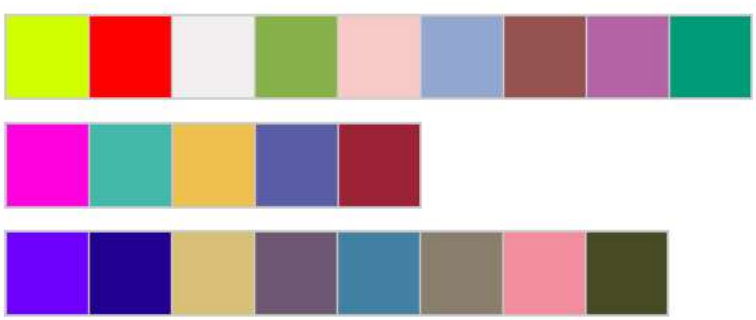
```
# -*- coding: utf-8 -*-

#import the libraries
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt #Data Visualization
import seaborn as sns  #Python library for Vidualization

from plotly.subplots import make_subplots
import plotly.graph_objects as go
import plotly.express as px
import plotly.io as pio
from termcolor import colored
import warnings
warnings.filterwarnings('ignore')

from sklearn.preprocessing import StandardScaler
from sklearn.impute import KNNImputer
from sklearn.cluster import KMeans, MiniBatchKMeans, MeanShift, estimate_bandwidth, AffinityPropagation, Birch, DBSCAN, OPTICS, AgglomerativeC
from yellowbrick.cluster import KElbowVisualizer
import scipy.cluster.hierarchy as sch
from sklearn.mixture import GaussianMixture, BayesianGaussianMixture
from sklearn.model_selection import GridSearchCV
from sklearn import set_config
from sklearn.metrics import silhouette_score, calinski_harabasz_score
from sklearn.decomposition import PCA
from itertools import product
from sklearn.neighbors import NearestNeighbors
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA
try:
    from kneed import KneeLocator
except:
    !pip install kneed
    from kneed import KneeLocator

dataset = pd.read_csv('./Customer_Data.csv')

dataset.head(10) #Printing first 10 rows of the dataset

#total rows and colums in the dataset
dataset.shape

# there are no missing values as all the columns has 200 entries properly
dataset.info()

#Missing values computation
dataset.isnull().sum()
```

```
Collecting kneed
  Downloading kneed-0.8.5-py3-none-any.whl (10 kB)
Requirement already satisfied: numpy>=1.14.2 in /usr/local/lib/python3.10/dist-packages (from kneed) (1.25.2)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from kneed) (1.11.4)
Installing collected packages: kneed
Successfully installed kneed-0.8.5
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8950 entries, 0 to 8949
Data columns (total 18 columns):
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
 0   cust_id                           8950 non-null   object
 1   balance                           8950 non-null   float64
 2   balance_frequency                 8950 non-null   float64
 3   purchases                         8950 non-null   float64
 4   oneoff_purchases                  8950 non-null   float64
 5   installments_purchases            8950 non-null   float64
 6   cash_advance                      8950 non-null   float64
 7   purchases_frequency               8950 non-null   float64
 8   oneoff_purchases_frequency        8950 non-null   float64
 9   purchases_installments_frequency  8950 non-null   float64
 10  cash_advance_frequency            8950 non-null   float64
 11  cash_advance_trx                  8950 non-null   int64
 12  purchases_trx                     8950 non-null   int64
 13  credit_limit                      8949 non-null   float64
 14  payments                          8950 non-null   float64
 15  minimum_payments                  8637 non-null   float64
 16  prc_full_payment                  8950 non-null   float64
 17  tenure                            8950 non-null   int64
```

```
dtypes: float64(14), int64(3), object(1)
memory usage: 1.2+ MB
cust_id                             0
balance                             0
balance_frequency                   0
purchases                           0
oneoff_purchases                    0
installments_purchases              0
cash_advance                        0
purchases_frequency                 0
oneoff_purchases_frequency          0
purchases_installments_frequency    0
cash_advance_frequency              0
cash_advance_trx                    0
purchases_trx                       0
credit_limit                        1
payments                            0
minimum_payments                  313
prc_full_payment                    0
tenure                              0
dtype: int64
```

```
# Define a custom function to display the color palettes
def custom_palette(custom_colors):
    """Show color palette that use in this notebook"""
    customPalette = sns.set_palette(sns.color_palette(custom_colors))
    sns.palplot(sns.color_palette(custom_colors), size=0.8)
    plt.tick_params(axis='both', labelsize=0, length=0)

# Define the FONT dictionary for plot titles
FONT = {'fontsize': 30, 'fontstyle': 'normal', 'fontfamily': 'DejaVu Sans', 'backgroundcolor': '#B41B10', 'color': '#E4C09E'}

# Create List of Color Palettes with premium and pastel colors
colors1 = ["#d0ff00", "#ff0000", "#F1F0F0", "#88B04B", "#F7CAC9", "#92A8D1", "#955251", "#B565A7", "#009B77"]
colors2 = ["#ff00e1", "#45B8AC", "#EFC050", "#5B5EA6", "#9B2335"]
dark_colors = ["#6f00ff","#240090", "#D9BF77", "#6E5773", "#4281A4", "#8A7E6D", "#F18D9E", "#474B24"]

# Plot Color Palettes
for color in [colors1, colors2, dark_colors]:
    custom_palette(color)
```



```
# Import customer dataset
dataset = pd.read_csv("./Customer_Data.csv")
dataset
```

	cust_id	balance	balance_frequency	purchases	oneoff_purchases	installments_purchases	cash_advance	purchases_frequency	or
0	C10001	40.900749	0.818182	95.40	0.00	95.40	0.000000	0.166667	
1	C10002	3202.467416	0.909091	0.00	0.00	0.00	6442.945483	0.000000	
2	C10003	2495.148862	1.000000	773.17	773.17	0.00	0.000000	1.000000	
3	C10004	1666.670542	0.636364	1499.00	1499.00	0.00	205.788017	0.083333	
4	C10005	817.714335	1.000000	16.00	16.00	0.00	0.000000	0.083333	
...	...	...	...	...	...	...	...	...	
8945	C19186	28.493517	1.000000	291.12	0.00	291.12	0.000000	1.000000	
8946	C19187	19.183215	1.000000	300.00	0.00	300.00	0.000000	1.000000	
8947	C19188	23.398673	0.833333	144.40	0.00	144.40	0.000000	0.833333	
8948	C19189	13.457564	0.833333	0.00	0.00	0.00	36.558778	0.000000	
8949	C19190	372.708075	0.666667	1093.25	1093.25	0.00	127.040008	0.666667	

8950 rows × 18 columns

```
# Overview of dataset
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8950 entries, 0 to 8949
Data columns (total 18 columns):
 #   Column                            Non-Null Count  Dtype  
---  ------                            --------------  -----  
 0   cust_id                           8950 non-null   object 
 1   balance                           8950 non-null   float64
 2   balance_frequency                 8950 non-null   float64
 3   purchases                         8950 non-null   float64
 4   oneoff_purchases                  8950 non-null   float64
 5   installments_purchases            8950 non-null   float64
 6   cash_advance                      8950 non-null   float64
 7   purchases_frequency               8950 non-null   float64
 8   oneoff_purchases_frequency        8950 non-null   float64
 9   purchases_installments_frequency  8950 non-null   float64
 10  cash_advance_frequency            8950 non-null   float64
 11  cash_advance_trx                  8950 non-null   int64  
 12  purchases_trx                     8950 non-null   int64  
 13  credit_limit                      8949 non-null   float64
 14  payments                          8950 non-null   float64
 15  minimum_payments                  8637 non-null   float64
 16  prc_full_payment                  8950 non-null   float64
 17  tenure                            8950 non-null   int64  
dtypes: float64(14), int64(3), object(1)
memory usage: 1.2+ MB
```

```
# Drop the cust_id columns because we do not neet it
dataset = dataset.drop('cust_id', axis=1)


def highlight_cells(val):
    """Highlight cells that is not zero"""
    f_color = '#FFFF00' if val != 0 else ''  # Yellow
    bg_color = '#8F00FF' if val != 0 else ''   # Violet
    w_font = 'bold'
    return 'background-color: {}; color: {}; font-weight: {}'.format(bg_color, f_color, w_font)

# Check missing values
is_nan = dataset.isna().sum().to_frame(name='Number_of_NaN')
is_nan.insert(1,'Percent(%)', [round(x/dataset.shape[0]*100,2) for x in is_nan.Number_of_NaN])
is_nan.style.applymap(highlight_cells)
```

	Number_of_NaN	Percent(%)
balance	0	0.000000
balance_frequency	0	0.000000
purchases	0	0.000000
oneoff_purchases	0	0.000000
installments_purchases	0	0.000000
cash_advance	0	0.000000
purchases_frequency	0	0.000000
oneoff_purchases_frequency	0	0.000000
purchases_installments_frequency	0	0.000000
cash_advance_frequency	0	0.000000
cash_advance_trx	0	0.000000
purchases_trx	0	0.000000
credit_limit	1	0.010000
payments	0	0.000000
minimum_payments	313	3.500000
prc_full_payment	0	0.000000
tenure	0	0.000000

```
# Overview nan value in credit_limit column
dataset[dataset.credit_limit.isna()]
```

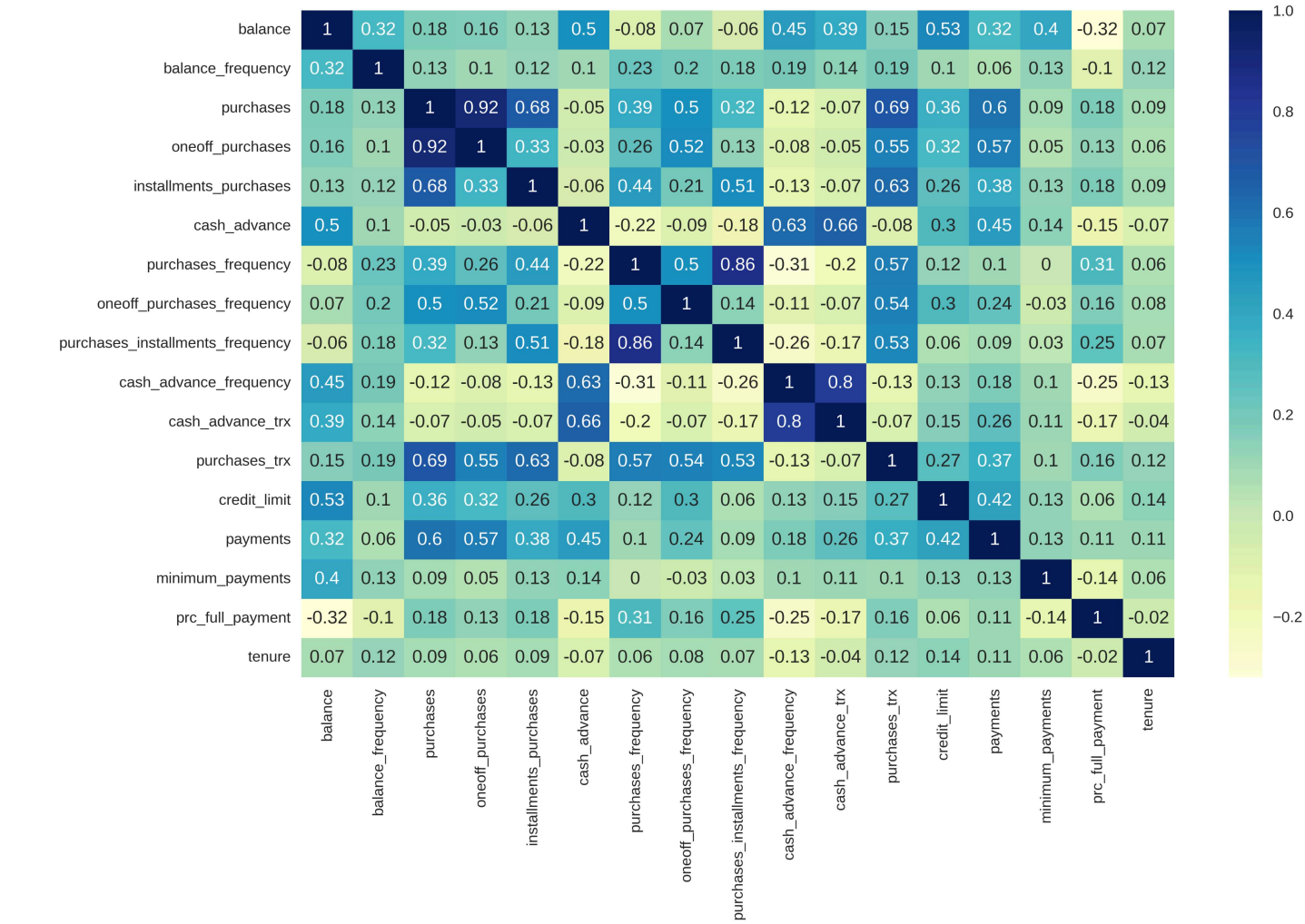
	balance	balance_frequency	purchases	oneoff_purchases	installments_purchases	cash_advance	purchases_frequency	oneoff_purcha
5203	18.400472	0.166667	0.0	0.0	0.0	186.853063	0.0	

```
# Drop row with nan value in credit_limit
dataset = dataset.drop(dataset[dataset.credit_limit.isna()].index, axis=0)


# Define the FONT dictionary for plot titles with a new purple background color
FONT = {'fontsize': 30, 'fontstyle': 'normal', 'fontfamily': 'DejaVu Sans', 'backgroundcolor': '#FFFFFF', 'color': '#002EFF'}

# Show details of data points with missing values
desc_cust = dataset[dataset.minimum_payments.isna()].describe().T
desc_cust_dataset = pd.DataFrame(index=desc_cust.index, columns=desc_cust.columns, data=desc_cust)

f, ax = plt.subplots(figsize=(11, 11), dpi=500)

sns.heatmap(desc_cust_dataset, annot=True, cmap="YlGnBu", fmt='.0f', ax=ax, linewidths=5, cbar=False, annot_kws={"size": 14})

ax.xaxis.set_ticks_position('top')
plt.tick_params(left=False, top=False)
plt.xticks(size=18)
plt.yticks(size=18)

plt.title("Descriptive Statistics of Missing Values", pad=30, x=0.31, y=1.02, fontdict=FONT)
plt.show()
```

Descriptive Statistics of Missing Values

	count	mean	std	min	25%	50%	75%	max
balance	313	555	1293	0	0	17	287	9165
balance_frequency	313	0	0	0	0	0	1	1
purchases	313	393	758	0	1	130	400	7597
oneoff_purchases	313	250	624	0	0	0	176	6761
installments_purchases	313	143	311	0	0	0	152	2959
cash_advance	313	559	1185	0	0	0	480	7616
purchases_frequency	313	0	0	0	0	0	1	1
oneoff_purchases_frequency	313	0	0	0	0	0	0	1
purchases_installments_frequency	313	0	0	0	0	0	0	1
cash_advance_frequency	313	0	0	0	0	0	0	1
cash_advance_trx	313	1	3	0	0	0	1	21
purchases_trx	313	6	10	0	1	2	8	77
credit_limit	313	3732	2925	500	1500	3000	5000	19500
payments	313	322	1997	0	0	0	0	29272
minimum_payments	0							
prc_full_payment	313	0	0	0	0	0	0	0
tenure	313	11	2	6	12	12	12	12

```
# Overview correlation between features
plt.figure(figsize=(13,8), dpi=500)
sns.heatmap(round(dataset.corr(),2), annot=True, cmap='YlGnBu')
plt.title("Correlation Matrix", pad=30, fontdict=FONT)
plt.show()
```




```
# Fill the remaining missing values with KNNimputer()
# Define imputer
imputer = KNNImputer()
# Fit on the dataset
imputer.fit(dataset)
# Transform the dataset
dataset1 = pd.DataFrame(imputer.transform(dataset),columns=dataset.columns)

# Print result
print(colored(f'Missing (before): {dataset.isna().sum().sum()}', 'red'))
print(colored(f'Missing (after): {dataset1.isna().sum().sum()}', 'green'))
```

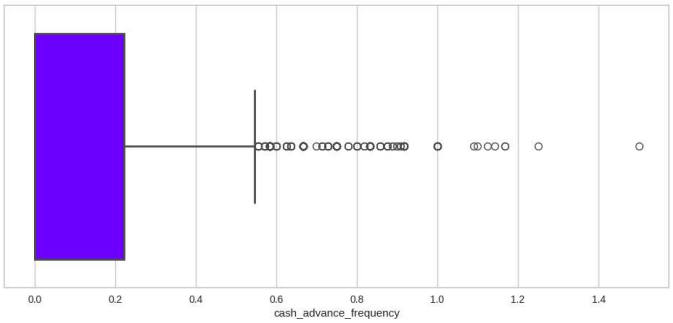
```
Missing (before): 313
Missing (after): 0
```

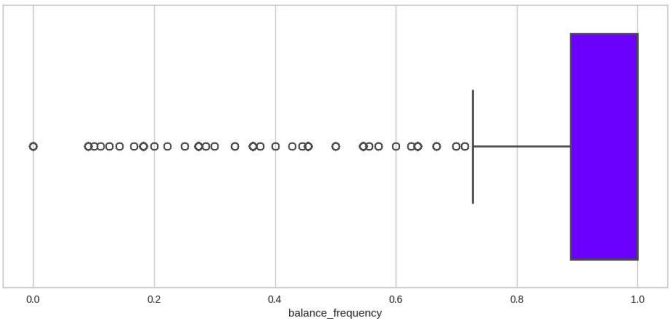
```
# Check duplicated data
print(colored(f"Number of dupilcated data: {dataset1.duplicated().sum()}", 'red'), )
```

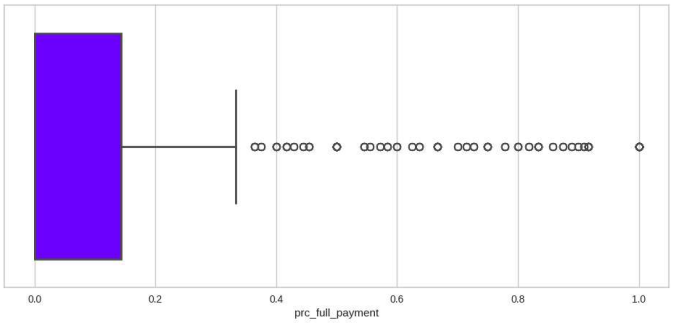
```
Number of dupilcated data: 0
```

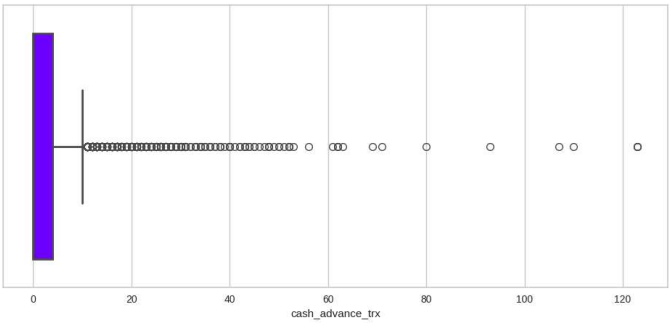
```
# Draw pair plot for check noise and
# pplot = sns.pairplot(dataset1, palette="#B41B10")
# pplot.fig.suptitle("Pariplot of Features", y=1.02, **FONT)
# pplot.tight_layout()
# plt.show()

# Draw boxplot for check outliers
fig ,ax = plt.subplots(8,2,figsize=(20,40))
for i, col in enumerate(set(dataset1.columns)-set(['tenure'])):
    sns.boxplot(data=dataset1, x=col, palette=dark_colors, saturation=1, linewidth = 2, ax=ax[i//2, i%2])
plt.suptitle("Boxplot for Outlier Treatment", y=1, **FONT)
plt.tight_layout(pad=3.0)
plt.show()
```

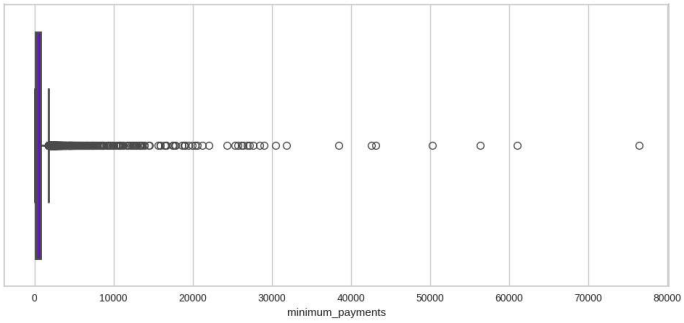
Boxplot for Outlier Treatment

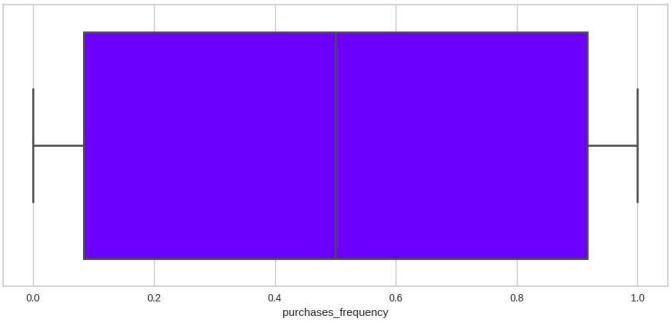


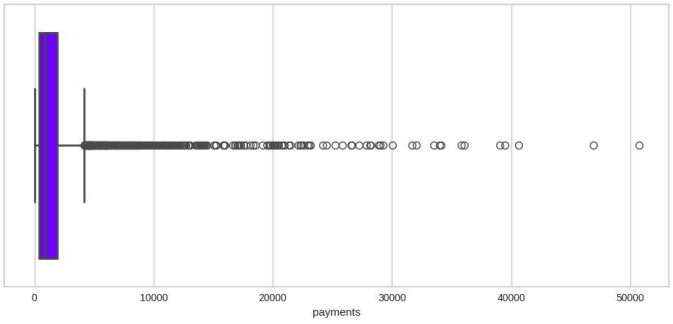


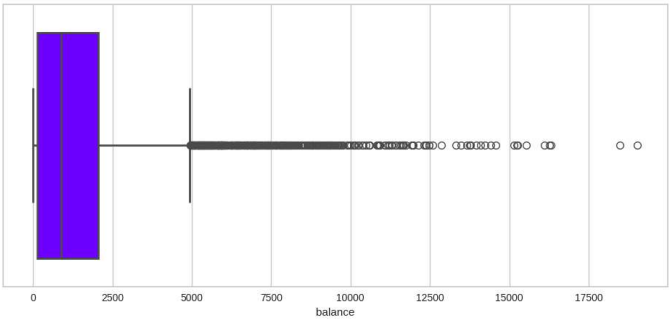


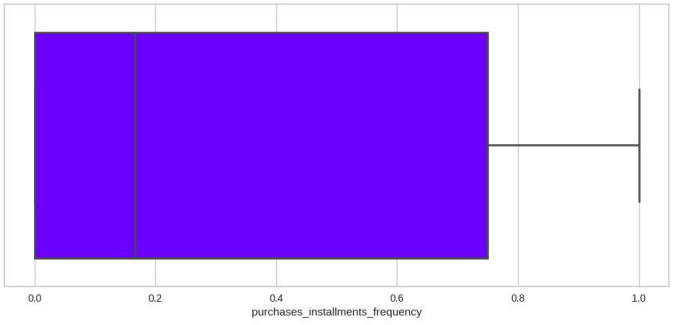


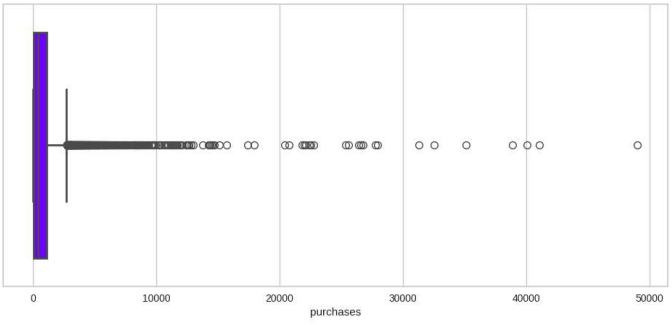


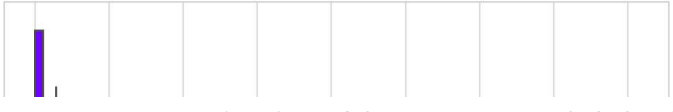


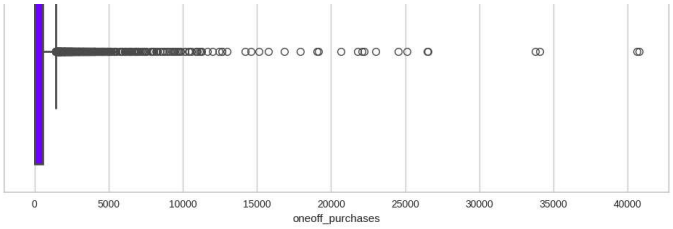


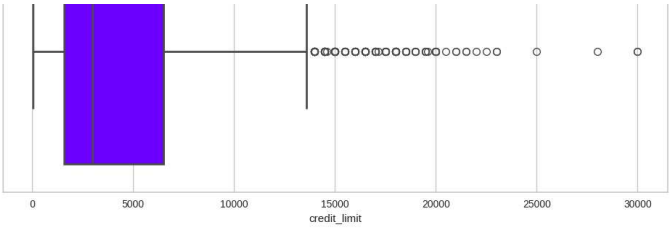


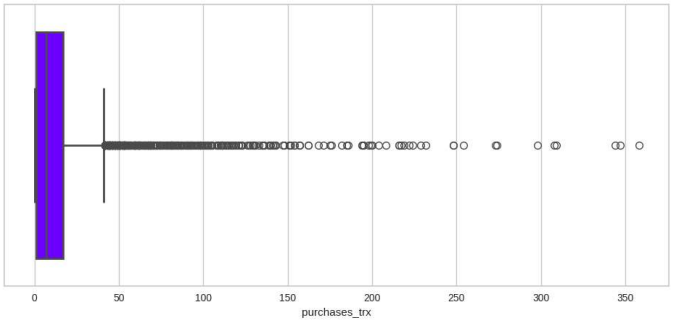


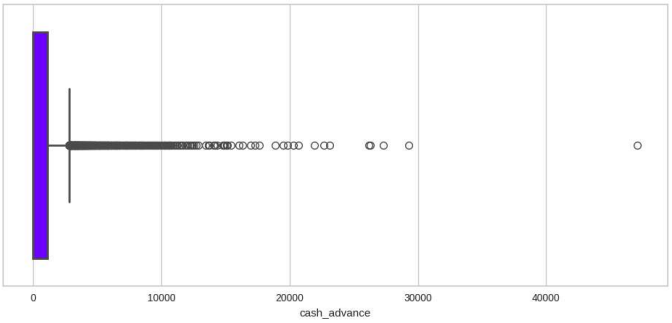


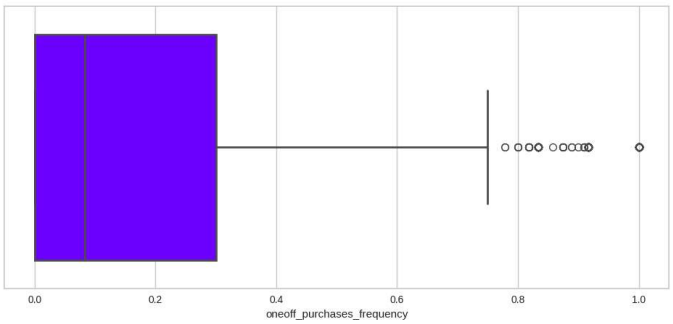


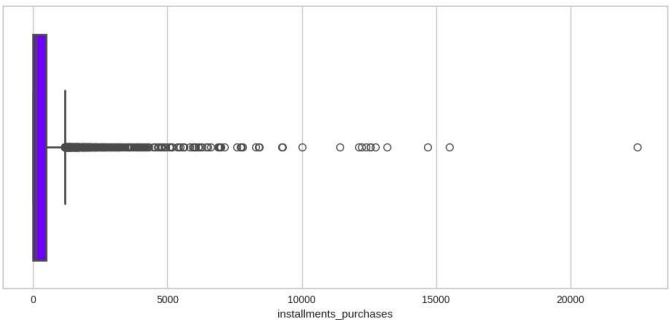













```
from sklearn.preprocessing import PowerTransformer

PT = PowerTransformer()

print(PT.fit_transform(dataset))
```

```
[[-1.23833786 -1.0801604  -0.36831098 ... -0.81526922 -0.67793662
   0.42210751]
 [ 1.05188287 -0.4256199  -1.50536123 ...  0.89703093  1.23484635
   0.42210751]
 [ 0.86050618  0.62852726  0.52149237 ...  0.462236   -0.67793662
   0.42210751]
 ...
 [-1.40957025 -0.9921333  -0.21655169 ... -1.28218108  1.32828513
  -2.52719186]
 [-1.55874115 -0.9921333  -1.50536123 ... -1.63468322  1.32828513
  -2.52719186]
 [-0.32454944 -1.6469605   0.70189133 ... -1.22061016 -0.67793662
  -2.52719186]]
```

```
dataset.shape
```

