E-Commerce Customer Segmentation

November 20, 2025

1 E-Commerce Customer Categorization and Analyzer

1: Data Preparation

2: Exploring the Contents of Variables

```
Analysis by Countries
Customers and Products
Cancelled Orders
StockCode
Basket Price
```

3: Products Description

```
Keywords
Categorical Keyword
Data encoding
```

Creating clusters of products

```
K-Means Clustering & silhouette_score
Word Cloud
```

4: Customer Categories

5: Classifying Customers *Train X, Y*

Models - Support Vector Machine Classifier (SVC) - Logistic Regression - k-Nearest Neighbors - Decision Tree - Random Forest

Conclusion

Project Overview

The E-Commerce Customer Categorization and Analyzer project focuses on analyzing a one-year purchasing database to provide insights into customer behavior and product categorization. The project utilizes Jupyter Notebook as the primary development environment and is optimized for version 7.0.8. However, certain commands or functions may differ in syntax or behavior depending on the version of Jupyter Notebook or the libraries being used.

Version Compatibility

If you are using a different version of Jupyter Notebook, you may need to adjust specific commands or functions. It is recommended to refer to the official documentation or search platforms like Stack Overflow for version-specific solutions. For example, some pandas methods or matplotlib plotting techniques might vary between versions.

Dependencies

This project requires the following libraries: - `numpy`: For numerical computations and handling arrays. - `pandas`: For data manipulation and analysis. - `matplotlib`: For creating static visualizations. - `seaborn`: For advanced data visualization based on matplotlib. - `scikit-learn`: For implementing machine learning algorithms and evaluation metrics. - `plotly`: For interactive and dynamic visualizations. - `nltk`: For natural language processing tasks. tasks.

Dataset Link: https://www.kaggle.com/datasets/carrie1/ecommerce-data

1.1 1: Data Preparation

[1]:
```
#Module#01 -> jawad1A
import sys
import pandas as pd
print("Location of Python interpreter:",sys.executable)

#jawad1A1 + jawad1A2
dataset = pd.read_csv('E-Commerce-Customers-data.csv',encoding="ISO-8859-1",␣
 ↪dtype={'CustomerID': str,'InvoiceID': str})

#jawad1A2
dataset['InvoiceDate'] = pd.to_datetime(dataset['InvoiceDate'])
print('\nDataframe dimensions:', dataset.shape)
print("\n First 5 Entries")
display(dataset.head(5))

#jawad1A3
print("\n\nmissing values")
col_types = pd.DataFrame(dataset.dtypes).T.rename(index={0: 'column type'})
missing_values = pd.DataFrame(dataset.isnull().sum()).T.rename(index={0: 'total␣
 ↪missing'})
missing_percentage = pd.DataFrame((dataset.isnull().sum() / len(dataset)) *␣
 ↪100).T.rename(index={0: 'missing percentage'})
col_info = pd.concat([col_types, missing_values, missing_percentage])
display(col_info)

#jawad1A4
dataset = dataset.dropna()
print('\nDataframe dimensions after dropping missing values:', dataset.shape)
print("\n\nmissing values after dropping missing values")
col_types = pd.DataFrame(dataset.dtypes).T.rename(index={0: 'column type'})
missing_values = pd.DataFrame(dataset.isnull().sum()).T.rename(index={0: 'total␣
 ↪missing'})
```

```
missing_percentage = pd.DataFrame((dataset.isnull().sum() / len(dataset)) *␣
 ↪100).T.rename(index={0: 'missing percentage'})
col_info = pd.concat([col_types, missing_values, missing_percentage])
display(col_info)

#jawad1A5
duplicates = dataset.duplicated()
print(f"\nNumber of duplicate rows: {duplicates.sum()}")
dataset = dataset.drop_duplicates()
print('\nDataframe dimensions after removing duplicates:', dataset.shape)
duplicates = dataset.duplicated().sum()
print(f"\nNumber of duplicate rows after cleaning: {duplicates.sum()}")
```

```
Location of Python interpreter:
C:\Users\rakes\AppData\Local\Programs\Python\Python312\python.exe

Dataframe dimensions: (541909, 8)

 First 5 Entries
```

	InvoiceNo	StockCode	Description	Quantity	\
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	
1	536365	71053	WHITE METAL LANTERN	6	
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	

	InvoiceDate	UnitPrice	CustomerID	Country
0	2010-12-01 08:26:00	2.55	17850	United Kingdom
1	2010-12-01 08:26:00	3.39	17850	United Kingdom
2	2010-12-01 08:26:00	2.75	17850	United Kingdom
3	2010-12-01 08:26:00	3.39	17850	United Kingdom
4	2010-12-01 08:26:00	3.39	17850	United Kingdom

```
missing values
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	\
column type	object	object	object	int64	datetime64[ns]	
total missing	0	0	1454	0	0	
missing percentage	0.0	0.0	0.268311	0.0	0.0	

	UnitPrice	CustomerID	Country
column type	float64	object	object
total missing	0	135080	0
missing percentage	0.0	24.926694	0.0

```
Dataframe dimensions after dropping missing values: (406829, 8)
```

```
missing values after dropping missing values

                    InvoiceNo StockCode Description Quantity     InvoiceDate  \
column type            object    object      object    int64  datetime64[ns]
total missing               0         0           0        0               0
missing percentage        0.0       0.0         0.0      0.0             0.0

                    UnitPrice CustomerID Country
column type           float64     object  object
total missing               0          0       0
missing percentage        0.0        0.0     0.0


Number of duplicate rows: 5225

Dataframe dimensions after removing duplicates: (401604, 8)

Number of duplicate rows after cleaning: 0
```

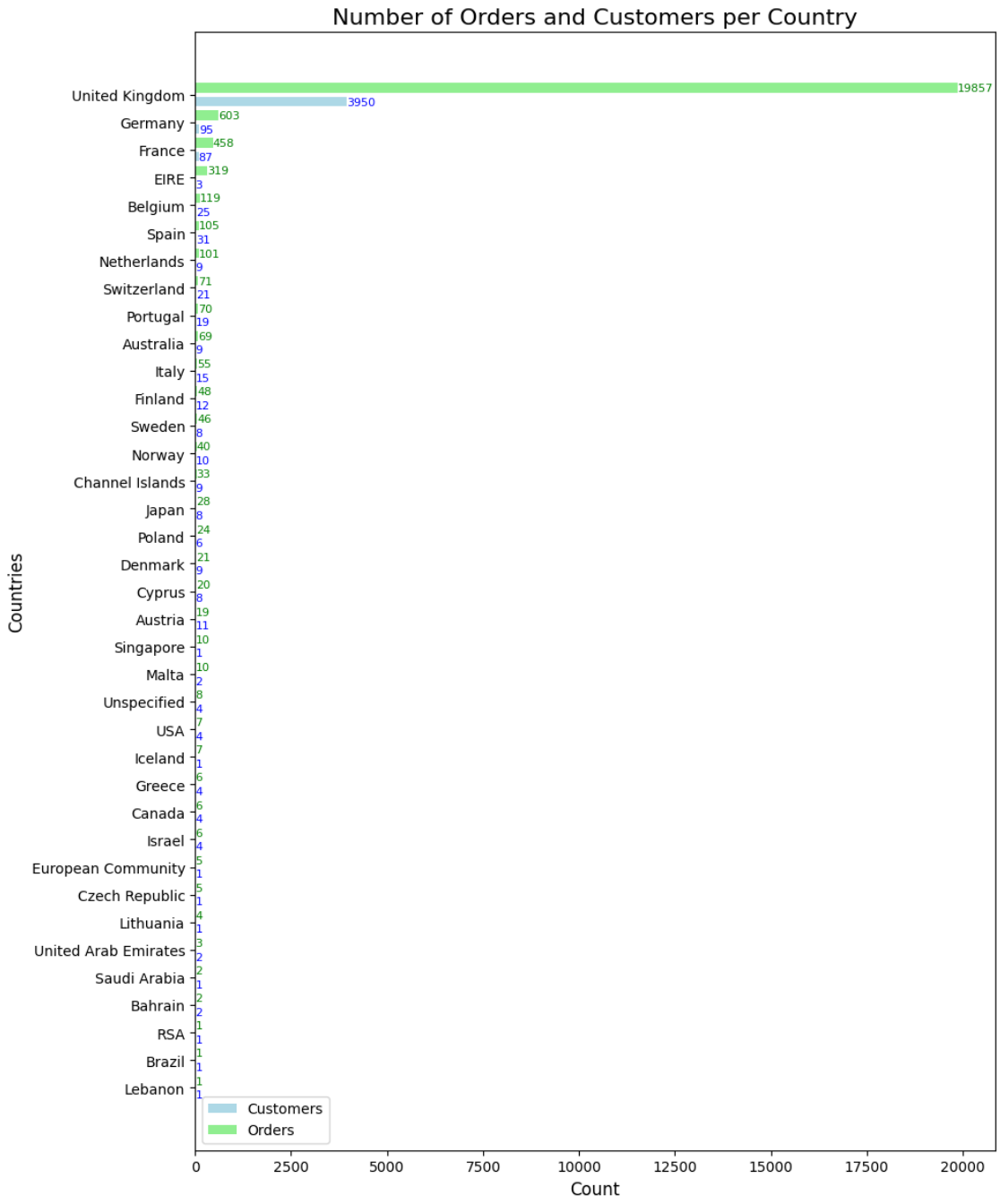
1.2 2: Exploring the Contents of variables

1.2.1 Analysis by Countries

[2]:
```
#Module#02 -> jawad2A
import matplotlib.pyplot as plt
import plotly.express as px
import pandas as pd
import numpy as np

#jawad2A1
customers_per_country = dataset.groupby('Country')['CustomerID'].nunique()
orders_per_country = dataset.groupby('Country')['InvoiceNo'].nunique()
df = pd.DataFrame({
    'Number of Customers': customers_per_country,
    'Number of Orders': orders_per_country
})
df = df.sort_values(by='Number of Orders')
print(f"No. of countries in the dataframe: {len(df)}")
fig, ax = plt.subplots(figsize=(10, 12))
bar_width = 0.35
index = np.arange(len(df))
bars1 = ax.barh(index, df['Number of Customers'], bar_width, color='lightblue',␣
 ↪label='Customers')
bars2 = ax.barh(index + 0.5, df['Number of Orders'], bar_width,␣
 ↪color='lightgreen', label='Orders')
for bar, count in zip(bars1, df['Number of Customers']):
```

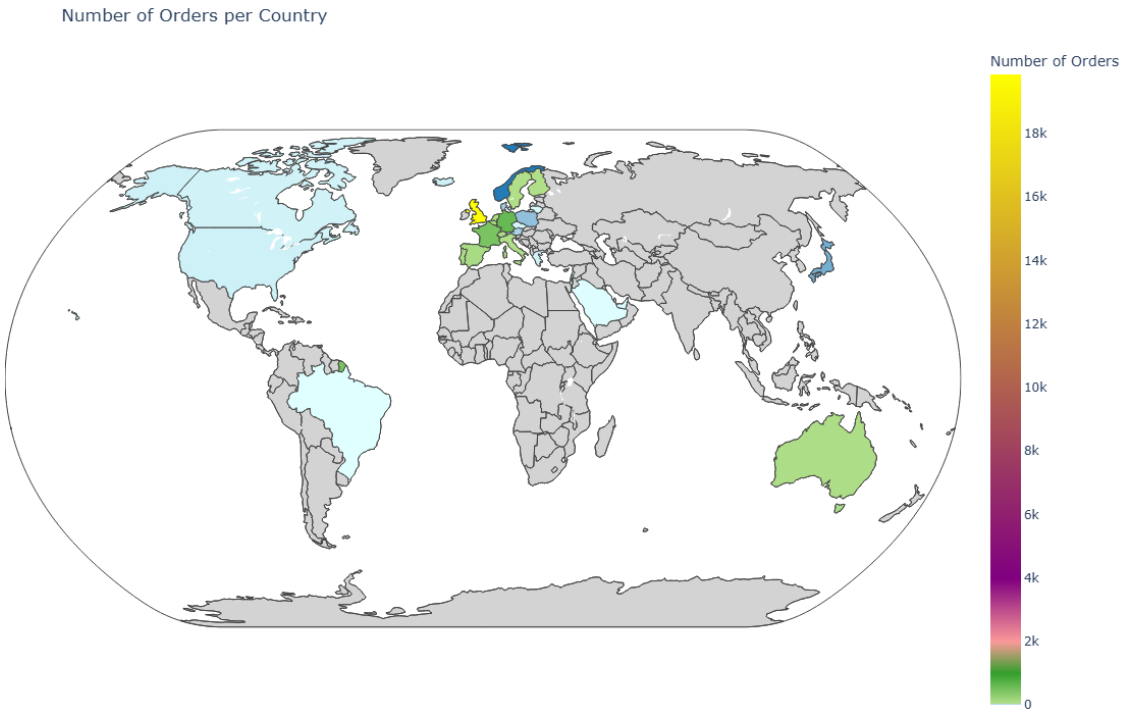
```
    ax.text(count + 1, bar.get_y() + bar.get_height() / 2, str(count),␣
 ↪va='center', fontsize=8, color='blue')
for bar, count in zip(bars2, df['Number of Orders']):
    ax.text(count + 1, bar.get_y() + bar.get_height() / 2, str(count),␣
 ↪va='center', fontsize=8, color='green')
ax.set_title('Number of Orders and Customers per Country', fontsize=16)
ax.set_xlabel('Count', fontsize=12)
ax.set_ylabel('Countries', fontsize=12)
ax.set_yticks(index + 0.25)
ax.set_yticklabels(df.index)
ax.legend()
plt.tight_layout()
plt.show()
orders_per_country = dataset.groupby('Country')['InvoiceNo'].nunique().
 ↪reset_index()
orders_per_country.columns = ['Country', 'Number of Orders']
custom_color_scale = [
    [0, 'rgb(224,255,255)'],
    [0.001, 'rgb(166,206,227)'],
    [0.002, 'rgb(31,120,180)'],
    [0.002, 'rgb(178,223,138)'],
    [0.05, 'rgb(51,160,44)'],
    [0.10, 'rgb(251,154,153)'],
    [0.20, 'rgb(128,0,128)'],
    [1, 'rgb(255,255,0)']
]
```

```
No. of countries in the dataframe: 37
```



[3]:
```
#Under -> Module#02 -> jawad2A
#jawad2A2
fig = px.choropleth(
    orders_per_country,
    locations="Country",
    locationmode="country names",
```

```
    color="Number of Orders",
    hover_name="Country",
    color_continuous_scale=custom_color_scale,
    title="Number of Orders per Country"
)
fig.update_geos(
    projection_type="natural earth",
    showcountries=True,
    showcoastlines=True,
    showland=True,
    landcolor="lightgrey"
)
fig.update_layout(
    margin={"r":0,"t":40,"l":0,"b":0},
    height=700,
    width=1000,
)
fig.show()
```



1.2.2 Customers and products

The dataframe contains `401,604 entries`. What are the number of users and products in these entries?

[4]:
```
#Module#03 -> jawad3A
#jawad3A1
pd.DataFrame([{'Products': len(dataset['StockCode'].value_counts()),
               'Transactions': len(dataset['InvoiceNo'].value_counts()),
               'Customers': len(dataset['CustomerID'].value_counts()),}],
             columns = ['Products', 'Transactions', 'Customers'], index =␣
 ↪['Quantity'])
```

[4]:

	Products	Transactions	Customers
Quantity	3684	22190	4372

[5]:
```
#jawad3A2
customer_summary = dataset.groupby(by=['CustomerID'], as_index=False).agg(
    Total_Purchases=('InvoiceNo', 'nunique'),
    Total_Quantity=('Quantity', 'sum')
)
customer_summary = customer_summary.sort_values('CustomerID', ascending=True)
display(customer_summary.head(10))
```

	CustomerID	Total_Purchases	Total_Quantity
0	12346	2	0
1	12347	7	2458
2	12348	4	2341
3	12349	1	631
4	12350	1	197
5	12352	11	470
6	12353	1	20
7	12354	1	530
8	12355	1	240
9	12356	3	1591

The first lines of this list shows Quantity **0** this indicates transactions that has been canceled.

[6]:
```
#Module#03 -> jawad3A2
print("Our Top 10 highr Purchases Customers")
customer_summary = dataset.groupby(by=['CustomerID'], as_index=False).agg(
    Total_Purchases=('InvoiceNo', 'nunique'),
    Total_Quantity=('Quantity', 'sum')
)
customer_summary = customer_summary.sort_values('Total_Quantity',␣
 ↪ascending=False)
display(customer_summary.head(10))
```

```
Our Top 10 highr Purchases Customers
```

	CustomerID	Total_Purchases	Total_Quantity
1703	14646	77	196719
55	12415	26	77242
1895	14911	248	77155

```
3758      17450                55           69009
4233      18102                62           64122
3801      17511                46           63012
1005      13694                60           61899
1447      14298                45           58021
1345      14156                66           56908
3202      16684                31           49390
```

1.2.3 Cancell orders

[7]:
```
display(dataset.sort_values('CustomerID')[:5])
```

```
       InvoiceNo StockCode                          Description  Quantity  \
61619     541431     23166       MEDIUM CERAMIC TOP STORAGE JAR     74215
61624    C541433     23166       MEDIUM CERAMIC TOP STORAGE JAR    -74215
286623    562032     22375    AIRLINE BAG VINTAGE JET SET BROWN         4
72260     542237     84991          60 TEATIME FAIRY CAKE CASES        24
14943     537626     22772  PINK DRAWER KNOB ACRYLIC EDWARDIAN        12

