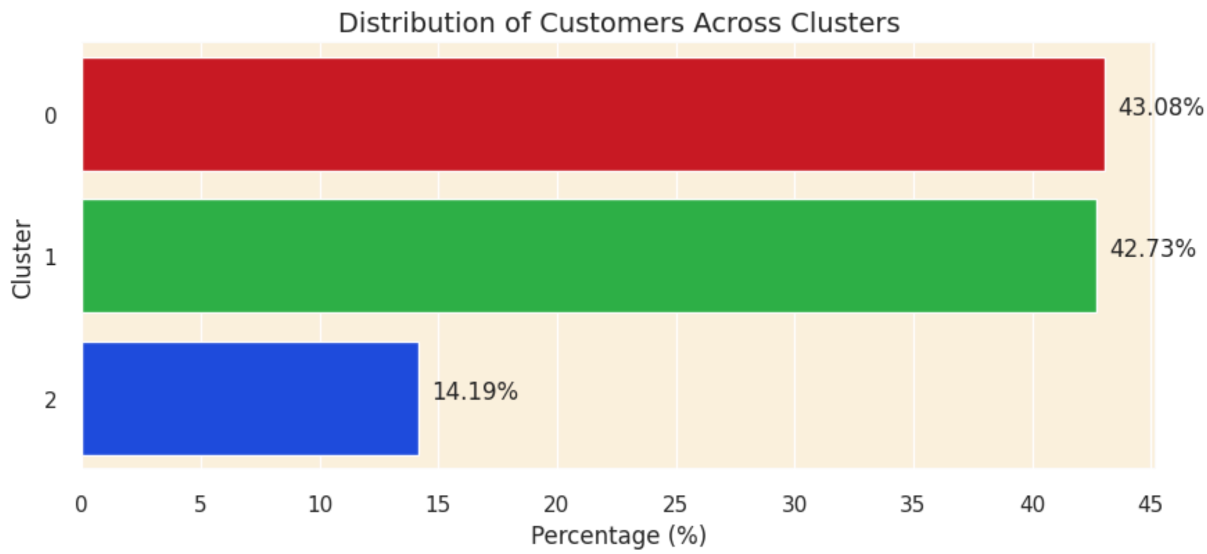
	CustomerID	Days_Since_Last_Purchase	Total_Transactions	Total_Products_I
0	12346.0	325	2	
1	12347.0	2	7	
2	12348.0	75	4	
3	12349.0	18	1	
4	12350.0	310	1	

Clustering Evaluation

Cluster Distribution Visualization

In [76]:
```
# Calculate the percentage of customers in each cluster
cluster_percentage = (customer_data_pca['cluster'].value_counts(normalize=Tr
cluster_percentage.columns = ['Cluster', 'Percentage']
cluster_percentage.sort_values(by='Cluster', inplace=True)

# Create a horizontal bar plot
plt.figure(figsize=(10, 4))
sns.barplot(x='Percentage', y='Cluster', data=cluster_percentage, orient='h'

# Adding percentages on the bars
```

```python
for index, value in enumerate(cluster_percentage['Percentage']):
    plt.text(value+0.5, index, f'{value:.2f}%')

plt.title('Distribution of Customers Across Clusters', fontsize=14)
plt.xticks(ticks=np.arange(0, 50, 5))
plt.xlabel('Percentage (%)')

# Show the plot
plt.show()
```



Evaluation Metrics

In [77]:
```python
# Compute number of customers
num_observations = len(customer_data_pca)

# Separate the features and the cluster labels
X = customer_data_pca.drop('cluster', axis=1)
clusters = customer_data_pca['cluster']

# Compute the metrics
sil_score = silhouette_score(X, clusters)
calinski_score = calinski_harabasz_score(X, clusters)
davies_score = davies_bouldin_score(X, clusters)

# Create a table to display the metrics and the number of observations
table_data = [
    ["Number of Observations", num_observations],
    ["Silhouette Score", sil_score],
    ["Calinski Harabasz Score", calinski_score],
    ["Davies Bouldin Score", davies_score]
]

# Print the table
print(tabulate(table_data, headers=["Metric", "Value"], tablefmt='pretty'))
```

```
+-------------------------+--------------------+
|         Metric          |       Value        |
+-------------------------+--------------------+
| Number of Observations  |        4067        |
|    Silhouette Score     | 0.2343605240053072 |
| Calinski Harabasz Score | 1275.485735208937  |
|  Davies Bouldin Score   | 1.3984053542704358 |
+-------------------------+--------------------+
```