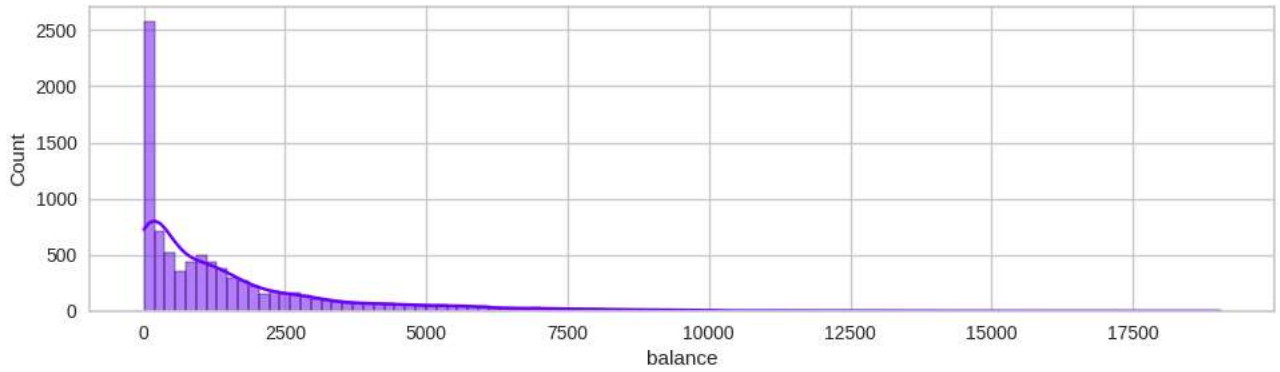
```
(8949, 17)
```

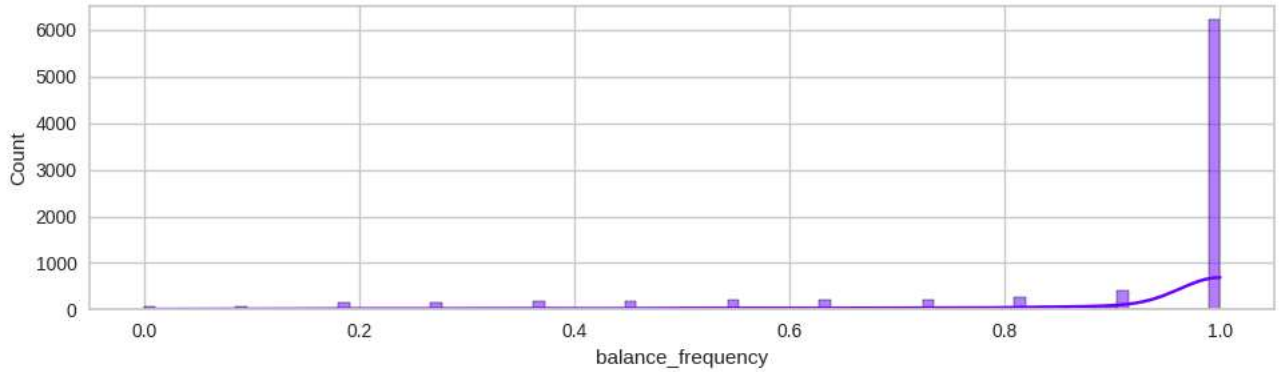
```
print(dataset.columns)
```

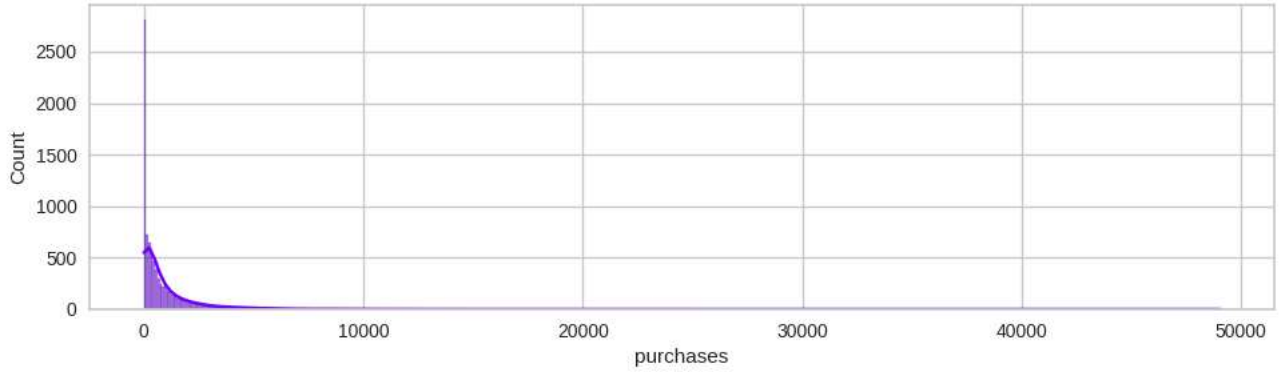
```
Index(['balance', 'balance_frequency', 'purchases', 'oneoff_purchases',
       'installments_purchases', 'cash_advance', 'purchases_frequency',
       'oneoff_purchases_frequency', 'purchases_installments_frequency',
       'cash_advance_frequency', 'cash_advance_trx', 'purchases_trx',
       'credit_limit', 'payments', 'minimum_payments', 'prc_full_payment',
       'tenure'],
      dtype='object')
```

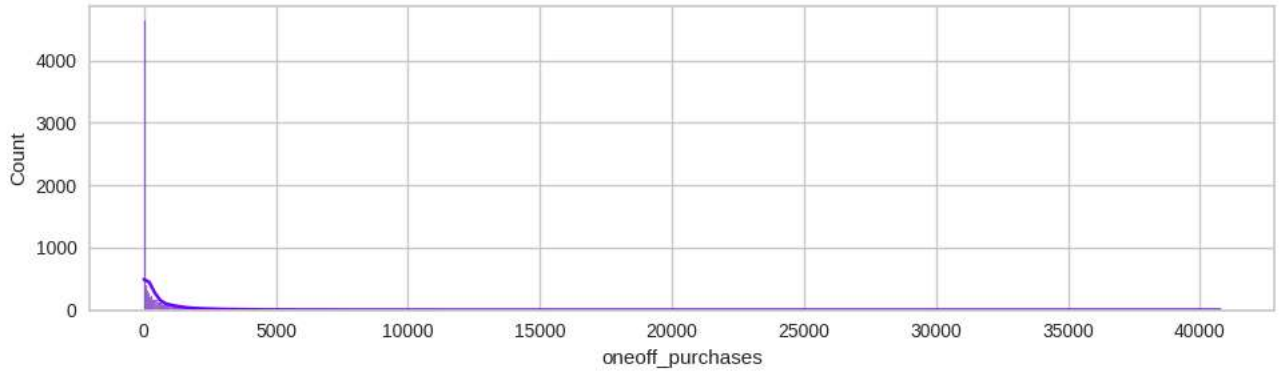
```
# Check distribution of features
fig, ax = plt.subplots(17, 1, figsize=(10,50))
for i, col in enumerate(dataset1):
    sns.histplot(dataset1[col], kde=True, ax=ax[i], palette=colors2)

fig.suptitle('Distribution of Features',y=1.002, **FONT)
fig.tight_layout()
plt.show()
```

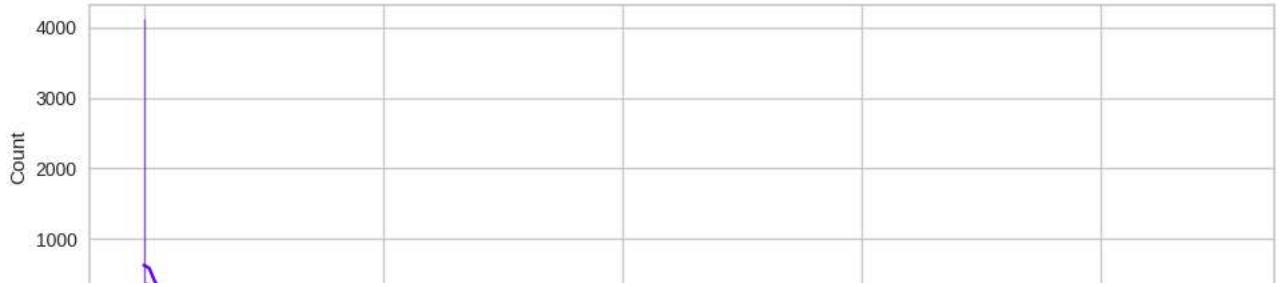
Distribution of Features

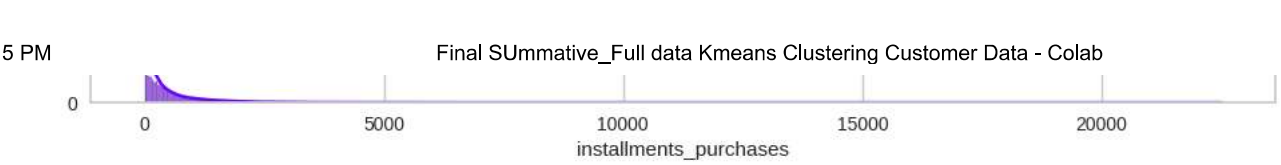


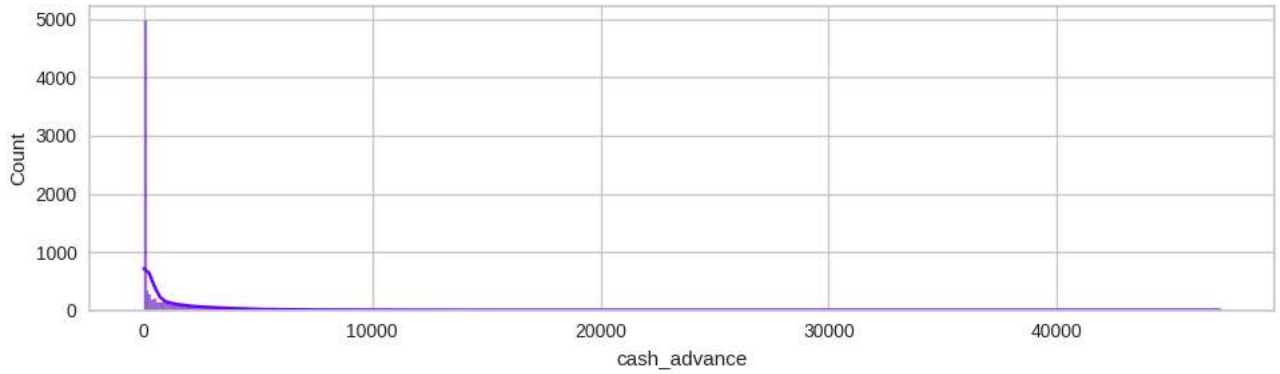


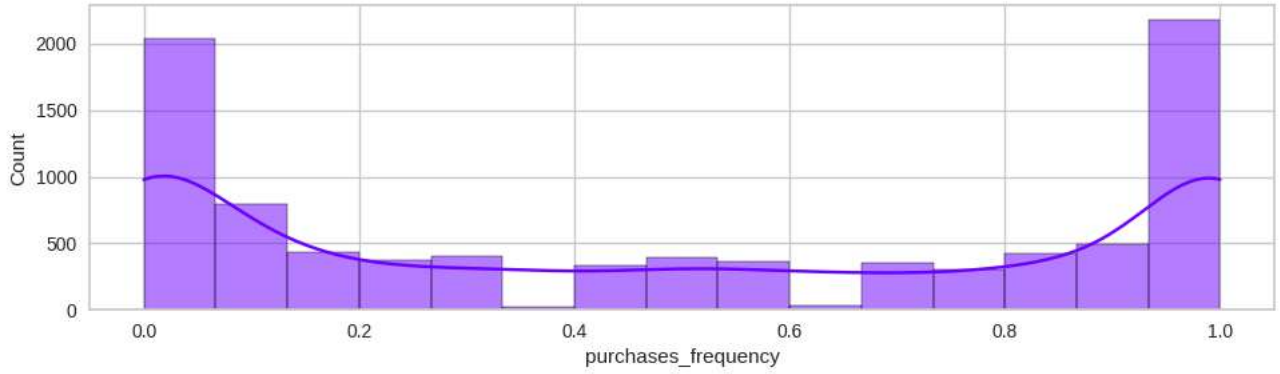


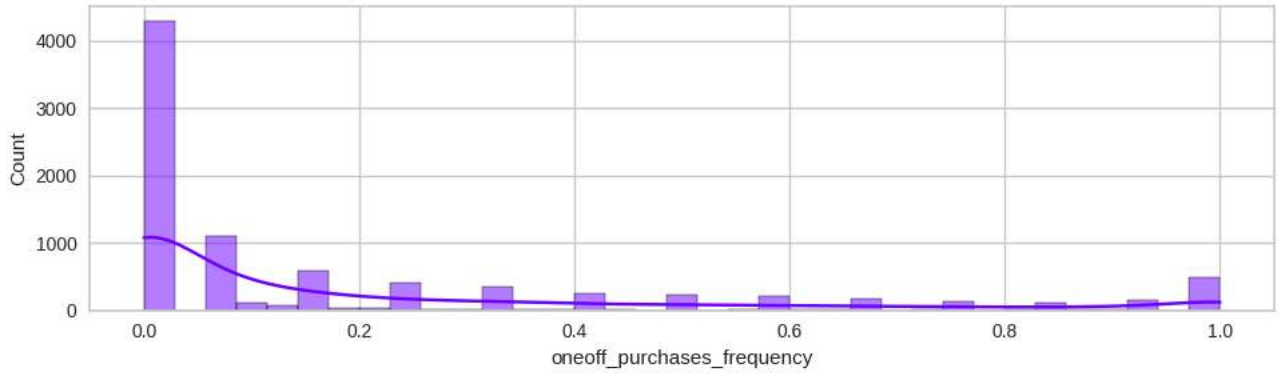


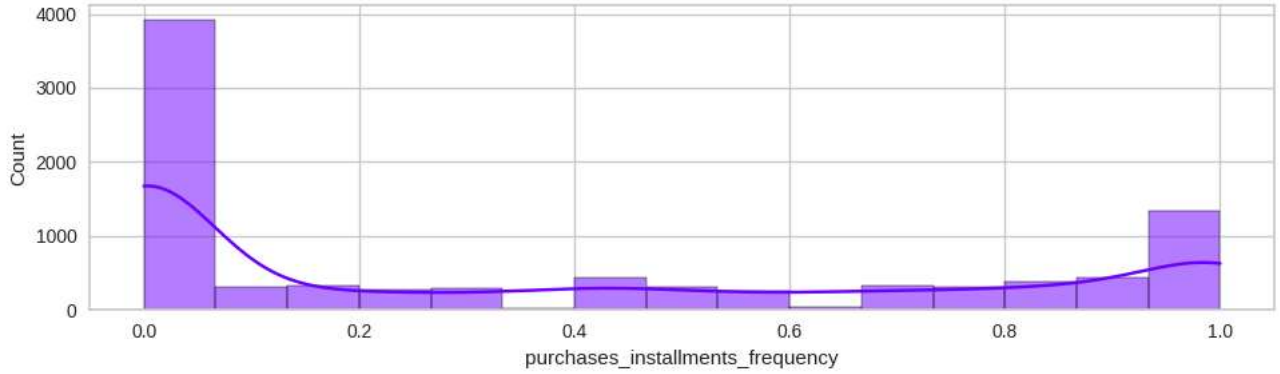


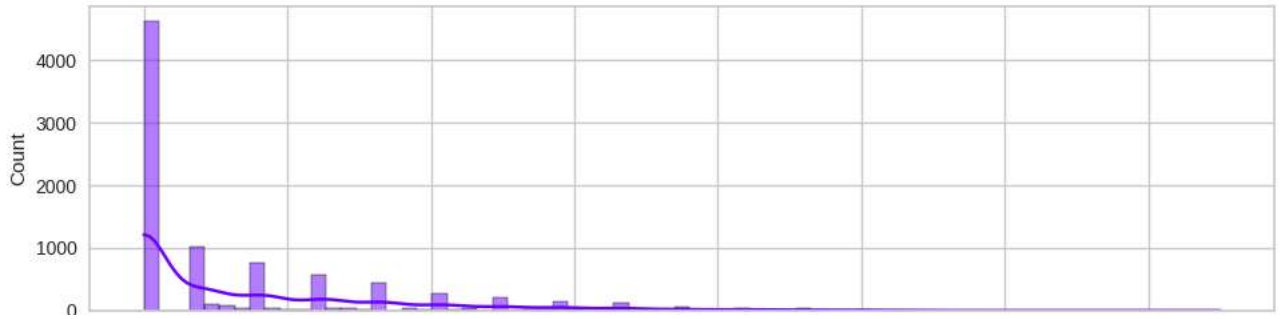

0
0
5000
10000
15000
20000
installments_purchases

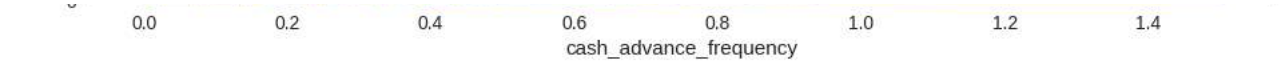

Count
5000
4000
3000
2000
1000
0
0
10000
20000
30000
40000
cash_advance

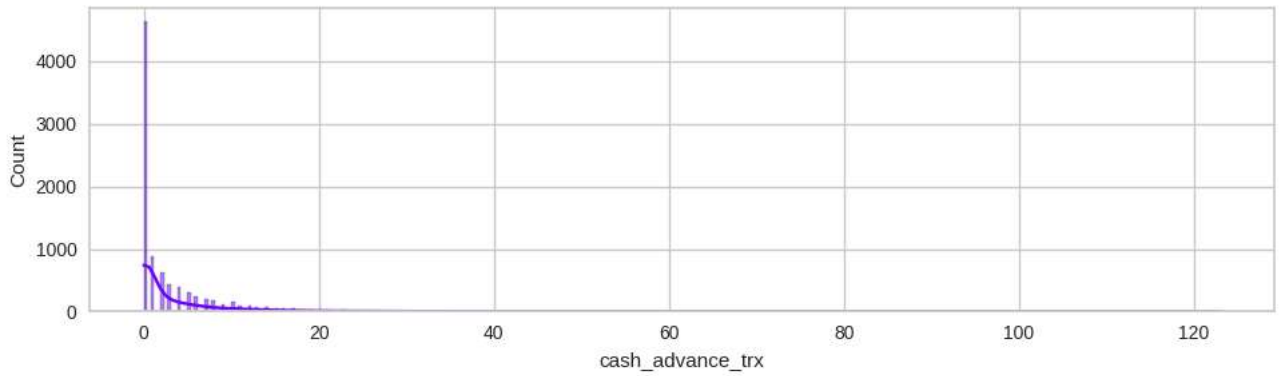

Count
2000
1500
1000
500
0
0.0
0.2
0.4
0.6
0.8
1.0
purchases_frequency

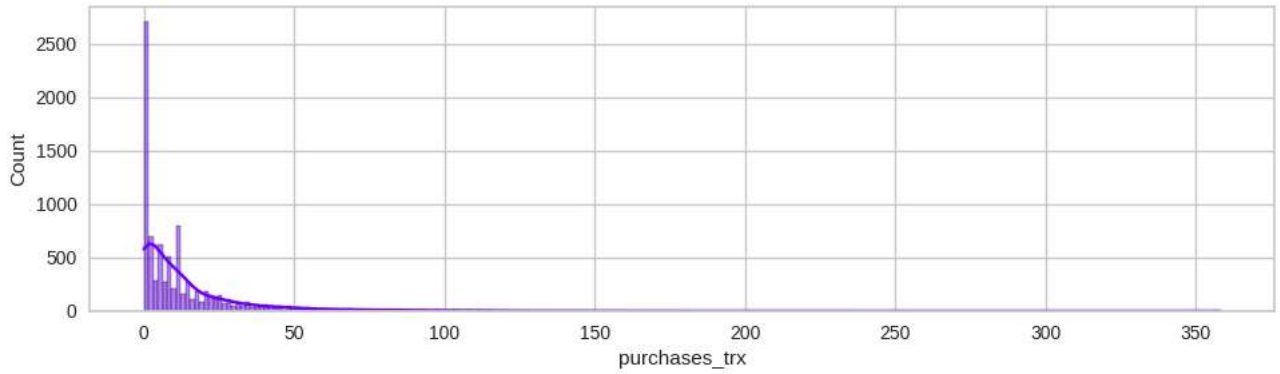

Count
4000
3000
2000
1000
0
0.0
0.2
0.4
0.6
0.8
1.0
oneoff_purchases_frequency

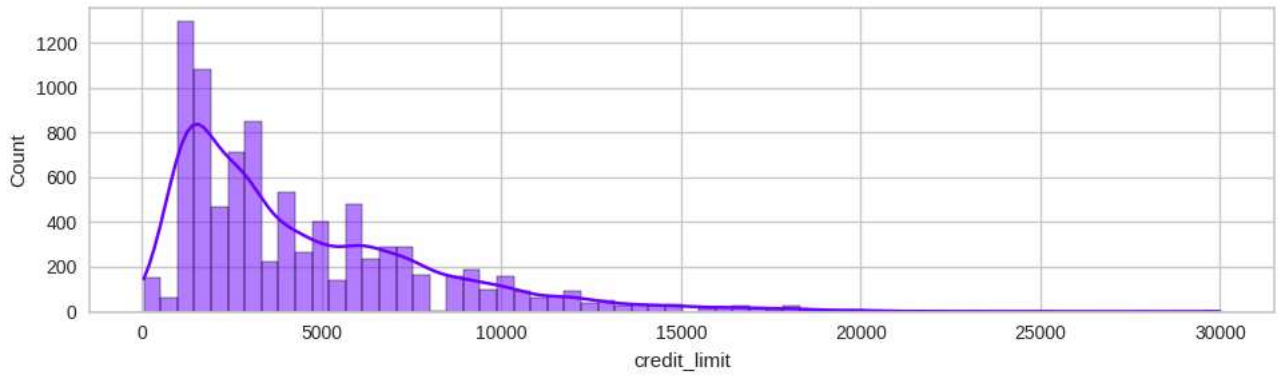

Count
4000
3000
2000
1000
0
0.0
0.2
0.4
0.6
0.8
1.0
purchases_installments_frequency

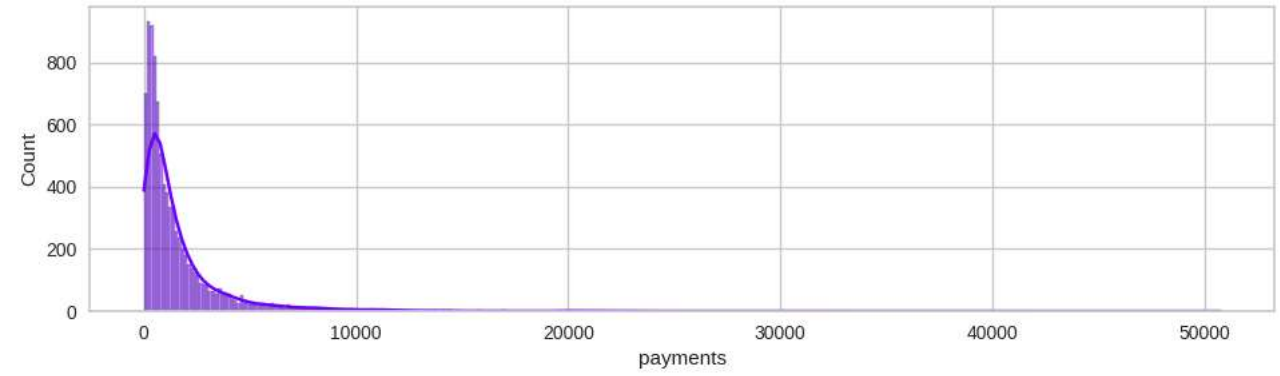

Count
4000
3000
2000
1000
0

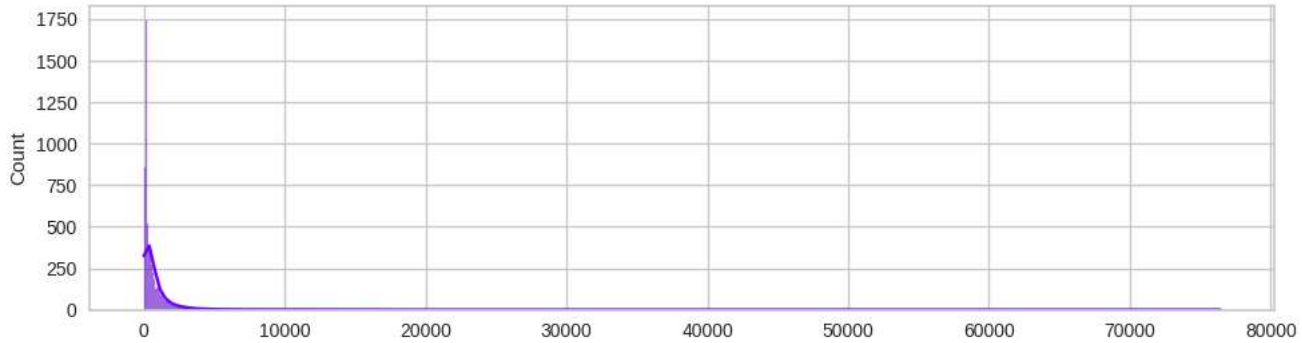

0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
cash_advance_frequency

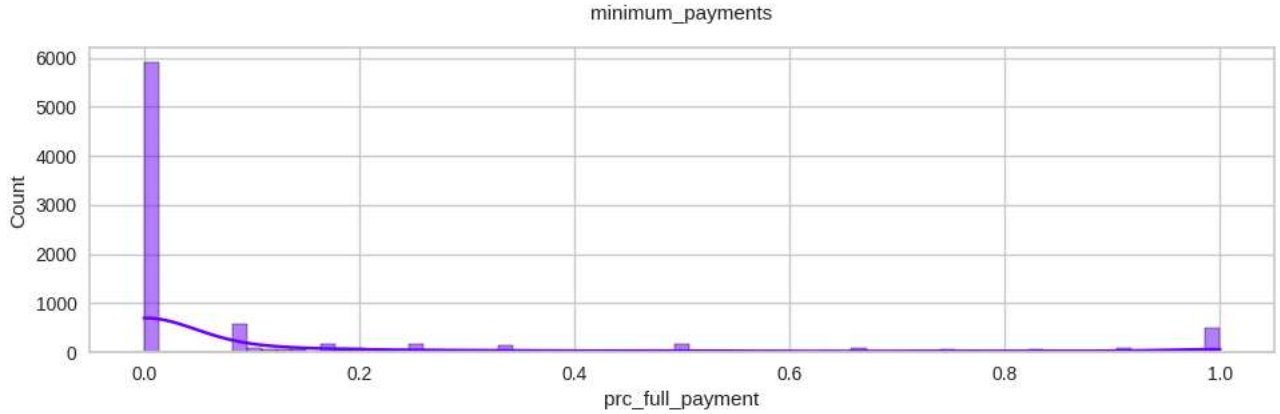

Count
4000
3000
2000
1000
0
0
20
40
60
80
100
120
cash_advance_trx

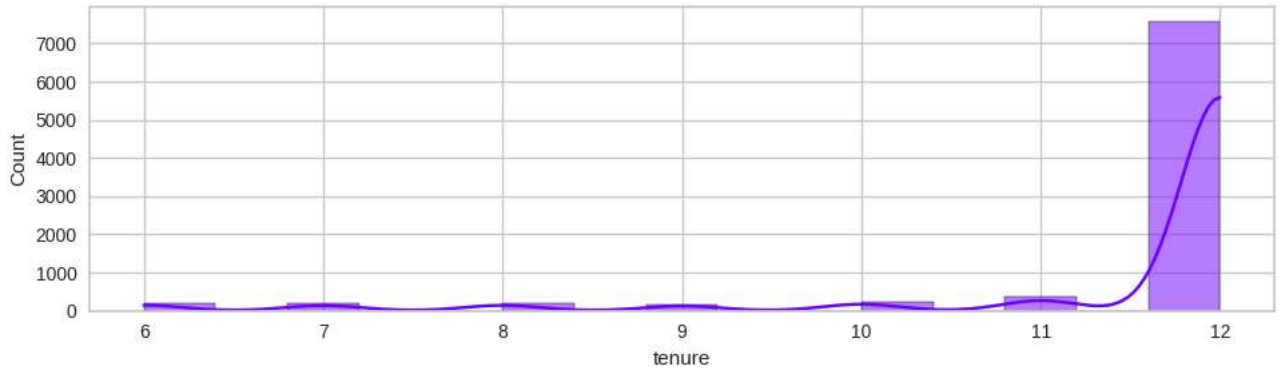

Count
2500
2000
1500
1000
500
0
0
50
100
150
200
250
300
350
purchases_trx


Count
1200
1000
800
600
400
200
0
0
5000
10000
15000
20000
25000
30000
credit_limit