                InvoiceDate  UnitPrice CustomerID         Country
61619   2011-01-18 10:01:00       1.04      12346  United Kingdom
61624   2011-01-18 10:17:00       1.04      12346  United Kingdom
286623  2011-08-02 08:48:00       4.25      12347         Iceland
72260   2011-01-26 14:30:00       0.55      12347         Iceland
14943   2010-12-07 14:57:00       1.25      12347         Iceland
```

The existence of entries with the prefix `C` for the `InvoiceNo` this indicates transactions that have been canceled.

[8]:
```
#Module#04 -> jawad4A
#jawad4A1
cancelled_orders = dataset[dataset['InvoiceNo'].str.startswith('C')]
total_orders = dataset['InvoiceNo'].nunique()
cancelled_count = cancelled_orders['InvoiceNo'].nunique()
print(f"Number of total orders: {total_orders}")
print(f"Number of cancelled orders: {cancelled_count}")
percentage_cancelled = (cancelled_count / total_orders) * 100
print(f"Percentage of cancelled orders: {percentage_cancelled:.2f}%")
display(cancelled_orders)
```

```
Number of total orders: 22190
Number of cancelled orders: 3654
Percentage of cancelled orders: 16.47%
```

```
       InvoiceNo StockCode                       Description  Quantity  \
141      C536379         D                          Discount        -1
154      C536383    35004C   SET OF 3 COLOURED  FLYING DUCKS        -1
235      C536391     22556    PLASTERS IN TIN CIRCUS PARADE        -12
236      C536391     21984  PACK OF 12 PINK PAISLEY TISSUES        -24
237      C536391     21983  PACK OF 12 BLUE PAISLEY TISSUES        -24
```

```
…          …       …                                  …       …
540449   C581490     23144   ZINC T-LIGHT HOLDER STARS SMALL       -11
541541   C581499         M                          Manual        -1
541715   C581568     21258       VICTORIAN SEWING BOX LARGE        -5
541716   C581569     84978  HANGING HEART JAR T-LIGHT HOLDER        -1
541717   C581569     20979     36 PENCILS TUBE RED RETROSPOT        -5

                InvoiceDate  UnitPrice CustomerID         Country
141    2010-12-01 09:41:00      27.50      14527  United Kingdom
154    2010-12-01 09:49:00       4.65      15311  United Kingdom
235    2010-12-01 10:24:00       1.65      17548  United Kingdom
236    2010-12-01 10:24:00       0.29      17548  United Kingdom
237    2010-12-01 10:24:00       0.29      17548  United Kingdom
…                      …         …          …              …
540449 2011-12-09 09:57:00       0.83      14397  United Kingdom
541541 2011-12-09 10:28:00     224.69      15498  United Kingdom
541715 2011-12-09 11:57:00      10.95      15311  United Kingdom
541716 2011-12-09 11:58:00       1.25      17315  United Kingdom
541717 2011-12-09 11:58:00       1.25      17315  United Kingdom

[8872 rows x 8 columns]
```

1.2.4 StockCode

Above, it has been seen that some values of the `StockCode` variable indicate a particular transaction (i.e. `D for Discount`). let's check the contents of this variable by looking for the set of codes that would contain only letters:

[9]:
```
#Module#05 -> jawad5A
#jawad5A1
list_special_codes = dataset[dataset['StockCode'].str.contains('^[a-zA-Z]+',␣
 ↪regex=True)]['StockCode'].unique()
list_special_codes
```

[9]:
```
array(['POST', 'D', 'C2', 'M', 'BANK CHARGES', 'PADS', 'DOT', 'CRUK'],
      dtype=object)
```

[10]:
```
#Under Module#05 -> jawad5A
#jawad5A2
for code in list_special_codes:
    print("{:<15} -> {:<30}".format(code, dataset[dataset['StockCode'] ==␣
 ↪code]['Description'].unique()[0]))
```

```
POST            -> POSTAGE
D               -> Discount
C2              -> CARRIAGE
M               -> Manual
BANK CHARGES    -> Bank Charges
PADS            -> PADS TO MATCH ALL CUSHIONS
```

```
DOT             -> DOTCOM POSTAGE
CRUK            -> CRUK Commission
```

1.2.5 Basket Price

Will create a `new variable` that indicates the `total price of every purchase`:

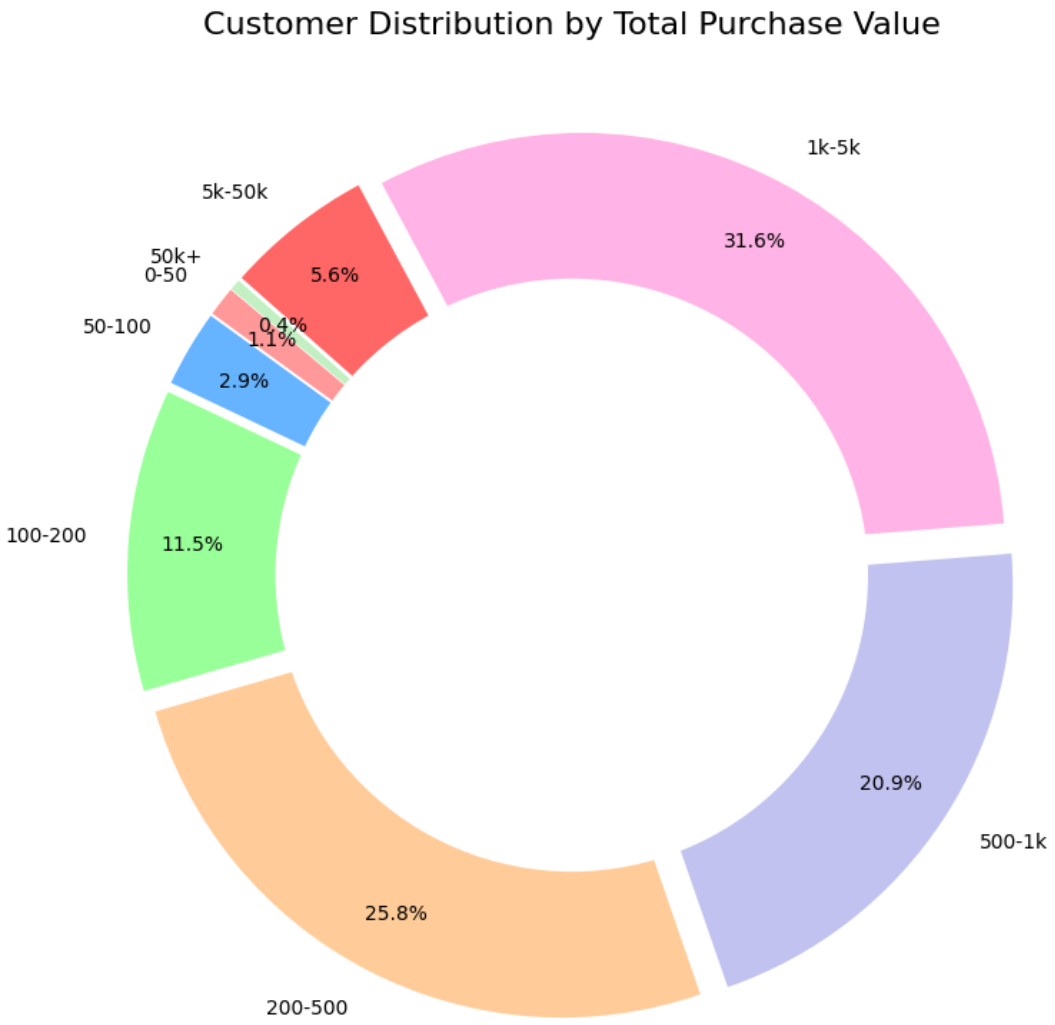
[11]:
```
#Module#06 -> jawad6A
#jawad6A1
dataset['TotalPrice']=dataset['Quantity']*dataset['UnitPrice']
print("Our Top 10 Customers")
customer_summary = dataset.groupby(by=['CustomerID'], as_index=False).agg(
    Total_Purchases=('InvoiceNo', 'nunique'),
    Total_Quantity=('Quantity', 'sum') ,
    Total_Price=('TotalPrice', 'sum')
)

#jawad6A2
customer_summary = customer_summary.sort_values('Total_Price', ascending=False)
display(customer_summary.head(10))
```

```
Our Top 10 Customers
```

	CustomerID	Total_Purchases	Total_Quantity	Total_Price
1703	14646	77	196719	279489.02
4233	18102	62	64122	256438.49
3758	17450	55	69009	187322.17
1895	14911	248	77155	132458.73
55	12415	26	77242	123725.45
1345	14156	66	56908	113214.59
3801	17511	46	63012	88125.38
3202	16684	31	49390	65892.08
1005	13694	60	61899	62690.54
2192	15311	118	37673	59284.19

[12]:
```
#Module#07 -> jawad7A
#jawad7A1
import matplotlib.pyplot as plt
bins = [0, 50, 100, 200, 500, 1000, 5000, 50000, float('inf')]
labels = ['0-50', '50-100', '100-200', '200-500','500-1k','1k-5k', '5k-50k',␣
 ↪'50k+']
customer_summary['Price_Category'] = pd.cut(customer_summary['Total_Price'],␣
 ↪bins=bins, labels=labels, right=False)
category_counts = customer_summary['Price_Category'].value_counts().sort_index()
category_percentages = (category_counts / category_counts.sum()) * 100

#jawad7A2
plt.figure(figsize=(8, 8))
```

```
colors = ['#ff9999', '#66b3ff', '#99ff99', '#ffcc99', '#c2c2f0', '#ffb3e6',␣
 ↪'#ff6666', '#c2f0c2']
explode = [0.05] * len(category_percentages)
plt.pie(
    category_percentages,
    labels=category_percentages.index,
    autopct='%1.1f%%',
    startangle=140,
    colors=colors,
    explode=explode,
    pctdistance=0.85
)
centre_circle = plt.Circle((0, 0), 0.70, fc='white')
plt.gca().add_artist(centre_circle)
plt.title('Customer Distribution by Total Purchase Value', fontsize=16)
plt.tight_layout()
plt.show()
```



[13]:
```
category_counts
```

[13]:
```
Price_Category
0-50          49
50-100       127
100-200      499
200-500     1118
500-1k       907
1k-5k       1368
5k-50k       244
50k+          18
Name: count, dtype: int64
```

1.3 3: Products Description

1.3.1 Keywords

[14]:
```
#Module#08 -> jawad8A
#jawad8A1
total_words = dataset['Description'].apply(lambda x: len(str(x).split())).sum()
print(f"Total number of words in the 'Description' column: {total_words}")
unique_words = set(" ".join(dataset['Description'].dropna()).split())
total_unique_words = len(unique_words)
list_words=unique_words
print(f"Total number of unique words in the 'Description' column:␣
 ↪{total_unique_words}")
```

```
Total number of words in the 'Description' column: 1763761
Total number of unique words in the 'Description' column: 2250
```

[15]:
```
#Under Module#08 -> jawad8A
#jawad8A2
from collections import Counter
import re
descriptions = " ".join(dataset['Description'].dropna()).lower()
descriptions = re.sub(r'[^a-z\s]', '', descriptions)
words = descriptions.split()
word_counts = Counter(words)
total_unique_words = len(word_counts)
print(f"Total keywords: {total_unique_words}")
word_counts
```

```
Total keywords: 2106
```

[15]:
```
Counter({'set': 47385,
         'of': 41471,
         'bag': 38090,
         'red': 32254,
         'heart': 29375,
         'retrospot': 26754,
         'vintage': 25761,
         'design': 23663,
         'pink': 20264,
         'christmas': 19082,
         'box': 18146,
         'cake': 16372,
         'white': 16178,
         'metal': 15739,
         'jumbo': 15587,
         'lunch': 14980,
         'blue': 13902,
         'hanging': 13025,
```

```
'holder': 12854,
'sign': 12682,
'pack': 11987,
'tlight': 11415,
'paper': 10827,
'small': 10441,
'card': 10080,
'wooden': 10067,
'cases': 9461,
'glass': 9125,
'tea': 8862,
'polkadot': 8852,
'decoration': 8636,
'spaceboy': 8611,
'bottle': 8585,
'in': 8456,
'and': 7909,
'home': 7901,
'hot': 7881,
'pantry': 7670,
'large': 7587,
'tin': 7539,
'water': 7509,
'regency': 7116,
'with': 7040,
'ceramic': 7005,
'doormat': 6808,
'paisley': 6616,
'dolly': 6594,
'ivory': 6580,
'cream': 6495,
'bunting': 6309,
'rose': 6250,
'wrap': 6249,
'green': 6239,
'mug': 6218,
'feltcraft': 6193,
'girl': 6115,
'assorted': 6054,
'mini': 6011,
'clock': 5991,
'love': 5844,
'party': 5810,
'wicker': 5786,
'tins': 5549,
'frame': 5543,
'kit': 5534,
```

```
'childrens': 5433,
's': 5245,
'antique': 5195,
'black': 5140,
'drawer': 5137,
'fairy': 5121,
'colour': 5082,
'garden': 5046,
'woodland': 4980,
'silver': 4780,
'bowl': 4752,
'wall': 4712,
'star': 4655,
'wood': 4627,
'charlotte': 4488,
'birthday': 4417,
'bird': 4363,
'strawberry': 4228,
'union': 4184,
'hand': 4136,
'skull': 4045,
'zinc': 4031,
'hearts': 4027,
'jam': 3936,
'jar': 3934,
'gift': 3900,
'cutlery': 3840,
'knob': 3824,
'candles': 3756,
'pencils': 3713,
'warmer': 3669,
'cup': 3604,
'alarm': 3598,
'bakelike': 3575,
'traditional': 3549,
'doily': 3536,
'baking': 3515,
'egg': 3509,
'sweetheart': 3509,
'chocolate': 3494,
'flower': 3471,
'london': 3468,
'french': 3444,
'storage': 3405,
'suki': 3404,
'trinket': 3402,
'sweet': 3394,
```

```
'candle': 3335,
'hook': 3219,
'tube': 3204,
'tier': 3200,
'tree': 3157,
'plasters': 3141,
'keep': 3140,
'travel': 3138,
'alphabet': 3137,
'calm': 3091,
'napkins': 3035,
'ribbons': 3022,
'apples': 3011,
'butterfly': 2994,
'i': 2956,
'jack': 2932,
'slate': 2932,
'craft': 2927,
'enamel': 2926,
'cover': 2918,
'tissues': 2871,
'round': 2845,
'kitchen': 2843,
'chain': 2803,
'photo': 2779,
'leaf': 2746,
'gingham': 2736,
'skulls': 2706,
'cushion': 2677,
'spot': 2673,
'dog': 2673,
'stand': 2671,
'cookie': 2623,
'plate': 2600,
'scandinavian': 2551,
'finish': 2509,
'natural': 2488,
'light': 2475,
'apron': 2456,
'lights': 2449,
'diner': 2444,
'magnets': 2436,
'saucer': 2430,
'cards': 2416,
'shopper': 2366,
'boxes': 2364,
'spotty': 2346,
```

```
'wallet': 2345,
'coffee': 2321,
'babushka': 2279,
'reel': 2269,
'ribbon': 2268,
'cakestand': 2250,
'sewing': 2224,
'door': 2196,
'edwardian': 2193,
'magic': 2183,
'teacup': 2180,
'english': 2170,
'garland': 2120,
'art': 2108,
'cabinet': 2085,
'circus': 2081,
'parade': 2081,
'painted': 2074,
'felt': 2070,
'notebook': 2032,
'the': 2011,
'jug': 2008,
'making': 2003,
'recipe': 1994,
'block': 1979,
'pot': 1958,
'lantern': 1952,
'doll': 1944,
'picture': 1932,
'picnic': 1932,
'rabbit': 1929,
'easter': 1913,
'chalkboard': 1893,
'classic': 1879,
'doilies': 1852,
'milk': 1838,
'pad': 1811,
'roses': 1801,
'piece': 1797,
'cat': 1784,
'victorian': 1783,
'happy': 1776,
'book': 1766,
'bell': 1763,
'rack': 1724,
'table': 1710,
'hanger': 1674,
```

```
'yellow': 1674,
'cutters': 1674,
'snack': 1668,
'key': 1660,
'scales': 1649,
'night': 1647,
'retro': 1625,
'kneeling': 1617,
'parasol': 1616,
'mirror': 1603,
'owl': 1591,
'herb': 1567,
'tlights': 1518,
'building': 1510,
'word': 1510,
'style': 1493,
'food': 1492,
'rustic': 1491,
'apple': 1489,
'bin': 1473,
'top': 1472,
'board': 1470,
'doiley': 1461,
'pen': 1460,
'toy': 1459,
'stripe': 1457,
'childs': 1448,
'dinosaur': 1432,
'tray': 1426,
'gardeners': 1422,
'lid': 1406,
'ornament': 1405,
'treasure': 1391,
'money': 1373,
'biscuit': 1371,
'pears': 1349,
'tissue': 1346,
'marker': 1331,
'time': 1330,
'basket': 1323,
'decorations': 1322,
'coat': 1320,
'playing': 1301,
'sugar': 1300,
'tonic': 1281,
'floral': 1272,
'led': 1265,
```

```
'game': 1260,
'bells': 1259,
'can': 1257,
'wreath': 1240,
'flag': 1238,
'kids': 1235,
'bread': 1227,
'roll': 1220,
'cakes': 1215,
'umbrella': 1214,
'postage': 1212,
'container': 1204,
'sketchbook': 1203,
'towels': 1199,
'your': 1191,
'decorative': 1191,
'dinner': 1187,
'bank': 1180,
'popcorn': 1178,
'jigsaw': 1177,
'lace': 1175,
'ball': 1168,
'drawing': 1159,
'up': 1159,
'animals': 1157,
'on': 1150,
'ant': 1143,
'beaker': 1140,
'mat': 1140,
'giant': 1137,
'wire': 1134,
'acrylic': 1127,
'font': 1118,
'bunny': 1115,
'scented': 1113,
'lola': 1110,
'doorstop': 1109,
'tall': 1109,
'empire': 1108,
'knick': 1102,
'knack': 1102,
'bathroom': 1100,
'baby': 1100,
'poppys': 1099,
'playhouse': 1099,
'harmonica': 1098,
'gin': 1097,
```