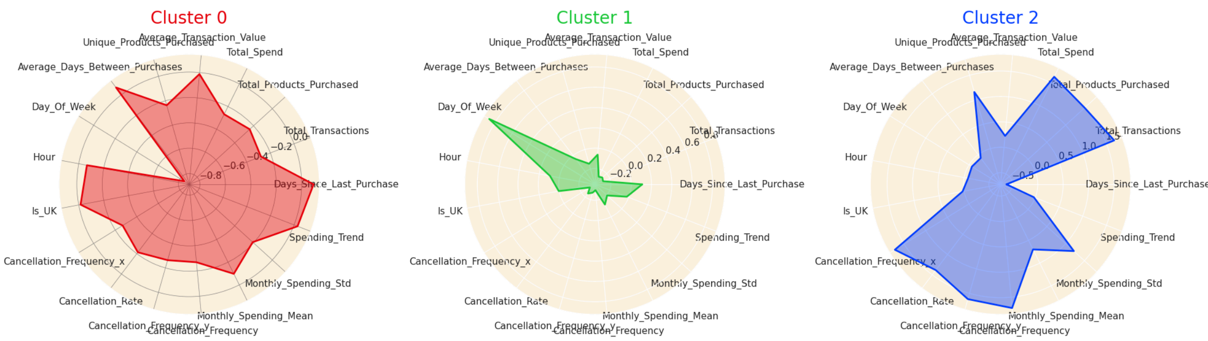
Cluster Analysis and Profiling

Radar Chart Approach

In [78]:

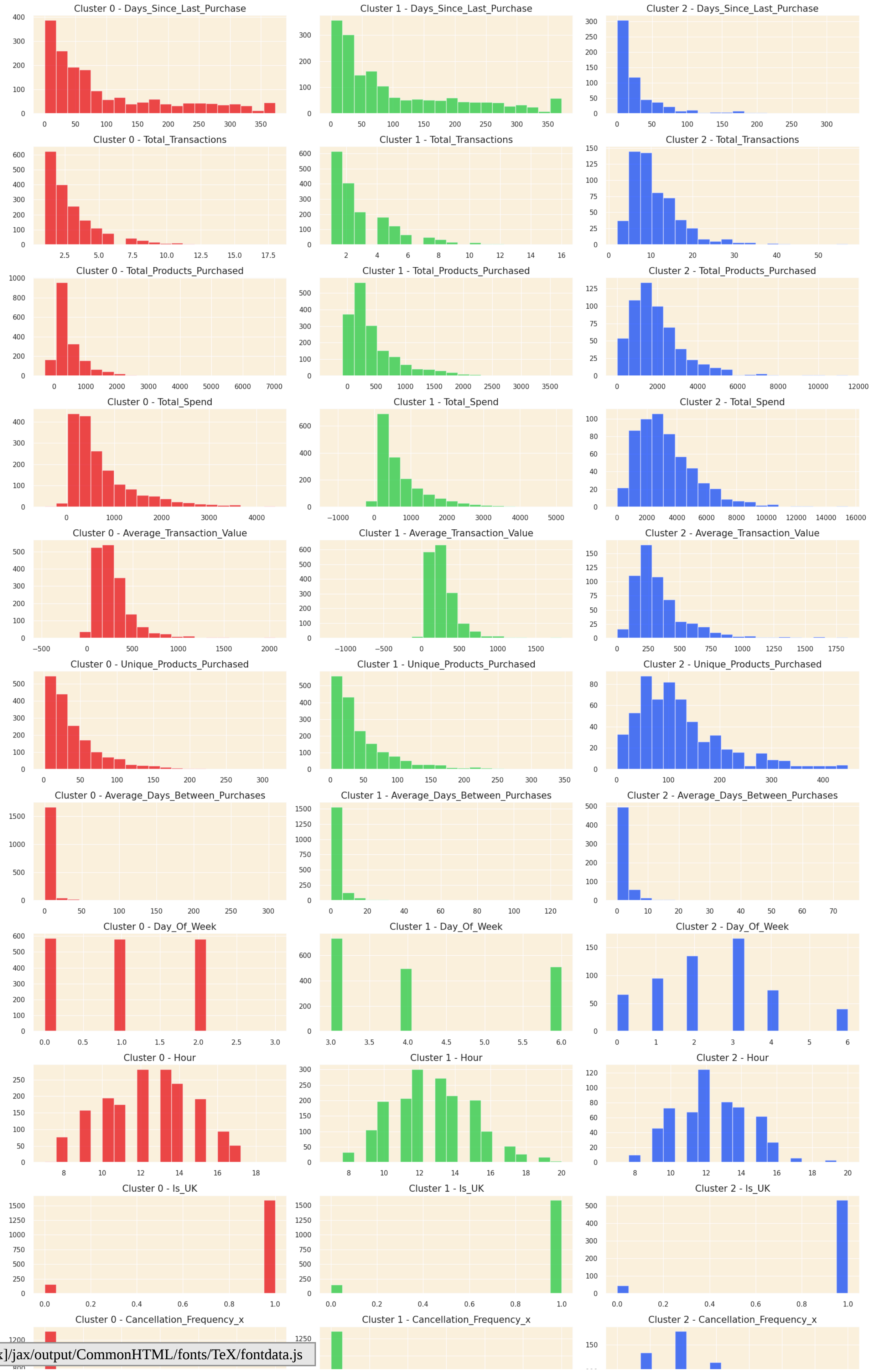
```python
# Setting 'CustomerID' column as index and assigning it to a new dataframe
df_customer = customer_data_cleaned.set_index('CustomerID')

# Standardize the data (excluding the cluster column)
scaler = StandardScaler()
df_customer_standardized = scaler.fit_transform(df_customer.drop(columns=['c

# Create a new dataframe with standardized values and add the cluster column
df_customer_standardized = pd.DataFrame(df_customer_standardized, columns=df
df_customer_standardized['cluster'] = df_customer['cluster']

# Calculate the centroids of each cluster
cluster_centroids = df_customer_standardized.groupby('cluster').mean()

# Function to create a radar chart
def create_radar_chart(ax, angles, data, color, cluster):
    # Plot the data and fill the area
    ax.fill(angles, data, color=color, alpha=0.4)
    ax.plot(angles, data, color=color, linewidth=2, linestyle='solid')

    # Add a title
    ax.set_title(f'Cluster {cluster}', size=20, color=color, y=1.1)

# Set data
labels=np.array(cluster_centroids.columns)
num_vars = len(labels)

# Compute angle of each axis
angles = np.linspace(0, 2 * np.pi, num_vars, endpoint=False).tolist()

# The plot is circular, so we need to "complete the loop" and append the sta
labels = np.concatenate((labels, [labels[0]]))
angles += angles[:1]

# Initialize the figure
fig, ax = plt.subplots(figsize=(20, 10), subplot_kw=dict(polar=True), nrows=

# Create radar chart for each cluster
for i, color in enumerate(colors):
    data = cluster_centroids.loc[i].tolist()
    data += data[:1]  # Complete the loop
    create_radar_chart(ax[i], angles, data, color, i)
```