Count
800
600
400
200
0
0
10000
20000
30000
40000
50000
payments


Count
1750
1500
1250
1000
750
500
250
0
0
10000
20000
30000
40000
50000
60000
70000
80000






```
# KPIs

# Check if the dataset contains all required columns (lowercase names)
required_columns = ['purchases', 'cash_advance', 'tenure', 'credit_limit', 'balance', 'payments', 'minimum_payments']
for col in required_columns:
    if col not in dataset.columns:
        raise ValueError(f"Column '{col}' not found in dataset")

# 1. Monthly Average Purchase
dataset['Monthly_Avg_Purchase'] = dataset['purchases'] / dataset['tenure']

# 2. Credit Utilization Ratio
dataset['Credit_Utilization_Ratio'] = (dataset['purchases'] + dataset['cash_advance']) / dataset['credit_limit']

# 3. Limit Usage (Balance to Credit Limit Ratio)
dataset['Limit_Usage'] = dataset['balance'] / dataset['credit_limit']

# 4. Payments to Minimum Payments Ratio
dataset['Pay_to_MinimumPay'] = dataset['payments'] / dataset['minimum_payments']

# 5. Purchase Type Category
dataset['Purchase_Type'] = np.where((dataset['oneoff_purchases'] == 0) & (dataset['installments_purchases'] == 0),
                                    'None_Of_the_Purchases',
                                    np.where((dataset['oneoff_purchases'] > 0) & (dataset['installments_purchases'] == 0),
                                             'One_Off_Purchase',
                                             np.where((dataset['oneoff_purchases'] == 0) & (dataset['installments_purchases'] > 0),
                                                      'Installment_Purchases',
                                                      'Both_Purchases')))

# Display the dataset with new KPIs
print(dataset[['Monthly_Avg_Purchase', 'Credit_Utilization_Ratio', 'Limit_Usage', 'Pay_to_MinimumPay', 'Purchase_Type']].head())
```

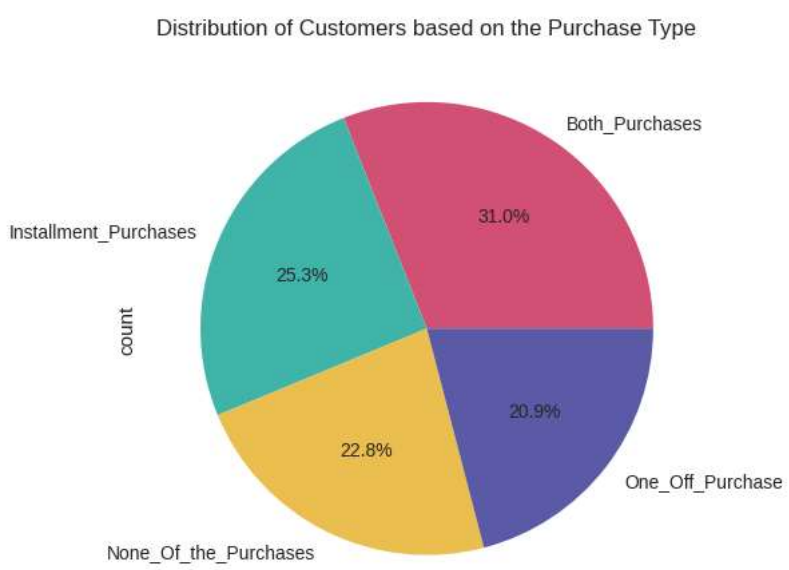
```
   Monthly_Avg_Purchase  Credit_Utilization_Ratio  Limit_Usage  \
0              7.950000                  0.095400     0.040901
1              0.000000                  0.920421     0.457495
2             64.430833                  0.103089     0.332687
3            124.916667                  0.227305     0.222223
4              1.333333                  0.013333     0.681429

   Pay_to_MinimumPay          Purchase_Type
0           1.446508  Installment_Purchases
1           3.826241  None_Of_the_Purchases
2           0.991682       One_Off_Purchase
```

```
3               NaN       One_Off_Purchase
4          2.771075       One_Off_Purchase
```

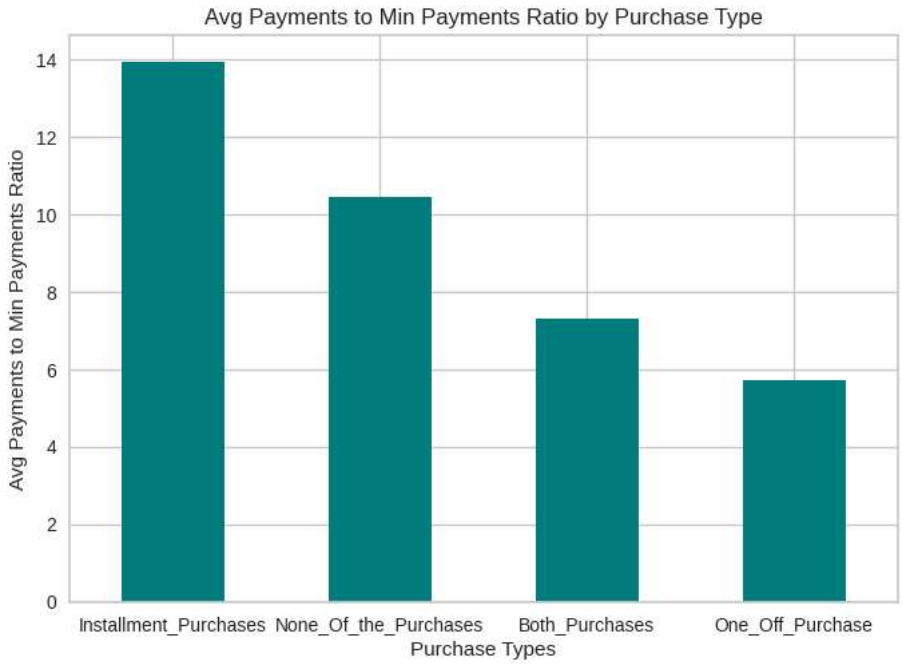
```
# Purchase Type Categories are as follows:
dataset['Purchase_Type'] .value_counts()
```

```
Purchase_Type
Both_Purchases           2774
Installment_Purchases    2260
None_Of_the_Purchases    2041
One_Off_Purchase         1874
Name: count, dtype: int64
```

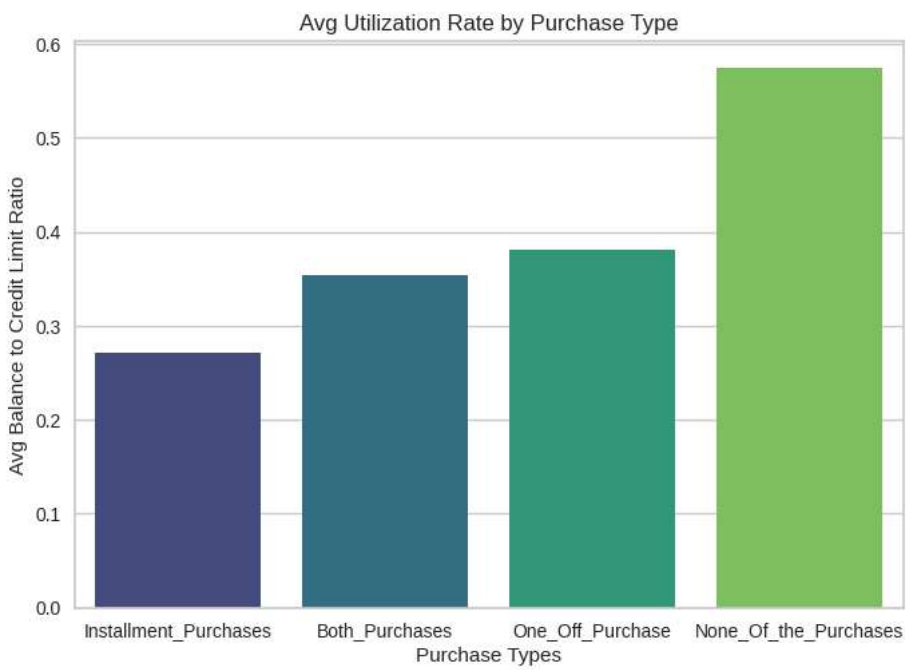
```
# Plotting the distribution of customer on basis of Purhcase Type
dataset['Purchase_Type'].value_counts().sort_index().plot(kind='pie',autopct='%1.01f%%',
                     colors =['#D65076', '#45B8AC', '#EFC050', '#5B5EA6'],fontsize=10,textprops = {'fontsize': 18})
plt.title('Distribution of Customers based on the Purchase Type')
plt.show()
```



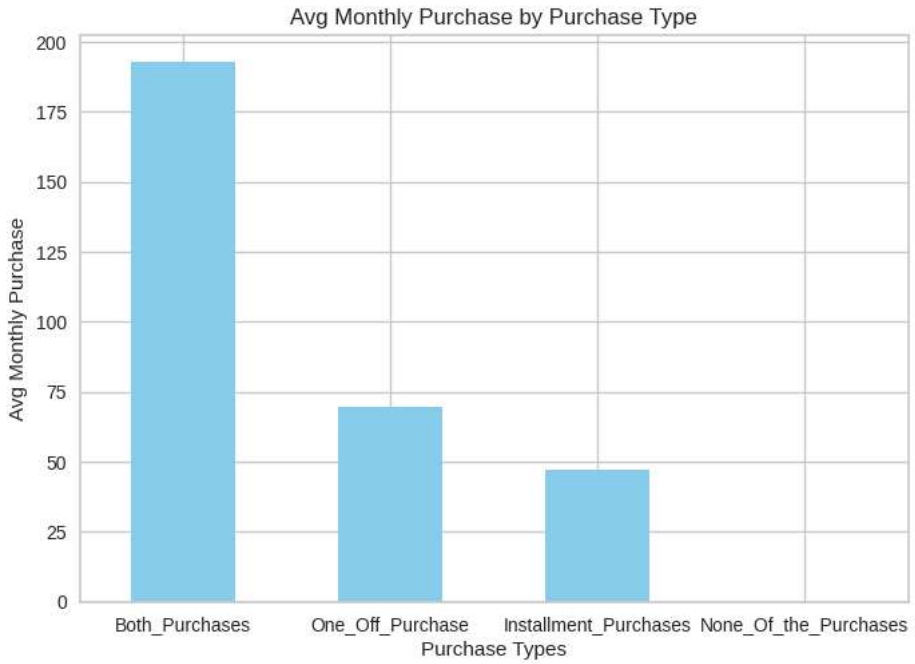

```
# Step 1: Calculate the average Pay_to_MinimumPay for each Purchase Type
pay_min_pay_avg = dataset.groupby('Purchase_Type')['Pay_to_MinimumPay'].mean().sort_values(ascending=False)

# Step 2: Plot the results
pay_min_pay_avg.plot(kind='bar', color='teal')
plt.title('Avg Payments to Min Payments Ratio by Purchase Type')
plt.xlabel('Purchase Types')
plt.ylabel('Avg Payments to Min Payments Ratio')
plt.xticks(rotation=0)
plt.show()

# Insight:
# Customers making installment purchases have the highest average payments to minimum payments ratio.
```



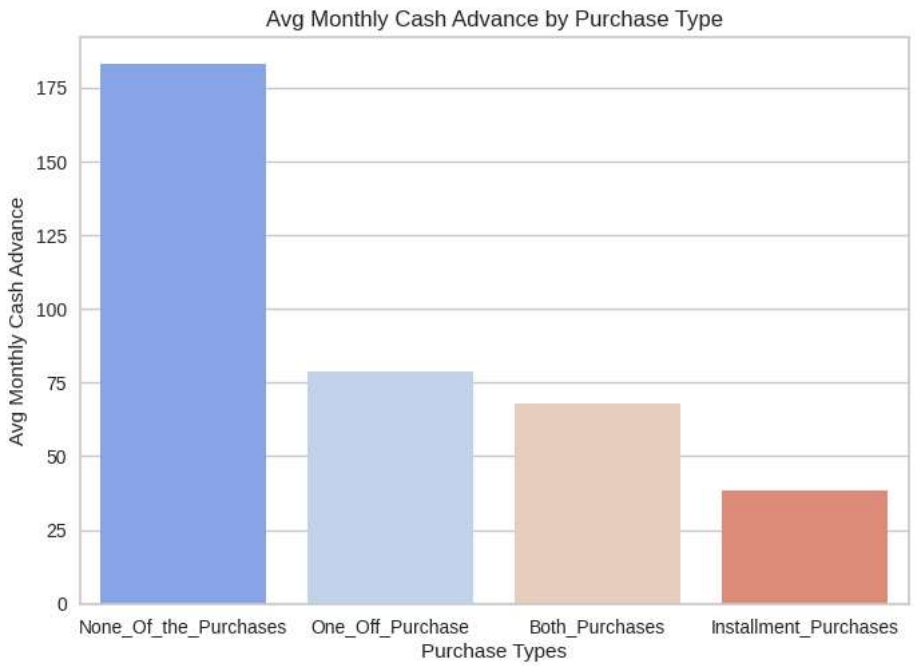

```
# Step 1: Calculate the average Limit Usage for each Purchase Type
limit_usage_avg = dataset.groupby('Purchase_Type')['Limit_Usage'].mean().sort_values(ascending=True).reset_index()

# Step 2: Plot the results
sns.barplot(data=limit_usage_avg, x='Purchase_Type', y='Limit_Usage', palette='viridis')
plt.title('Avg Utilization Rate by Purchase Type')
plt.xlabel('Purchase Types')
plt.ylabel('Avg Balance to Credit Limit Ratio')
plt.show()

# Insight:
# Customers who favor installment purchases show the lowest credit utilization rate, indicating better credit management.
```



```
# Step 1: Calculate the average Monthly_Avg_Purchase for each Purchase Type
monthly_avg_purchase_avg = dataset.groupby('Purchase_Type')['Monthly_Avg_Purchase'].mean().sort_values(ascending=False)

# Step 2: Plot the results
monthly_avg_purchase_avg.plot(kind='bar', color='skyblue')
plt.title('Avg Monthly Purchase by Purchase Type')
plt.xlabel('Purchase Types')
plt.ylabel('Avg Monthly Purchase')
plt.xticks(rotation=0)
plt.show()

# Insight:
# Customers who engage in both one-off and installment purchases have the highest average monthly purchase amounts.
```




```
# Step 1: Calculate the average Monthly_Avg_Cash for each Purchase Type

dataset['Monthly_Avg_Cash'] = dataset['cash_advance'] / dataset['tenure']

monthly_avg_cash_avg = dataset.groupby('Purchase_Type')['Monthly_Avg_Cash'].mean().sort_values(ascending=False).reset_index()

# Step 2: Plot the results
sns.barplot(data=monthly_avg_cash_avg, x='Purchase_Type', y='Monthly_Avg_Cash', palette='coolwarm')
plt.title('Avg Monthly Cash Advance by Purchase Type')
plt.xlabel('Purchase Types')
plt.ylabel('Avg Monthly Cash Advance')
plt.show()

# Insight:
# Customers who neither make one-off purchases nor installment purchases tend to have the highest monthly cash advance amounts.
```



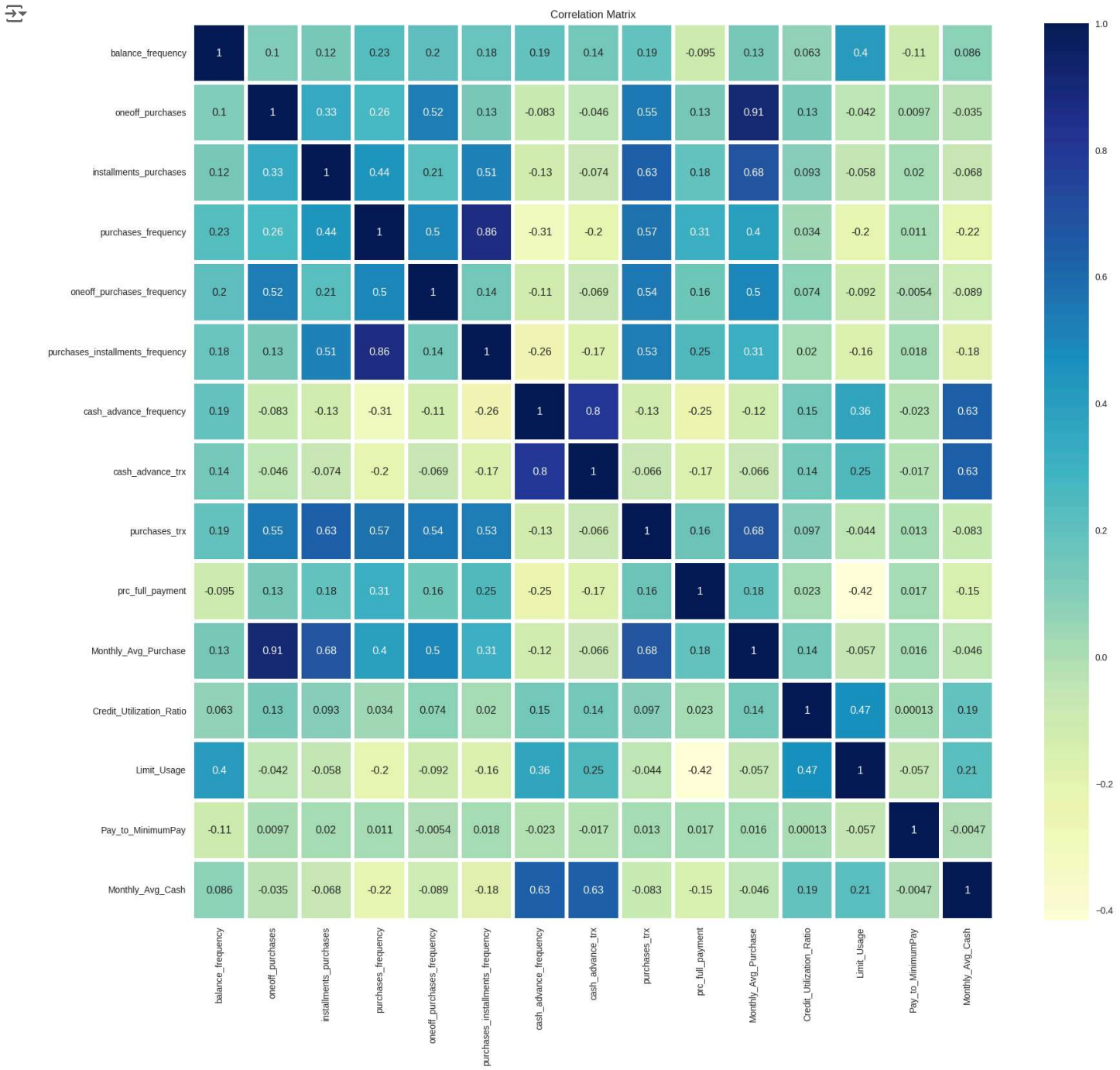

```
# Correlation Matrix

# Display column names to verify
print(dataset.columns)
```

```
Index(['balance', 'balance_frequency', 'purchases', 'oneoff_purchases',
       'installments_purchases', 'cash_advance', 'purchases_frequency',
       'oneoff_purchases_frequency', 'purchases_installments_frequency',
       'cash_advance_frequency', 'cash_advance_trx', 'purchases_trx',
       'credit_limit', 'payments', 'minimum_payments', 'prc_full_payment',
       'tenure', 'Monthly_Avg_Purchase', 'Credit_Utilization_Ratio',
       'Limit_Usage', 'Pay_to_MinimumPay', 'Purchase_Type',
       'Monthly_Avg_Cash'],
      dtype='object')
```

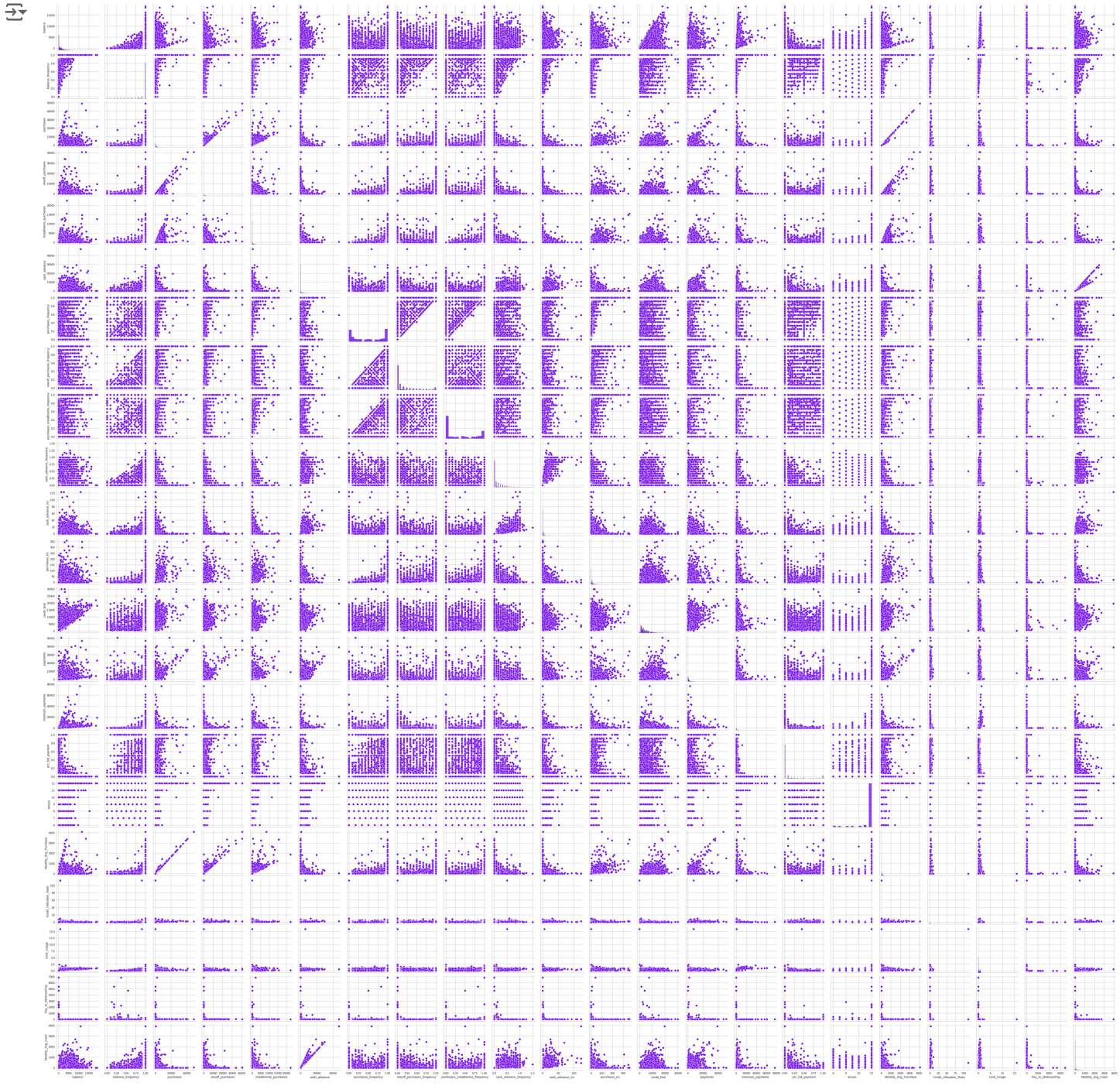
```
# Dropping the original variables and the Purchase_Type column
columns_to_drop = ['balance', 'credit_limit', 'purchases', 'payments', 'minimum_payments', 'tenure', 'cash_advance', 'Purchase_Type']
columns_to_drop = [col for col in columns_to_drop if col in dataset.columns]  # Ensure only existing columns are dropped

dataset_reduced = dataset.drop(columns=columns_to_drop, axis=1)

# Calculate the correlation matrix
corr_matrix = dataset_reduced.corr()

# Display column names to verify
print(dataset.columns)
```

```
Index(['balance', 'balance_frequency', 'purchases', 'oneoff_purchases',
       'installments_purchases', 'cash_advance', 'purchases_frequency',
       'oneoff_purchases_frequency', 'purchases_installments_frequency',
       'cash_advance_frequency', 'cash_advance_trx', 'purchases_trx',
       'credit_limit', 'payments', 'minimum_payments', 'prc_full_payment',
       'tenure', 'Monthly_Avg_Purchase', 'Credit_Utilization_Ratio',
       'Limit_Usage', 'Pay_to_MinimumPay', 'Purchase_Type',
       'Monthly_Avg_Cash'],
      dtype='object')
```

```
# Plotting the correlation matrix
plt.figure(figsize=(20,18))
sns.heatmap(corr_matrix, annot=True, cmap='YlGnBu', linewidths=3, fmt='.2g')
plt.title('Correlation Matrix')
plt.show()
```