```
'medium': 1095,
'jars': 1089,
'stickers': 1082,
'watering': 1079,
'tags': 1074,
'dish': 1065,
'shed': 1063,
'one': 1063,
'toadstool': 1061,
'stars': 1060,
'single': 1058,
'pan': 1051,
'chick': 1050,
'princess': 1045,
'teatime': 1045,
'friends': 1044,
'measuring': 1038,
'recycled': 1026,
'own': 1017,
'cornice': 1015,
'ring': 1007,
'balloons': 1001,
'cars': 1000,
'purse': 990,
'moulds': 989,
'shape': 984,
'pegs': 977,
'plant': 976,
'sticks': 972,
'parisienne': 958,
'folding': 955,
'fob': 952,
'botanical': 946,
'acapulco': 945,
'grey': 941,
'tidy': 932,
'a': 932,
'orange': 931,
'person': 931,
'glitter': 927,
'purple': 923,
'shoulder': 920,
'park': 913,
'breakfast': 912,
'frying': 901,
'rex': 900,
'cashcarry': 900,
```

```
'cotton': 899,
'toilet': 898,
'colouring': 897,
'diet': 896,
'school': 894,
'ruler': 894,
'jelly': 890,
'teapot': 887,
'printed': 886,
'pc': 882,
'fancy': 881,
'mint': 881,
'sticker': 878,
'cutter': 873,
'folkart': 872,
'peg': 872,
'oven': 867,
'charm': 864,
'or': 863,
'welcome': 861,
'owls': 859,
'please': 857,
'airline': 849,
'washing': 848,
'bowls': 846,
'pencil': 839,
'house': 836,
'list': 835,
'baroque': 833,
'chest': 832,
'brush': 831,
'magnetic': 829,
'lovebird': 828,
'chalk': 827,
'gumball': 817,
'scottie': 812,
'd': 807,
'cosy': 805,
'billboard': 799,
'gold': 798,
'daisy': 796,
'filigree': 791,
'angel': 789,
'stamp': 789,
'greeting': 786,
'flowers': 783,
'lip': 776,
```

```
'gloss': 776,
'triple': 774,
'shopping': 766,
'letters': 765,
'matches': 757,
'bundle': 755,
'books': 755,
'square': 752,
'calendar': 752,
'tape': 745,
'jingle': 740,
'snap': 735,
'stationery': 732,
'man': 728,
'charlie': 727,
'morris': 724,
'horse': 722,
'cottage': 721,
'crackers': 721,
'mould': 718,
'it': 716,
'feather': 713,
'dominoes': 712,
'lavender': 712,
'ice': 711,
'stocking': 706,
'asstd': 706,
'new': 703,
'place': 699,
'glove': 690,
'coloured': 679,
'fan': 679,
'plates': 676,
'memo': 675,
'spice': 675,
'heads': 674,
'tails': 674,
'candleholder': 674,
'bucket': 673,
'jewellery': 672,
'wine': 669,
'loaf': 669,
'skittles': 668,
'size': 668,
'cups': 662,
'shelf': 662,
'come': 659,
```

```
'no': 658,
'clothes': 655,
'gloves': 655,
'trellis': 646,
'honeycomb': 640,
'funky': 638,
'bath': 632,
'record': 632,
'thermometer': 631,
'tile': 631,
'knitting': 627,
'apothecary': 624,
'brown': 622,
'cook': 618,
'glaze': 616,
'sleigh': 615,
'grow': 613,
'stencil': 612,
'elephant': 609,
'swirly': 607,
'family': 602,
'for': 602,
'shaped': 600,
'rocking': 599,
'flock': 598,
'village': 597,
'crackle': 596,
'over': 595,
'g': 593,
'letter': 593,
'england': 589,
'jewelled': 584,
'pottering': 584,
'doughnut': 583,
'napkin': 583,
'world': 579,
'charlielola': 577,
'soap': 571,
'marbles': 568,
'gingerbread': 567,
'gymkhana': 559,
'backpack': 557,
'reindeer': 549,
'dove': 545,
'vanilla': 543,
'soft': 542,
'kings': 541,
```

```
'choice': 541,
'bicycle': 541,
'tv': 536,
'charms': 536,
'pattern': 535,
'crystal': 534,
'multi': 534,
'cocktail': 534,
'ladies': 532,
'embroidered': 531,
'holiday': 531,
'fruit': 530,
'collage': 528,
'chilli': 527,
'boxed': 527,
'homemade': 526,
'jazz': 524,
'towel': 524,
'cm': 523,
'nancy': 523,
'paint': 522,
'sympathy': 520,
'area': 519,
'cherry': 517,
'spoons': 516,
'am': 512,
'so': 512,
'poorly': 512,
'silk': 511,
'baths': 507,
'exercise': 507,
'writing': 506,
'fun': 506,
'images': 505,
'pocket': 504,
'deluxe': 501,
'organiser': 498,
'scent': 497,
'raffia': 497,
'fridge': 490,
'buffalo': 490,
'bill': 490,
'disco': 489,
'fruits': 489,
'scotty': 488,
'designs': 488,
'four': 487,
```

```
'you': 482,
'swallows': 477,
'war': 477,
'gliders': 477,
'string': 472,
'spring': 471,
'knitted': 467,
'toadstools': 467,
'ladder': 465,
'first': 464,
'me': 463,
'holly': 462,
'patterns': 461,
'blocks': 460,
'strongman': 460,
'manual': 460,
'make': 459,
'luggage': 459,
'hldr': 458,
'straws': 458,
'multicolour': 457,
'balloon': 453,
'stripey': 453,
'rotating': 453,
'singing': 453,
'clear': 452,
'little': 452,
'santa': 449,
'tag': 449,
'bracelet': 446,
'birdhouse': 446,
'stripes': 444,
'biscuits': 444,
'tote': 443,
'save': 440,
'planet': 440,
'mirrored': 440,
'fork': 440,
'day': 438,
'blossom': 433,
'to': 432,
'willie': 431,
'winkie': 431,
'rain': 427,
'rope': 425,
'potting': 418,
'chicken': 417,
```

```
'invites': 415,
'setting': 415,
'wick': 415,
'recycling': 414,
'coaster': 413,
'dairy': 413,
'maid': 413,
'mice': 410,
'office': 410,
'seaside': 407,
'fabric': 405,
'portrait': 405,
'trim': 405,
'flannel': 403,
'crochet': 400,
'pens': 400,
'fluted': 400,
'tutti': 399,
'frutti': 399,
'pannetone': 397,
'fawn': 397,
'aid': 396,
'enchanted': 396,
'cafe': 396,
'way': 390,
'polyester': 389,
'filler': 389,
'bunnies': 388,
'advent': 388,
'bottles': 388,
'pudding': 387,
'hen': 387,
'decoupage': 386,
'sundae': 386,
'chateau': 383,
'this': 382,
'turquoise': 381,
'patrolled': 380,
'teacoffeesugar': 378,
'mushroom': 375,
'open': 375,
'gentlemen': 372,
'spinning': 372,
'tops': 372,
'engraved': 372,
'parasols': 372,
'lipstick': 372,
```

```
'chicks': 372,
'mobile': 371,
'rectangle': 370,
'caravan': 369,
'cigar': 369,
'mix': 369,
'sack': 368,
'beaded': 367,
'jardin': 366,
'xcm': 366,
'beware': 366,
'basil': 364,
'toys': 363,
'w': 362,
'lamp': 362,
'hairband': 362,
'laundry': 362,
'spoon': 362,
'windmill': 360,
'incense': 359,
'poppies': 359,
'take': 358,
'leave': 358,
'diamante': 356,
'pin': 354,
'youre': 353,
'confusing': 353,
'sock': 353,
'bauble': 353,
'jet': 353,
'magazine': 353,
'skipping': 351,
'nesting': 350,
'paris': 348,
'danish': 348,
'sponge': 347,
'soldier': 344,
'romantic': 344,
'postcard': 344,
'reds': 343,
'leaves': 343,
'closed': 343,
'grand': 341,
'butter': 340,
'our': 340,
'soldiers': 339,
'birdcage': 337,
```

```
'modelling': 337,
'clay': 337,
'carry': 336,
'birdy': 334,
'print': 334,
'im': 332,
'colours': 331,
'woolly': 329,
'fashion': 329,
'notebooks': 329,
'parcel': 329,
'robot': 329,
'hottie': 328,
'petit': 328,
'bon': 328,
'fonts': 326,
'only': 326,
'brocante': 325,
'etched': 324,
'album': 324,
'angels': 323,
'crayons': 323,
'bedroom': 321,
'emily': 320,
'cut': 319,
'cats': 319,
'erasers': 318,
'stool': 318,
'cracker': 317,
'curio': 316,
'passport': 315,
'embossed': 311,
'rosie': 309,
'rounders': 308,
'choc': 308,
'blackblue': 304,
'ma': 304,
'campagne': 304,
'catch': 303,
'toast': 302,
'its': 302,
'ticket': 302,
'molly': 301,
'revolver': 300,
'strawbery': 298,
'marshmallows': 298,
'candy': 297,
```

```
'washroom': 296,
'big': 296,
'clips': 295,
'crate': 295,
'snowmen': 294,
'landmark': 294,
'magnet': 293,
'modern': 291,
'settings': 291,
'gazebo': 290,
'aged': 288,
'keepsake': 287,
'pinks': 287,
'piggy': 285,
'foil': 285,
'spade': 284,
'cupid': 283,
'baubles': 283,
'bingo': 283,
'stamped': 282,
'ollie': 282,
'beak': 282,
'folk': 281,
'bus': 281,
'secret': 281,
'carriage': 280,
'thank': 280,
'amelie': 279,
'number': 278,
'back': 278,
'eggs': 278,
'bull': 276,
'stick': 273,
'cowboys': 273,
'cacti': 273,
'bead': 273,
'calculator': 272,
'girls': 272,
'necklace': 272,
'swallow': 271,
'life': 270,
'mistletoe': 270,
'lolly': 269,
'snakes': 268,
'ladders': 268,
'placemats': 268,
'ass': 267,
```

```
'c': 267,
'envelopes': 266,
'lovebirds': 265,
'citronella': 265,
'oval': 264,
'psychedelic': 264,
'abc': 264,
'occupied': 264,
'vacant': 264,
'sheet': 263,
'pizza': 263,
'rolling': 263,
'is': 261,
'moody': 260,
'spots': 259,
'tokyo': 258,
'chocolatecandle': 258,
'spotted': 258,
'bags': 257,
'de': 256,
'sunflower': 256,
'livingroom': 255,
'puppies': 255,
'olivia': 254,
'drop': 253,
'crossbones': 251,
'scrubbing': 250,
'musical': 249,
'dispenser': 249,
'tealight': 248,
'keys': 248,
'croquet': 248,
'note': 248,
'dinosaurs': 247,
'boy': 244,
'giraffe': 244,
'miniature': 244,
'rocket': 243,
'wish': 243,
'refectory': 243,
'flytrap': 241,
'point': 241,
'pick': 241,
'queens': 240,
'guard': 240,
'porcelain': 239,
'teddy': 238,
```

```
'notepad': 237,
'boys': 236,
'coasters': 235,
'forest': 231,
'bomb': 231,
'pirate': 230,
'island': 229,
'cupcake': 228,
'badges': 228,
'quilted': 227,
'wool': 226,
'best': 225,
'garage': 225,
'deco': 224,
'donkey': 224,
'tail': 224,
'dec': 223,
'monkey': 223,
'journal': 222,
'le': 221,
'naughts': 221,
'crosses': 221,
'cardholder': 220,
'canvas': 219,
'chinese': 218,
'dressing': 217,
'plastic': 217,
'trowel': 217,
'tablecloth': 217,
'origami': 217,
'bubblegum': 214,
'salt': 213,
'pepper': 213,
'wc': 212,
'wash': 212,
'laurel': 212,
'brocade': 211,
'planter': 211,
'posy': 211,
'beads': 211,
'gardenia': 210,
'transport': 210,
'beach': 209,
'hammock': 209,
'memoboard': 209,
'chic': 209,
'flying': 207,
```

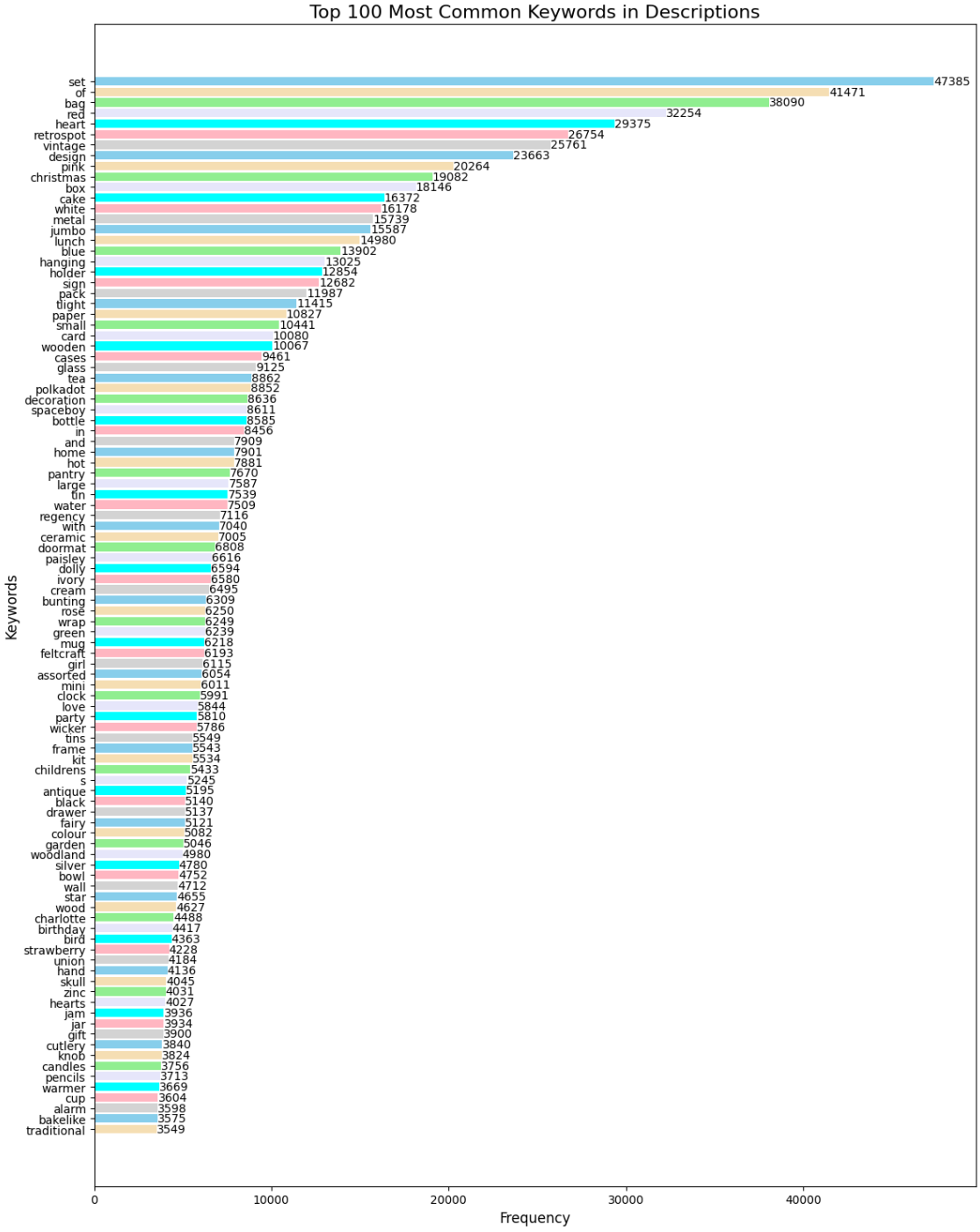
```
'gaolers': 207,
'purdey': 206,
'perfume': 205,
'repair': 204,
'six': 204,
'indians': 203,
'frog': 203,
'opener': 203,
'circular': 202,
'pearl': 202,
'personal': 200,
'doorsign': 200,
'chopsticks': 200,
'pillar': 200,
'circles': 200,
'two': 199,
'pancake': 199,
'jubilee': 199,
'flora': 198,
'votive': 198,
'stop': 197,
'handbag': 196,
'coathanger': 196,
'petals': 196,
'animal': 195,
'housework': 195,
'cube': 194,
'message': 194,
'merry': 193,
'wastepaper': 192,
'rosemary': 192,
'thyme': 192,
'poncho': 192,
'present': 191,
'parsley': 191,
'monsters': 190,
'lucky': 190,
'grocery': 188,
'topiary': 187,
'penny': 187,
'farthing': 187,
'neighbourhood': 187,
'witch': 187,
'nicole': 187,
'chopping': 187,
'boom': 186,
'fine': 186,
```

```
'placemat': 185,
'address': 185,
'drawers': 184,
'washbag': 184,
'parlour': 184,
'stuff': 183,
'sporting': 182,
'confectionery': 182,
'filigris': 182,
'hour': 181,
'my': 181,
'belle': 181,
'jardiniere': 181,
'foot': 181,
'snowy': 179,
'slice': 179,
'secateurs': 179,
'mushrooms': 179,
'tradtional': 179,
'noel': 178,
'hooks': 178,
'ludo': 178,
'mitt': 178,
'goose': 178,
'motorbike': 177,
'medina': 176,
'camouflage': 176,
'smiley': 176,
'children': 176,
'chandelier': 176,
'chrome': 176,
'cloche': 175,
'baseball': 175,
'boot': 175,
'pedestal': 174,
'rest': 174,
'pants': 174,
'football': 173,
'junk': 173,
'mail': 173,
'clip': 172,
'funny': 172,
'face': 172,
'bone': 172,
'china': 172,
'earrings': 171,
'sharpener': 171,
```