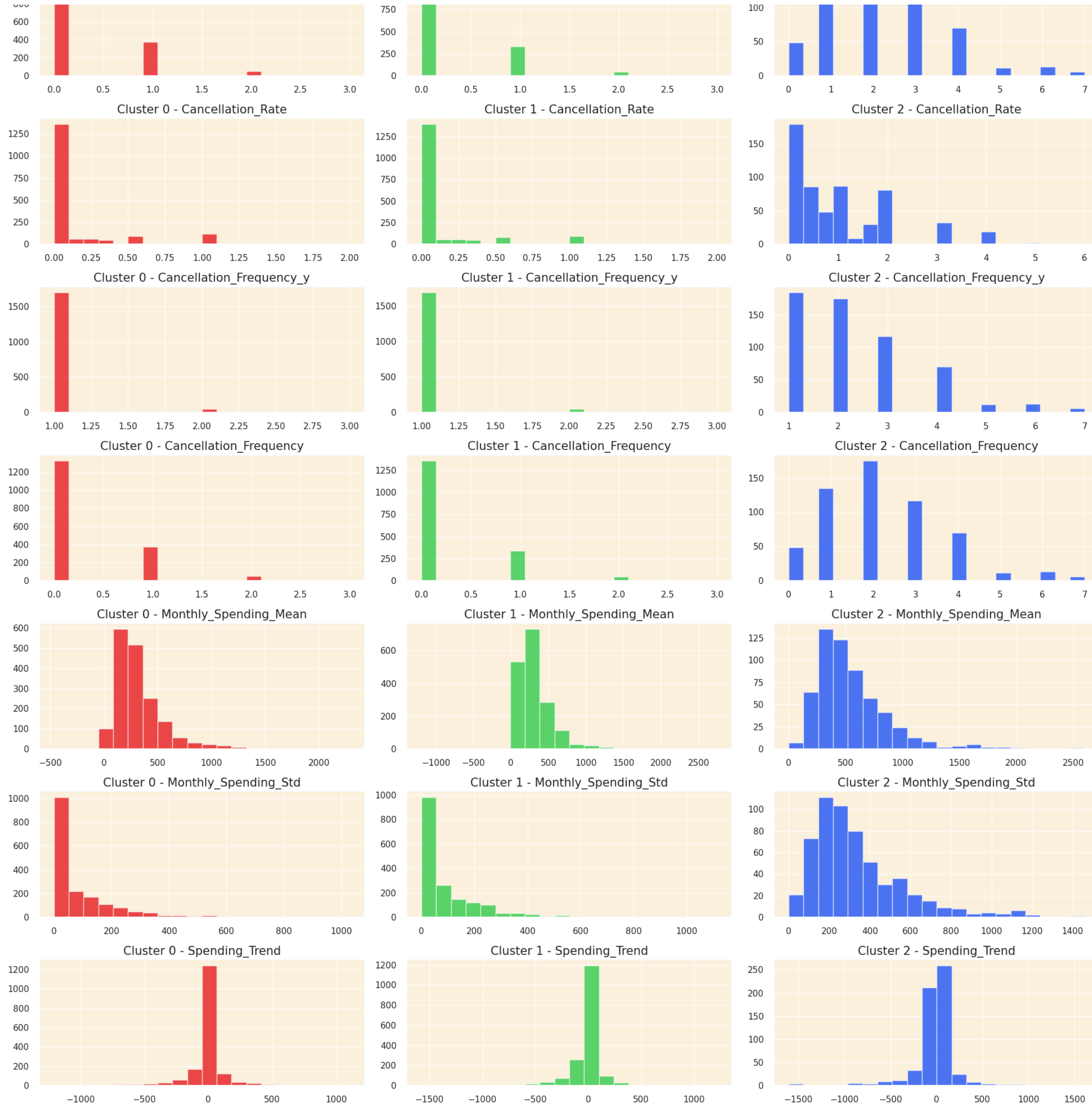
```
# Add input data
ax[0].set_xticks(angles[:-1])
ax[0].set_xticklabels(labels[:-1])

ax[1].set_xticks(angles[:-1])
ax[1].set_xticklabels(labels[:-1])

ax[2].set_xticks(angles[:-1])
ax[2].set_xticklabels(labels[:-1])

# Add a grid
ax[0].grid(color='grey', linewidth=0.5)

# Display the plot
plt.tight_layout()
plt.show()
```