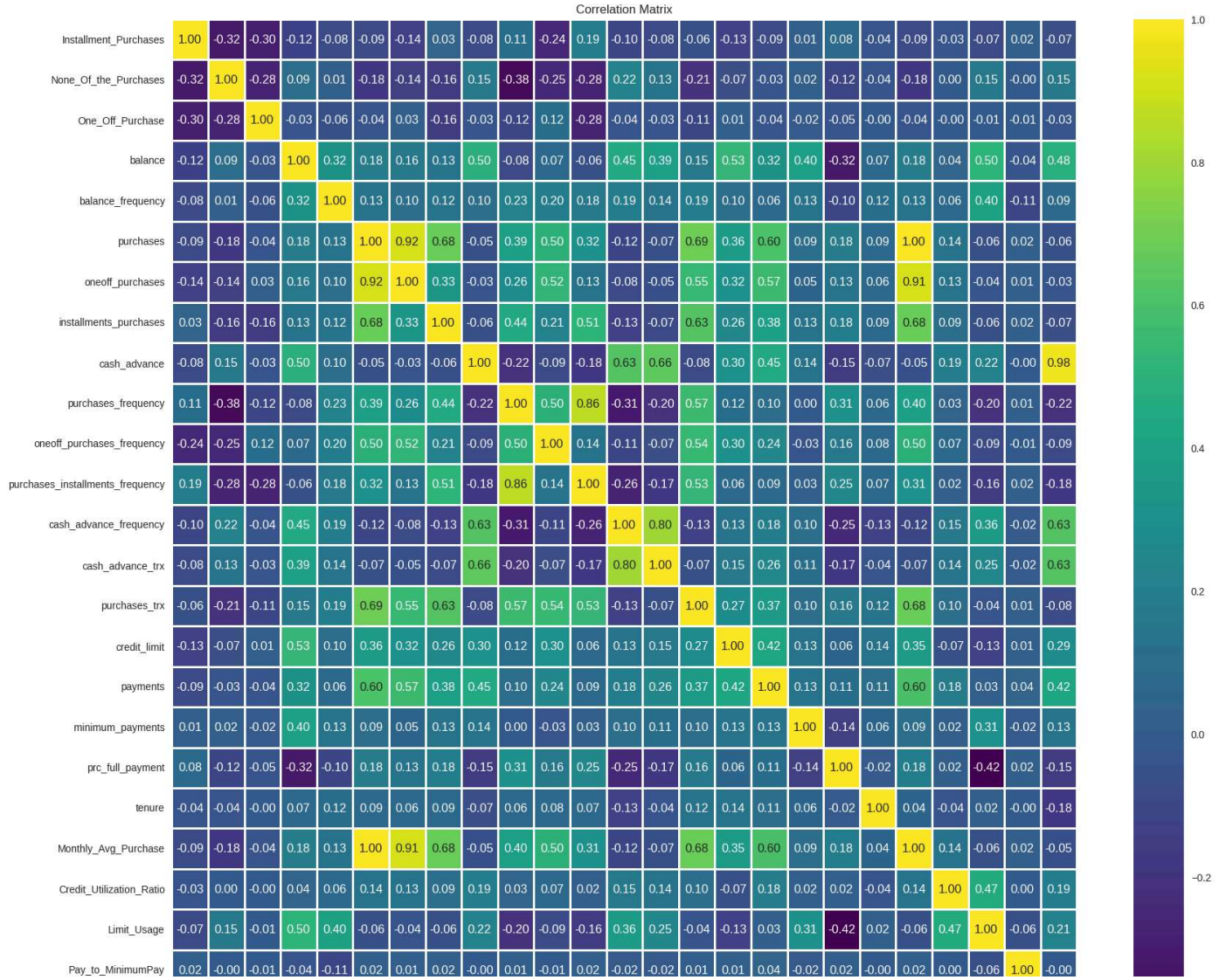


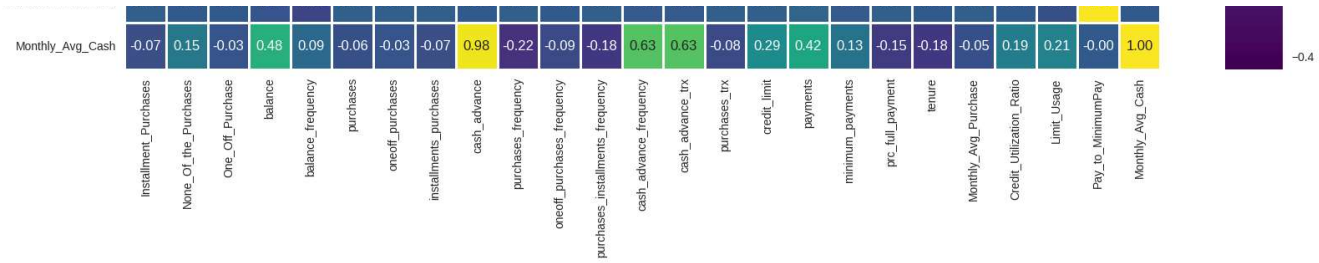

```
# Plotting Pair Plot
sns.pairplot(dataset)
plt.show()
```



```
# Data Modeling
# Create dummy variables for Purchase_Type
categorical_features = pd.get_dummies(dataset['Purchase_Type'], drop_first=True)

# Select numerical variables
numerical_features = dataset.select_dtypes(include=[np.number]).columns.tolist()

# Scale the numerical variables
scaler = StandardScaler()
scaled_numerical_features = pd.DataFrame(scaler.fit_transform(dataset[numerical_features]), columns=numerical_features)

# Combine the categorical and scaled numerical datasets
prepared_dataset = pd.concat([categorical_features, scaled_numerical_features], axis=1)

# Display the head of the combined dataset
print(prepared_dataset.head())

# Plotting the correlation matrix
plt.figure(figsize=(20,18))
sns.heatmap(prepared_dataset.corr(), annot=True, cmap='viridis', linewidths=2, fmt='.2f')
plt.title('Correlation Matrix')
plt.show()
```

```
  Installment_Purchases None_Of_the_Purchases One_Off_Purchase   balance  \
0                  True                 False            False -0.732054
1                 False                  True            False  0.786858
2                 False                 False             True  0.447041
3                 False                 False             True  0.049015
4                 False                 False             True -0.358849

   balance_frequency  purchases  oneoff_purchases  installments_purchases  \
0          -0.249881  -0.424934         -0.356957               -0.349114
1           0.134049  -0.469584         -0.356957               -0.454607
2           0.517980  -0.107716          0.108843               -0.454607
3          -1.017743   0.231995          0.546123               -0.454607
4           0.517980  -0.462095         -0.347317               -0.454607

   cash_advance  purchases_frequency  ...  credit_limit  payments  \
0     -0.466805            -0.806649  ...     -0.960380 -0.529026
1      2.605438            -1.221928  ...      0.688601  0.818546
2     -0.466805             1.269742  ...      0.826016 -0.383857
3     -0.368678            -1.014290  ...      0.826016 -0.598733
4     -0.466805            -1.014290  ...     -0.905414 -0.364421

   minimum_payments  prc_full_payment    tenure  Monthly_Avg_Purchase  \
0         -0.305508         -0.525588  0.360541             -0.433418
1          0.087689          0.234159  0.360541             -0.477461
2         -0.099906         -0.525588  0.360541             -0.120516
3               NaN         -0.525588  0.360541              0.214573
4         -0.261131         -0.525588  0.360541             -0.470074

   Credit_Utilization_Ratio  Limit_Usage  Pay_to_MinimumPay  Monthly_Avg_Cash
0                 -0.316498    -0.893059          -0.065714         -0.460737
1                  0.309706     0.175953          -0.045931          2.319245
2                 -0.310661    -0.144316          -0.069496         -0.460737
3                 -0.216380    -0.427774                NaN         -0.371944
4                 -0.378787     0.750582          -0.054703         -0.460737

[5 rows x 25 columns]
```






```
# Fill missing values with the mean of each column
prepared_dataset = prepared_dataset.fillna(prepared_dataset.mean())

# Verify that there are no more missing values
print(prepared_dataset.isna().sum())

# Assuming `prepared_dataset` is the dataset we have after combining dummy variables and scaling numerical features
scaler = StandardScaler()
scaled_data = scaler.fit_transform(prepared_dataset)

# Display the first few rows of the scaled data to verify
print(pd.DataFrame(scaled_data, columns=prepared_dataset.columns).head())
```

```
balance_frequency                   0
purchases                           0
oneoff_purchases                    0
installments_purchases              0
cash_advance                        0
purchases_frequency                 0
oneoff_purchases_frequency          0
purchases_installments_frequency    0
cash_advance_frequency              0
cash_advance_trx                    0
purchases_trx                       0
credit_limit                        0
payments                            0
minimum_payments                    0
prc_full_payment                    0
tenure                              0
Monthly_Avg_Purchase                0
Credit_Utilization_Ratio            0
Limit_Usage                         0
Pay_to_MinimumPay                   0
Monthly_Avg_Cash                    0
dtype: int64
   Installment_Purchases  None_Of_the_Purchases  One_Off_Purchase   balance  \
0               1.720484              -0.543588         -0.514690 -0.732095
1              -0.581297               1.839835         -0.514690  0.786902
2              -0.581297              -0.543588          1.943134  0.447066
3              -0.581297              -0.543588          1.943134  0.049018
4              -0.581297              -0.543588          1.943134 -0.358869

   balance_frequency  purchases  oneoff_purchases  installments_purchases  \
0          -0.249895  -0.424957         -0.356976               -0.349134
1           0.134057  -0.469610         -0.356976               -0.454632
2           0.518009  -0.107722          0.108849               -0.454632
3          -1.017800   0.232008          0.546153               -0.454632
4           0.518009  -0.462121         -0.347337               -0.454632

   cash_advance  purchases_frequency  ...  credit_limit  payments  \
0     -0.466831            -0.806694  ...     -0.960433 -0.529056
1      2.605583            -1.221997  ...      0.688639  0.818592
2     -0.466831             1.269813  ...      0.826062 -0.383879
3     -0.368698            -1.014347  ...      0.826062 -0.598767
4     -0.466831            -1.014347  ...     -0.905464 -0.364442

   minimum_payments  prc_full_payment    tenure  Monthly_Avg_Purchase  \
0     -3.110121e-01         -0.525618  0.360561             -0.433442
1      8.926865e-02          0.234172  0.360561             -0.477488
2     -1.017062e-01         -0.525618  0.360561             -0.120523
3     -1.469296e-18         -0.525618  0.360561              0.214585
```

```
1                 0.309723     0.175963      -4.675807e-02          2.319374
2                -0.310679    -0.144324      -7.074764e-02         -0.460762
3                -0.216392    -0.427798       2.203944e-19         -0.371965
4                -0.378809     0.750624      -5.568820e-02         -0.460762

[5 rows x 25 columns]
```

```
# Calculating the covariance matrix
cov_matrix = np.cov(scaled_data.T)
print('Shape of Covariance Matrix:', cov_matrix.shape)
print('Covariance Matrix:\n', cov_matrix)
```

```
  5.96549138e-01  1.79266083e-01  2.87505550e-02  3.55008830e-02
  4.15735020e-01]
 [ 5.88379992e-03  1.96623471e-02 -1.98857932e-02  3.94309815e-01
  1.14195087e-01  9.35072549e-02  4.85886477e-02  1.31685588e-01
  1.39224981e-01  2.92584603e-03 -2.99949695e-02  2.95571761e-02
  9.79162109e-02  1.09185209e-01  9.58471323e-02  1.25148198e-01
  1.25038131e-01  1.00011174e+00 -1.39715182e-01  5.71499251e-02
  9.17894948e-02  2.20254870e-02  3.07798261e-01 -2.20708718e-02
  1.27640150e-01]
 [ 8.30050936e-02 -1.21326123e-01 -4.50176845e-02 -3.19053225e-01
 -9.53186547e-02  1.80376644e-01  1.32759634e-01  1.82567987e-01
 -1.52978189e-01  3.05794990e-01  1.57515021e-01  2.50077148e-01
 -2.49796091e-01 -1.69826390e-01  1.62055265e-01  5.56783676e-02
  1.12119826e-01 -1.39715182e-01  1.00011174e+00 -1.67458809e-02
  1.81775374e-01  2.26445395e-02 -4.15747870e-01  1.71688846e-02
 -1.51402838e-01]
 [-3.83630205e-02 -4.36249616e-02 -1.02531749e-03  7.24291910e-02
  1.18579402e-01  8.61644241e-02  6.40540499e-02  8.60257851e-02
 -6.85597785e-02  6.10126519e-02  8.22430422e-02  7.29338543e-02
 -1.33441631e-01 -4.36190289e-02  1.21732934e-01  1.39182258e-01
  1.05976543e-01  5.71499251e-02 -1.67458809e-02  1.00011174e+00
  4.41154765e-02 -4.09505239e-02  2.41694184e-02 -6.63092550e-04
 -1.77988736e-01]
 [-9.23330313e-02 -1.79151217e-01 -4.07908456e-02  1.79699642e-01
  1.31203106e-01  9.95956432e-01  9.13160942e-01  6.77092126e-01
 -4.75280755e-02  3.95354076e-01  4.99749803e-01  3.14141265e-01
 -1.16115161e-01 -6.57233204e-02  6.82649385e-01  3.53656805e-01
  5.96549138e-01  9.17894948e-02  1.81775374e-01  4.41154765e-02
  1.00011174e+00  1.39692876e-01 -5.71030087e-02  1.54835243e-02
 -4.57749318e-02]
 [-2.72820740e-02  2.94169742e-03 -3.20146418e-03  3.54710787e-02
  6.32377611e-02  1.37356341e-01  1.26306142e-01  9.26711607e-02
  1.88489877e-01  3.43923836e-02  7.35447290e-02  2.02983193e-02
  1.49664140e-01  1.40539279e-01  9.73520912e-02 -6.68867469e-02
  1.79266083e-01  2.20254870e-02  2.26445395e-02 -4.09505239e-02
  1.39692876e-01  1.00011174e+00  4.65310028e-01  1.30729624e-04
  1.86064407e-01]
 [-7.45481321e-02  1.52149822e-01 -7.69247629e-03  5.03574082e-01
  4.04602172e-01 -5.74703761e-02 -4.22589638e-02 -5.83241585e-02
  2.15195212e-01 -2.01966545e-01 -9.20990100e-02 -1.61554506e-01
  3.60208268e-01  2.52623830e-01 -4.37993824e-02 -1.29327893e-01
  2.87505550e-02  3.07798261e-01 -4.15747870e-01  2.41694184e-02
 -5.71030087e-02  4.65310028e-01  1.00011174e+00 -5.60940618e-02
  2.11183455e-01]
 [ 1.59357050e-02 -4.30488903e-03 -7.95783041e-03 -4.05775764e-02
 -9.40544692e-02  1.59243764e-02  9.64775910e-03  1.99145380e-02
 -3.79944148e-03  1.05375989e-02 -5.30275449e-03  1.72452742e-02
 -2.26160853e-02 -1.67082611e-02  1.26323627e-02  1.07673529e-02
  3.55008830e-02 -2.20708718e-02  1.71688846e-02 -6.63092550e-04
  1.54835243e-02  1.30729624e-04 -5.60940618e-02  1.00011174e+00
 -4.71183280e-03]
 [-7.21194732e-02  1.53303175e-01 -2.81738788e-02  4.75476071e-01
  8.59727574e-02 -5.54953756e-02 -3.45611038e-02 -6.78062010e-02
  9.76473121e-01 -2.15870275e-01 -8.90692918e-02 -1.79393447e-01
  6.28391248e-01  6.33361808e-01 -8.34199367e-02  2.85156923e-01
  4.15735020e-01  1.27640150e-01 -1.51402838e-01 -1.77988736e-01
 -4.57749318e-02  1.86064407e-01  2.11183455e-01 -4.71183280e-03
  1.00011174e+00]]
```

```
# Calculating eigenvalues and eigenvectors
eig_val, eig_vec = np.linalg.eig(cov_matrix)
print('Number of Eigenvalues:', len(eig_val))
print('Shape of Eigenvectors:', eig_vec.shape)
print('Eigenvalues:\n', eig_val)
print('Eigenvectors:\n', eig_vec)
```

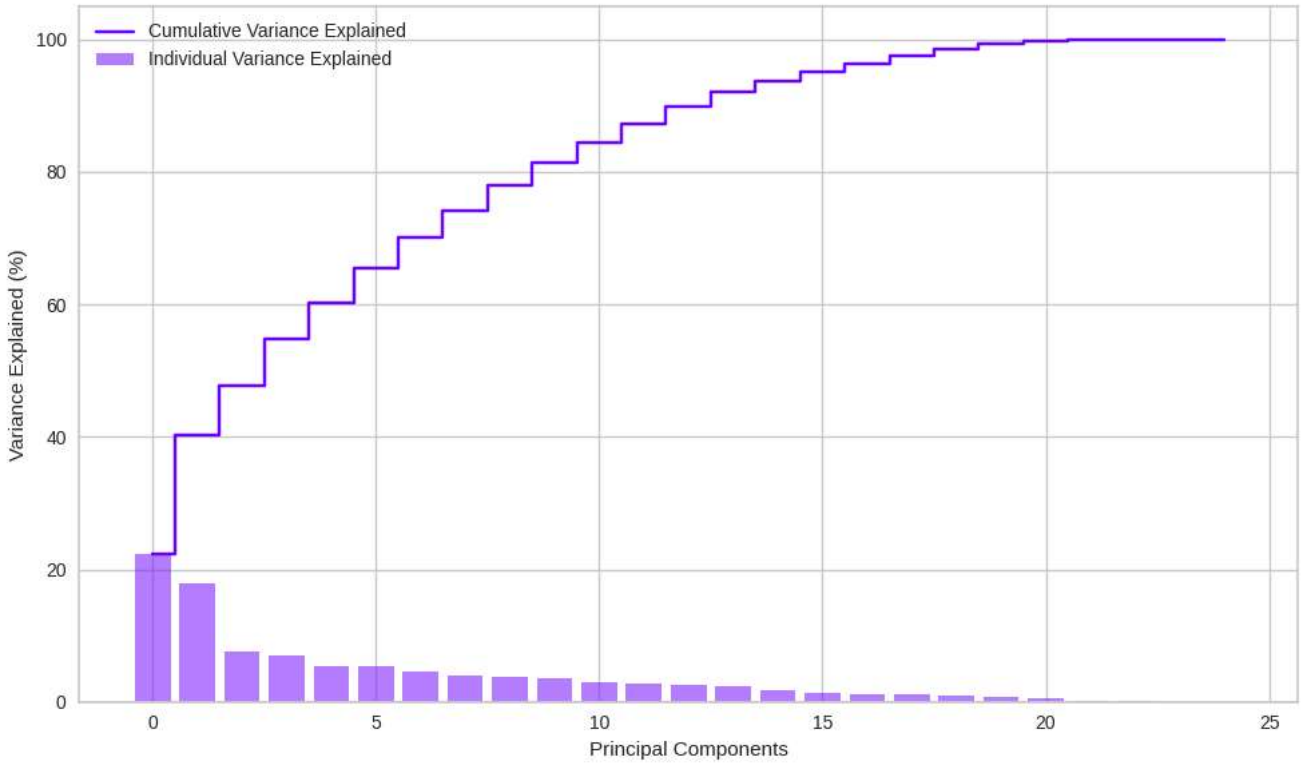
```
 -1.23382037e-01 -2.47911104e-01 -3.49753882e-01  4.49234772e-02
 -1.98941043e-01  4.99860183e-01 -1.33501791e-01  4.27386569e-01
  3.39833589e-01  1.06994659e-01 -3.26936256e-02 -2.47740129e-01
  8.00808892e-02  2.48396893e-02 -6.97218032e-02 -5.62923497e-02
  3.82512982e-05  1.38556950e-02 -7.96112395e-04  1.38171904e-04
 -3.23584880e-05]
[ 1.26870977e-01  1.53574333e-01 -6.18055812e-02 -3.42154431e-01
  6.83738748e-02  1.09310092e-01  1.64094222e-01  3.49169687e-03
  2.22973861e-01  3.56048374e-01 -3.97756961e-01  4.69162241e-01
 -2.26754261e-01 -2.38647692e-01 -4.50944526e-03  3.10942319e-01
 -1.51161122e-01  9.69041533e-02  3.36067144e-02 -5.60602754e-04
  4.87871550e-02  1.50444154e-02  1.48133739e-03 -7.02477208e-04
 -1.63770454e-05]
[ 6.42323550e-02  1.71880564e-02  2.79686599e-02  1.88475175e-01
 -2.81385486e-01 -2.22077440e-01  1.22065555e-02 -3.02780121e-01
  7.82239586e-01 -5.56233871e-02  1.90383709e-01  8.83755166e-02
  1.60577341e-01  4.12769351e-02  1.20183195e-03  4.65042955e-02
  5.37127807e-02  1.42634461e-01  1.08401517e-01 -9.23758696e-02
  8.89117086e-03 -1.64065835e-02 -7.35952321e-02  3.11839901e-02
 -5.28499709e-05]
[ 3.86152089e-01 -6.39958684e-02 -1.45830721e-01  4.14192125e-02
  1.79407133e-01 -7.31723481e-02 -8.08756544e-02  8.53185900e-02
 -5.69354712e-02 -1.02137195e-01  6.17703670e-02  7.03430190e-02
 -6.78623447e-02  3.06666171e-03  7.83919998e-02 -4.53226859e-02
  4.04428640e-02  1.58388388e-01  1.38899078e-01  5.69745025e-03
  3.98085030e-02 -7.14630112e-03 -2.22023066e-02  8.30070801e-01
 -1.40867894e-03]
[ 4.66985583e-02 -1.27038773e-01  1.07252770e-01  1.24503368e-01
  5.03598706e-01  3.76610436e-01 -8.89045036e-02 -1.57334149e-01
  2.92116023e-01  2.63679655e-01 -1.69660274e-01 -2.84196107e-01
  7.30220877e-02 -2.07016303e-01  7.06460672e-02 -2.67682603e-01
  1.73730996e-01  4.47116385e-02 -9.89213307e-02 -1.04317017e-02
 -3.07934440e-01  1.24463968e-02 -1.64396228e-02  1.67605496e-03
 -1.35452374e-05]
[-5.96055603e-02 -2.39384528e-01  2.54069980e-01  4.66373412e-01
  2.21243268e-01  1.08122670e-01 -1.08646458e-01 -5.49177736e-02
  3.40179259e-02  4.48743973e-03 -9.23923159e-02 -5.28707129e-02
 -8.45368620e-02 -5.01658689e-02  2.28415725e-02  3.21528490e-01
 -2.35662962e-01 -1.01799436e-01  1.01821222e-01  7.63864001e-02
  6.09865757e-01 -2.68517943e-02  2.77024092e-02 -5.07009588e-03
  1.34961389e-05]
[ 8.75634411e-03  1.59520589e-02 -4.90808943e-02 -1.07678629e-01
  6.73769779e-02 -6.66348603e-02 -1.52801171e-01 -9.00468762e-01
 -2.97360165e-01 -1.55453000e-01 -1.05025166e-01  1.03830872e-01
 -8.51070236e-02  5.77197909e-03 -1.08504751e-02 -5.69436368e-03
 -2.26331621e-03  2.41631441e-02  2.36656671e-04  6.56497487e-03
 -5.76183021e-03  1.11700987e-03 -2.89989666e-04  5.68315804e-05
 -8.03640082e-06]
[-5.32989955e-02 -3.92073011e-01  1.16217404e-02 -2.94958171e-01
 -2.26218329e-02  1.07424200e-01  1.14838315e-03  5.55763418e-03
 -1.75800995e-02  1.04202725e-01 -2.60453322e-02 -1.31637782e-01
 -2.07204484e-02  3.05040531e-01 -1.73786610e-01  4.49369828e-02
  6.32491495e-02  2.94087496e-01  9.31169415e-02 -4.11824395e-02
  1.35916778e-01  1.31937567e-02 -6.84644332e-01 -3.32322454e-02
  2.26646829e-05]]
```

```
# Pairing eigenvalues with eigenvectors
eigen_pairs = [(eig_val[i], eig_vec[:,i]) for i in range(len(eig_val))]
eigen_pairs_sorted = sorted(eigen_pairs, key=lambda x: x[0], reverse=True)

# Extracting sorted eigenvalues and eigenvectors
eig_val_sorted = [eigen_pairs_sorted[i][0] for i in range(len(eig_val))]
eig_vec_sorted = [eigen_pairs_sorted[i][1] for i in range(len(eig_val))]

# Displaying sorted eigenvalues
print('Sorted Eigenvalues:', eig_val_sorted)
```

```
Sorted Eigenvalues: [5.581919169593167, 4.477602813706266, 1.8835434257128085, 1.7544348237976444, 1.366237076215946, 1.3243086956726584
```

```
# Calculating explained variance
total_variance = np.sum(eig_val)
explained_variance = [(i/total_variance) * 100 for i in eig_val_sorted]
cumulative_variance = np.cumsum(explained_variance)

print('Cumulative Variance Explained:', cumulative_variance)

# Plotting the explained variance
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
plt.bar(range(len(explained_variance)), explained_variance, alpha=0.5, align='center', label='Individual Variance Explained')
plt.step(range(len(cumulative_variance)), cumulative_variance, where='mid', label='Cumulative Variance Explained')
plt.xlabel('Principal Components')
plt.ylabel('Variance Explained (%)')
plt.legend(loc='best')
plt.tight_layout()
plt.show()
```

```
Cumulative Variance Explained: [ 22.32518197  40.23359206  47.76692395  54.78387914  60.24821684
  65.54485975  70.20787825  74.19578624  78.00163967  81.49372586
  84.46564349  87.25204596  89.83276734  92.11835544  93.7907198
  95.17417713  96.42473273  97.55073461  98.55968074  99.25589337
  99.75043418  99.91801667  99.98234533  99.99995347 100.        ]
```




```
from sklearn.decomposition import PCA

# Applying PCA
pca_model = PCA(n_components=17)  # Start with all components
pca_data = pca_model.fit_transform(scaled_data)

# Checking the shape of the PCA transformed data
print('Shape of PCA Data:', pca_data.shape)

# Cumulative variance explained by the PCA components
cumulative_variance_sklearn = np.cumsum(pca_model.explained_variance_ratio_ * 100)
print('Cumulative Variance Explained (sklearn):', cumulative_variance_sklearn)
```

```
Shape of PCA Data: (8950, 17)
Cumulative Variance Explained (sklearn): [22.32518197 40.23359206 47.76692395 54.78387914 60.24821684 65.54485975
 70.20787825 74.19578624 78.00163967 81.49372586 84.46564349 87.25204596
 89.83276734 92.11835544 93.7907198  95.17417713 96.42473273]
```

```
# Choosing 7 components to explain around 85% of the variance
pca_model_7 = PCA(n_components=7)
pca_data_7 = pca_model_7.fit_transform(scaled_data)

# Creating a DataFrame for the 7 principal components
principal_components_df = pd.DataFrame(data=pca_data_7, columns=['PC1', 'PC2', 'PC3', 'PC4', 'PC5', 'PC6', 'PC7'])
print(principal_components_df.head())
```

```
        PC1       PC2       PC3       PC4       PC5       PC6       PC7
0 -1.491047 -1.735931  0.198349 -0.417853  0.281674 -0.551636  1.200690
1 -1.681891  3.333998 -0.646026 -1.109394  0.271567 -1.079268 -0.837768
2  0.864777 -0.587862 -1.346529  1.617340 -2.264929  1.624998 -0.316576
3 -0.594057 -0.373111 -2.311467  0.774321 -0.874506  0.090165  1.262935
4 -1.594493 -0.794978 -1.059536  1.796223 -0.467301  0.855967  0.655905
```

```
# List of original features
original_features = prepared_dataset.columns

# Creating the loadings matrix
loadings_matrix = pd.DataFrame(pca_model_7.components_.T, columns=['PC'+str(i+1) for i in range(7)], index=original_features)

print(loadings_matrix)

# Exporting the loadings matrix to CSV
loadings_matrix.to_csv('loadings_matrix.csv')
```

```
                                     PC1       PC2       PC3       PC4  \
Installment_Purchases            -0.018854 -0.101792  0.352960 -0.265752
None_Of_the_Purchases            -0.135025  0.115968 -0.080866  0.077529
One_Off_Purchase                 -0.032152 -0.007618 -0.359430  0.195992
balance                           0.062844  0.350748  0.108951  0.172693
balance_frequency                 0.086618  0.132744  0.294998  0.306084
purchases                         0.387953  0.063088 -0.145484  0.048657
oneoff_purchases                  0.332047  0.075462 -0.272032  0.091802
installments_purchases            0.307192  0.010471  0.155709 -0.053634
cash_advance                     -0.048420  0.399297  0.014749 -0.282256
purchases_frequency               0.284789 -0.137187  0.316217 -0.091337
oneoff_purchases_frequency        0.265658  0.003015 -0.157235  0.123110
purchases_installments_frequency  0.240285 -0.126920  0.443874 -0.157499
cash_advance_frequency           -0.102085  0.367812  0.059077 -0.094897
cash_advance_trx                 -0.064959  0.353314  0.063571 -0.180761
purchases_trx                     0.348849  0.014112  0.096967  0.035525
credit_limit                      0.176158  0.190301 -0.137306 -0.120633
payments                          0.234690  0.242193 -0.157340 -0.169582
minimum_payments                  0.040038  0.144028  0.185523  0.218242
prc_full_payment                  0.126871 -0.153574 -0.061630 -0.342165
tenure                            0.064232 -0.017188  0.028178  0.188468
Monthly_Avg_Purchase              0.386152  0.063996 -0.145734  0.041421
Credit_Utilization_Ratio          0.046699  0.127039  0.107152  0.124510
Limit_Usage                      -0.059606  0.239384  0.254174  0.466370
Pay_to_MinimumPay                 0.008756 -0.015952 -0.049065 -0.107679
Monthly_Avg_Cash                 -0.053299  0.392073  0.011823 -0.294965

                                     PC5       PC6       PC7
Installment_Purchases             0.069176 -0.049338  0.484905
None_Of_the_Purchases             0.297807 -0.437369 -0.501847
One_Off_Purchase                 -0.276477  0.444537  0.232223
balance                          -0.254158 -0.175231  0.099621
balance_frequency                -0.150934  0.132067 -0.311841
purchases                         0.170364 -0.085895  0.080473
oneoff_purchases                  0.165365 -0.022639  0.073962
installments_purchases            0.099066 -0.161499  0.054428
cash_advance                     -0.053327  0.083225 -0.000416
purchases_frequency              -0.165751  0.227163 -0.169817
oneoff_purchases_frequency       -0.175340  0.293251 -0.220814
purchases_installments_frequency -0.071135  0.043099 -0.127183
cash_advance_frequency            0.024839  0.123236 -0.115318
cash_advance_trx                 -0.021210  0.149328 -0.096514
purchases_trx                    -0.002764  0.015882 -0.096340
credit_limit                     -0.399579 -0.194595  0.006109
payments                          0.102489 -0.084685  0.112339
minimum_payments                 -0.123322 -0.248043  0.349461
prc_full_payment                  0.068217  0.109537 -0.163719
tenure                           -0.281783 -0.221643 -0.011513
Monthly_Avg_Purchase              0.179259 -0.073107  0.080988
Credit_Utilization_Ratio          0.503789  0.376350  0.088722
Limit_Usage                       0.221140  0.108199  0.108866
Pay_to_MinimumPay                 0.067357 -0.066651  0.152794
Monthly_Avg_Cash                 -0.022839  0.107693 -0.000770
```

```
from scipy.cluster.hierarchy import linkage, cophenet
from scipy.spatial.distance import pdist

# Finding the Cophenetic Distance Correlation Coefficient for different Linkages
linkage_methods = ['single', 'complete', 'average', 'ward']
distance_metrics = ['euclidean', 'cityblock', 'cosine']

for method in linkage_methods:
    print(f'Linkage: {method}')
    for metric in distance_metrics:
        if method == 'ward' and metric != 'euclidean':
            continue  # Ward linkage only works with Euclidean distance
        Z = linkage(pca_data, method=method, metric=metric)
        c, coph_dist = cophenet(Z, pdist(pca_data))
        print(f'Cophenetic Distance Correlation Coefficient for {metric} distance: {c:.4f}')
    print()
```