```
 'pieces': 171,
 'pearls': 170,
 'strainer': 170,
 'market': 169,
 'crates': 169,
 'fuschia': 169,
 'chives': 168,
 'fringe': 168,
 'paperweight': 168,
 'screwdriver': 168,
 'botanique': 168,
 'sleeping': 167,
 'swiss': 167,
 'lanterns': 167,
 'full': 166,
 'ivoryred': 165,
 'wedding': 165,
 'front': 165,
 'elixir': 165,
 'served': 163,
 'padded': 163,
 'cones': 162,
 'bendy': 162,
 'ocean': 162,
 'flask': 162,
 'ducks': 160,
 'candlestick': 160,
 'cooking': 160,
 'holders': 160,
 'salad': 160,
 'chocolates': 160,
 'blackboard': 160,
 'scarf': 159,
 'airmail': 159,
 'meter': 159,
 'helicopter': 158,
 'penhot': 158,
 'sandcastle': 158,
 'flags': 158,
 'diva': 158,
 'boudoir': 156,
 'cowboy': 156,
 …})
```

[16]:
```
#Under Module#08 -> jawad8A
#jawad8A3
import matplotlib.pyplot as plt
```

```
from collections import Counter
import re
top_100_words = word_counts.most_common(100)
words, counts = zip(*top_100_words)
#jawad8A4
colors = ['skyblue', 'wheat', 'lightgreen', 'lavender', 'Aqua', 'lightpink',␣
 ↪'lightgray']
bar_colors = [colors[i % len(colors)] for i in range(len(words))]
plt.figure(figsize=(12, 15))
bars = plt.barh(words, counts, color=bar_colors)
plt.xlabel('Frequency', fontsize=12)
plt.ylabel('Keywords', fontsize=12)
plt.title('Top 100 Most Common Keywords in Descriptions', fontsize=16)
plt.gca().invert_yaxis()
for bar, count in zip(bars, counts):
    plt.text(
        bar.get_width() + 1,
        bar.get_y() + bar.get_height() / 2,
        str(count),
        va='center',
        fontsize=10,
        color='black'
    )
plt.tight_layout()
plt.show()
```



1.3.2 Categorical Keywords

The list that was obtained contains more than `1400 keywords` and the `most frequent` ones appear in more than `200 products`. However, while examinating the content of the list, I noticed that some names doesn't `depict any information`. Others do not carry enough information, like `colors`.

Therefore, I'll discard these words from the analysis that follows.

[17]:
```
#Module#9 -> jawad9A
#jawad9A1
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
import nltk
import re
nltk.download('averaged_perceptron_tagger')
descriptions = dataset['Description'].dropna().tolist()
tfidf = TfidfVectorizer(max_features=300, stop_words='english')
tfidf_matrix = tfidf.fit_transform(descriptions)
tfidf_keywords = tfidf.get_feature_names_out()
tfidf_frequencies = tfidf_matrix.sum(axis=0).A1
keywords_with_freq = list(zip(tfidf_keywords, tfidf_frequencies))
stop_words = {'pink', 'blue', 'green', 'orange', 'tag', 'red'}
filtered_keywords = [
    (word, freq) for word, freq in keywords_with_freq
    if word not in stop_words and len(word) >= 3 and not bool(re.search(r'\d',␣
 ↪word))
]
pos_tags = nltk.pos_tag([word for word, _ in filtered_keywords])
category_keywords = [
    (word, freq) for (word, freq), (word_pos, tag) in zip(filtered_keywords,␣
 ↪pos_tags)
    if tag.startswith('NN')
]
category_keywords.sort(key=lambda x: x[1], reverse=True)
print("Total Categorical Keywords:",len(category_keywords))
print("Refined Categorical Keywords:")
for keyword, freq in category_keywords:
    print(f"{keyword}: {freq}")
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\rakes\AppData\Roaming\nltk_data…
[nltk_data]   Unzipping taggers\averaged_perceptron_tagger.zip.
```

```
Total Categorical Keywords: 187
Refined Categorical Keywords:
set: 17920.538047142556
bag: 15924.837613297987
heart: 12521.306927096728
retrospot: 12223.547052361158
vintage: 11251.405054756398
design: 9714.163300781209
metal: 8655.874213482692
box: 8621.612398811834
christmas: 8605.288033067014
```

sign: 7666.536577806468
lunch: 7502.024831339033
cake: 6685.151180598999
holder: 6128.041055572321
pack: 5608.959231997891
paper: 5240.94553660635
glass: 4980.075188656418
pantry: 4686.459580532208
spaceboy: 4622.4469183408355
doormat: 4552.8109710768495
bottle: 4287.504143065913
tin: 4248.156218520388
cases: 4232.512261678444
mug: 4007.821990618543
wrap: 3827.89271425745
water: 3753.869513625468
party: 3674.069341048369
regency: 3590.9750327852166
cream: 3509.105162013254
ivory: 3465.5421271215105
mini: 3449.2858835939346
wicker: 3433.0062035766914
home: 3306.6553444433735
frame: 3244.327803645802
girl: 3125.3742893154194
clock: 3104.9300274703987
love: 3059.1336029011927
garden: 2951.84016444478
tins: 2903.992884451498
silver: 2874.0420966676234
woodland: 2830.621777979856
bowl: 2773.8908691171687
charlotte: 2731.774679552
birthday: 2730.2721181342627
fairy: 2722.666174405316
star: 2704.6742849404322
wall: 2699.7146108801753
lights: 2671.484621734424
drawer: 2611.6985822687907
colour: 2591.642431821527
wood: 2579.337933790848
candle: 2397.1554301918372
candles: 2376.169744135309
skull: 2342.9934383579202
zinc: 2342.695178873921
alphabet: 2298.387314442803
hand: 2271.854410453117
cutlery: 2243.4790125678473

gift: 2238.084032390835
jar: 2227.00163707953
ribbons: 2189.2135343623436
union: 2173.5538308723962
chocolate: 2148.0434982308248
suki: 2145.3480324631664
egg: 2120.6625023506017
jam: 2114.1530787991987
flower: 2061.2451491172733
cup: 2049.6478112269924
apples: 2013.7786156091154
hook: 1986.458201865573
alarm: 1955.5909291918197
tree: 1946.5335176835647
sweetheart: 1944.2109893779923
knob: 1892.414133676343
pencils: 1891.878708595461
storage: 1884.183089774677
plasters: 1867.5195243318992
craft: 1865.529896874035
tier: 1835.1536026359179
trinket: 1814.2129165577871
gingham: 1788.1538216284744
enamel: 1768.8211622534084
tube: 1749.0450323517762
magnets: 1745.2712023978684
cover: 1739.754679755072
leaf: 1720.3842457463563
skulls: 1691.916368079779
cookie: 1686.5842753513855
travel: 1657.428707862564
jack: 1592.6120336748356
cushion: 1590.1920895379092
tissues: 1585.0584064214659
stand: 1567.9797631985416
cards: 1555.445304899063
dog: 1537.4287438831855
chain: 1535.6324810034077
coffee: 1513.571005466523
reel: 1476.3411590729422
ribbon: 1475.5225712358736
babushka: 1441.9280710058733
spot: 1435.8940916242718
boxes: 1377.4939082789597
notebook: 1329.9385220433999
saucer: 1306.0757947925122
diner: 1302.7924660772649
wallet: 1292.8718047930006

cabinet: 1279.5092183279364
rabbit: 1248.2737507292966
picture: 1224.5218551125336
postage: 1212.0
teacup: 1188.8261943664918
milk: 1186.4394778606807
rack: 1173.2607599045018
recipe: 1168.0676596905594
block: 1154.9882712556146
piece: 1154.532851804523
bell: 1154.3858569475078
pot: 1147.680805034801
easter: 1143.4960074890355
mirror: 1141.4663945944098
retro: 1140.5200187172657
parade: 1132.8917204741572
book: 1118.0875155425133
scales: 1107.0164598541467
chalkboard: 1105.0932944139593
roses: 1085.9185461304785
pad: 1084.8521257115617
parasol: 1071.1570705768656
doilies: 1051.6407892699283
doiley: 1048.2357429261883
apple: 1041.9605495828448
herb: 1034.5054817401322
hanger: 1029.41964115334
night: 1013.6565437378796
yellow: 1009.2943553202344
tray: 993.1304504061043
sugar: 991.4916940135755
treasure: 986.548530015668
childs: 985.7307165720416
mat: 983.5170690316253
table: 982.5817247766222
owl: 979.3842964124119
cutters: 972.4735491768415
board: 956.1846544151699
marker: 953.4075390210711
snack: 932.1849441453826
game: 926.0779501011239
decorations: 921.3508720143072
wreath: 915.9165248884381
food: 904.5625986488172
sketchbook: 892.1526198409182
dinosaur: 887.2629230408422
coat: 885.9018138468291
biscuit: 879.8196969291032

```
pears: 877.55493419215
stickers: 865.4726720414119
umbrella: 861.1775291880006
bells: 858.5458592611991
time: 855.5264586603233
money: 831.0198567519957
charlie: 818.9995415564815
ornament: 818.1764525301458
doorstop: 814.7295595705874
word: 811.7486432470639
flag: 794.5626371754317
dinner: 789.8959762843199
style: 784.5510873526979
kids: 783.4398870363842
beaker: 782.5660266927729
baby: 772.9738877199135
basket: 771.7034503891151
lace: 765.9610883893564
bank: 754.231393761077
font: 749.0433587181784
bathroom: 742.3499880672254
animals: 724.9578755593714
gardeners: 722.1232297884211
towels: 718.9895672079059
empire: 714.9549279060946
cakes: 714.7195454761055
bunny: 706.1667145933088
tags: 705.8152871731933
wire: 692.3506107798588
medium: 688.2973346100456
container: 679.1716624887556
gin: 675.1249743768417
jars: 652.8499420708799
knack: 600.4734757150183
```

1.3.3 Data encoding

Now I will use these keywords to create groups of product.

[18]:
```
#Module#10 -> jawad10A
#jawad10A1
import pandas as pd
liste_produits = dataset['Description'].unique()
display(liste_produits)
X = []
for key, occurence in category_keywords:
    X.append(pd.DataFrame({key: list(map(lambda x: int(key.upper() in x),␣
 ↪liste_produits))}))
```

```
X = pd.concat(X, axis=1)
X
```

```
array(['WHITE HANGING HEART T-LIGHT HOLDER', 'WHITE METAL LANTERN',
       'CREAM CUPID HEARTS COAT HANGER', …,
       'PINK CRYSTAL SKULL PHONE CHARM',
       'CREAM HANGING HEART T-LIGHT HOLDER',
       'PAPER CRAFT , LITTLE BIRDIE'], dtype=object)
```

[18]:

	set	bag	heart	retrospot	vintage	design	metal	box	christmas	…	sign	…	empire	cakes	bunny	tags	wire	medium	container	gin	jars	knack
0	0	0	1	0	0	0	0	0	0		0	…	0	0	0	0	0	0	0	1	0	0
1	0	0	0	0	0	0	1	0	0		0	…	0	0	0	0	0	0	0	0	0	0
2	0	0	1	0	0	0	0	0	0		0	…	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0		0	…	0	0	0	0	0	0	0	0	0	0
4	0	0	1	0	0	0	0	0	0		0	…	0	0	0	0	0	0	0	0	0	0
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
3891	0	0	0	0	0	0	0	0	0		0	…	0	0	0	0	0	0	0	0	0	0
3892	0	0	0	0	0	0	0	0	0		0	…	0	0	0	0	0	0	0	0	0	0
3893	0	0	0	0	0	0	0	0	0		0	…	0	0	0	0	0	0	0	0	0	0
3894	0	0	1	0	0	0	0	0	0		0	…	0	0	0	0	0	0	0	1	0	0
3895	0	0	0	0	0	0	0	0	0		0	…	0	0	0	0	0	0	0	0	0	0

```
[3896 rows x 187 columns]
```

[19]:
```
import pandas as pd
threshold = [0, 1, 2, 3, 5, 10]
label_col = ['{}<.<{}'.format(threshold[i], threshold[i+1]) if i <␣
 ↪len(threshold) - 1 else '.>{}'.format(threshold[i]) for i in␣
 ↪range(len(threshold))]
new_cols = pd.DataFrame(0, index=X.index, columns=label_col)
for i, prod in enumerate(liste_produits):
    prix = dataset[dataset['Description'] == prod]['UnitPrice'].mean()
    j = 0
    while j < len(threshold) and prix > threshold[j]:
        j += 1
    if j > 0:
        new_cols.loc[i, label_col[j-1]] = 1
X = pd.concat([X, new_cols], axis=1)
print(X.head())
```

```
   set  bag  heart  retrospot  vintage  design  metal  box  christmas  sign  \
0    0    0      1          0        0       0      0    0          0     0
1    0    0      0          0        0       0      1    0          0     0
2    0    0      1          0        0       0      0    0          0     0
3    0    0      0          0        0       0      0    0          0     0
4    0    0      1          0        0       0      0    0          0     0

   …  container  gin  jars  knack  0<.<1  1<.<2  2<.<3  3<.<5  5<.<10  .>10
0  …          0    1     0      0      0      0      1      0       0     0
1  …          0    0     0      0      0      0      0      1       0     0
2  …          0    0     0      0      0      0      0      1       0     0
3  …          0    0     0      0      0      0      0      1       0     0
4  …          0    0     0      0      0      0      0      1       0     0

[5 rows x 193 columns]
```

The `X matrix` indicates the words contained in the description.

[20]:
```
print("{:<8} {:<20} \n".format('range', 'no. products') + 20*'-')
for i in range(len(threshold)):
    if i == len(threshold)-1:
        col = '.>{}'.format(threshold[i])
    else:
        col = '{}<.<{}'.format(threshold[i], threshold[i+1])
    col_sum = X[col].sum()
    if isinstance(col_sum, pd.Series):
        col_sum = col_sum.sum()
    print("{:<10}  {:<20}".format(col, int(col_sum)))
```

```
range    no. products
```

```
--------------------
0<.<1       965
1<.<2       1013
2<.<3       674
3<.<5       611
5<.<10      476
.>10        157
```

1.3.4 Creating clusters of products

In this section, I will group the products into different classes. In order to define (approximately) the number of clusters that best represents the data, I use the `silhouette score`:

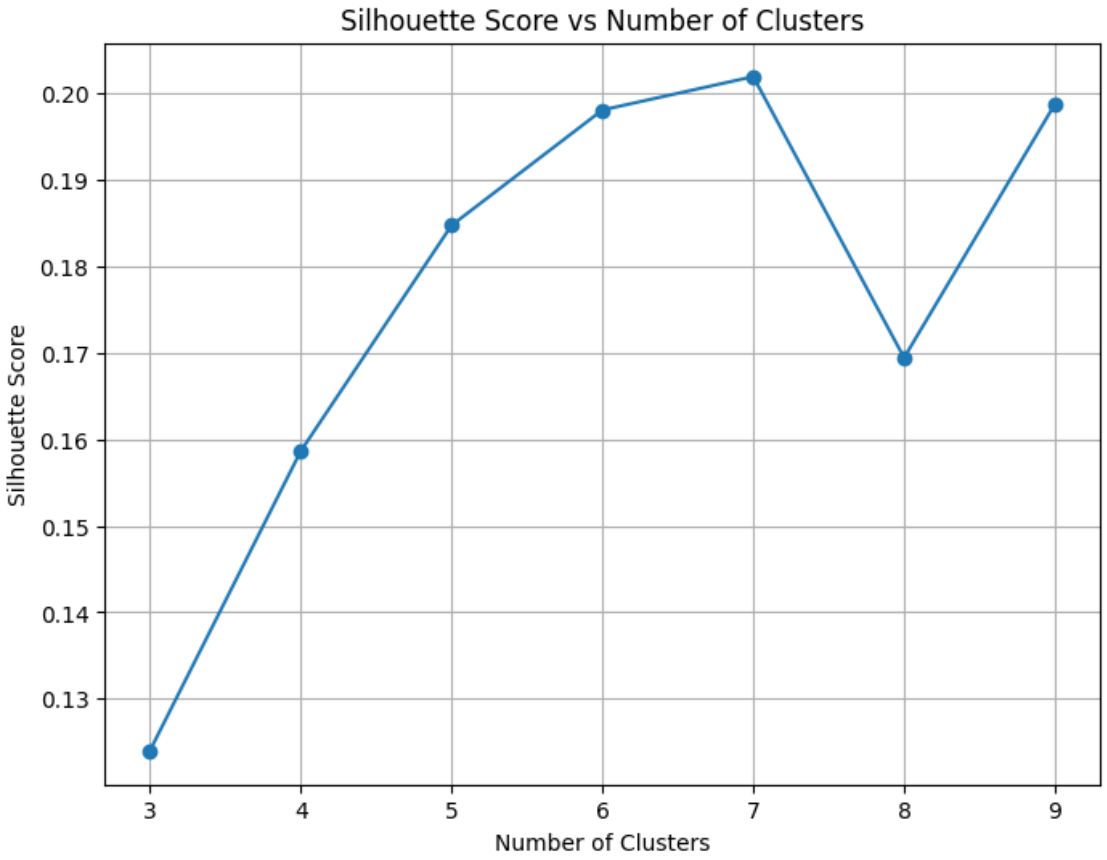
K-Means Clustering & silhouette_score

[21]:
```
import numpy as np
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
matrix = X.values
for n_clusters in range(3, 10):
    kmeans = KMeans(init='k-means++', n_clusters=n_clusters, n_init=30)
    kmeans.fit(matrix)
    clusters = kmeans.predict(matrix)
    silhouette_avg = silhouette_score(matrix, clusters)
    print("For n_clusters =", n_clusters, "The average silhouette_score is :",␣
 ↪silhouette_avg)
```

```
C:\Users\rakes\AppData\Local\Programs\Python\Python312\Lib\site-
packages\joblib\externals\loky\backend\context.py:131: UserWarning:

Could not find the number of physical cores for the following reason:
[WinError 2] The system cannot find the file specified
Returning the number of logical cores instead. You can silence this warning by
setting LOKY_MAX_CPU_COUNT to the number of cores you want to use.