Histogram Chart Approach

In [79]:
```
# Plot histograms for each feature segmented by the clusters
features = customer_data_cleaned.columns[1:-1]
clusters = customer_data_cleaned['cluster'].unique()
clusters.sort()

# Setting up the subplots
n_rows = len(features)
n_cols = len(clusters)
fig, axes = plt.subplots(n_rows, n_cols, figsize=(20, 3*n_rows))

# Plotting histograms
for i, feature in enumerate(features):
    for j, cluster in enumerate(clusters):
        data = customer_data_cleaned[customer_data_cleaned['cluster'] == clu
        axes[i, j].hist(data, bins=20, color=colors[j], edgecolor='w', alpha
        axes[i, j].set_title(f'Cluster {cluster} - {feature}', fontsize=15)
        axes[i, j].set_xlabel('')
        axes[i, j].set_ylabel('')

# Adjusting layout to prevent overlapping
plt.tight_layout()
plt.show()
```


Cluster 0 - Days_Since_Last_Purchase
Cluster 1 - Days_Since_Last_Purchase
Cluster 2 - Days_Since_Last_Purchase
Cluster 0 - Total_Transactions
Cluster 1 - Total_Transactions
Cluster 2 - Total_Transactions
Cluster 0 - Total_Products_Purchased
Cluster 1 - Total_Products_Purchased
Cluster 2 - Total_Products_Purchased
Cluster 0 - Total_Spend
Cluster 1 - Total_Spend
Cluster 2 - Total_Spend
Cluster 0 - Average_Transaction_Value
Cluster 1 - Average_Transaction_Value
Cluster 2 - Average_Transaction_Value
Cluster 0 - Unique_Products_Purchased
Cluster 1 - Unique_Products_Purchased
Cluster 2 - Unique_Products_Purchased
Cluster 0 - Average_Days_Between_Purchases
Cluster 1 - Average_Days_Between_Purchases
Cluster 2 - Average_Days_Between_Purchases
Cluster 0 - Day_Of_Week
Cluster 1 - Day_Of_Week
Cluster 2 - Day_Of_Week
Cluster 0 - Hour
Cluster 1 - Hour
Cluster 2 - Hour
Cluster 0 - Is_UK
Cluster 1 - Is_UK
Cluster 2 - Is_UK
Cluster 0 - Cancellation_Frequency_x
Cluster 1 - Cancellation_Frequency_x
Cluster 2 - Cancellation_Frequency_x