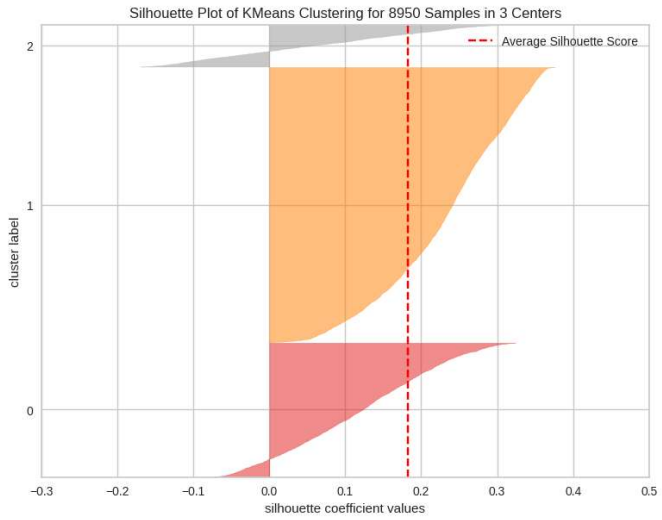
```
Linkage: single
Cophenetic Distance Correlation Coefficient for euclidean distance: 0.7661
Cophenetic Distance Correlation Coefficient for cityblock distance: 0.7297
Cophenetic Distance Correlation Coefficient for cosine distance: 0.1087

Linkage: complete
Cophenetic Distance Correlation Coefficient for euclidean distance: 0.6741
Cophenetic Distance Correlation Coefficient for cityblock distance: 0.7133
Cophenetic Distance Correlation Coefficient for cosine distance: 0.1683

Linkage: average
Cophenetic Distance Correlation Coefficient for euclidean distance: 0.8971
Cophenetic Distance Correlation Coefficient for cityblock distance: 0.8839
Cophenetic Distance Correlation Coefficient for cosine distance: 0.2027

Linkage: ward
Cophenetic Distance Correlation Coefficient for euclidean distance: 0.3434
```

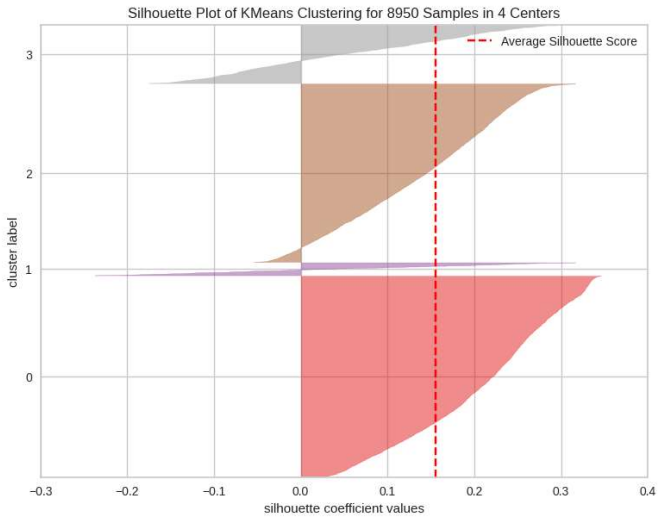
```
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt

# Initialize lists to store the results
wcss = []
sil_kmeans = []
sil_agc = []

# Apply clustering for 3 to 6 clusters
for n_clusters in range(3, 7):
    # KMeans Clustering
    kmeans = KMeans(n_clusters=n_clusters, n_init=100, init='k-means++', random_state=42)
    kmeans_labels = kmeans.fit_predict(pca_data)

    # Inertia and Silhouette Score for KMeans
    wcss.append(kmeans.inertia_)
    sil_kmeans.append(silhouette_score(pca_data, kmeans_labels))

    # Agglomerative Clustering
    agc = AgglomerativeClustering(n_clusters=n_clusters, affinity='cityblock', linkage='average')
    agc_labels = agc.fit_predict(pca_data)
    sil_agc.append(silhouette_score(pca_data, agc_labels))

    # Print results for each cluster number
    print(f'Number of clusters: {n_clusters}')
    print(f'KMeans Inertia: {kmeans.inertia_:.2f}')
    print(f'Silhouette Score for KMeans: {silhouette_score(pca_data, kmeans_labels):.4f}')
    print(f'Silhouette Score for Agglomerative Clustering: {silhouette_score(pca_data, agc_labels):.4f}')
    print()
```

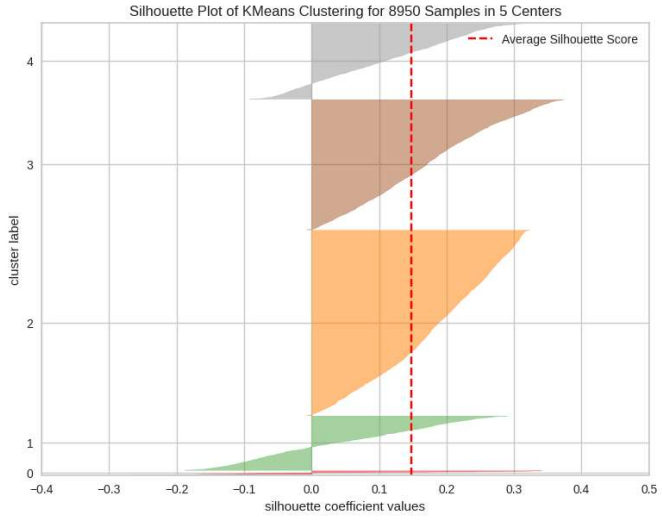
```
Number of clusters: 3
KMeans Inertia: 166202.79
Silhouette Score for KMeans: 0.1826
Silhouette Score for Agglomerative Clustering: 0.8256

Number of clusters: 4
KMeans Inertia: 151694.96
Silhouette Score for KMeans: 0.1553
Silhouette Score for Agglomerative Clustering: 0.8261
```

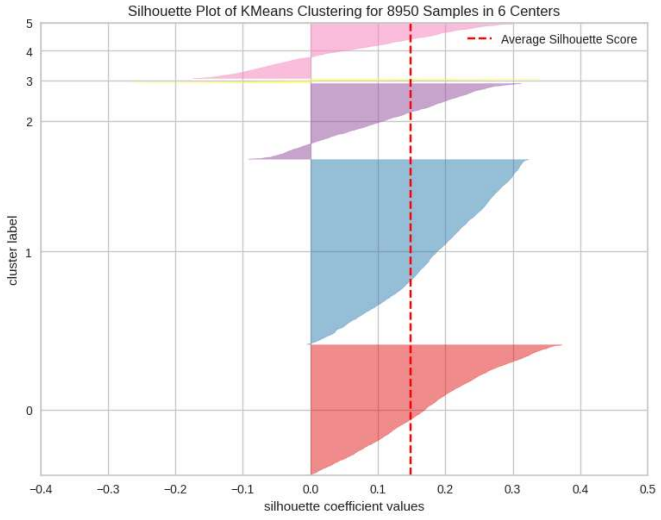
```
Number of clusters: 5
KMeans Inertia: 141824.19
Silhouette Score for KMeans: 0.1478
Silhouette Score for Agglomerative Clustering: 0.7816

Number of clusters: 6
KMeans Inertia: 132987.34
Silhouette Score for KMeans: 0.1489
Silhouette Score for Agglomerative Clustering: 0.7811
```

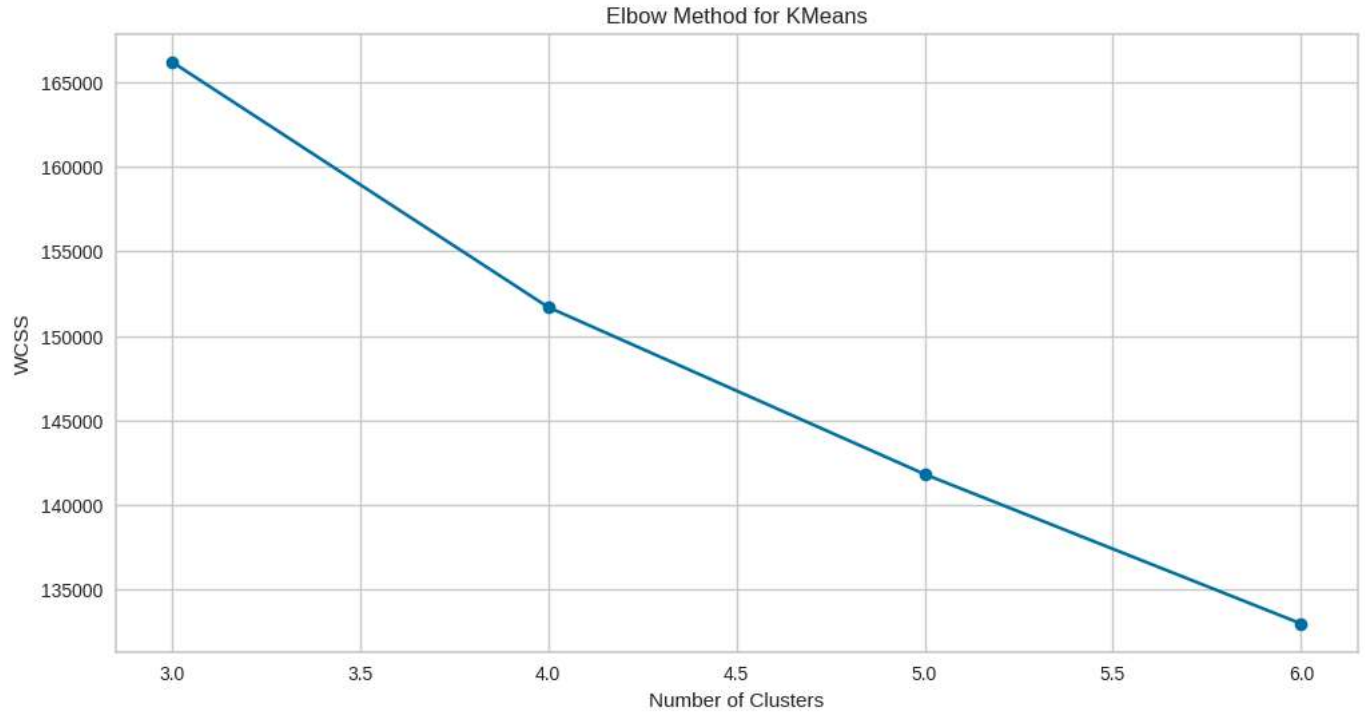
```
from yellowbrick.cluster import SilhouetteVisualizer

# Silhouette Visualizer for KMeans
fig, axs = plt.subplots(2, 2, figsize=(20, 15))
axs = axs.flatten()
for i, n_clusters in enumerate(range(3, 7)):
    ax = axs[i]
    sil_visualizer = SilhouetteVisualizer(KMeans(n_clusters=n_clusters, n_init=100, init='k-means++', random_state=42), ax=ax)
    sil_visualizer.fit(pca_data)
    sil_visualizer.finalize()

# Elbow Method Plot for KMeans
plt.figure(figsize=(12, 6))
plt.plot(range(3, 7), wcss, marker='o', color='b')
plt.title('Elbow Method for KMeans')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.show()

# Silhouette Scores for KMeans and Agglomerative Clustering
plt.figure(figsize=(12, 6))
plt.plot(range(3, 7), sil_kmeans, marker='s', color='purple', label='KMeans')
plt.plot(range(3, 7), sil_agc, marker='o', color='green', label='Agglomerative')
plt.title('Silhouette Scores for KMeans and Agglomerative Clustering')
plt.xlabel('Number of Clusters')
plt.ylabel('Silhouette Score')
plt.legend()
plt.show()
```

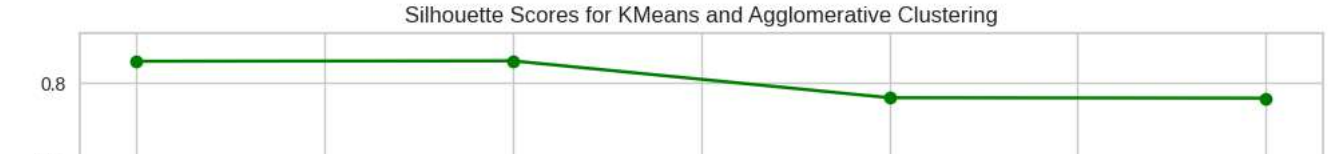

Silhouette Plot of KMeans Clustering for 8950 Samples in 3 Centers
Average Silhouette Score
cluster label
silhouette coefficient values

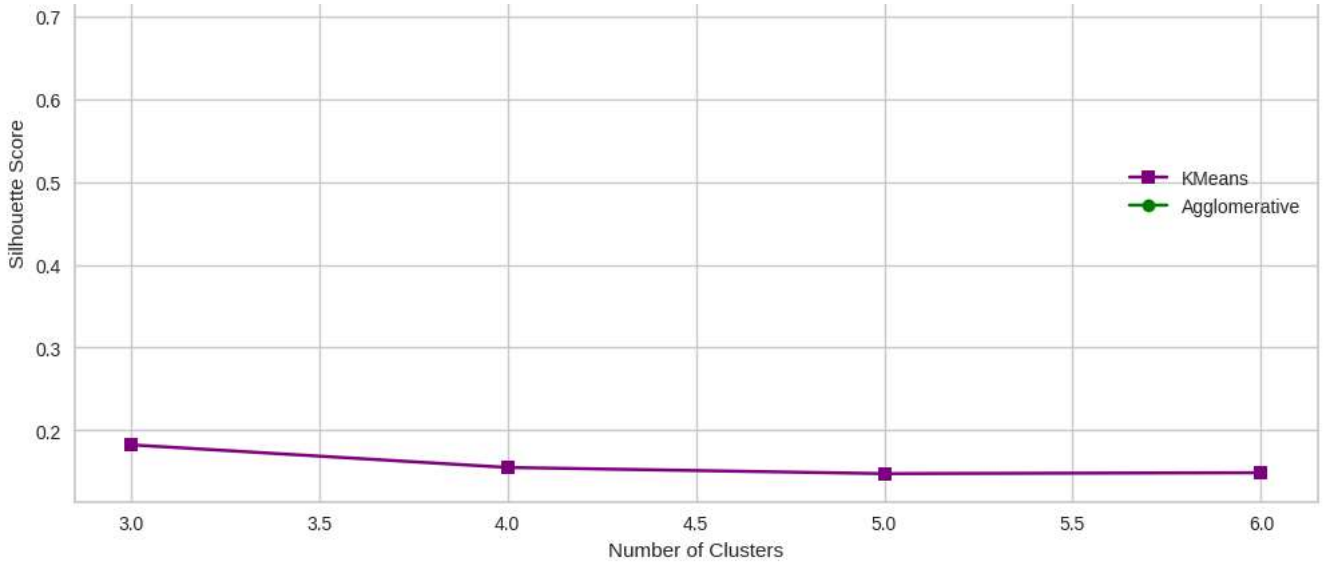

Silhouette Plot of KMeans Clustering for 8950 Samples in 4 Centers
Average Silhouette Score
cluster label
silhouette coefficient values


Silhouette Plot of KMeans Clustering for 8950 Samples in 5 Centers
Average Silhouette Score
cluster label
silhouette coefficient values

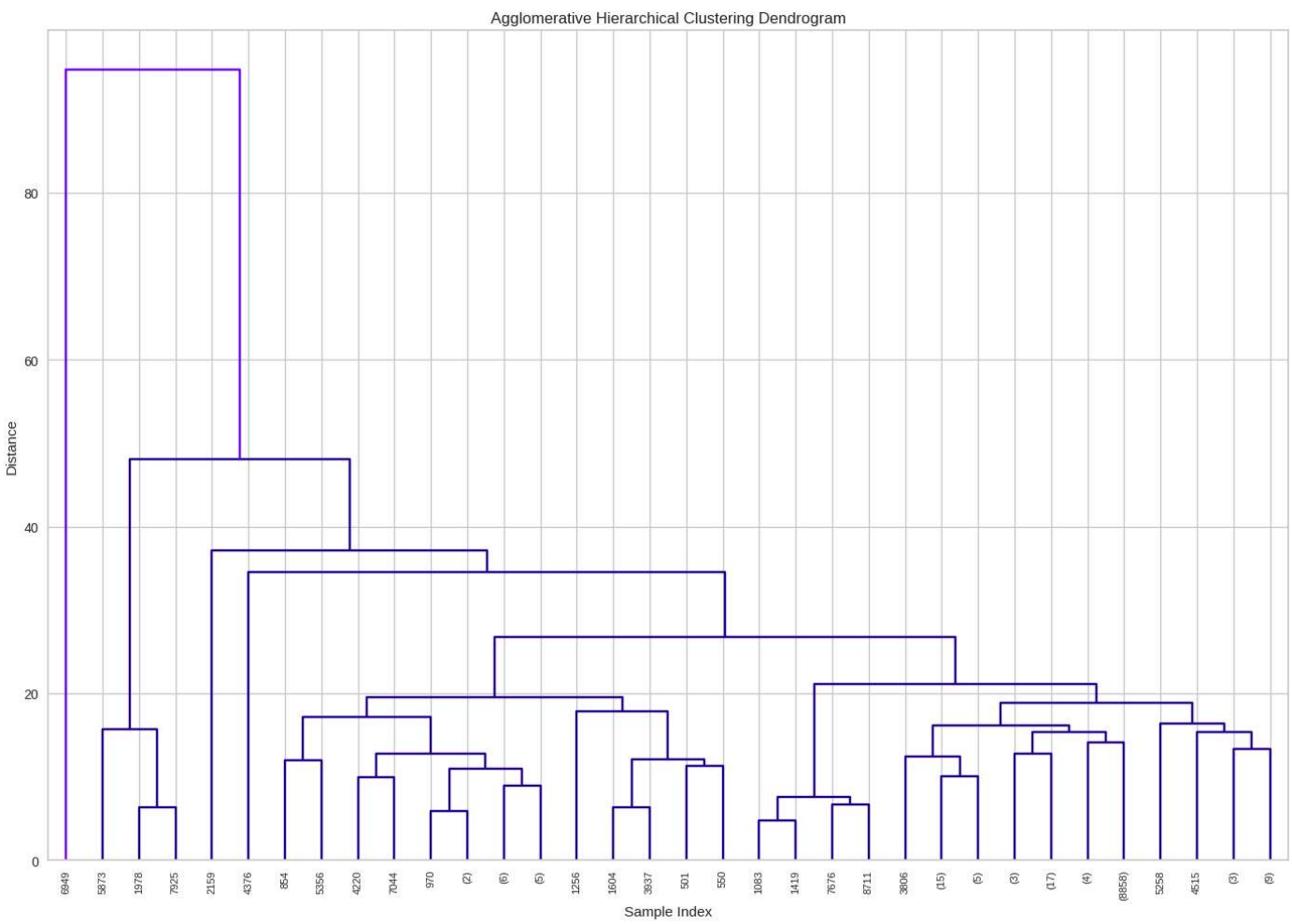

Silhouette Plot of KMeans Clustering for 8950 Samples in 6 Centers
Average Silhouette Score
cluster label
silhouette coefficient values


Elbow Method for KMeans
WCSS
Number of Clusters

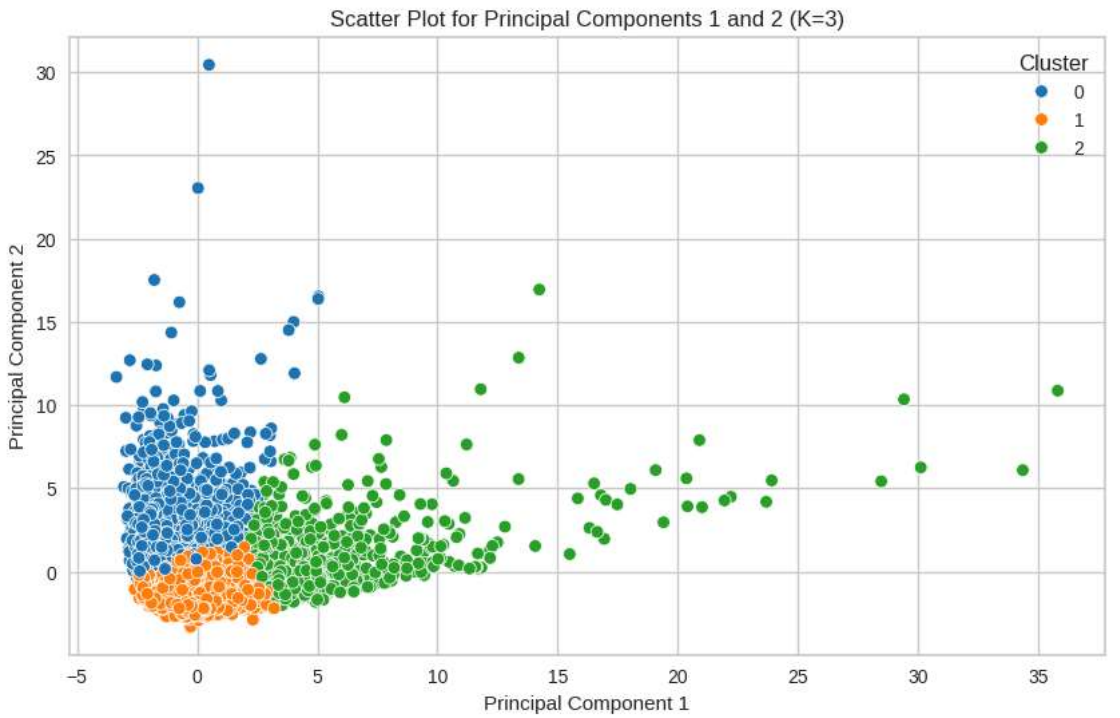

Silhouette Scores for KMeans and Agglomerative Clustering

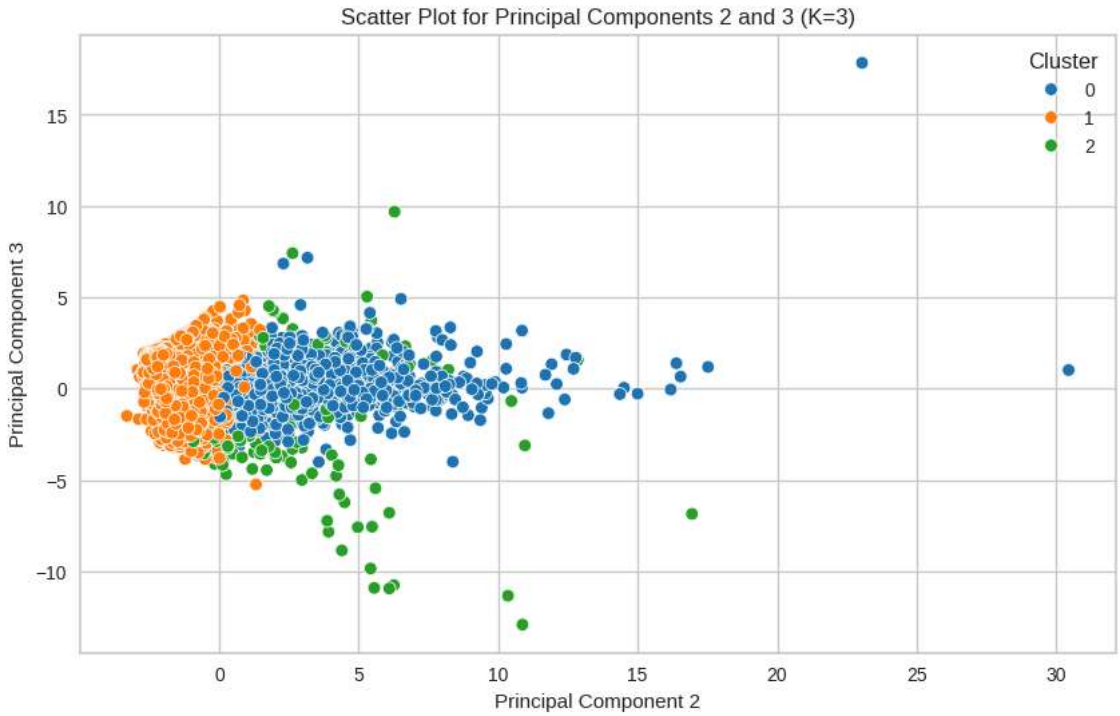


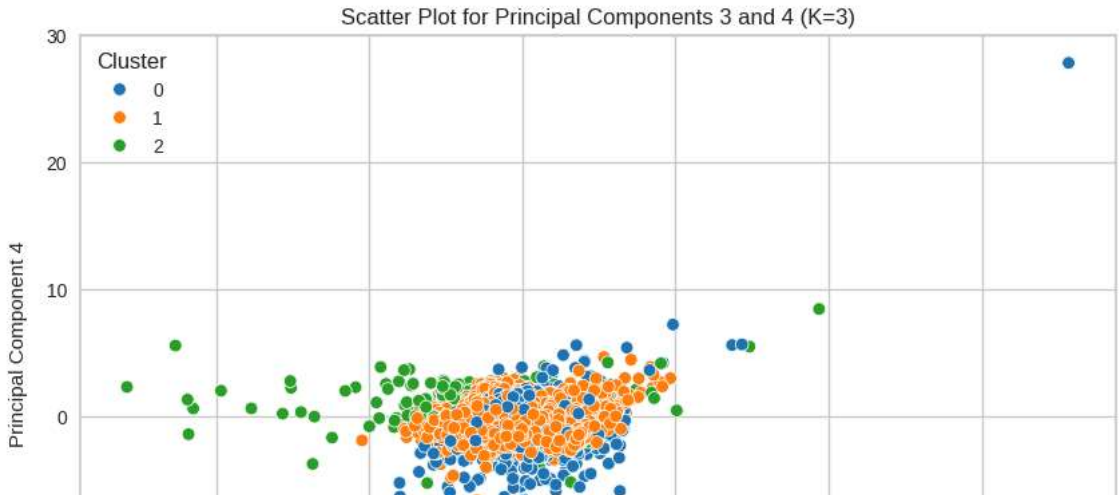

```
# Step 4: Plotting the Dendrogram

import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage

# Plotting Dendrogram for Agglomerative Hierarchical Clustering
Z = linkage(pca_data, method='average', metric='euclidean')
plt.figure(figsize=(14, 10))
plt.title('Agglomerative Hierarchical Clustering Dendrogram')
plt.xlabel('Sample Index')
plt.ylabel('Distance')
dendrogram(Z, leaf_rotation=90, leaf_font_size=8, truncate_mode='level', p=9)
plt.tight_layout()
plt.show()
```

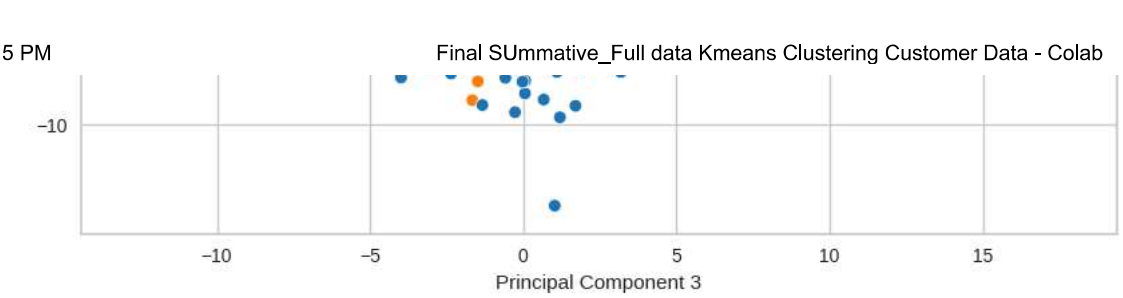


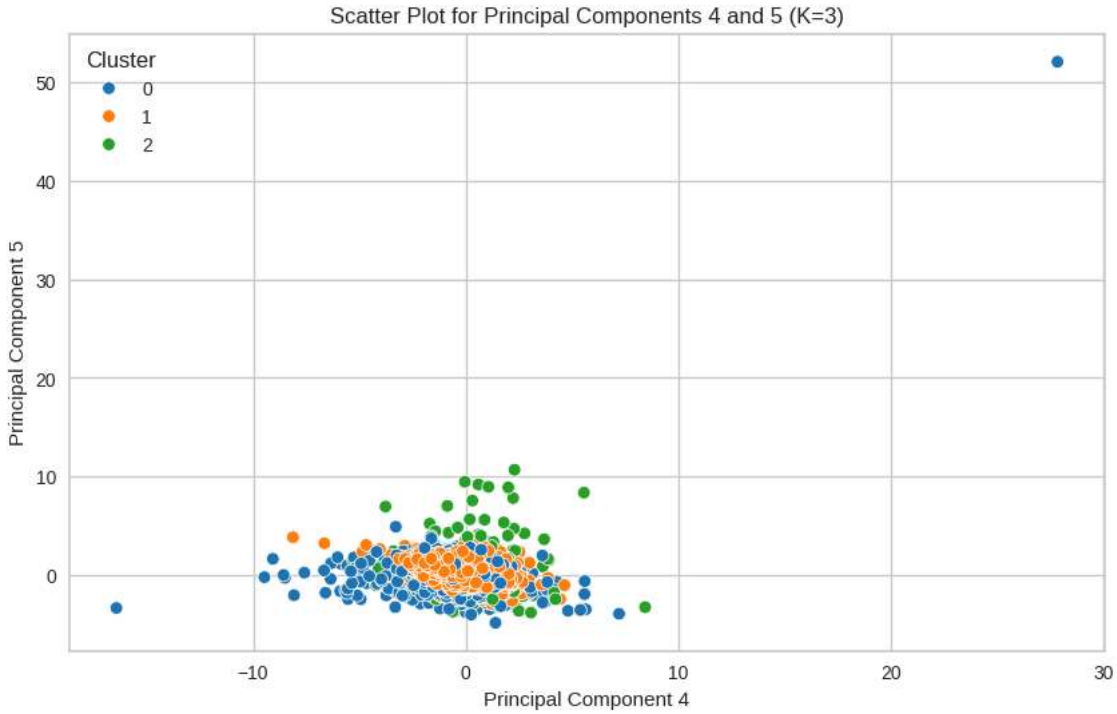

```
# Function to apply KMeans clustering and visualize the results
def apply_kmeans_and_visualize(pca_data, n_clusters, palette):
    # Applying KMeans clustering
    kmeans = KMeans(n_clusters=n_clusters, n_init=100, init='k-means++', random_state=42)
    cluster_labels = kmeans.fit_predict(pca_data)

    # Creating a DataFrame for PCA data and cluster labels
    pca_df = pd.DataFrame(pca_data, columns=[f'PC{i+1}' for i in range(pca_data.shape[1])])
    pca_df['Cluster'] = cluster_labels

    # Visualizing the scatter plots for principal component pairs
    for i in range(min(6, pca_data.shape[1] - 1)):
        plt.figure(figsize=(10, 6))
        sns.scatterplot(x=pca_df[f'PC{i+1}'], y=pca_df[f'PC{i+2}'], hue=pca_df['Cluster'], palette=palette)
        plt.title(f'Scatter Plot for Principal Components {i+1} and {i+2} (K={n_clusters})')
        plt.xlabel(f'Principal Component {i+1}')
        plt.ylabel(f'Principal Component {i+2}')
        plt.legend(title='Cluster')
        plt.show()

# Applying KMeans clustering and visualizing for 3, 4, and 5 clusters
apply_kmeans_and_visualize(pca_data, n_clusters=3, palette=['#1f77b4', '#ff7f0e', '#2ca02c'])  # Blue, Orange, Green
apply_kmeans_and_visualize(pca_data, n_clusters=4, palette=['#d62728', '#9467bd', '#8c564b', '#e377c2'])  # Red, Purple, Brown, Pink
apply_kmeans_and_visualize(pca_data, n_clusters=5, palette=['#7f7f7f', '#bcbd22', '#17becf', '#1f77b4', '#ff7f0e'])  # Gray, Olive, Cyan, Bl
```