  File "C:\Users\rakes\AppData\Local\Programs\Python\Python312\Lib\site-
packages\joblib\externals\loky\backend\context.py", line 247, in
_count_physical_cores
    cpu_count_physical = _count_physical_cores_win32()
                         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

  File "C:\Users\rakes\AppData\Local\Programs\Python\Python312\Lib\site-
packages\joblib\externals\loky\backend\context.py", line 299, in
_count_physical_cores_win32
    cpu_info = subprocess.run(
               ^^^^^^^^^^^^^^

  File
"C:\Users\rakes\AppData\Local\Programs\Python\Python312\Lib\subprocess.py", line
548, in run
    with Popen(*popenargs, **kwargs) as process:
```

```
        ^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File
"C:\Users\rakes\AppData\Local\Programs\Python\Python312\Lib\subprocess.py", line
1026, in __init__
    self._execute_child(args, executable, preexec_fn, close_fds,
  File
"C:\Users\rakes\AppData\Local\Programs\Python\Python312\Lib\subprocess.py", line
1538, in _execute_child
    hp, ht, pid, tid = _winapi.CreateProcess(executable, args,
                       ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

For n_clusters = 3 The average silhouette_score is : 0.12826217521096547
For n_clusters = 4 The average silhouette_score is : 0.1586333784336075
For n_clusters = 5 The average silhouette_score is : 0.18611347783779264
For n_clusters = 6 The average silhouette_score is : 0.1918291436141626
For n_clusters = 7 The average silhouette_score is : 0.20387376524975115
For n_clusters = 8 The average silhouette_score is : 0.20278823994388054
For n_clusters = 9 The average silhouette_score is : 0.19824369470755035
```

[22]:
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
matrix = X.values
sil_scores = []
for n_clusters in range(3, 10):
    kmeans = KMeans(init='k-means++', n_clusters=n_clusters, n_init=30)
    kmeans.fit(matrix)
    clusters = kmeans.predict(matrix)
    silhouette_avg = silhouette_score(matrix, clusters)
    sil_scores.append(silhouette_avg)
plt.figure(figsize=(8, 6))
plt.plot(range(3, 10), sil_scores, marker='o')
plt.title('Silhouette Score vs Number of Clusters')
plt.xlabel('Number of Clusters')
plt.ylabel('Silhouette Score')
plt.grid(True)
plt.show()
```



[23]:
```
n_clusters = 5
silhouette_avg = -1
while silhouette_avg < 0.145:
    kmeans = KMeans(init='k-means++', n_clusters = n_clusters, n_init=30)
    kmeans.fit(matrix)
    clusters = kmeans.predict(matrix)
    silhouette_avg = silhouette_score(matrix, clusters)
    print("For n_clusters =", n_clusters, "The average silhouette_score is :",␣
 ↪silhouette_avg)
```

```
For n_clusters = 5 The average silhouette_score is : 0.185030831739071
```

[24]:
```
X
```

[24]:

	set	bag	heart	retrospot	vintage	design	metal	box	christmas	\
0	0	0	1	0	0	0	0	0	0	
1	0	0	0	0	0	0	1	0	0	
2	0	0	1	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	
4	0	0	1	0	0	0	0	0	0	

```
…   …  …    …         …      …     …    …  …         …
3891   0    0      0          0        0       0      0    0          0
3892   0    0      0          0        0       0      0    0          0
3893   0    0      0          0        0       0      0    0          0
3894   0    0      1          0        0       0      0    0          0
3895   0    0      0          0        0       0      0    0          0

      sign  …  container  gin  jars  knack  0<.<1  1<.<2  2<.<3  3<.<5  \
0        0  …          0    1     0      0      0      0      1      0
1        0  …          0    0     0      0      0      0      0      1
2        0  …          0    0     0      0      0      0      0      1
3        0  …          0    0     0      0      0      0      0      1
4        0  …          0    0     0      0      0      0      0      1
…    …  …        …  …   …    …     …    …     …    …
3891     0  …          0    0     0      0      1      0      0      0
3892     0  …          0    0     0      0      1      0      0      0
3893     0  …          0    0     0      0      1      0      0      0
3894     0  …          0    1     0      0      0      0      1      0
3895     0  …          0    0     0      0      0      0      1      0

      5<.<10  .>10
0          0     0
1          0     0
2          0     0
3          0     0
4          0     0
…       …   …
3891       0     0
3892       0     0
3893       0     0
3894       0     0
3895       0     0

[3896 rows x 193 columns]
```

[25]:
```
pd.Series(clusters).value_counts()
```

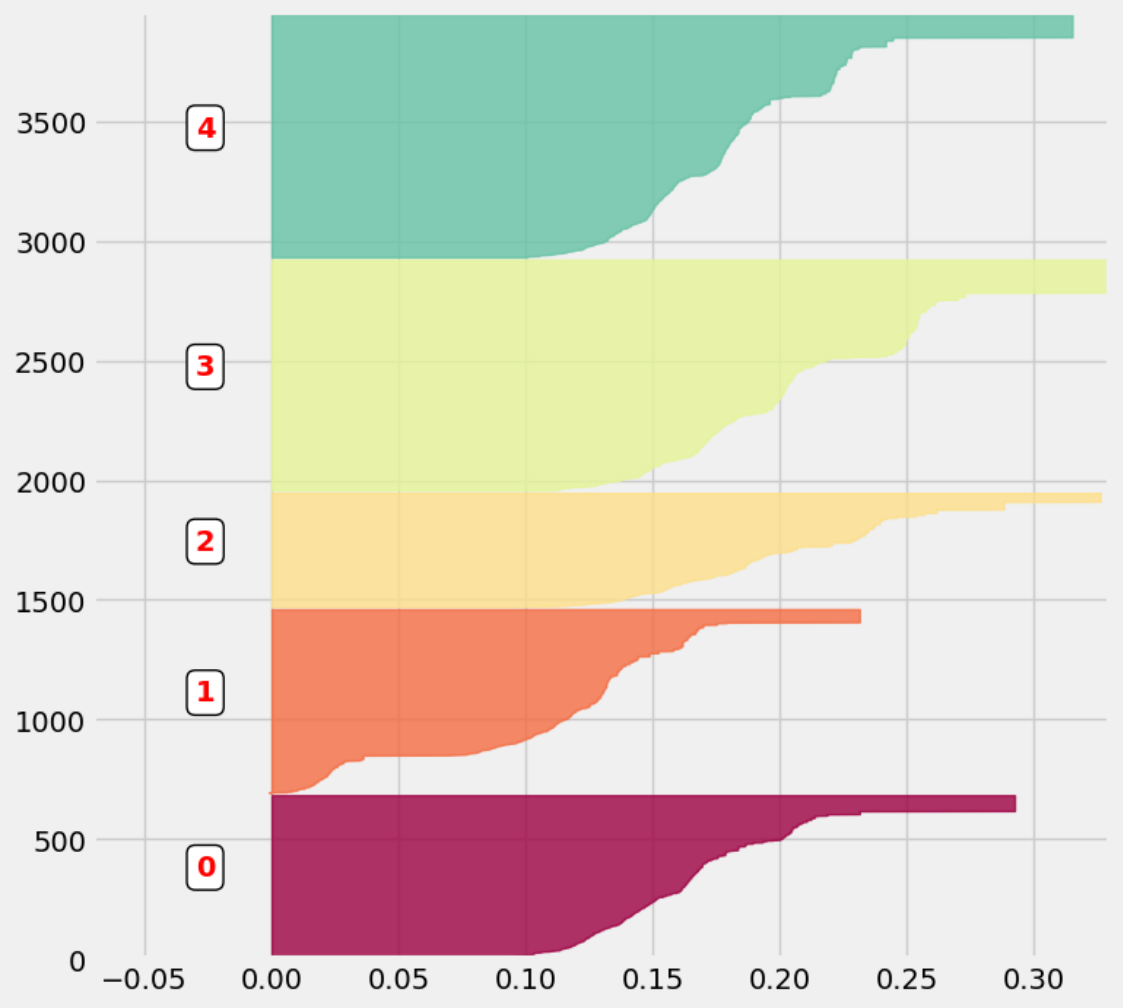
[25]:
```
4    1013
3     965
1     768
0     674
2     476
Name: count, dtype: int64
```

[26]:
```
import matplotlib as mpl
import matplotlib.cm as cm
from matplotlib import colormaps
```

```
def graph_component_silhouette(n_clusters, lim_x, mat_size,␣
 ↪sample_silhouette_values, clusters):
    plt.rcParams["patch.force_edgecolor"] = True
    plt.style.use('fivethirtyeight')
    mpl.rc('patch', edgecolor = 'dimgray', linewidth=1)
    fig, ax1 = plt.subplots(1, 1)
    fig.set_size_inches(8, 8)
    ax1.set_xlim([lim_x[0], lim_x[1]])
    ax1.set_ylim([0, mat_size + (n_clusters + 1) * 10])
    y_lower = 10
    for i in range(n_clusters):
        ith_cluster_silhouette_values = sample_silhouette_values[clusters == i]
        ith_cluster_silhouette_values.sort()
        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i
        cmap = cm.get_cmap("Spectral")
        color = cmap(float(i) / n_clusters)
        ax1.fill_betweenx(np.arange(y_lower, y_upper), 0,␣
 ↪ith_cluster_silhouette_values,
                          facecolor=color, edgecolor=color, alpha=0.8)
        ax1.text(-0.03, y_lower + 0.5 * size_cluster_i, str(i), color = 'red',␣
 ↪fontweight = 'bold',
                bbox=dict(facecolor='white', edgecolor='black',␣
 ↪boxstyle='round, pad=0.3'))
        y_lower = y_upper + 10
```

[27]:
```
#jawad11A5
from sklearn.metrics import silhouette_samples
sample_silhouette_values = silhouette_samples(matrix, clusters)
graph_component_silhouette(n_clusters, [-0.07, 0.33], len(X),␣
 ↪sample_silhouette_values, clusters)
```

```
C:\Users\rakes\AppData\Local\Temp\ipykernel_17100\2448983036.py:20:
MatplotlibDeprecationWarning:

The get_cmap function was deprecated in Matplotlib 3.7 and will be removed two
minor releases later. Use ``matplotlib.colormaps[name]`` or
``matplotlib.colormaps.get_cmap(obj)`` instead.
```



Word Cloud

[28]:
```
liste = pd.DataFrame(liste_produits)
liste_words = [word for (word, occurence) in category_keywords]
occurence = [dict() for _ in range(n_clusters)]
for i in range(n_clusters):
    liste_cluster = liste.loc[clusters == i]
    for word in liste_words:
        if word in ['art', 'set', 'heart', 'pink', 'blue', 'tag']: continue
        occurence[i][word] = sum(liste_cluster.loc[:, 0].str.contains(word.
 ↪upper()))
from wordcloud import WordCloud
def random_color_func(word=None, font_size=None, position=None,
                      orientation=None, font_path=None, random_state=None):
    h = int(360.0 * tone / 255.0)
    s = int(100.0 * 255.0 / 255.0)
    l = int(100.0 * float(random_state.randint(70, 120)) / 255.0)
```

```
    return "hsl({}, {}%, {}%)".format(h, s, l)
def make_wordcloud(liste, increment):
    ax1 = fig.add_subplot(4,2,increment)
    words = dict()
    trunc_occurences = liste[0:150]
    for s in trunc_occurences:
        words[s[0]] = s[1]
    wordcloud = WordCloud(width=1000,height=400, background_color='lightgrey',
                          max_words=1628,relative_scaling=1,
                          color_func = random_color_func,
                          normalize_plurals=False)
    wordcloud.generate_from_frequencies(words)
    ax1.imshow(wordcloud, interpolation="bilinear")
    ax1.axis('off')
    plt.title('cluster {}'.format(increment-1))
fig = plt.figure(1, figsize=(14,14))
color = [0, 160, 130, 95, 280, 40, 330, 110, 25]
for i in range(n_clusters):
    list_cluster_occurences = occurence[i]
    tone = color[i]
    liste = []
    for key, value in list_cluster_occurences.items():
        liste.append([key, value])
    liste.sort(key = lambda x:x[1], reverse = True)
    make_wordcloud(liste, i+1)
```

From the above representation, we can see that for example, one of the clusters contains objects that could be associated with `gifts` * (`keywords: Birthday, candle, cake, ...`). Nevertheless, it can also be observed that many words appear in various clusters and it is therefore difficult to clearly distinguish them.

1.4 4: Customer categories

`Formatting data`

In the previous section, the different `products` were grouped in `five clusters`. In order to prepare the rest of the analysis, a first step consists in introducing this information into the dataframe. To do this, I create the categorical variable `categ_product` where I indicate the cluster of each product:

[29]:
```
#jawad13A1
corresp = dict()
for key, val in zip (liste_produits, clusters):
    corresp[key] = val
#jawad13A2
dataset['categ_product'] = dataset.loc[:, 'Description'].map(corresp)
for i in range(5):
    col = f'categ_{i}'
    df_temp = dataset[dataset['categ_product'] == i]
    price_temp = df_temp['UnitPrice'] * df_temp['Quantity']
    price_temp = price_temp.apply(lambda x: x if x > 0 else 0).astype(float)
    dataset[col] = 0.0
    dataset.loc[df_temp.index, col] = price_temp
    dataset[col] = dataset[col].fillna(0.0)
dataset[['InvoiceNo', 'Description', 'categ_product', 'categ_0', 'categ_1',␣
 ↪'categ_2', 'categ_3', 'categ_4']]
```

[29]:

	InvoiceNo	Description	categ_product	categ_0	\
0	536365	WHITE HANGING HEART T-LIGHT HOLDER	0	15.3	
1	536365	WHITE METAL LANTERN	1	0.0	
2	536365	CREAM CUPID HEARTS COAT HANGER	1	0.0	
3	536365	KNITTED UNION FLAG HOT WATER BOTTLE	1	0.0	
4	536365	RED WOOLLY HOTTIE WHITE HEART.	1	0.0	
…	…	…	…	…	
541904	581587	PACK OF 20 SPACEBOY NAPKINS	3	0.0	
541905	581587	CHILDREN'S APRON DOLLY GIRL	0	12.6	
541906	581587	CHILDRENS CUTLERY DOLLY GIRL	1	0.0	
541907	581587	CHILDRENS CUTLERY CIRCUS PARADE	1	0.0	
541908	581587	BAKING SET 9 PIECE RETROSPOT	1	0.0	

	categ_1	categ_2	categ_3	categ_4
0	0.00	0.0	0.0	0.0
1	20.34	0.0	0.0	0.0

```
2          22.00      0.0      0.0      0.0
3          20.34      0.0      0.0      0.0
4          20.34      0.0      0.0      0.0
…            …        …        …        …
541904      0.00      0.0     10.2      0.0
541905      0.00      0.0      0.0      0.0
541906     16.60      0.0      0.0      0.0
541907     16.60      0.0      0.0      0.0
541908     14.85      0.0      0.0      0.0

[401604 rows x 8 columns]
```

I decided to create the `categ_N` variables (with `N[0:4]`) that contains the `amount spent in each product category`:

[30]:
```
temp = dataset.groupby(by=['CustomerID', 'InvoiceNo'],␣
 ↪as_index=False)['TotalPrice'].sum()
basket_price = temp.rename(columns={'TotalPrice': 'Basket Price'})
for i in range(5):
    col = 'categ_{}'.format(i)
    temp = dataset.groupby(by=['CustomerID', 'InvoiceNo'], as_index=False)[col].
 ↪sum()
    basket_price[col] = temp[col]
dataset['InvoiceDate_int'] = dataset['InvoiceDate'].astype('int64')
temp = dataset.groupby(by=['CustomerID', 'InvoiceNo'],␣
 ↪as_index=False)['InvoiceDate_int'].mean()
dataset.drop('InvoiceDate_int', axis=1, inplace=True)
basket_price['InvoiceDate'] = pd.to_datetime(temp['InvoiceDate_int'])
basket_price = basket_price[basket_price['Basket Price'] > 0]
basket_price = basket_price.sort_values('CustomerID', ascending=True)
basket_price
```

[30]:
```
       CustomerID InvoiceNo  Basket Price  categ_0  categ_1  categ_2  categ_3  \
0           12346    541431      77183.60     0.00     0.00     0.00     0.00
2           12347    537626        711.79    83.40   293.35   124.44    23.40
3           12347    542237        475.39    53.10   207.45     0.00    84.34
4           12347    549222        636.25    71.10   153.25     0.00    81.00
5           12347    556201        382.52    78.06   168.76    19.90    41.40
…             …         …             …        …        …        …        …
22177       18283    557956        192.80    69.26    17.40     0.00    43.34
22186       18283    580872        208.00   119.03     0.00     0.00    27.07
22188       18287    570715       1001.32   326.04    32.00     0.00   256.84
22187       18287    554065        765.28   134.70    15.00    34.00    40.68
22189       18287    573167         70.68     0.00     0.00     0.00    25.68

        categ_4                    InvoiceDate
0      77183.60 2011-01-18 10:01:00.000000000
```

```
2         187.20 2010-12-07 14:57:00.000000000
3         130.50 2011-01-26 14:29:59.999999744
4         330.90 2011-04-07 10:43:00.000000000
5          74.40 2011-06-09 13:01:00.000000000
…           …                              …
22177      62.80 2011-06-23 19:20:00.000000000
22186      61.90 2011-12-06 12:02:00.000000000
22188     386.44 2011-10-12 10:23:00.000000000
22187     540.90 2011-05-22 10:39:00.000000000
22189      45.00 2011-10-28 09:29:00.000000000

[18532 rows x 9 columns]
```