Recommendation System

In [80]:

```
# Step 1: Extract the CustomerIDs of the outliers and remove their transacti
outlier_customer_ids = outliers_data['CustomerID'].astype('float').unique()
df_filtered = df[~df['CustomerID'].isin(outlier_customer_ids)]

# Step 2: Ensure consistent data type for CustomerID across both dataframes
customer_data_cleaned['CustomerID'] = customer_data_cleaned['CustomerID'].as

# Step 3: Merge the transaction data with the customer data to get the clust
merged_data = df_filtered.merge(customer_data_cleaned[['CustomerID', 'cluste

# Step 4: Identify the top 10 best-selling products in each cluster based or
best_selling_products = merged_data.groupby(['cluster', 'StockCode', 'Descri
best_selling_products = best_selling_products.sort_values(by=['cluster', 'Qu
top_products_per_cluster = best_selling_products.groupby('cluster').head(10)
```

```
# Step 5: Create a record of products purchased by each customer in each clu
customer_purchases = merged_data.groupby(['CustomerID', 'cluster', 'StockCoc

# Step 6: Generate recommendations for each customer in each cluster
recommendations = []
for cluster in top_products_per_cluster['cluster'].unique():
    top_products = top_products_per_cluster[top_products_per_cluster['cluste
    customers_in_cluster = customer_data_cleaned[customer_data_cleaned['clus

    for customer in customers_in_cluster:
        # Identify products already purchased by the customer
        customer_purchased_products = customer_purchases[(customer_purchases
                                                         (customer_purchases

        # Find top 3 products in the best-selling list that the customer has
        top_products_not_purchased = top_products[~top_products['StockCode']
        top_3_products_not_purchased = top_products_not_purchased.head(3)

        # Append the recommendations to the list
        recommendations.append([customer, cluster] + top_3_products_not_purc

# Step 7: Create a dataframe from the recommendations list and merge it with
recommendations_df = pd.DataFrame(recommendations, columns=['CustomerID', 'c
                                                  'Rec2_StockCode', 'Rec2_Des
customer_data_with_recommendations = customer_data_cleaned.merge(recommendat
```

In [81]:

```
# Display 10 random rows from the customer_data_with_recommendations datafra
customer_data_with_recommendations.set_index('CustomerID').iloc[:, -6:].samp
```

Out[81]:

CustomerID	Rec1_StockCode	Rec1_Description	Rec2_StockCode	Rec2_Descri
15721.0	84077	WORLD WAR 2 GLIDERS ASSTD DESIGNS	84879	ASSC COLOUR ORNA
15713.0	84077	WORLD WAR 2 GLIDERS ASSTD DESIGNS	84879	ASSC COLOUR ORNA
16877.0	84077	WORLD WAR 2 GLIDERS ASSTD DESIGNS	84879	ASSC COLOUR ORNA
16833.0	84077	WORLD WAR 2 GLIDERS ASSTD DESIGNS	84879	ASSC COLOUR ORNA
14757.0	84077	WORLD WAR 2 GLIDERS ASSTD DESIGNS	85123A	WHITE HAN HEART T- HC
14297.0	84077	WORLD WAR 2 GLIDERS ASSTD DESIGNS	17003	BROCADE F
14560.0	22616	PACK OF 12 LONDON TISSUES	84077	WORLD V GLIDERS A DES
17391.0	84077	WORLD WAR 2 GLIDERS ASSTD DESIGNS	84879	ASSC COLOUR ORNA
17716.0	22616	PACK OF 12 LONDON TISSUES	84077	WORLD V GLIDERS A DES
13080.0	84077	WORLD WAR 2 GLIDERS ASSTD DESIGNS	84879	ASSC COLOUR ORNA