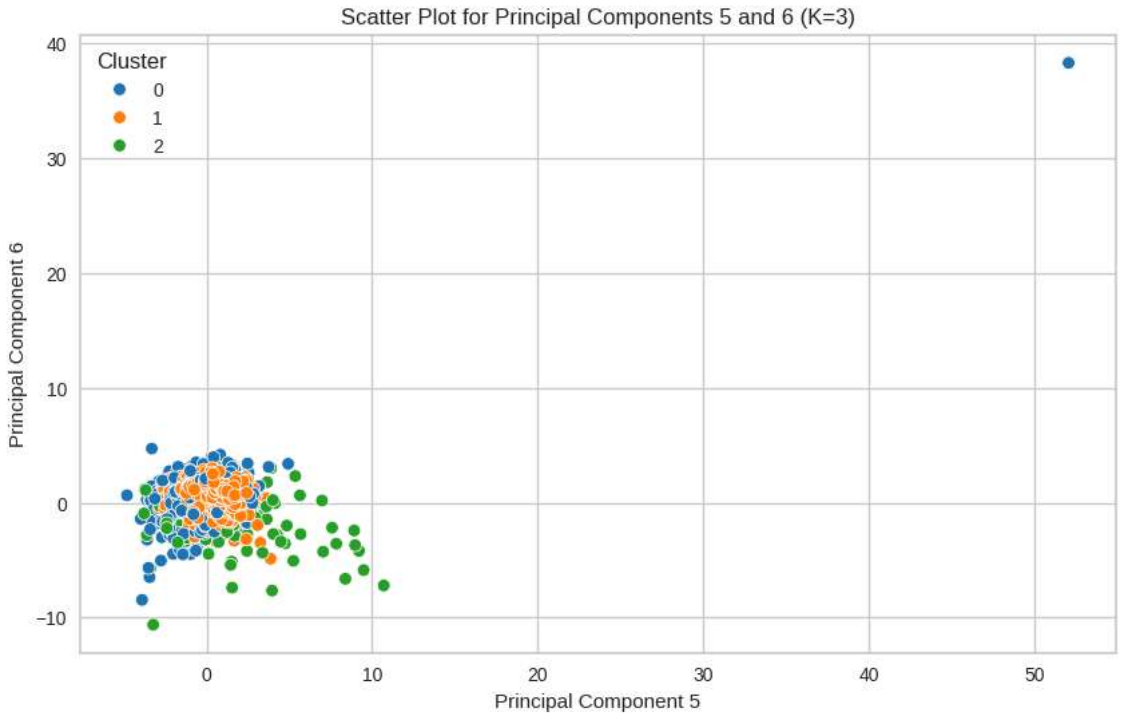

Scatter Plot for Principal Components 1 and 2 (K=3)
Cluster
0
1
2
Principal Component 2
Principal Component 1

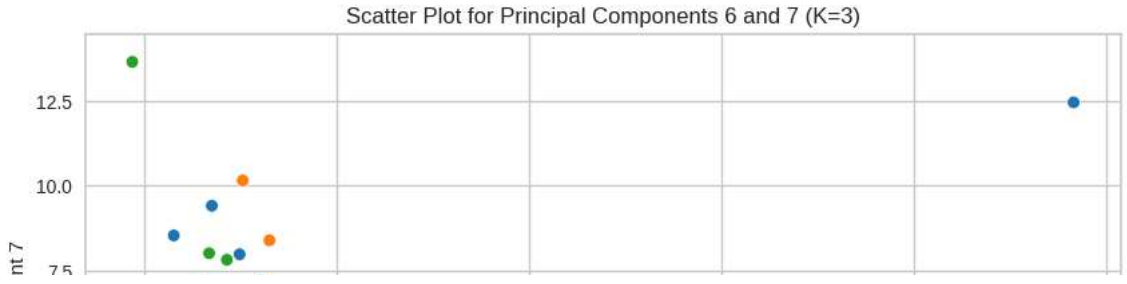

Scatter Plot for Principal Components 2 and 3 (K=3)
Cluster
0
1
2
Principal Component 3
Principal Component 2

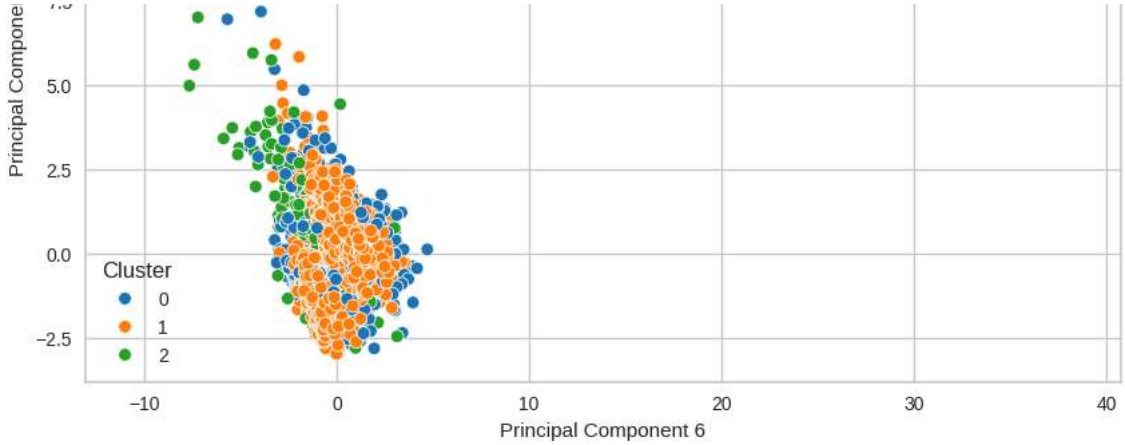

Scatter Plot for Principal Components 3 and 4 (K=3)
Cluster
0
1
2
Principal Component 4

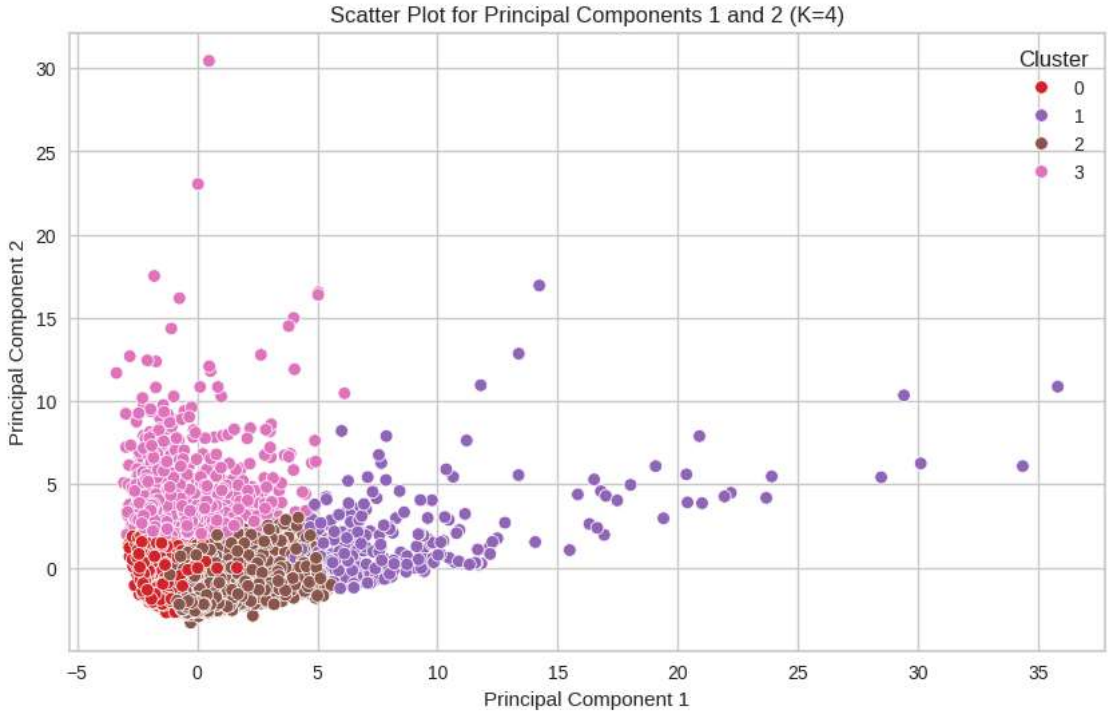

−10
−10
−5
0
5
10
15
Principal Component 3

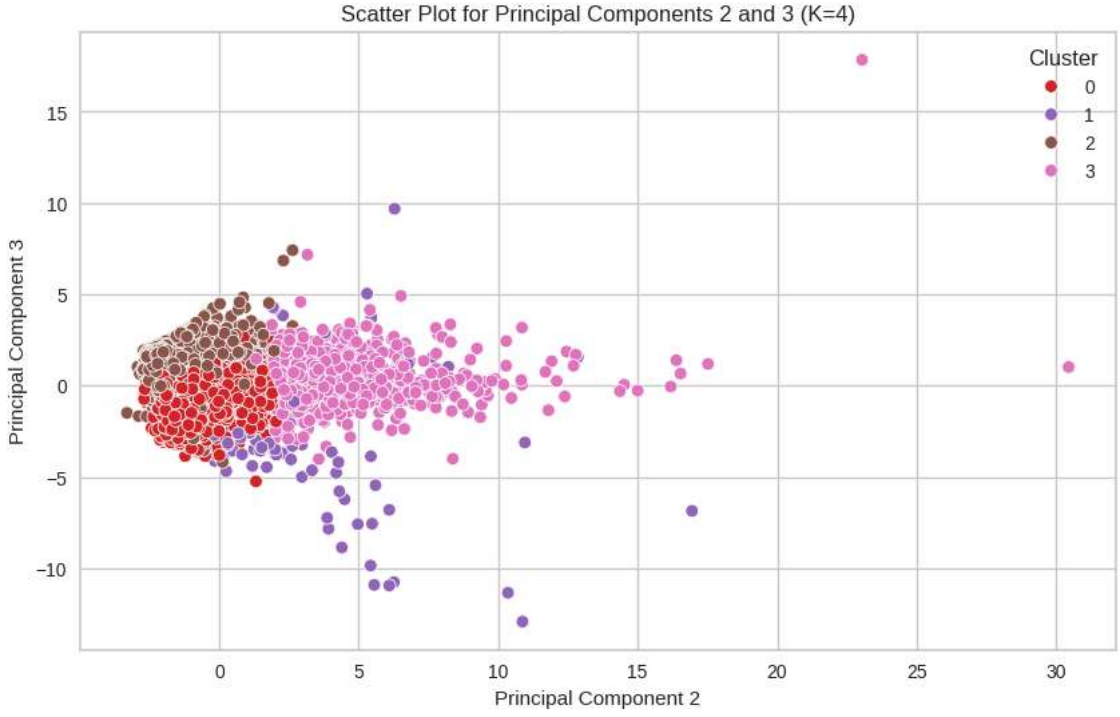

Scatter Plot for Principal Components 4 and 5 (K=3)
Cluster
0
1
2
Principal Component 5
Principal Component 4

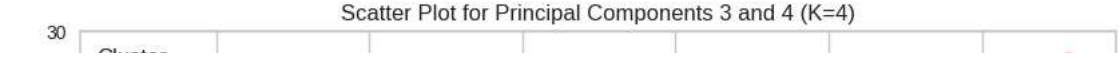

Scatter Plot for Principal Components 5 and 6 (K=3)
Cluster
0
1
2
Principal Component 6
Principal Component 5

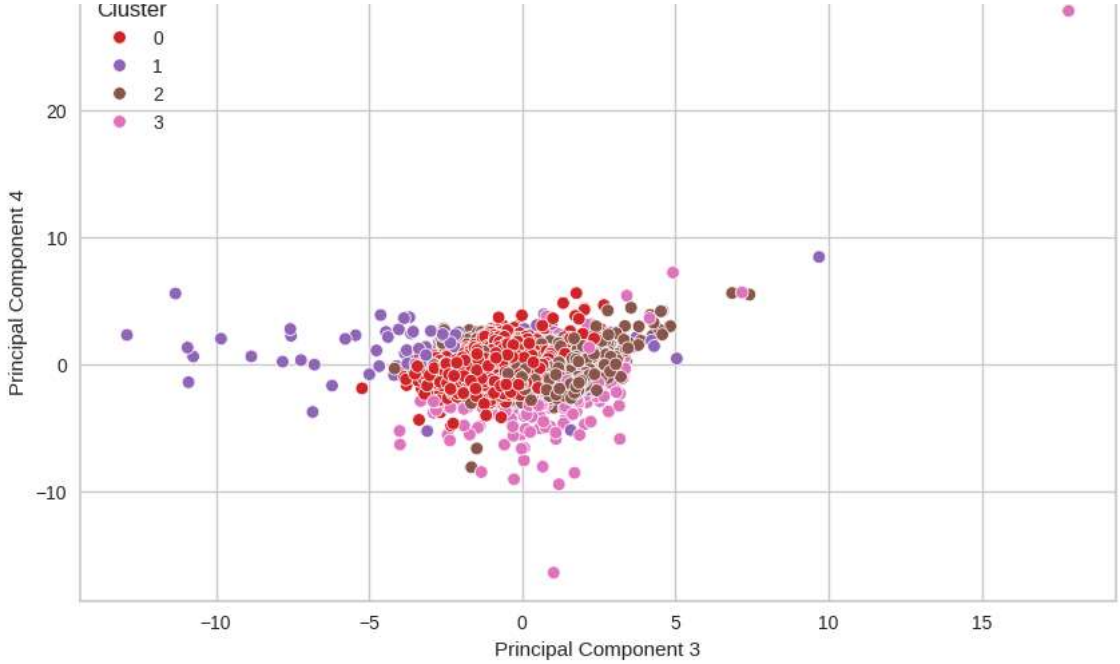

Scatter Plot for Principal Components 6 and 7 (K=3)


Principal Compone
5.0
2.5
0.0
−2.5
Cluster
0
1
2
−10
0
10
20
30
40
Principal Component 6

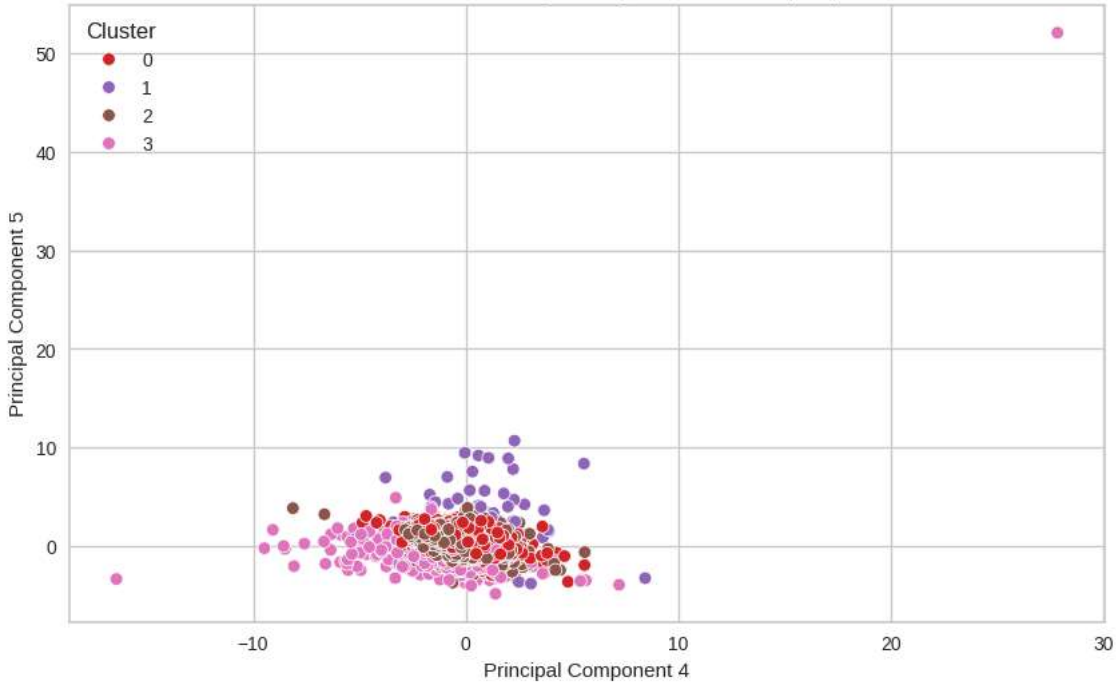

Scatter Plot for Principal Components 1 and 2 (K=4)
Cluster
0
1
2
3
Principal Component 2
30
25
20
15
10
5
0
−5
0
5
10
15
20
25
30
35
Principal Component 1


Scatter Plot for Principal Components 2 and 3 (K=4)
Cluster
0
1
2
3
Principal Component 3
15
10
5
0
−5
−10
0
5
10
15
20
25
30
Principal Component 2

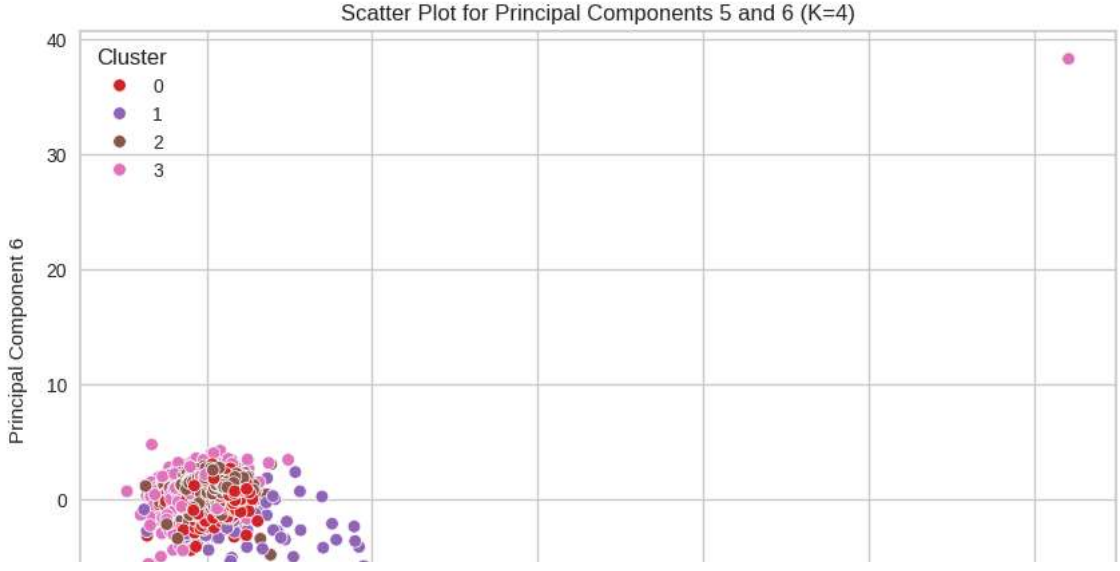

Scatter Plot for Principal Components 3 and 4 (K=4)
30

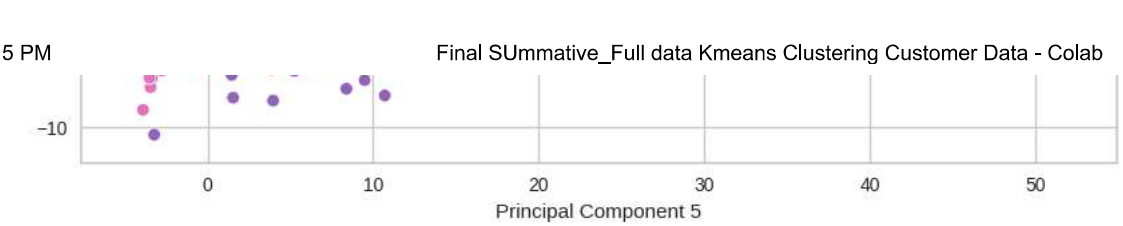


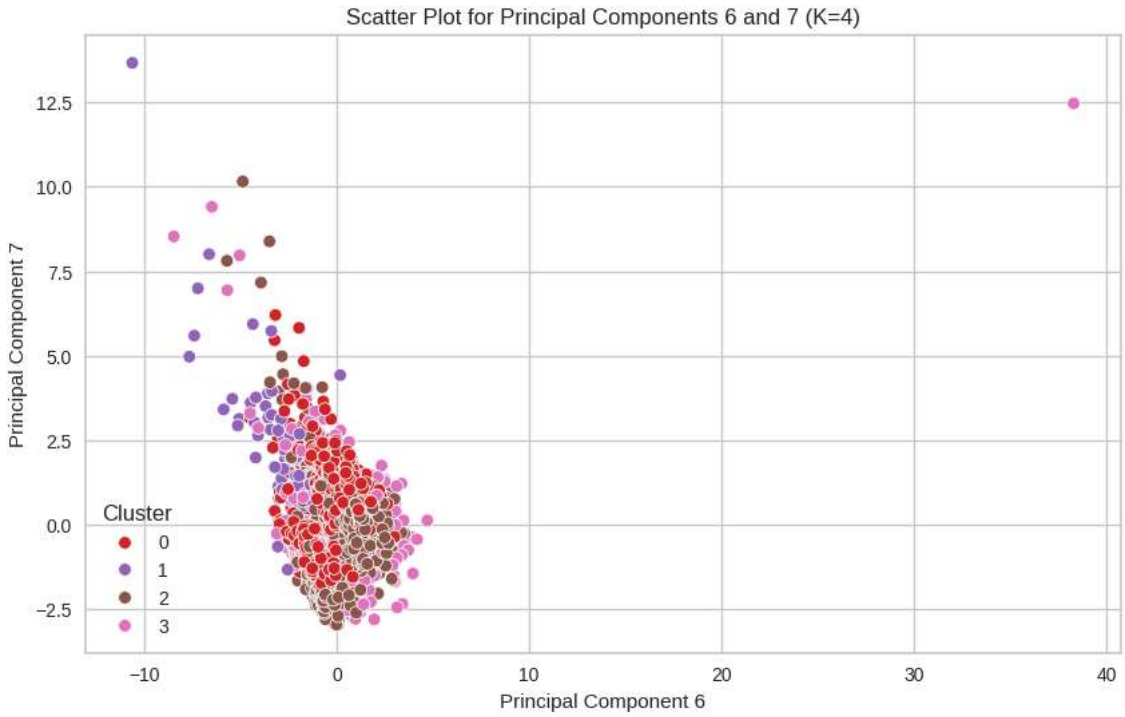
Scatter Plot for Principal Components 4 and 5 (K=4)

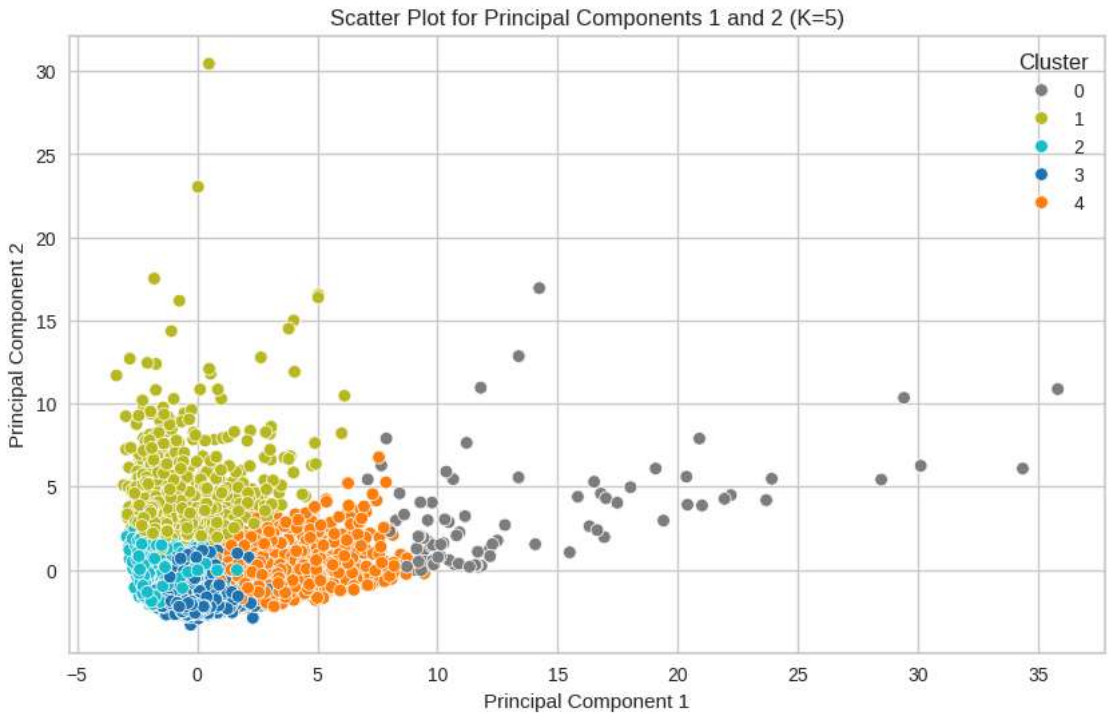


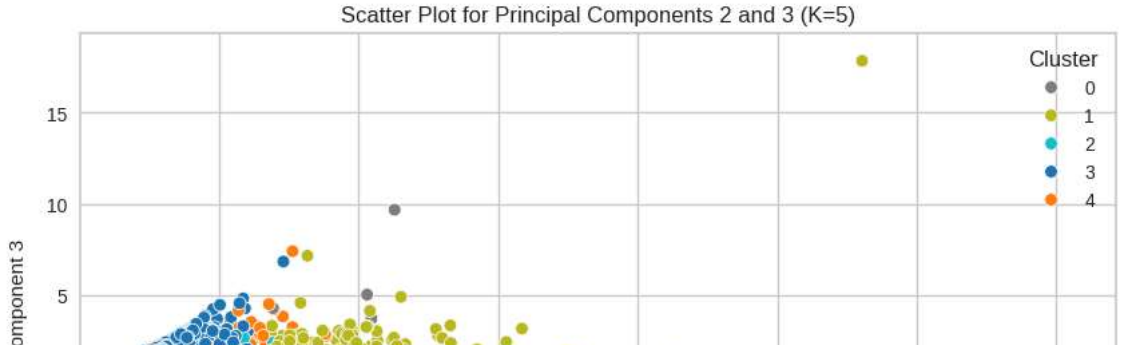
Scatter Plot for Principal Components 5 and 6 (K=4)