[31]:
```
basket_price.head(10)
```

[31]:
```
    CustomerID InvoiceNo  Basket Price  categ_0  categ_1  categ_2  categ_3  \
0        12346    541431      77183.60     0.00     0.00     0.00     0.00
2        12347    537626        711.79    83.40   293.35   124.44    23.40
3        12347    542237        475.39    53.10   207.45     0.00    84.34
4        12347    549222        636.25    71.10   153.25     0.00    81.00
5        12347    556201        382.52    78.06   168.76    19.90    41.40
6        12347    562032        584.91   119.70   196.41    97.80    61.30
7        12347    573511       1294.32   435.90   445.22    55.60   154.30
8        12347    581180        224.82    55.44     0.00    30.00    38.58
12       12348    568172        310.00     0.00    40.00     0.00     0.00
11       12348    548955        367.00     0.00    40.00     0.00    17.00

    categ_4                  InvoiceDate
0   77183.6 2011-01-18 10:01:00.000000000
2     187.2 2010-12-07 14:57:00.000000000
3     130.5 2011-01-26 14:29:59.999999744
4     330.9 2011-04-07 10:43:00.000000000
5      74.4 2011-06-09 13:01:00.000000000
6     109.7 2011-08-02 08:48:00.000000000
7     203.3 2011-10-31 12:25:00.000000000
8     100.8 2011-12-07 15:52:00.000000000
12    270.0 2011-09-25 13:13:00.000000000
11    310.0 2011-04-05 10:47:00.000000000
```

[32]:
```
print(basket_price['InvoiceDate'].min(), '->',  basket_price['InvoiceDate'].
  ↪max())
```

```
2010-12-01 08:26:00 -> 2011-12-09 12:50:00
```

[33]:
```
import pandas as pd
import datetime
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
```

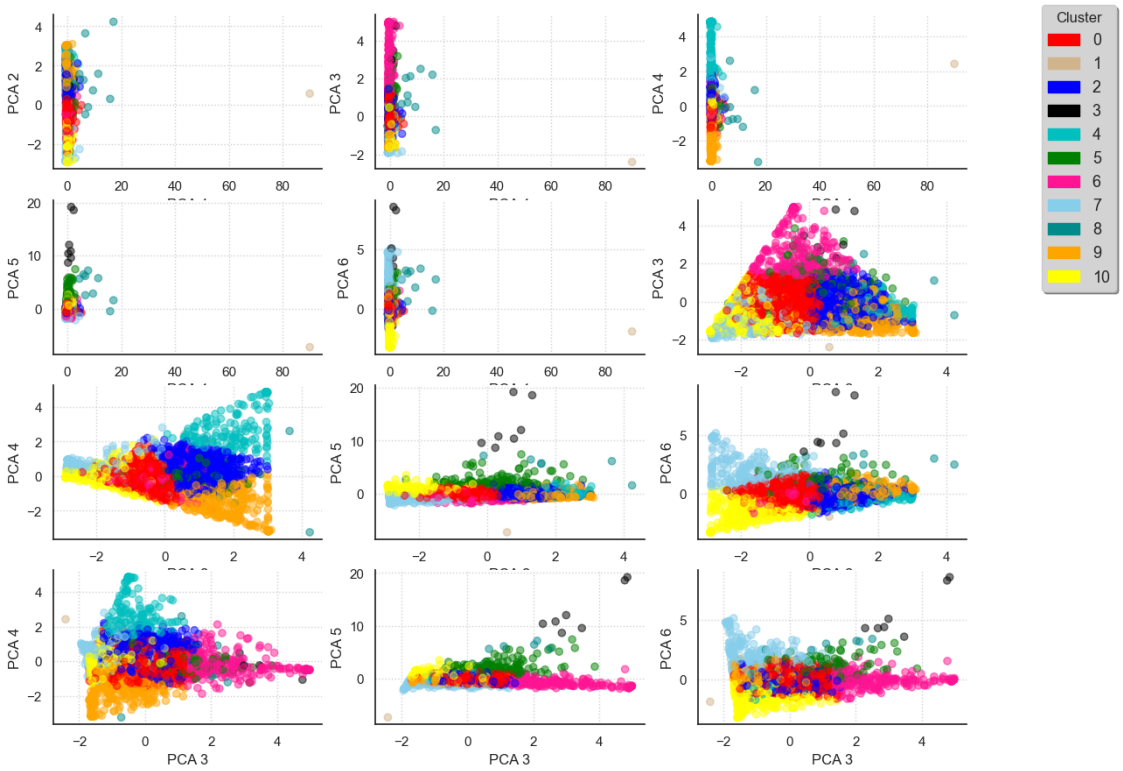
```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
comparison_date = datetime.datetime(2011, 10, 1)
set_training = basket_price[basket_price['InvoiceDate'] < comparison_date]
set_test = basket_price[basket_price['InvoiceDate'] >= comparison_date]
basket_price = set_training.copy(deep=True)
transactions_per_user = basket_price.groupby(by=['CustomerID'])['Basket Price'].
 ↪agg(['count','min','max','mean','sum'])
for i in range(5):
    col = 'categ_{}'.format(i)
    transactions_per_user.loc[:, col] = basket_price.
 ↪groupby(by=['CustomerID'])[col].sum() / transactions_per_user['sum'] * 100
transactions_per_user.reset_index(drop=False, inplace=True)
first_registration = basket_price.groupby('CustomerID')['InvoiceDate'].min()
last_purchase = basket_price.groupby('CustomerID')['InvoiceDate'].max()
last_date = basket_price['InvoiceDate'].max().date()
test = first_registration.map(lambda x: (last_date - x.date()).days)
test2 = last_purchase.map(lambda x: (last_date - x.date()).days)
transactions_per_user.loc[:, 'LastPurchase'] = test2.reset_index(drop=True)
transactions_per_user.loc[:, 'FirstPurchase'] = test.reset_index(drop=True)
list_cols = ['count', 'min', 'max', 'mean', 'categ_0', 'categ_1', 'categ_2',␣
 ↪'categ_3', 'categ_4']
selected_customers = transactions_per_user.copy(deep=True)
matrix = selected_customers[list_cols].to_numpy()
scaler = StandardScaler()
scaled_matrix = scaler.fit_transform(matrix)
```

[34]:
```
n_clusters = 11
kmeans = KMeans(init='k-means++', n_clusters=n_clusters, n_init=30)
kmeans.fit(scaled_matrix)
clusters_clients = kmeans.predict(scaled_matrix)
selected_customers['cluster'] = clusters_clients
silhouette_avg = silhouette_score(scaled_matrix, clusters_clients)
print(f'Silhouette score: {silhouette_avg:.3f}')
pca = PCA(n_components=6)
matrix_3D = pca.fit_transform(scaled_matrix)
mat = pd.DataFrame(matrix_3D)
mat['cluster'] = clusters_clients
print(mat.head())
```

```
Silhouette score: 0.209
           0         1         2         3         4         5  cluster
0  90.068270  0.579163 -2.405882  2.427933 -7.307199 -1.907765        1
1   0.184024  0.177598 -0.447042 -0.744057  0.356803 -0.131933        0
2   0.153364 -1.643836 -1.589003 -0.322343  0.249575  0.742674        7
3   0.006210 -1.497986  0.183042 -0.433247 -0.053036 -1.007364        0
4  -0.040701  1.965504 -0.922057 -1.496328  0.673890  0.098844        9
```

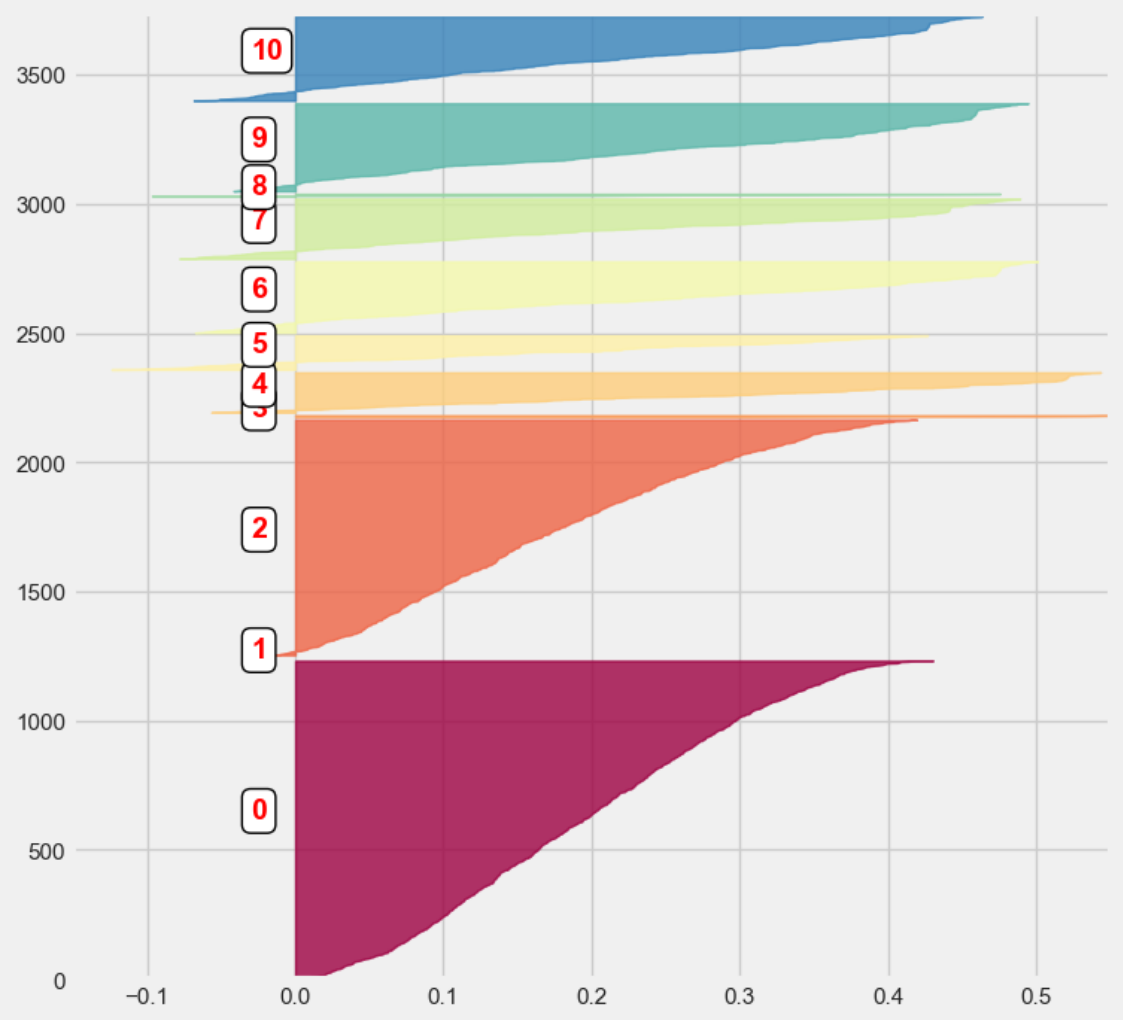
[35]:
```
import matplotlib.patches as mpatches
import seaborn as sns
sns.set_style("white")
sns.set_context("notebook", font_scale=1, rc={"lines.linewidth": 2.5})
LABEL_COLOR_MAP = {0:'r', 1:'tan', 2:'b', 3:'k', 4:'c', 5:'g', 6:'deeppink', 7:
 ↪'skyblue', 8:'darkcyan', 9:'orange',
                   10:'yellow', 11:'tomato', 12:'seagreen'}
label_color = [LABEL_COLOR_MAP[l] for l in mat['cluster']]
fig = plt.figure(figsize = (12,10))
increment = 0
for ix in range(6):
    for iy in range(ix+1, 6):
        increment += 1
        ax = fig.add_subplot(4,3,increment)
        ax.scatter(mat[ix], mat[iy], c= label_color, alpha=0.5)
        plt.ylabel('PCA {}'.format(iy+1), fontsize = 12)
        plt.xlabel('PCA {}'.format(ix+1), fontsize = 12)
        ax.yaxis.grid(color='lightgray', linestyle=':')
        ax.xaxis.grid(color='lightgray', linestyle=':')
        ax.spines['right'].set_visible(False)
        ax.spines['top'].set_visible(False)
        if increment == 12: break
    if increment == 12: break
comp_handler = []
for i in range(n_clusters):
    comp_handler.append(mpatches.Patch(color = LABEL_COLOR_MAP[i], label = i))
plt.legend(handles=comp_handler, bbox_to_anchor=(1.1, 0.9),
           title='Cluster', facecolor = 'lightgrey',
           shadow = True, frameon = True, framealpha = 1,
           fontsize = 13, bbox_transform = plt.gcf().transFigure)
plt.tight_layout()
```

```
C:\Users\rakes\AppData\Local\Temp\ipykernel_17100\1062018531.py:30: UserWarning:

Tight layout not applied. tight_layout cannot make axes height small enough to
accommodate all axes decorations.
```



[36]:
```
sample_silhouette_values = silhouette_samples(scaled_matrix, clusters_clients)
sample_silhouette_values = silhouette_samples(scaled_matrix, clusters_clients)
graph_component_silhouette(n_clusters, [-0.15, 0.55], len(scaled_matrix),␣
 ↪sample_silhouette_values, clusters_clients)
```

```
C:\Users\rakes\AppData\Local\Temp\ipykernel_17100\2448983036.py:20:
MatplotlibDeprecationWarning:

The get_cmap function was deprecated in Matplotlib 3.7 and will be removed two
minor releases later. Use ``matplotlib.colormaps[name]`` or
``matplotlib.colormaps.get_cmap(obj)`` instead.
```



1.5 5:Classifying customers

1.5.1 Train X,Y

[37]:
```
columns = ['mean', 'categ_0', 'categ_1', 'categ_2', 'categ_3', 'categ_4']
X = selected_customers[columns]
Y = selected_customers['cluster']
print(X.head())
print(Y.value_counts())
```

```
           mean    categ_0    categ_1    categ_2    categ_3     categ_4
0  77183.600000   0.000000   0.000000   0.000000   0.000000  100.000000
1    558.172000  14.524555  36.519926   8.676179  10.442659   29.836681
2    449.310000   0.000000  20.030714   0.000000  38.016069   41.953217
3    334.400000  27.900718  11.961722   0.000000  11.692584   48.444976
4    313.472857   4.798775  65.840743  13.516777   0.464839   15.378866
```

```
cluster
0     1223
2      912
9      340
10     326
6      276
7      233
4      156
5      133
8        9
3        7
1        1
Name: count, dtype: int64
```

[38]:
```
category_counts = mat['cluster'].value_counts()
category_counts_df = pd.DataFrame(category_counts).reset_index()
category_counts_df.columns = ['Cluster', 'No. of Customers']
display(category_counts_df)
```

	Cluster	No. of Customers
0	0	1223
1	2	912
2	9	340
3	10	326
4	6	276
5	7	233
6	4	156
7	5	133
8	8	9
9	3	7
10	1	1

[39]:
```
class Class_Fit(object):
    def __init__(self, clf, params=None):
        if params:
            self.clf = clf(**params)
        else:
            self.clf = clf
    def train(self, x_train, y_train):
        self.clf.fit(x_train, y_train)
    def predict(self, x):
        return self.clf.predict(x)
    def grid_search(self, parameters, Kfold):
        self.grid = GridSearchCV(estimator=self.clf, param_grid=parameters,␣
 ↪cv=Kfold)
    def grid_fit(self, X, Y):
        self.grid.fit(X, Y)
    def grid_predict(self, X, Y):
```

```
        if not hasattr(self.grid, 'best_estimator_'):
            raise NotFittedError("GridSearchCV is not fitted yet. Please call␣
 ↪grid_fit first.")
        self.predictions = self.grid.predict(X)
        print("Precision: {:.2f} % ".format(100 * accuracy_score(Y, self.
 ↪predictions)))
```

[40]:
```
X
```

[40]:

	mean	categ_0	categ_1	categ_2	categ_3	categ_4
0	77183.600000	0.000000	0.000000	0.000000	0.000000	100.000000
1	558.172000	14.524555	36.519926	8.676179	10.442659	29.836681
2	449.310000	0.000000	20.030714	0.000000	38.016069	41.953217
3	334.400000	27.900718	11.961722	0.000000	11.692584	48.444976
4	313.472857	4.798775	65.840743	13.516777	0.464839	15.378866
…	…	…	…	…	…	…
3611	180.600000	41.140642	24.833887	34.025471	0.000000	0.000000
3612	80.820000	18.930958	41.945063	0.000000	18.708241	20.415739
3613	100.210000	17.662908	38.918272	0.000000	20.516914	22.901906
3614	108.683000	34.549102	8.837629	2.245061	17.204163	37.164046
3615	765.280000	17.601401	1.960067	4.442818	5.315701	70.680013

[3616 rows x 6 columns]

[41]:
```
Y.value_counts()
```

[41]:
```
cluster
0     1223
2      912
9      340
10     326
6      276
7      233
4      156
5      133
8        9
3        7
1        1
Name: count, dtype: int64
```

[42]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, train_size=0.8,␣
 ↪random_state=0)
```

1.5.2 Models

Support Vector Machine Classifier (SVC)
Logistic regression