In [1]:

```
import zipfile
import os
import pandas as pd

# Step 1: Unzip the file
zip_file_path = '/content/archive (8).zip'  # Path to your zip file
extract_folder = '/content/extracted_files/'

# Create a directory for extracted files
os.makedirs(extract_folder, exist_ok=True)

# Extract the zip file
try:
    with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
        zip_ref.extractall(extract_folder)
    print("Files extracted successfully.")
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
except FileNotFoundError:
    print("The zip file was not found. Please check the file path.")
    exit()
except Exception as e:
    print(f"An error occurred while extracting the zip file: {e}")
    exit()

# List extracted files
extracted_files = os.listdir(extract_folder)
print("Extracted files:", extracted_files)

# Step 2: Load the CSV file
csv_file_name = 'data.csv'  # Replace with the actual CSV file name
csv_file_path = os.path.join(extract_folder, csv_file_name)

try:
    # Try reading the CSV file with different encodings
    customer_data_with_recommendations = pd.read_csv(csv_file_path, encoding
    print("CSV file loaded successfully.")
except FileNotFoundError:
    print("The CSV file was not found. Please check the file path.")
    exit()
except UnicodeDecodeError:
    print("Encoding error. Try a different encoding like 'ISO-8859-1'.")
    exit()
except Exception as e:
    print(f"An unexpected error occurred while reading the CSV file: {e}")
    exit()

# Confirm column names
print("Column names in the dataset:", customer_data_with_recommendations.col

# Step 3: Filter and Display Data
# Prompt user for CustomerID and Country
customer_id = input("Enter CustomerID: ")
country = input("Enter Country: ")

# Convert CustomerID to integer
try:
    customer_id = int(customer_id)
except ValueError:
    print("Invalid CustomerID. Please enter a valid integer.")
    exit()

# Check if 'Country' column exists
if 'Country' not in customer_data_with_recommendations.columns:
    print("'Country' column not found in the dataset. Please check the datas
    exit()

# Check if the entered country is valid
if country not in customer_data_with_recommendations['Country'].values:
    print(f"No data found for the country '{country}'. Please check the cour
    exit()

# Filter the dataframe based on CustomerID and Country
filtered_data = customer_data_with_recommendations[
```

```
    (customer_data_with_recommendations['CustomerID'] == customer_id) &
    (customer_data_with_recommendations['Country'] == country)
]

# Display 10 random rows of recommendations (or fewer if we don’t have that
if not filtered_data.empty:
    recommendations = filtered_data.set_index('CustomerID').iloc[:, -6:].sam
    print(recommendations)
else:
    print(f"No data found for CustomerID {customer_id} in {country}")
```

```
Files extracted successfully.
Extracted files: ['data.csv']
CSV file loaded successfully.
Column names in the dataset: Index(['InvoiceNo', 'StockCode', 'Description',
'Quantity', 'InvoiceDate',
       'UnitPrice', 'CustomerID', 'Country'],
      dtype='object')
Enter CustomerID: 17850
Enter Country: United Kingdom
           StockCode                         Description  Quantity  \
CustomerID
17850.0       85123A  WHITE HANGING HEART T-LIGHT HOLDER         8
17850.0        20679                 EDWARDIAN PARASOL RED         6
17850.0        71053                   WHITE METAL LANTERN        12
17850.0        22632               HAND WARMER RED POLKA DOT         6
17850.0       85123A  WHITE HANGING HEART T-LIGHT HOLDER         6
17850.0        21730   GLASS STAR FROSTED T-LIGHT HOLDER         8
17850.0        37370              RETRO COFFEE MUGS ASSORTED         6
17850.0        21871                   SAVE THE PLANET MUG         6
17850.0        82486   WOOD S/3 CABINET ANT WHITE FINISH         4
17850.0        21730   GLASS STAR FROSTED T-LIGHT HOLDER         6

                InvoiceDate  UnitPrice         Country
CustomerID
17850.0     12/1/2010 11:33       2.55  United Kingdom
17850.0     12/2/2010 14:04       4.95  United Kingdom
17850.0     12/2/2010 12:23       3.39  United Kingdom
17850.0     12/2/2010 12:24       1.85  United Kingdom
17850.0      12/2/2010 8:34       2.55  United Kingdom
17850.0     12/2/2010 14:04       4.25  United Kingdom
17850.0      12/2/2010 9:44       1.06  United Kingdom
17850.0      12/2/2010 9:41       1.06  United Kingdom
17850.0     12/1/2010 11:33       6.95  United Kingdom
17850.0     12/2/2010 14:06       4.25  United Kingdom
```