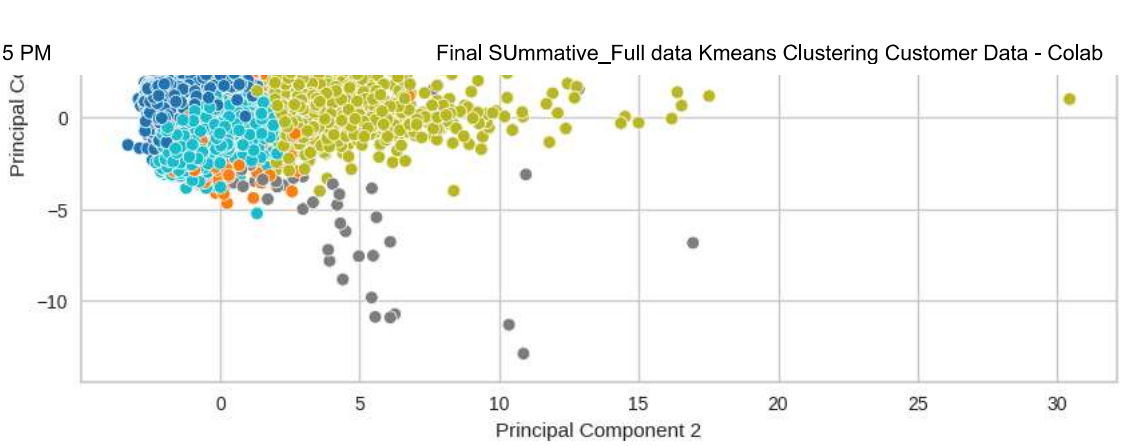


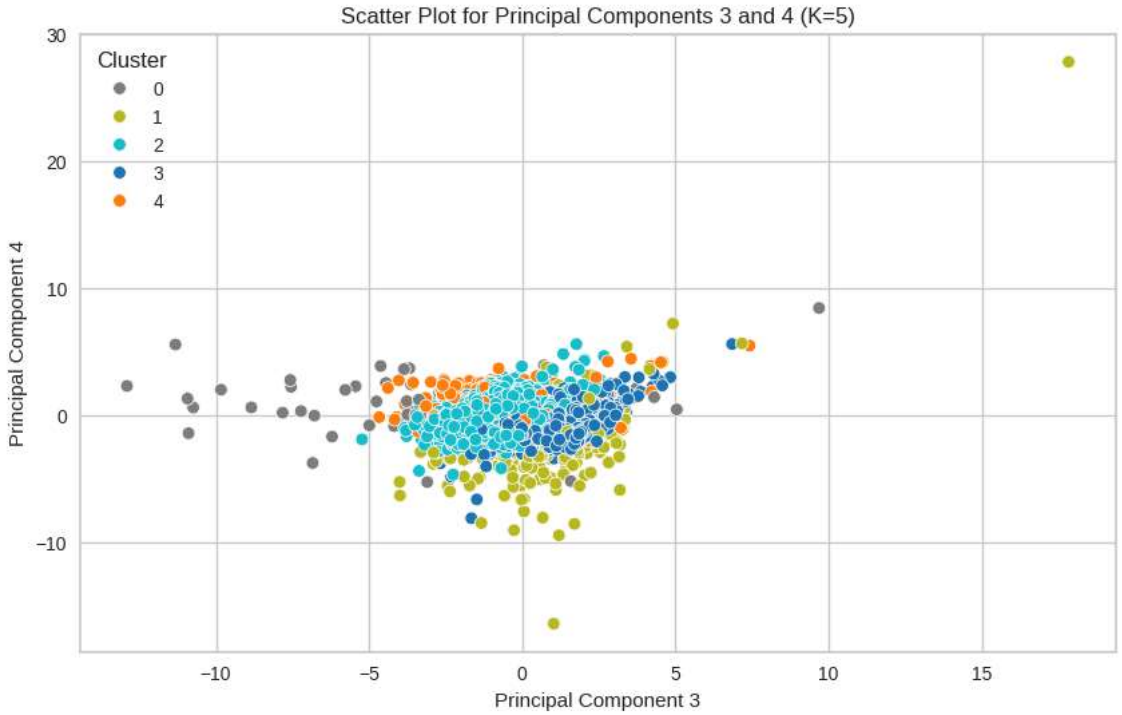

−10
0
10
20
30
40
50
Principal Component 5

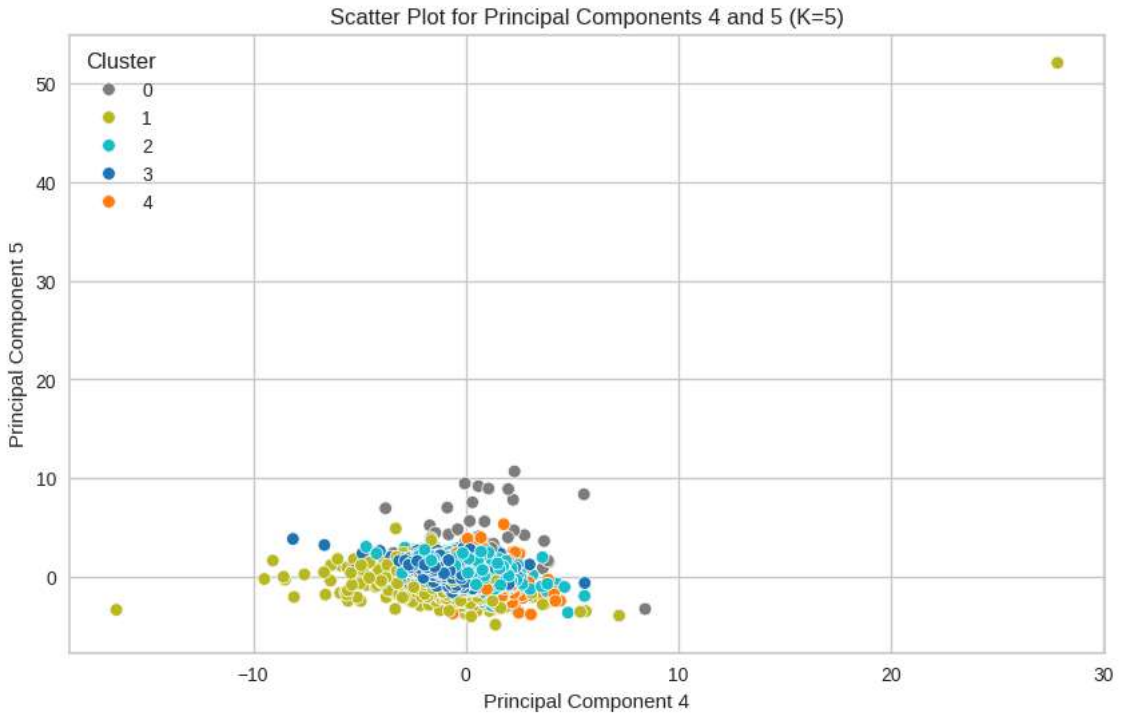

Scatter Plot for Principal Components 6 and 7 (K=4)
Principal Component 7
Principal Component 6
Cluster
0
1
2
3

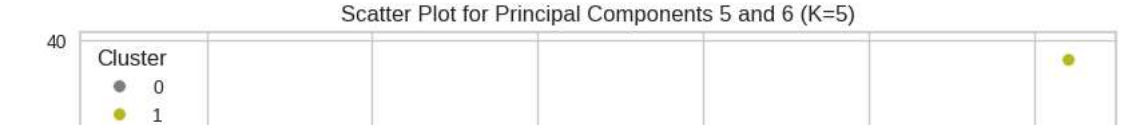

Scatter Plot for Principal Components 1 and 2 (K=5)
Principal Component 2
Principal Component 1
Cluster
0
1
2
3
4

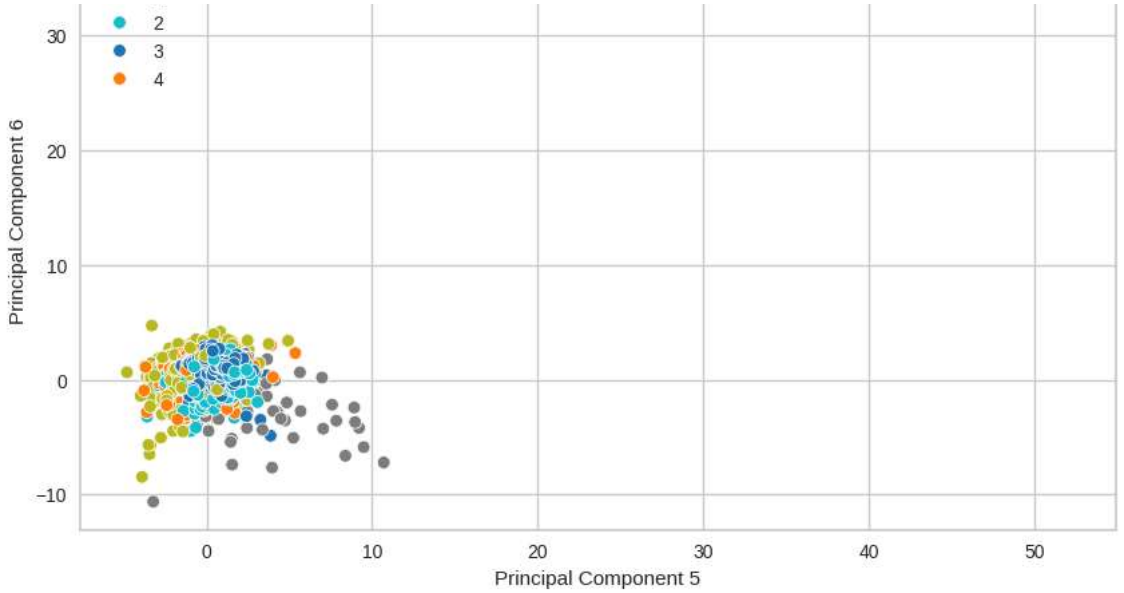

Scatter Plot for Principal Components 2 and 3 (K=5)
Cluster
0
1
2
3
4

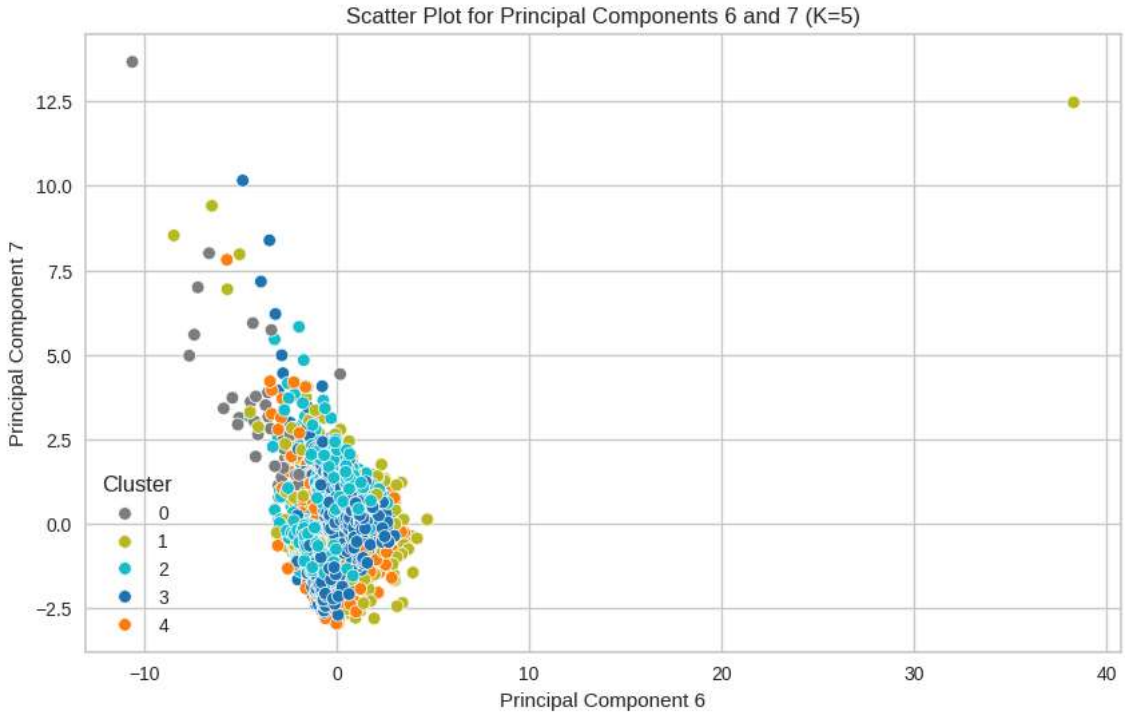

Principal C
0
−5
−10
0
5
10
15
20
25
30
Principal Component 2


Scatter Plot for Principal Components 3 and 4 (K=5)
Cluster
0
1
2
3
4
Principal Component 4
30
20
10
0
−10
−10
−5
0
5
10
15
Principal Component 3


Scatter Plot for Principal Components 4 and 5 (K=5)
Cluster
0
1
2
3
4
Principal Component 5
50
40
30
20
10
0
−10
0
10
20
30
Principal Component 4

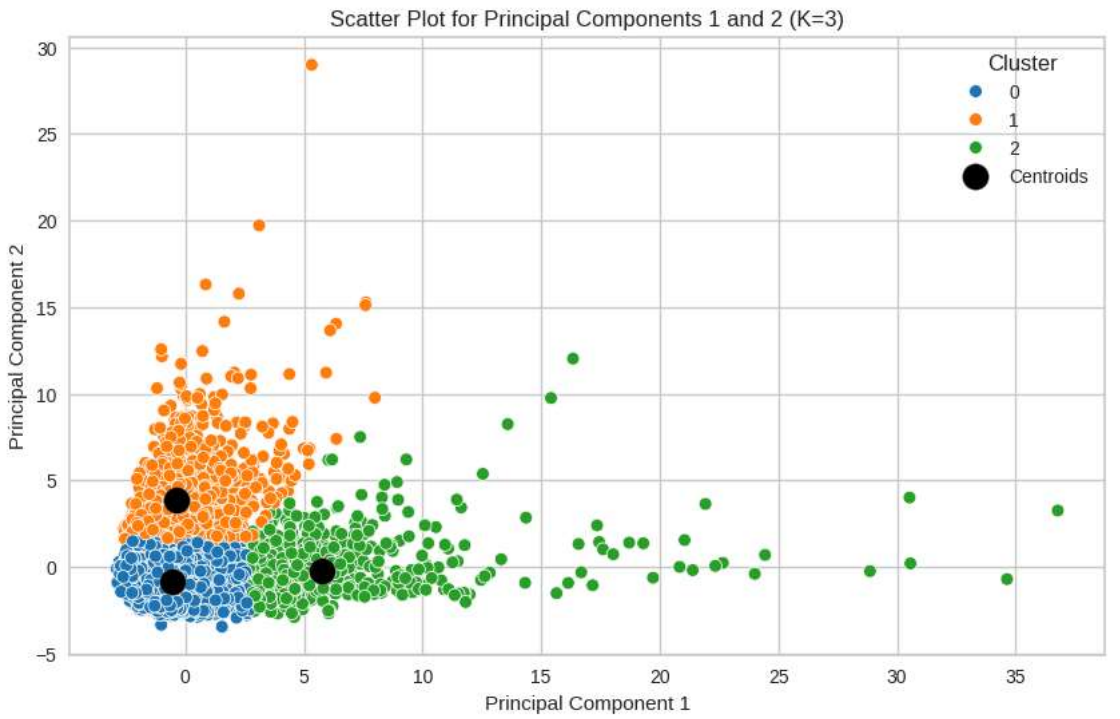

Scatter Plot for Principal Components 5 and 6 (K=5)
40
Cluster
0
1

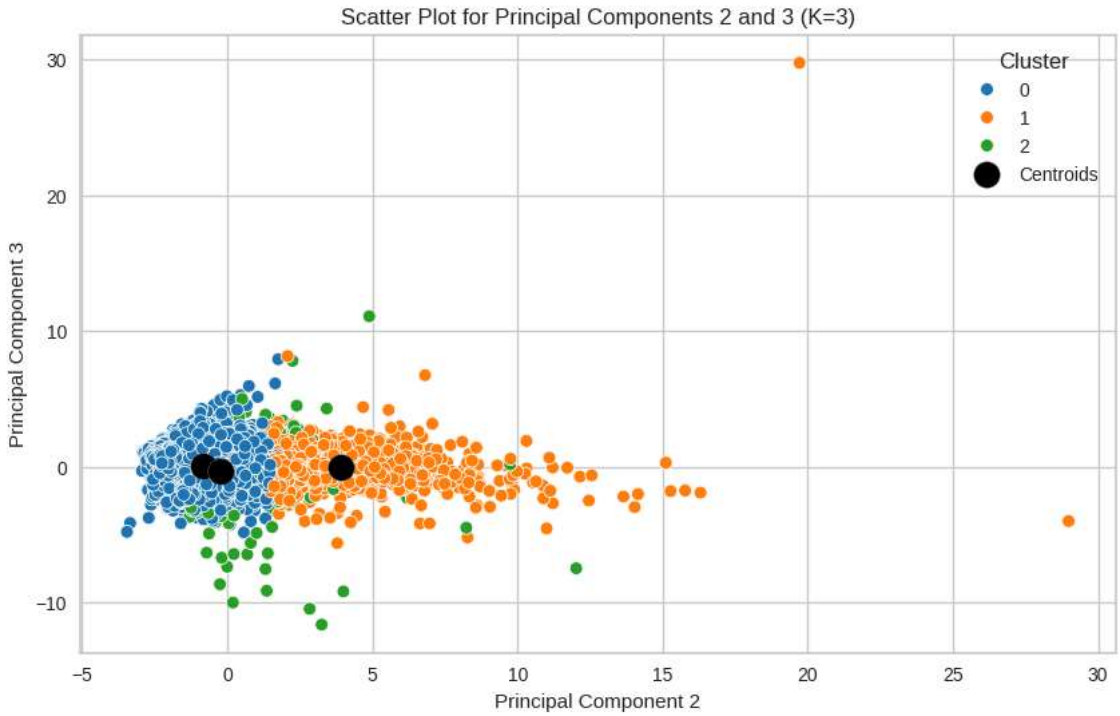


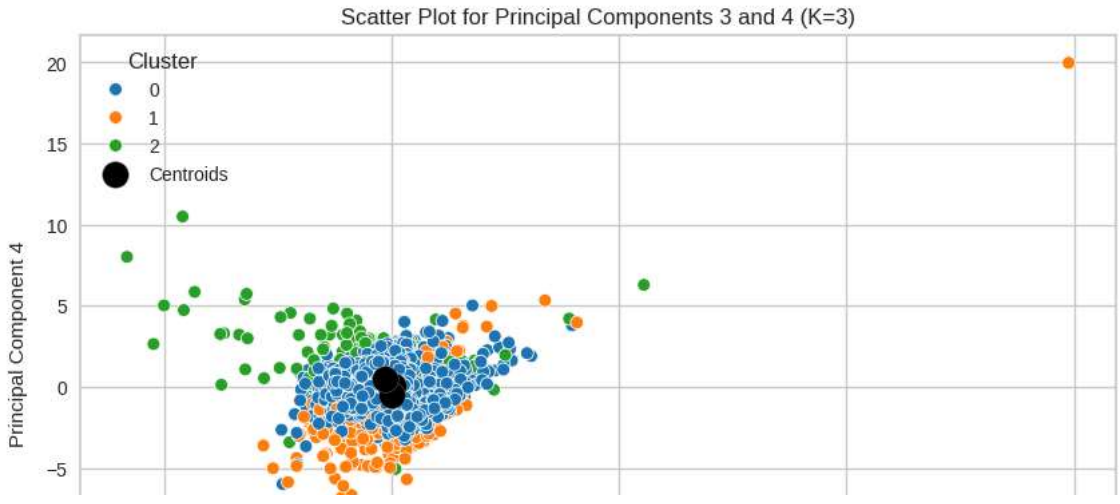


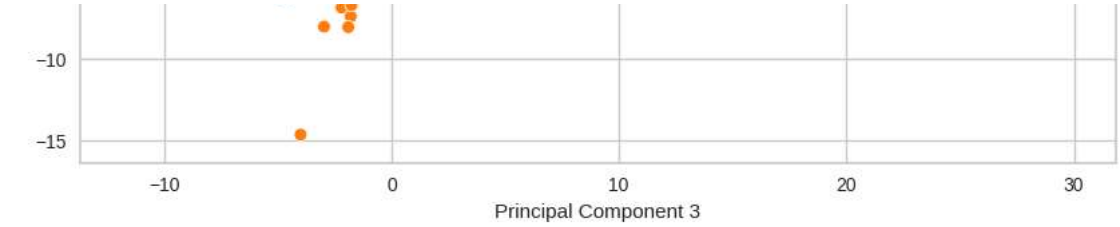

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.impute import SimpleImputer

# Function to apply KMeans clustering and visualize the results
def apply_kmeans_and_visualize(pca_data, n_clusters, palette):
    # Applying KMeans clustering
    kmeans = KMeans(n_clusters=n_clusters, n_init=100, init='k-means++', random_state=42)
    cluster_labels = kmeans.fit_predict(pca_data)
    centroids = kmeans.cluster_centers_

    # Creating a DataFrame for PCA data and cluster labels
    pca_df = pd.DataFrame(pca_data, columns=[f'PC{i+1}' for i in range(pca_data.shape[1])])
    pca_df['Cluster'] = cluster_labels
    centroids_df = pd.DataFrame(centroids, columns=[f'PC{i+1}' for i in range(pca_data.shape[1])])

    # Visualizing the scatter plots for principal component pairs
    for i in range(min(6, pca_data.shape[1] - 1)):
        plt.figure(figsize=(10, 6))
```

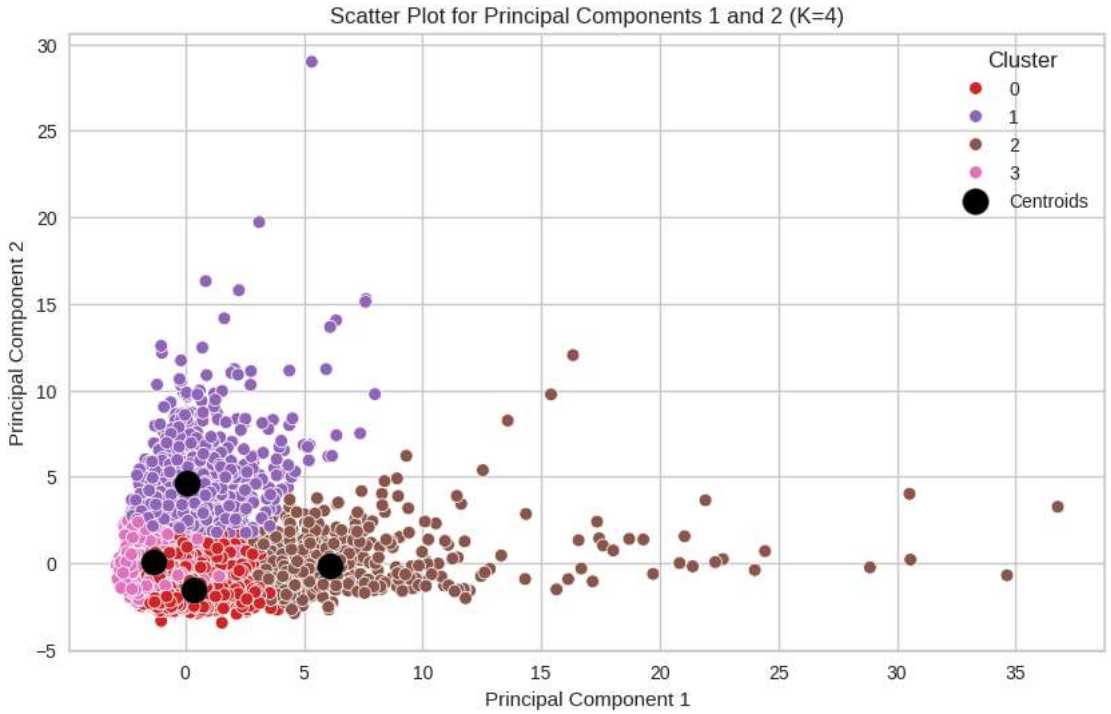
```
            sns.scatterplot(x=pca_df[f'PC{i+1}'], y=pca_df[f'PC{i+2}'], hue=pca_df['Cluster'], palette=palette, s=50)
            plt.scatter(centroids[:, i], centroids[:, i+1], c='black', edgecolor='grey', s=200, label='Centroids')
            plt.title(f'Scatter Plot for Principal Components {i+1} and {i+2} (K={n_clusters})')
            plt.xlabel(f'Principal Component {i+1}')
            plt.ylabel(f'Principal Component {i+2}')
            plt.legend(title='Cluster')
            plt.show()

    # Assuming dataset is already loaded and processed with numeric columns only

    # Handle missing values
    imputer = SimpleImputer(strategy='mean')
    imputed_data = imputer.fit_transform(dataset.select_dtypes(include=[np.number]))