```
k-Nearest Neighbors
Decision Tree
Random Forest
```

[43]:
```
from sklearn.model_selection import learning_curve
def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                        n_jobs=-1, train_sizes=np.linspace(.1, 1.0, 10)):
    """Generate a simple plot of the test and training learning curve"""
    plt.figure()
    plt.title(title)
    if ylim is not None:
        plt.ylim(*ylim)
    plt.xlabel("Training examples")
    plt.ylabel("Score")
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    plt.grid()
    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1, color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r", label="Training␣
 ↪score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",␣
 ↪label="Cross-validation score")
    plt.legend(loc="best")
    return plt
```

[44]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
logreg = LogisticRegression(max_iter=20000)
logreg.fit(X_train, Y_train)
predictions = logreg.predict(X_test)
accuracy = accuracy_score(Y_test, predictions)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

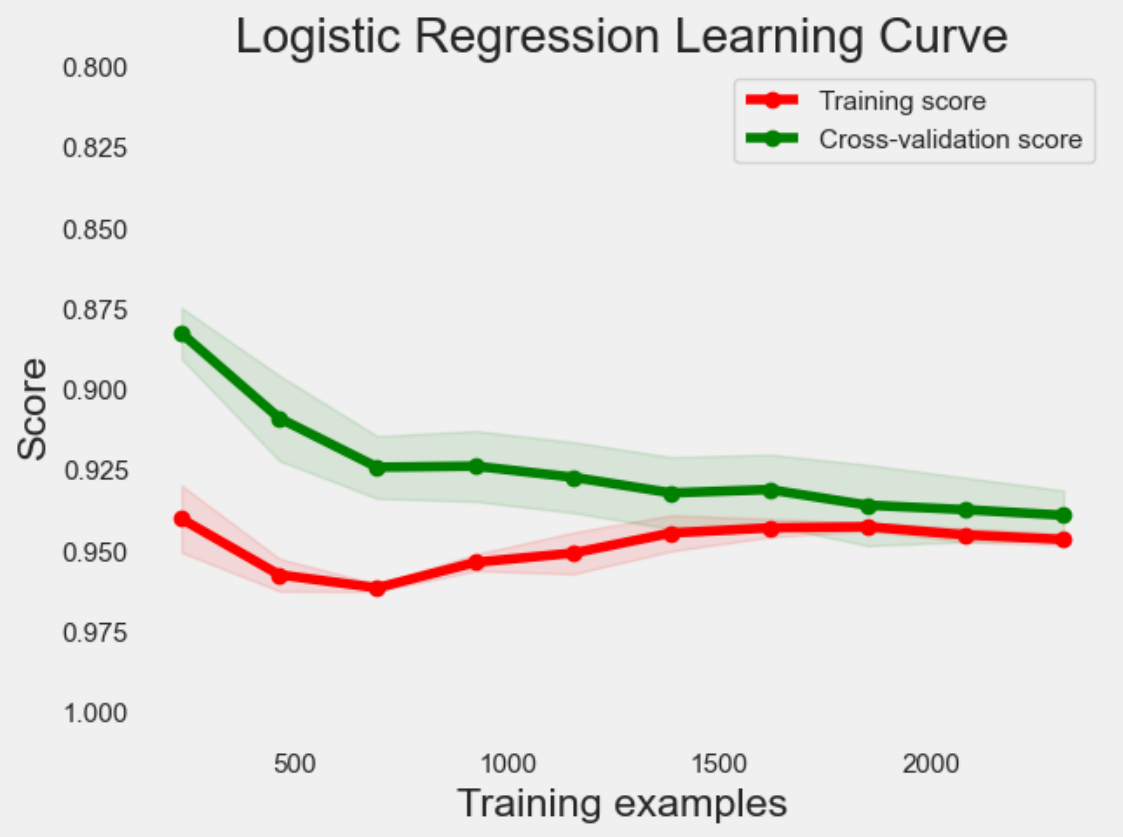
```
Accuracy: 93.65%
```

[45]:
```
plot_learning_curve(logreg, "Logistic Regression Learning Curve", X_train,␣
 ↪Y_train, ylim=[1.01, 0.8], cv=5)
```

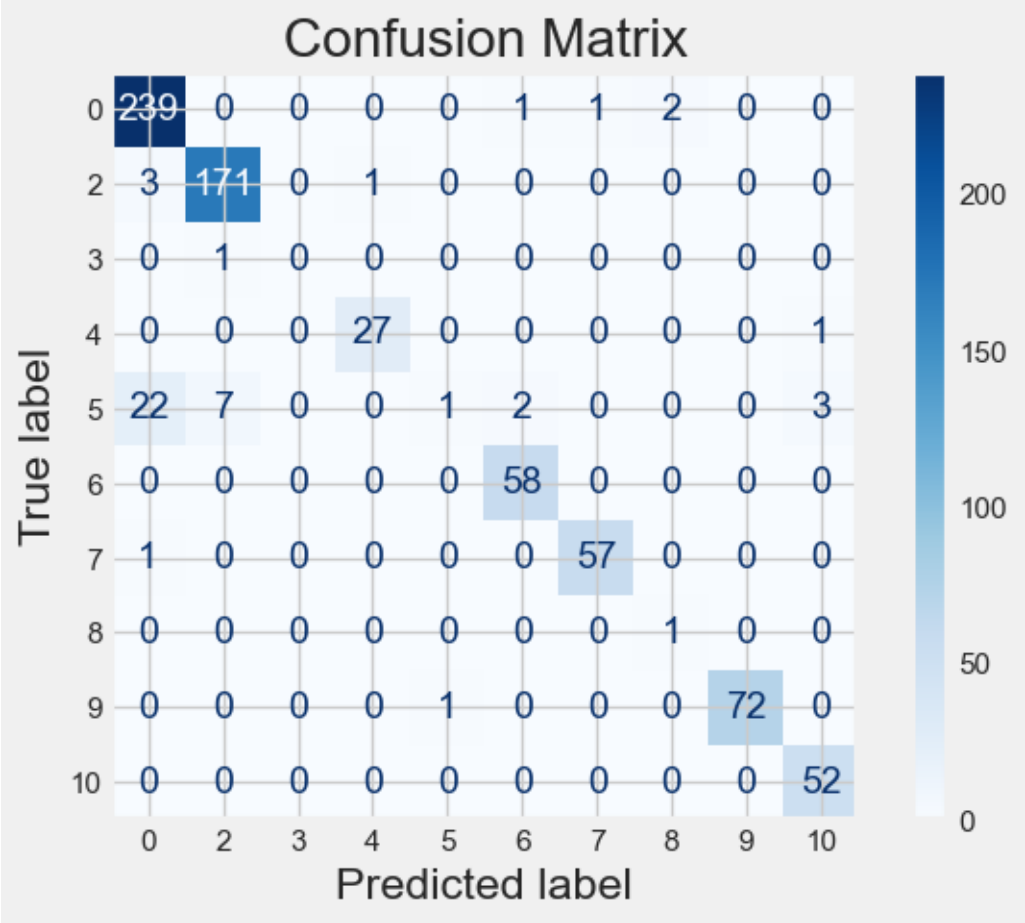
```
C:\Users\rakes\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\model_selection\_split.py:776: UserWarning:

The least populated class in y has only 1 members, which is less than
```

```
n_splits=5.
```

[45]:
```
<module 'matplotlib.pyplot' from
'C:\\Users\\rakes\\AppData\\Local\\Programs\\Python\\Python312\\Lib\\site-
packages\\matplotlib\\pyplot.py'>
```



[46]:
```
from sklearn.metrics import ConfusionMatrixDisplay
ConfusionMatrixDisplay.from_estimator(logreg, X_test, Y_test, cmap='Blues')
plt.title("Confusion Matrix")
plt.show()
```



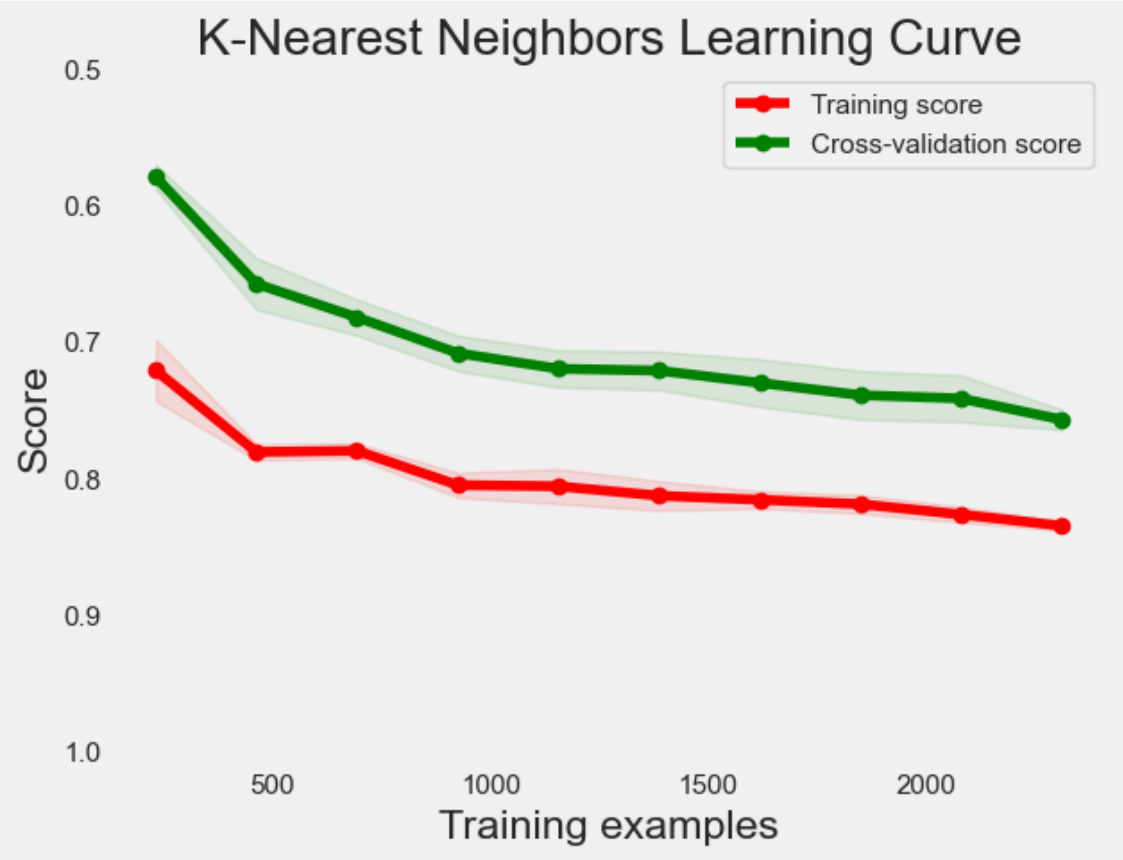
[47]:
```
from sklearn.neighbors import KNeighborsClassifier
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train, Y_train)
predictions = knn_model.predict(X_test)
accuracy = accuracy_score(Y_test, predictions)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
Accuracy: 78.31%
```

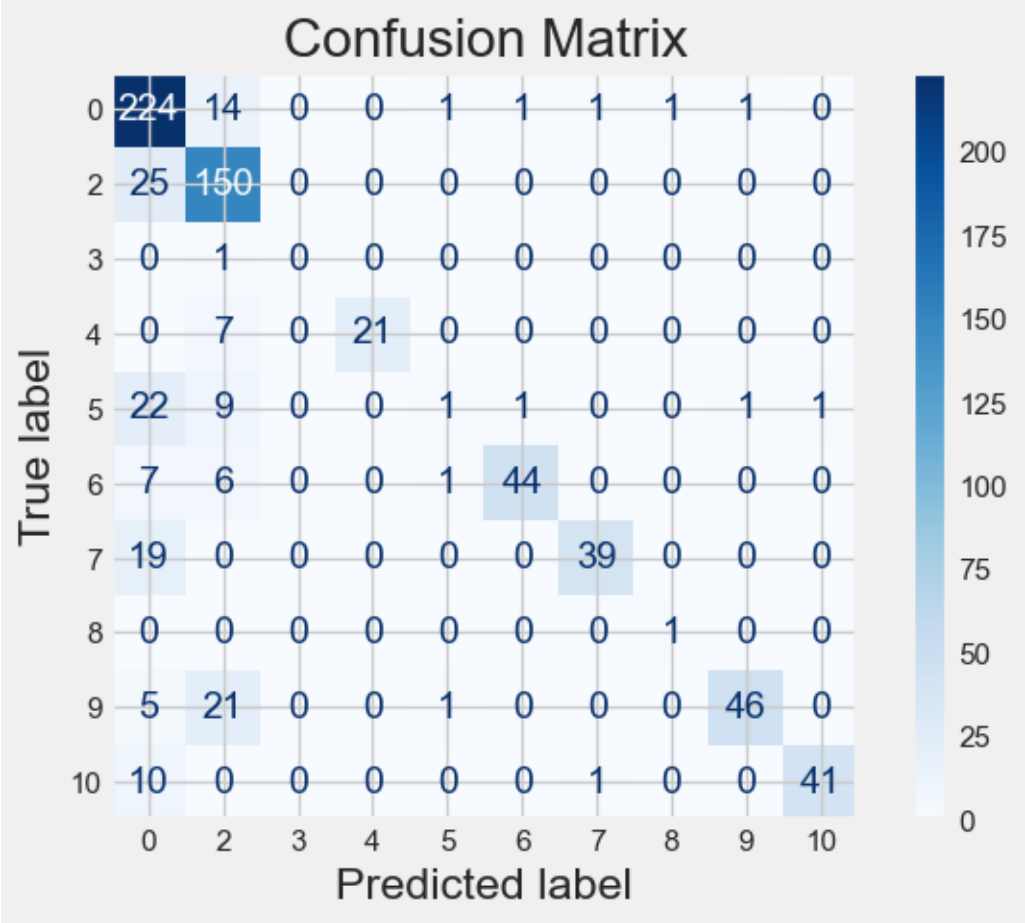
[48]:
```
plot_learning_curve(knn_model, "K-Nearest Neighbors Learning Curve", X_train,␣
↪Y_train, ylim=[1.01, 0.5], cv=5)
```

```
C:\Users\rakes\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\model_selection\_split.py:776: UserWarning:

The least populated class in y has only 1 members, which is less than
n_splits=5.
```

[48]:
```
<module 'matplotlib.pyplot' from
'C:\\Users\\rakes\\AppData\\Local\\Programs\\Python\\Python312\\Lib\\site-
packages\\matplotlib\\pyplot.py'>
```



[49]:
```
from sklearn.metrics import ConfusionMatrixDisplay
ConfusionMatrixDisplay.from_estimator(knn_model, X_test, Y_test, cmap='Blues')
plt.title("Confusion Matrix")
plt.show()
```



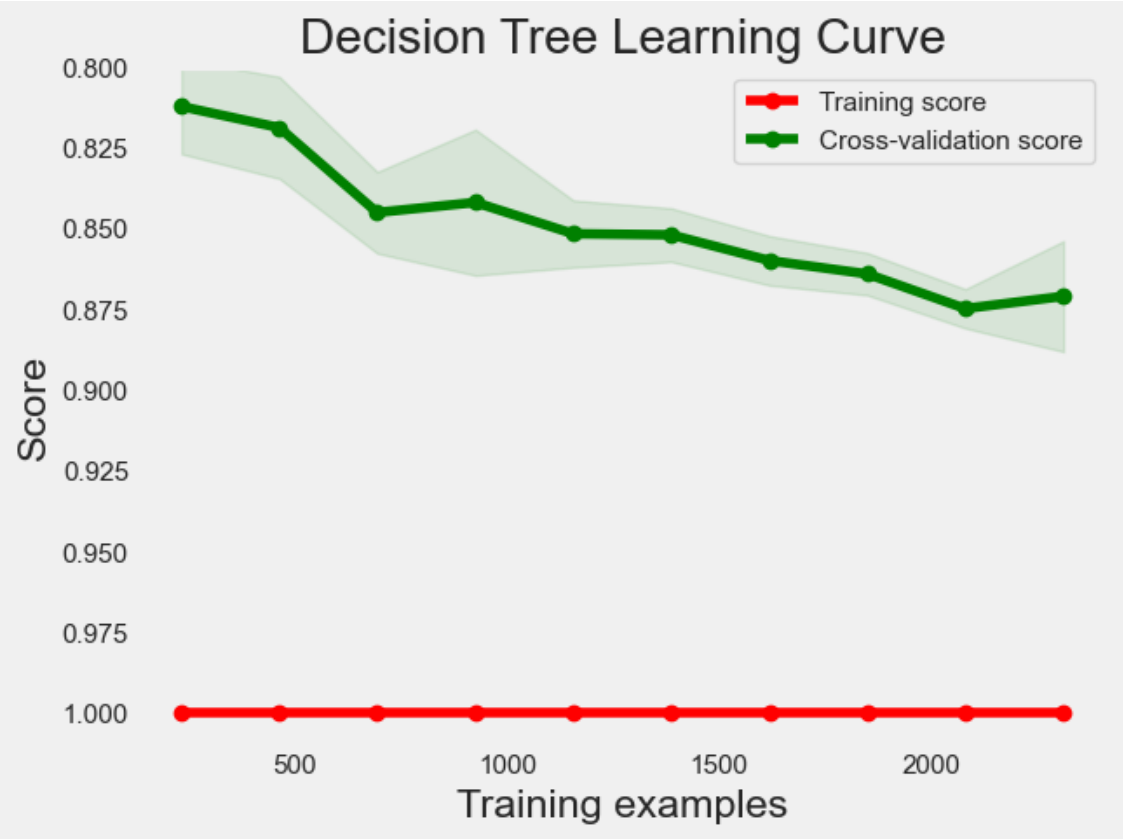
[50]:
```
from sklearn.tree import DecisionTreeClassifier
dt_model = DecisionTreeClassifier(random_state=0)
dt_model.fit(X_train, Y_train)
predictions = dt_model.predict(X_test)
accuracy = accuracy_score(Y_test, predictions)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
Accuracy: 85.77%
```

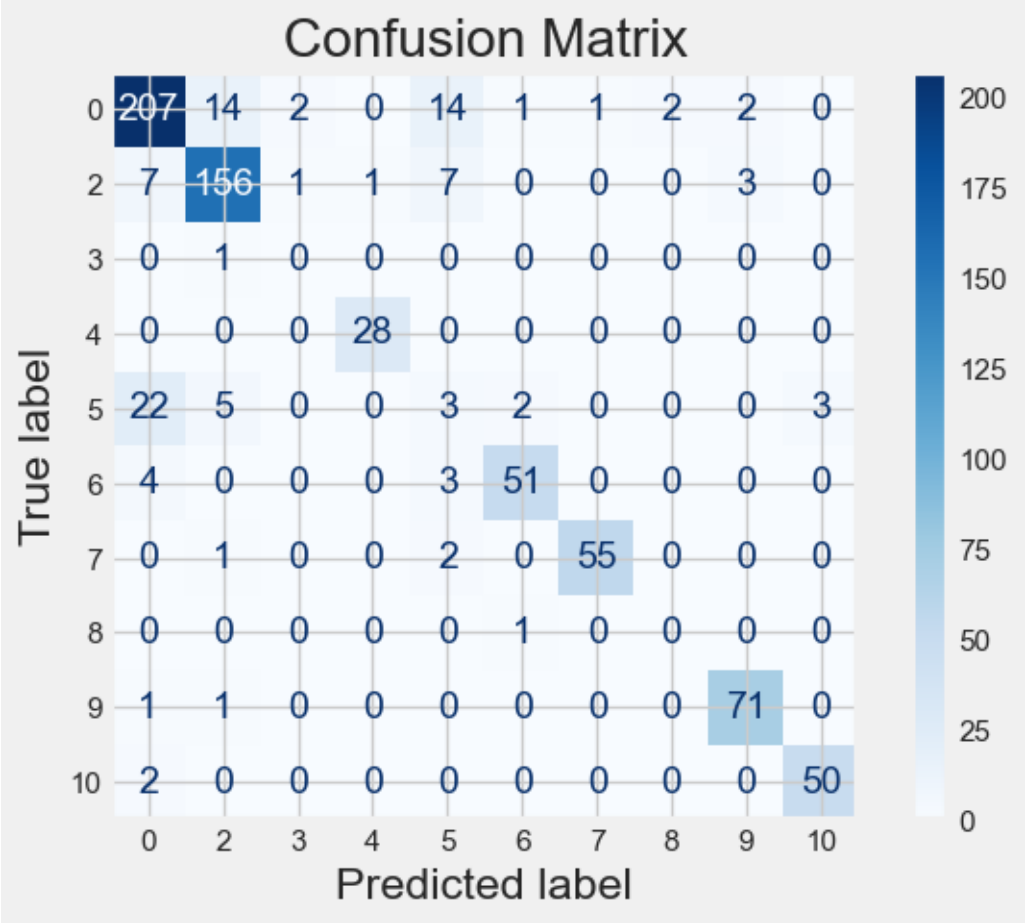
[51]:
```
plot_learning_curve(dt_model, "Decision Tree Learning Curve", X_train, Y_train,␣
↪ylim=[1.01, 0.8], cv=5)
```

