Credid card Customer Segmentation

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import plotly.graph_objects as go
import plotly.express as px
from plotly.subplots import make_subplots
from matplotlib import colors
import matplotlib.pyplot as plt
%matplotlib inline
from itertools import product
import missingno

# Preprocessing Data
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer

# Modeling
from sklearn_extra.cluster import KMedoids
from sklearn.cluster import KMeans
from scipy.spatial.distance import cdist, pdist
from sklearn.decomposition import PCA
from sklearn.cluster import AgglomerativeClustering, Birch
from sklearn.metrics import silhouette_score, davies_bouldin_score, calinski_harab

# Handling Warning
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
pip install scikit-learn-extra
```

```
Requirement already satisfied: scikit-learn-extra in c:\users\dell\appdata\local\pr
ograms\python\python311\lib\site-packages (0.3.0)
Requirement already satisfied: numpy>=1.13.3 in c:\users\dell\appdata\local\program
s\python\python311\lib\site-packages (from scikit-learn-extra) (1.24.2)
Requirement already satisfied: scipy>=0.19.1 in c:\users\dell\appdata\local\program
s\python\python311\lib\site-packages (from scikit-learn-extra) (1.10.1)
Requirement already satisfied: scikit-learn>=0.23.0 in c:\users\dell\appdata\local
\programs\python\python311\lib\site-packages (from scikit-learn-extra) (1.2.2)
Requirement already satisfied: joblib>=1.1.1 in c:\users\dell\appdata\local\program
s\python\python311\lib\site-packages (from scikit-learn>=0.23.0->scikit-learn-extr
a) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\dell\appdata\local
\programs\python\python311\lib\site-packages (from scikit-learn>=0.23.0->scikit-lea
rn-extra) (3.1.0)
Note: you may need to restart the kernel to use updated packages.
[notice] A new release of pip available: 22.3.1 -> 23.0.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

Load DataSet

In [3]:
```
df=pd.read_csv('C:/Users/DELL/Downloads/Data/Credit Card_Clustering/Credit Card_Cl
```

In [4]:
```
df.head()
```

Out[4]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INST/
0	C10001	40.900749	0.818182	95.40	0.00	
1	C10002	3202.467416	0.909091	0.00	0.00	
2	C10003	2495.148862	1.000000	773.17	773.17	
3	C10004	1666.670542	0.636364	1499.00	1499.00	
4	C10005	817.714335	1.000000	16.00	16.00	

Exploratory Data Analysis

In [5]:
```
# convert name of feature to lower font
df.columns = map(str.lower, df.columns)
df.head()
```

Out[5]:

	cust_id	balance	balance_frequency	purchases	oneoff_purchases	installments_purc
0	C10001	40.900749	0.818182	95.40	0.00	
1	C10002	3202.467416	0.909091	0.00	0.00	
2	C10003	2495.148862	1.000000	773.17	773.17	
3	C10004	1666.670542	0.636364	1499.00	1499.00	
4	C10005	817.714335	1.000000	16.00	16.00	