    # Standardize the dataset
    scaler = StandardScaler()
    scaled_data = scaler.fit_transform(imputed_data)

    # Apply PCA
    pca = PCA(n_components=4)
    pca_data = pca.fit_transform(scaled_data)

    # Applying KMeans clustering and visualizing for 3, 4, and 5 clusters
    apply_kmeans_and_visualize(pca_data, n_clusters=3, palette=['#1f77b4', '#ff7f0e', '#2ca02c'])  # Blue, Orange, Green
    apply_kmeans_and_visualize(pca_data, n_clusters=4, palette=['#d62728', '#9467bd', '#8c564b', '#e377c2'])  # Red, Purple, Brown, Pink
    apply_kmeans_and_visualize(pca_data, n_clusters=5, palette=['#7f7f7f', '#bcbd22', '#17becf', '#1f77b4', '#ff7f0e'])  # Gray, Olive, Cyan, Blu
```

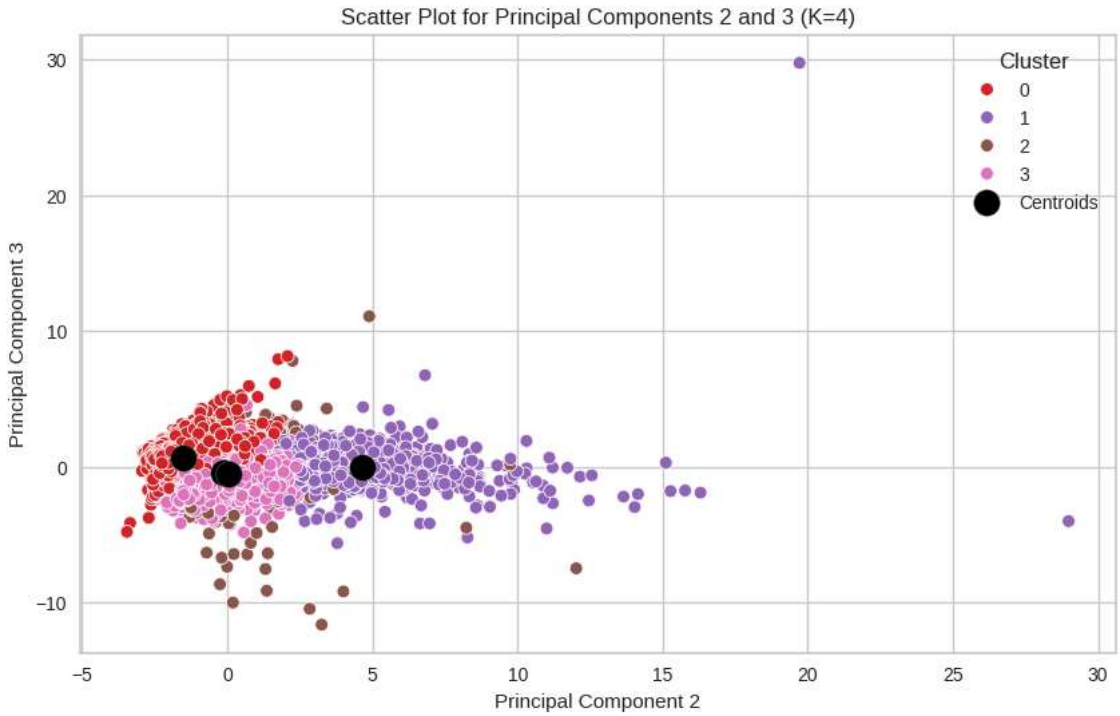

Scatter Plot for Principal Components 1 and 2 (K=3)
Cluster
0
1
2
Centroids
Principal Component 2
Principal Component 1

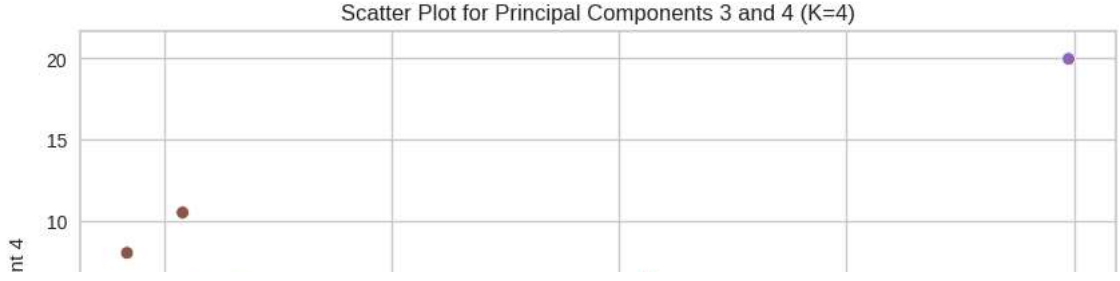

Scatter Plot for Principal Components 2 and 3 (K=3)
Cluster
0
1
2
Centroids
Principal Component 3
Principal Component 2

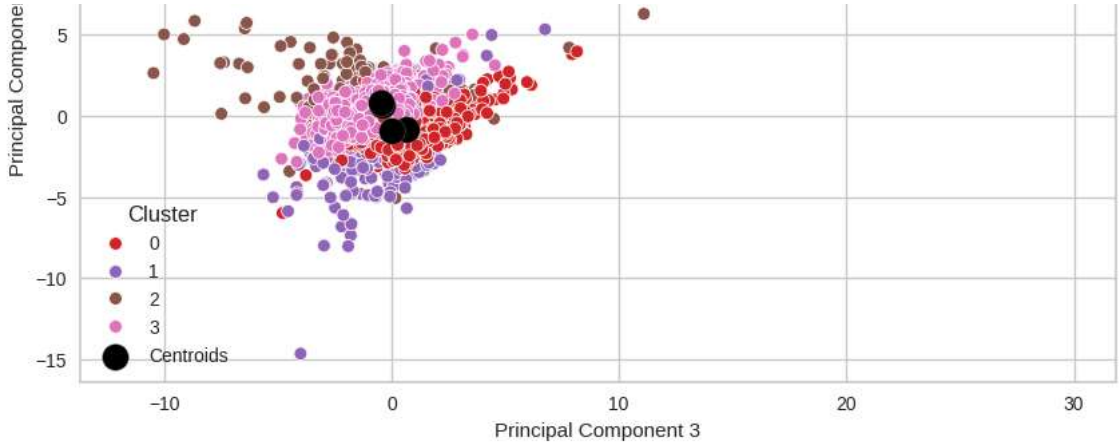

Scatter Plot for Principal Components 3 and 4 (K=3)
Cluster
0
1
2
Centroids
Principal Component 4

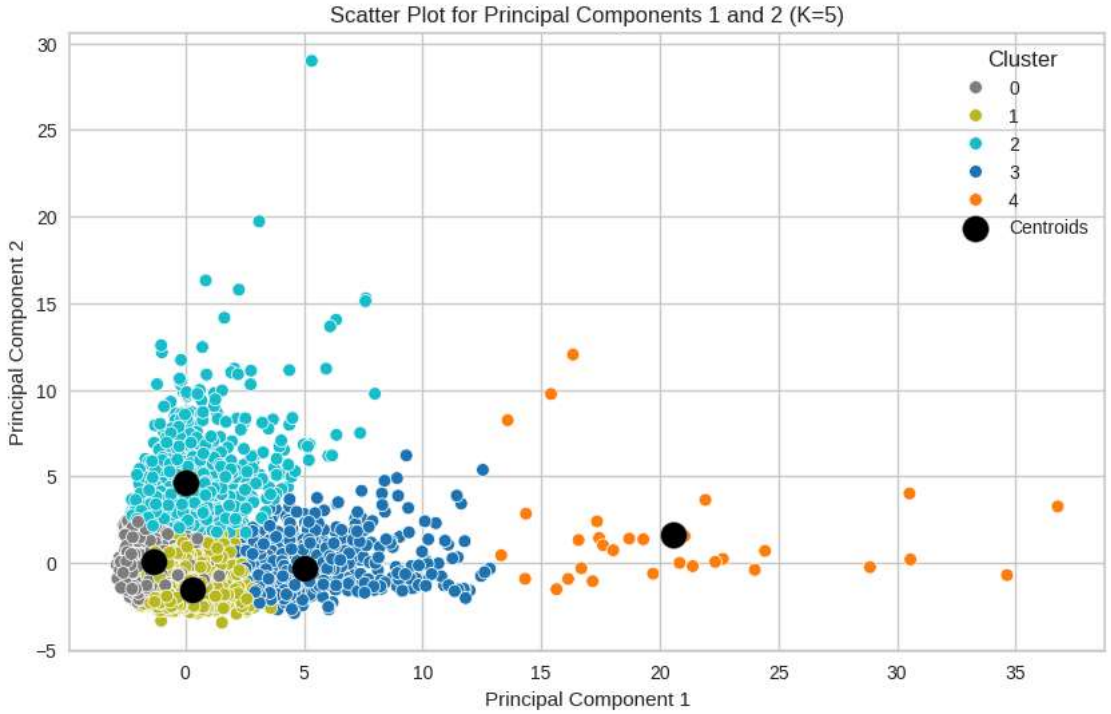

−10
−15
−10
0
10
20
30
Principal Component 3

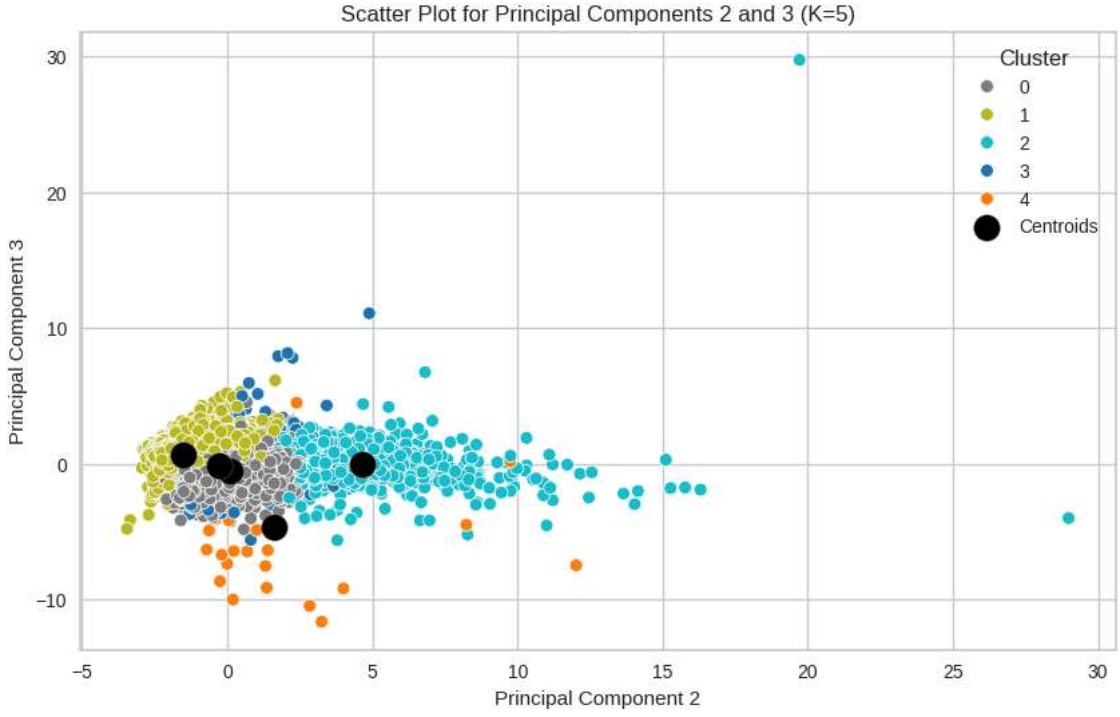

Scatter Plot for Principal Components 1 and 2 (K=4)
Cluster
0
1
2
3
Centroids
Principal Component 2
Principal Component 1
30
25
20
15
10
5
0
−5
0
5
10
15
20
25
30
35

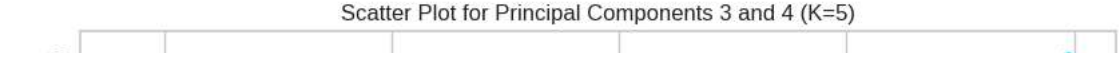

Scatter Plot for Principal Components 2 and 3 (K=4)
Cluster
0
1
2
3
Centroids
Principal Component 3
Principal Component 2
30
20
10
0
−10
−5
0
5
10
15
20
25
30

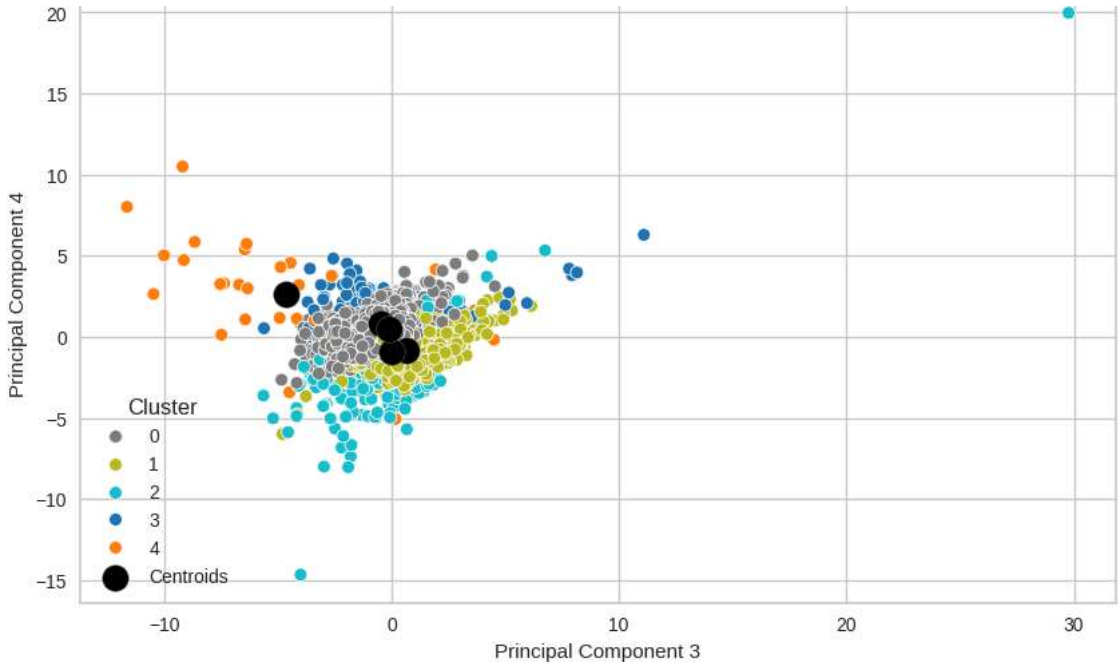

Scatter Plot for Principal Components 3 and 4 (K=4)
nt 4
20
15
10


Principal Compone
Cluster
0
1
2
3
Centroids
Principal Component 3

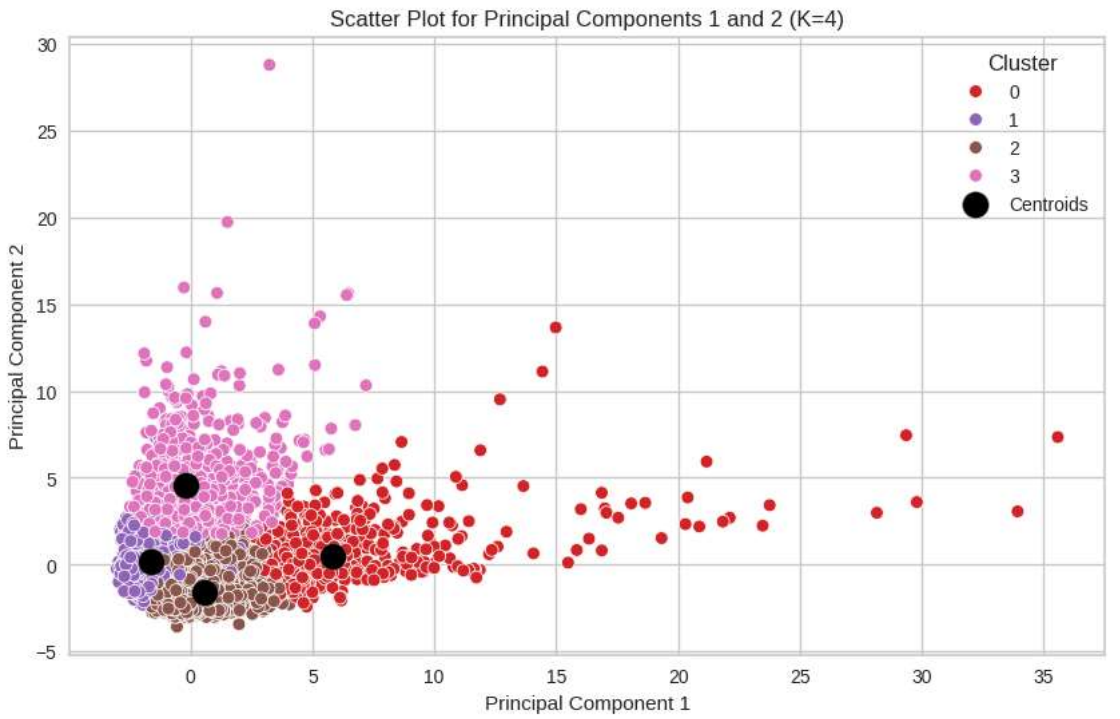

Scatter Plot for Principal Components 1 and 2 (K=5)
Cluster
0
1
2
3
4
Centroids
Principal Component 2
Principal Component 1

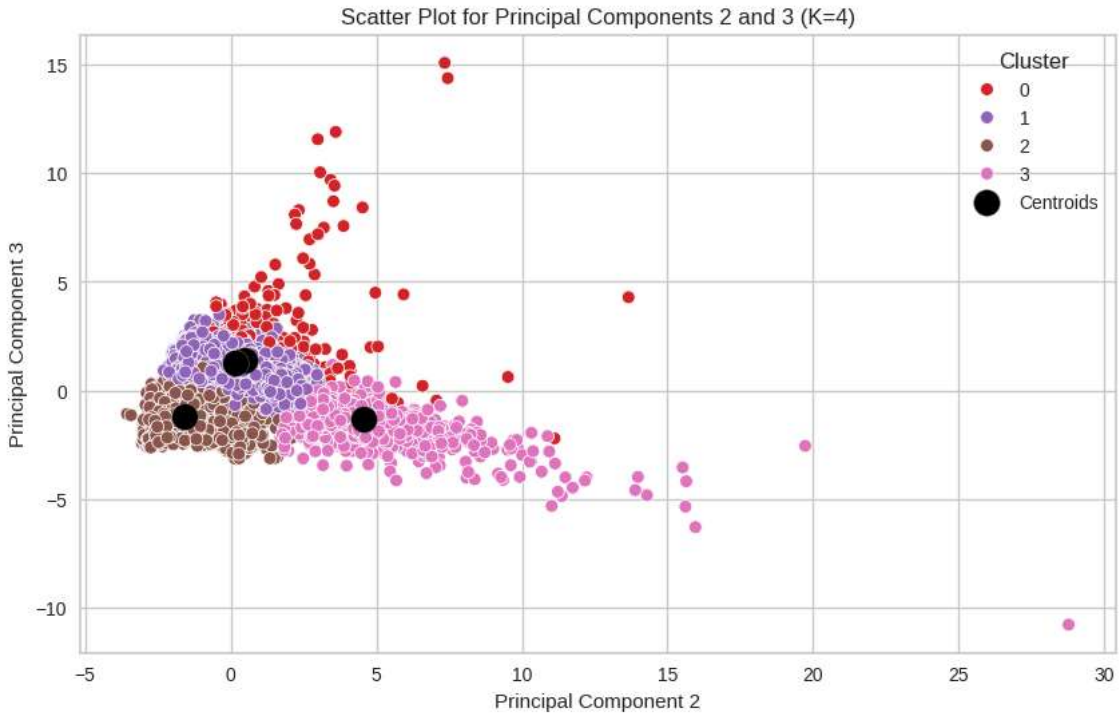

Scatter Plot for Principal Components 2 and 3 (K=5)
Cluster
0
1
2
3
4
Centroids
Principal Component 3
Principal Component 2

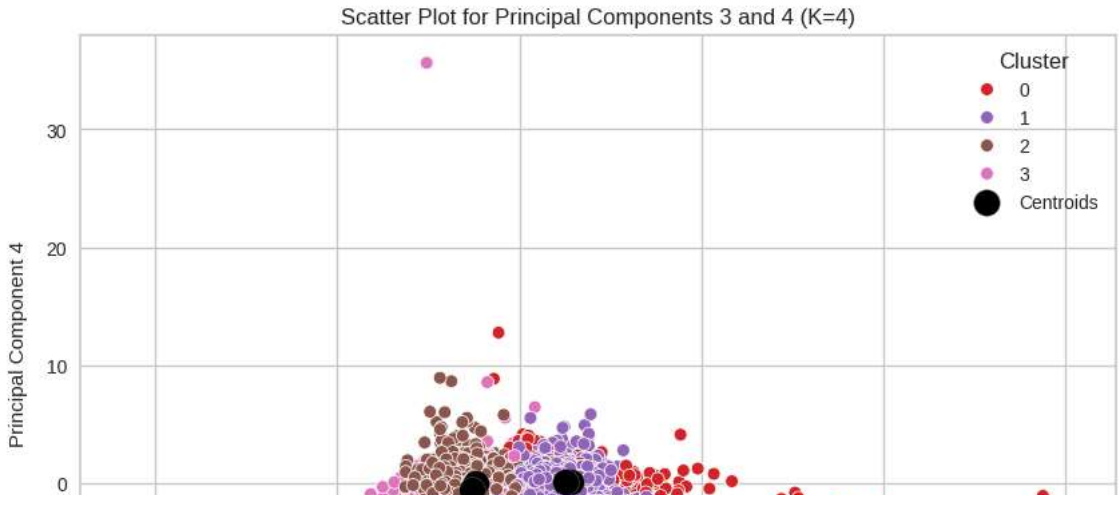

Scatter Plot for Principal Components 3 and 4 (K=5)

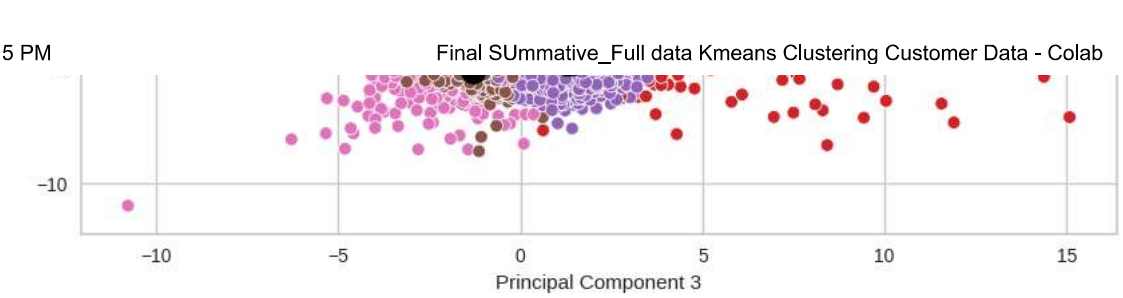

Principal Component 4
Principal Component 3
Cluster
0
1
2
3
4
Centroids
20
15
10
5
0
−5
−10
−15
−10
0
10
20
30

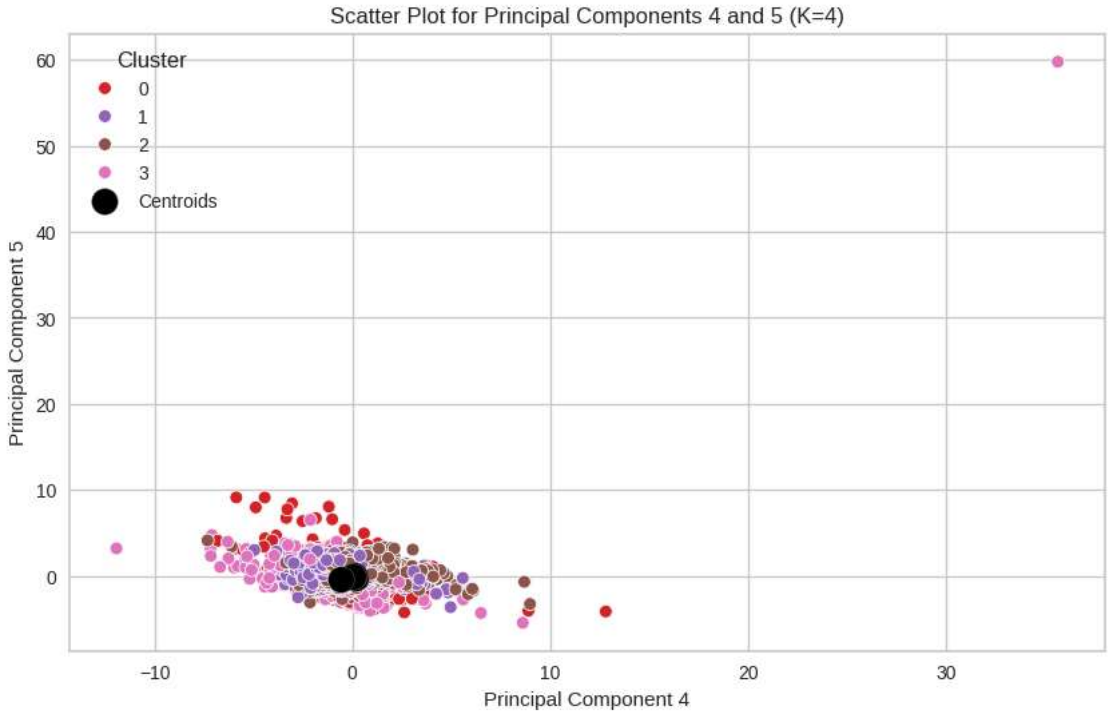
```
#only for cluster group 4 and between all 7 PCs

import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.impute import SimpleImputer

# Function to apply KMeans clustering and visualize the results
def apply_kmeans_and_visualize(pca_data, n_clusters, palette):
    # Applying KMeans clustering
    kmeans = KMeans(n_clusters=n_clusters, n_init=100, init='k-means++', random_state=42)
    cluster_labels = kmeans.fit_predict(pca_data)
    centroids = kmeans.cluster_centers_

    # Creating a DataFrame for PCA data and cluster labels
    pca_df = pd.DataFrame(pca_data, columns=[f'PC{i+1}' for i in range(pca_data.shape[1])])
    pca_df['Cluster'] = cluster_labels
    centroids_df = pd.DataFrame(centroids, columns=[f'PC{i+1}' for i in range(pca_data.shape[1])])

    # Visualizing the scatter plots for principal component pairs
    for i in range(pca_data.shape[1] - 1):
        plt.figure(figsize=(10, 6))
        sns.scatterplot(x=pca_df[f'PC{i+1}'], y=pca_df[f'PC{i+2}'], hue=pca_df['Cluster'], palette=palette, s=50)
        plt.scatter(centroids[:, i], centroids[:, i+1], c='black', edgecolor='grey', s=200, label='Centroids')
        plt.title(f'Scatter Plot for Principal Components {i+1} and {i+2} (K={n_clusters})')
        plt.xlabel(f'Principal Component {i+1}')
        plt.ylabel(f'Principal Component {i+2}')
        plt.legend(title='Cluster')
        plt.show()

# Assuming dataset is already loaded and processed with numeric columns only

# Handle missing values
imputer = SimpleImputer(strategy='mean')
imputed_data = imputer.fit_transform(dataset.select_dtypes(include=[np.number]))

# Standardize the dataset
scaler = StandardScaler()
scaled_data = scaler.fit_transform(imputed_data)

# Apply PCA
pca = PCA(n_components=7)  # Using 7 principal components
pca_data = pca.fit_transform(scaled_data)

# Applying KMeans clustering and visualizing for 4 clusters
apply_kmeans_and_visualize(pca_data, n_clusters=4, palette=['#d62728', '#9467bd', '#8c564b', '#e377c2'])  # Red, Purple, Brown, Pink
```

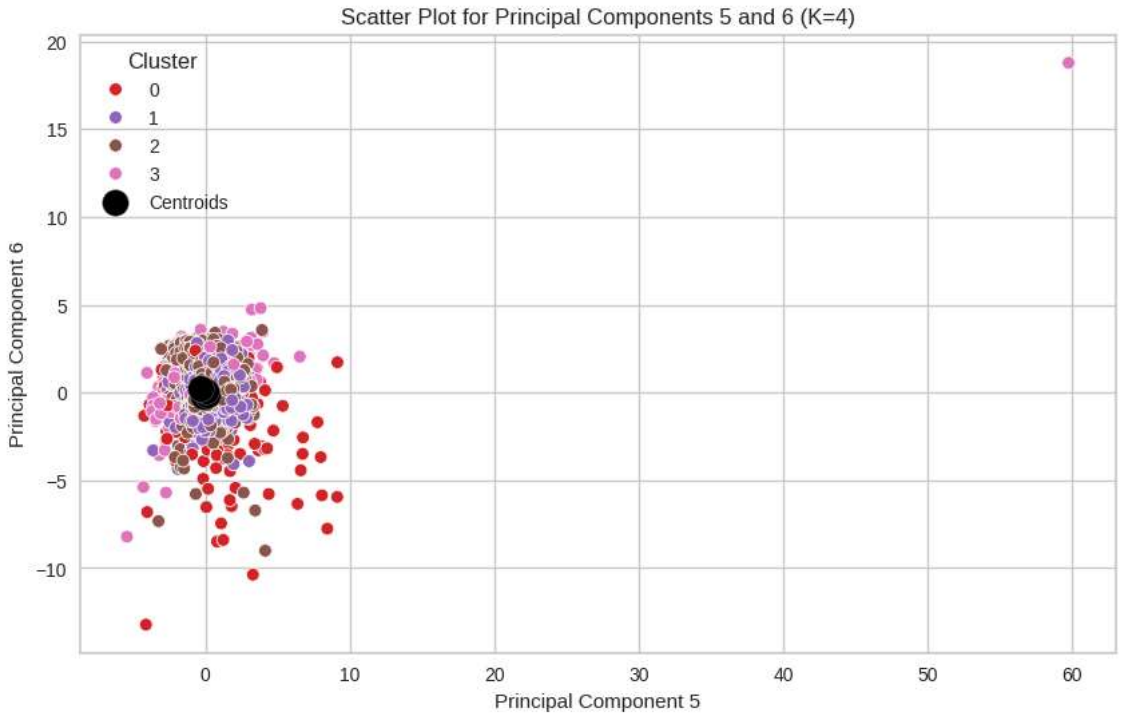

Scatter Plot for Principal Components 1 and 2 (K=4)
Cluster
0
1
2
3
Centroids
Principal Component 2
Principal Component 1

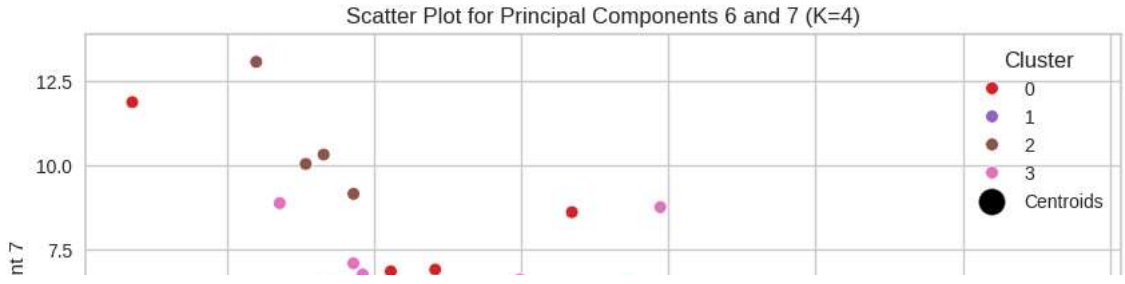

Scatter Plot for Principal Components 2 and 3 (K=4)
Cluster
0
1
2
3
Centroids
Principal Component 3
Principal Component 2


Scatter Plot for Principal Components 3 and 4 (K=4)
Cluster
0
1
2
3
Centroids
Principal Component 4


−10
−10
−5
0
5
10
15
Principal Component 3


Scatter Plot for Principal Components 4 and 5 (K=4)
Cluster
0
1
2
3
Centroids
Principal Component 5
Principal Component 4


Scatter Plot for Principal Components 5 and 6 (K=4)
Cluster
0
1
2
3
Centroids
Principal Component 6
Principal Component 5


Scatter Plot for Principal Components 6 and 7 (K=4)
Cluster
0
1
2
3
Centroids