```
C:\Users\rakes\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\model_selection\_split.py:776: UserWarning:

The least populated class in y has only 1 members, which is less than
n_splits=5.
```

[51]:
```
<module 'matplotlib.pyplot' from
'C:\\Users\\rakes\\AppData\\Local\\Programs\\Python\\Python312\\Lib\\site-
packages\\matplotlib\\pyplot.py'>
```



[53]:
```
from sklearn.metrics import ConfusionMatrixDisplay
ConfusionMatrixDisplay.from_estimator(dt_model, X_test, Y_test, cmap='Blues')
plt.title("Confusion Matrix")
plt.show()
```



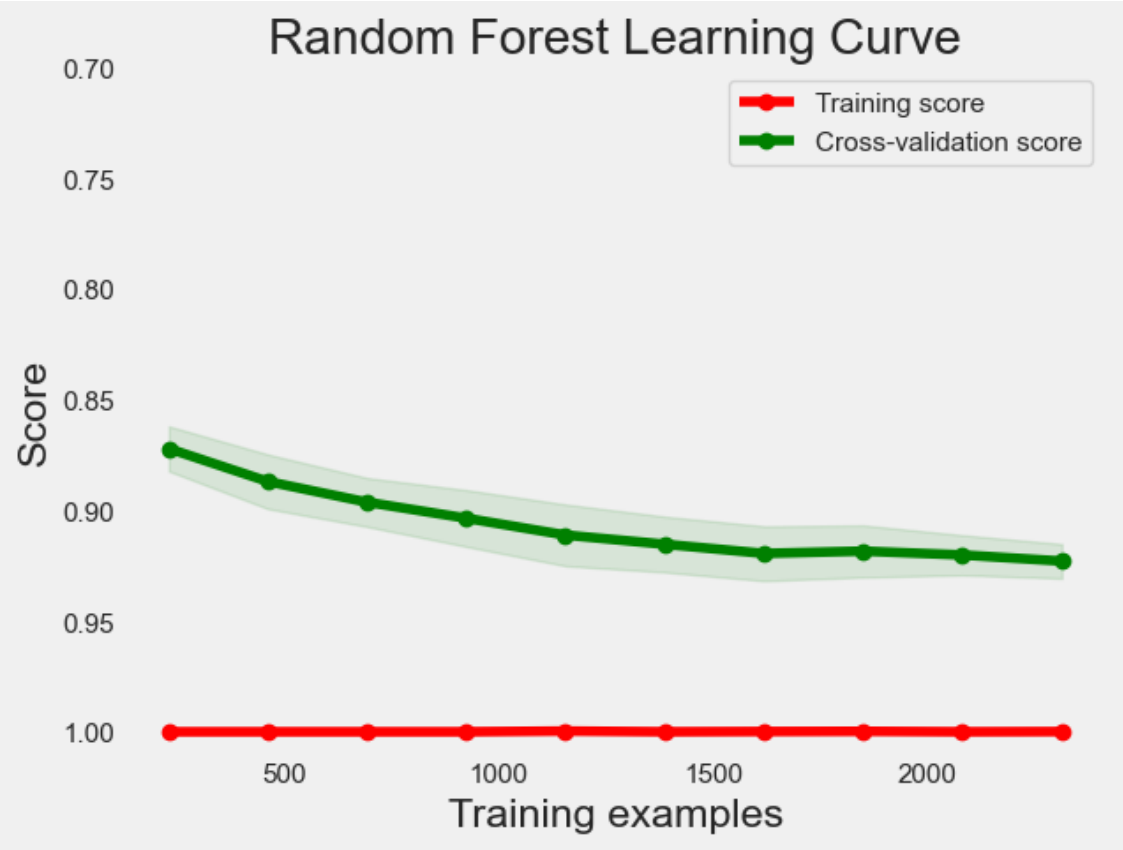
[54]:
```
from sklearn.ensemble import RandomForestClassifier
rf_model = RandomForestClassifier(random_state=0)
rf_model.fit(X_train, Y_train)
predictions = rf_model.predict(X_test)
accuracy = accuracy_score(Y_test, predictions)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
Accuracy: 92.54%
```

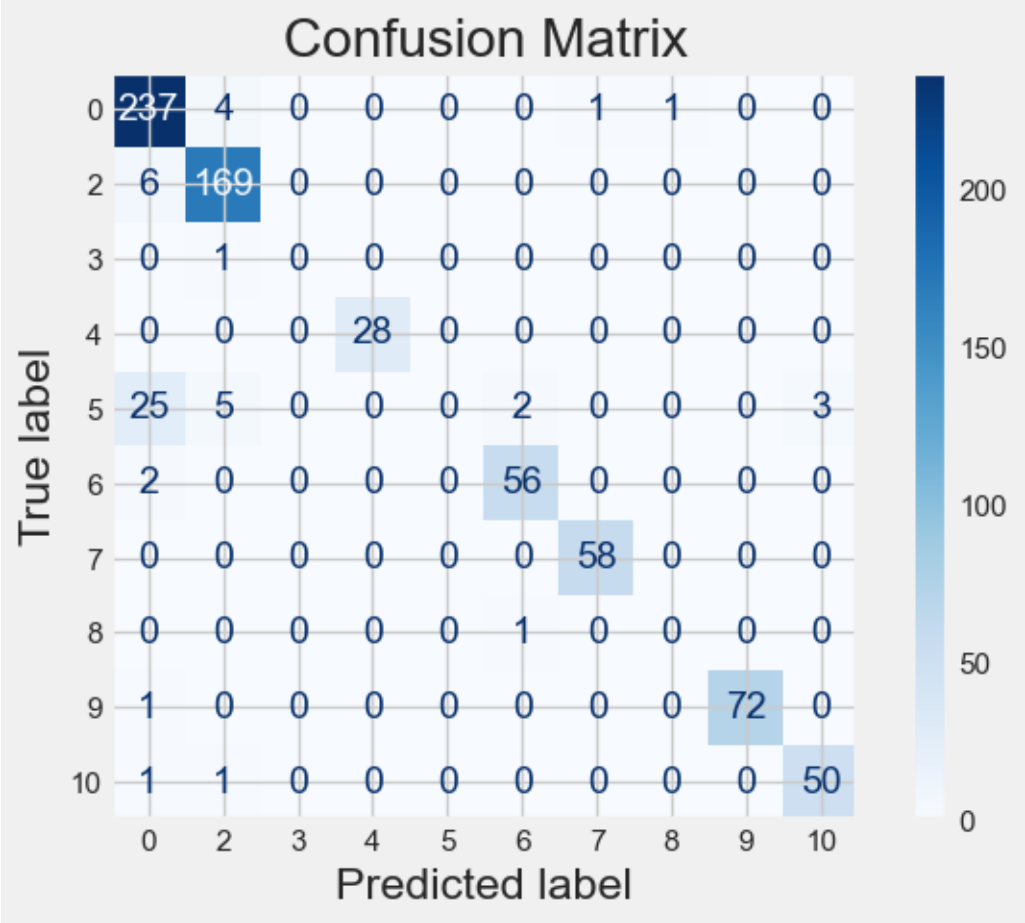
[55]:
```
plot_learning_curve(rf_model, "Random Forest Learning Curve", X_train, Y_train,␣
↪ylim=[1.01, 0.7], cv=5)
```

```
C:\Users\rakes\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\model_selection\_split.py:776: UserWarning:

The least populated class in y has only 1 members, which is less than
n_splits=5.
```

[55]:
```
<module 'matplotlib.pyplot' from
'C:\\Users\\rakes\\AppData\\Local\\Programs\\Python\\Python312\\Lib\\site-
packages\\matplotlib\\pyplot.py'>
```



[56]:
```
from sklearn.metrics import ConfusionMatrixDisplay
ConfusionMatrixDisplay.from_estimator(rf_model, X_test, Y_test, cmap='Blues')
plt.title("Confusion Matrix")
plt.show()
```



[57]:
```
#time consume
import numpy as np
from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
svc_model = svm.SVC(kernel='linear')
svc_model.fit(X_train, Y_train)
predictions = svc_model.predict(X_test)
accuracy = accuracy_score(Y_test, predictions)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

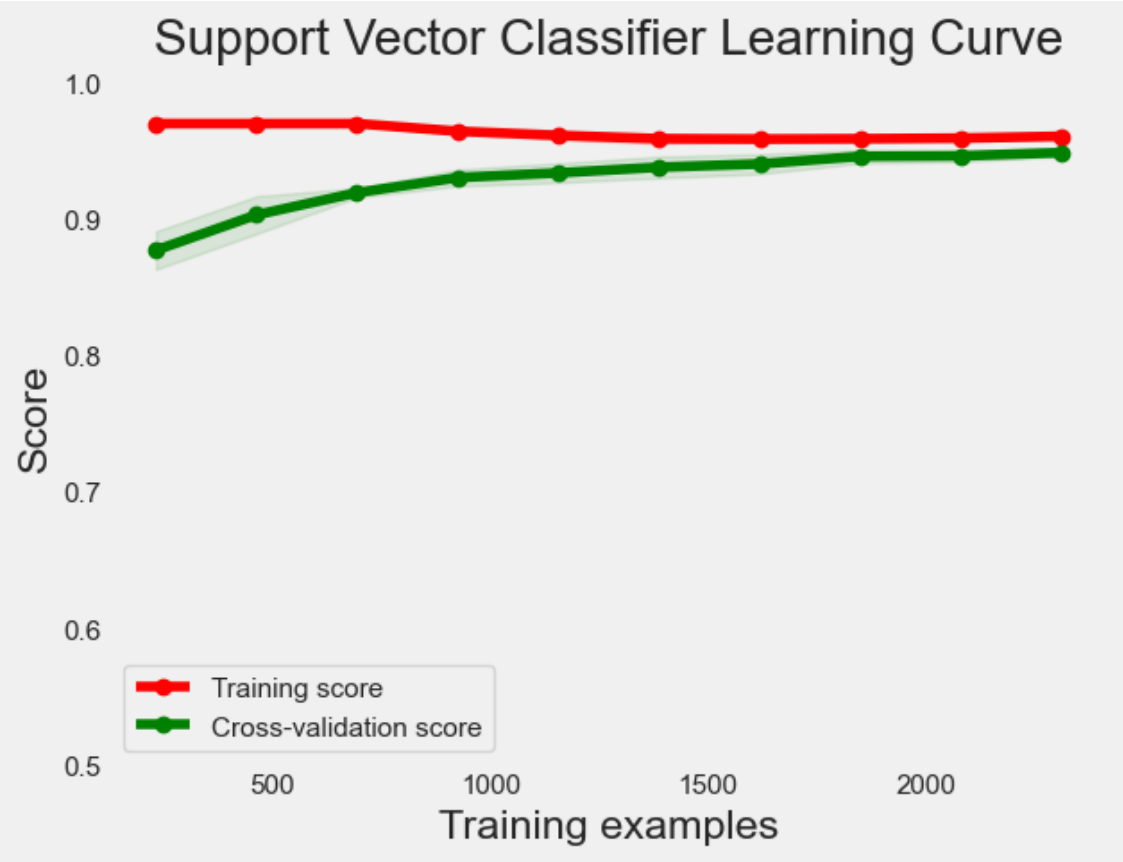
```
Accuracy: 93.65%
```

[58]:
```
plot_learning_curve(svc_model, "Support Vector Classifier Learning Curve",␣
↪X_train, Y_train, ylim=[0.5, 1.01], cv=5)
```

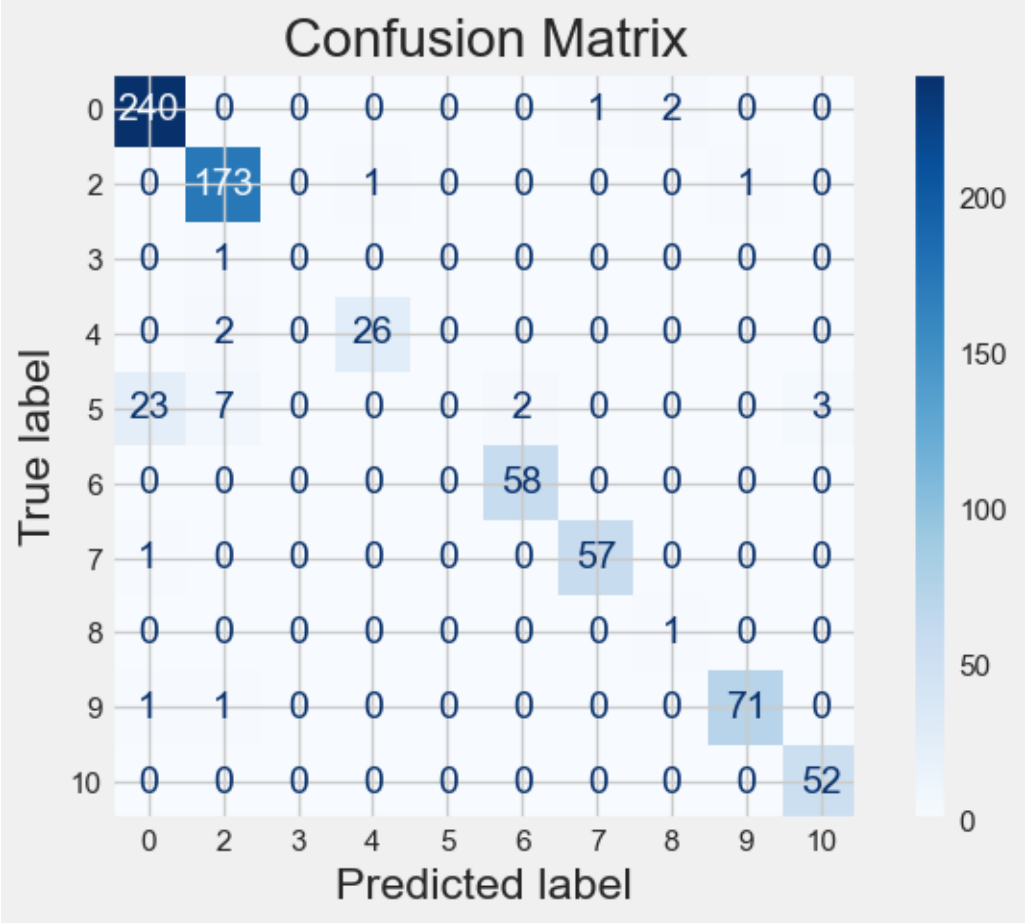
```
C:\Users\rakes\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\model_selection\_split.py:776: UserWarning:
```

```
The least populated class in y has only 1 members, which is less than
n_splits=5.
```

[58]:
```
<module 'matplotlib.pyplot' from
'C:\\Users\\rakes\\AppData\\Local\\Programs\\Python\\Python312\\Lib\\site-
packages\\matplotlib\\pyplot.py'>
```



[59]:
```
from sklearn.metrics import ConfusionMatrixDisplay
ConfusionMatrixDisplay.from_estimator(svc_model, X_test, Y_test, cmap='Blues')
plt.title("Confusion Matrix")
plt.show()
```



1.6 Conclusion:

The analysis and development conducted in this project demonstrate the potential of leveraging purchase data from an e-commerce platform to gain valuable insights into customer behavior and product trends. By grouping products into five main categories and classifying customers into 11 distinct segments based on their purchasing habits, spending patterns, and visit frequencies, the study provides a foundation for predictive modeling. The trained classifiers, utilizing key variables such as average basket amount and category spending percentages, achieved a classification accuracy of 90%, indicating promising results despite potential limitations in the dataset.

The analysis highlights critical areas for improvement, such as addressing seasonal variations in purchasing behavior to refine customer categorization. Expanding the dataset to cover a more extended time frame could significantly enhance the accuracy and robustness of the predictions. Overall, this project demonstrates the viability of customer classification and purchase prediction in e-commerce, paving the way for more personalized marketing strategies and improved customer engagement.