In [6]:
```
df_wo_id = df.drop('cust_id', axis=1)
```

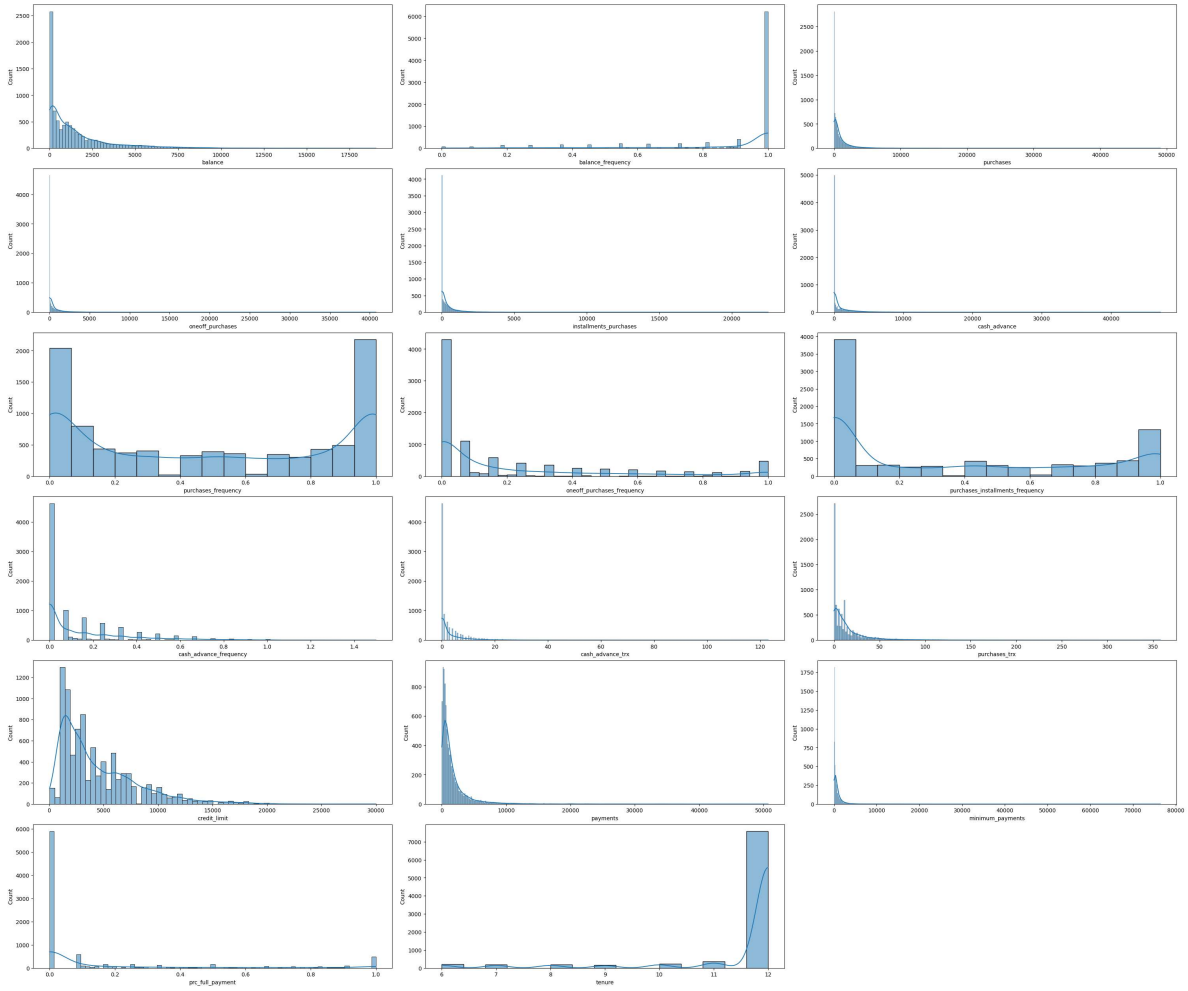
In [7]:
```
df.head()
```

Out[7]:

	cust_id	balance	balance_frequency	purchases	oneoff_purchases	installments_purc
0	C10001	40.900749	0.818182	95.40	0.00	
1	C10002	3202.467416	0.909091	0.00	0.00	
2	C10003	2495.148862	1.000000	773.17	773.17	
3	C10004	1666.670542	0.636364	1499.00	1499.00	
4	C10005	817.714335	1.000000	16.00	16.00	

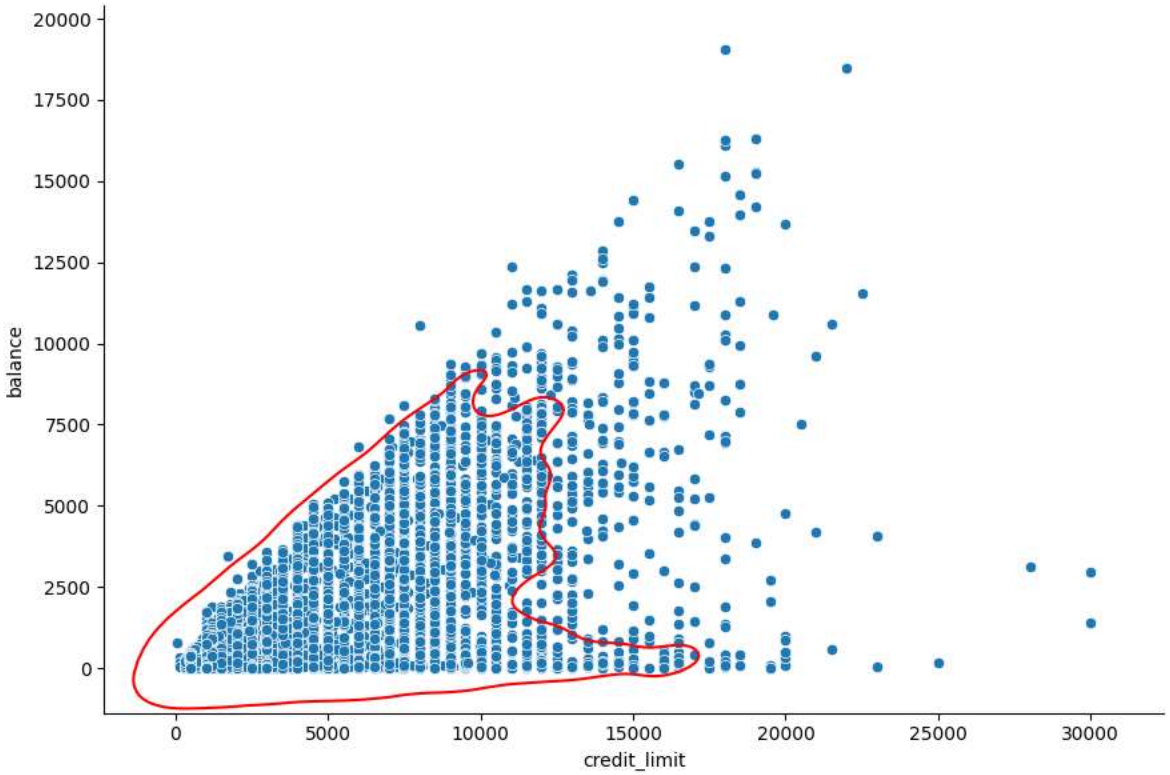
Univariate Analysis

In [8]:
```
# creating distribution plot for analysis
fig = plt.figure(figsize=(30, 25), constrained_layout=True)
for i in range(len(df_wo_id.columns)):
    plt.subplot(6, 3, i+1)
    sns.histplot(df_wo_id[df_wo_id.columns[i]], kde=True)
```

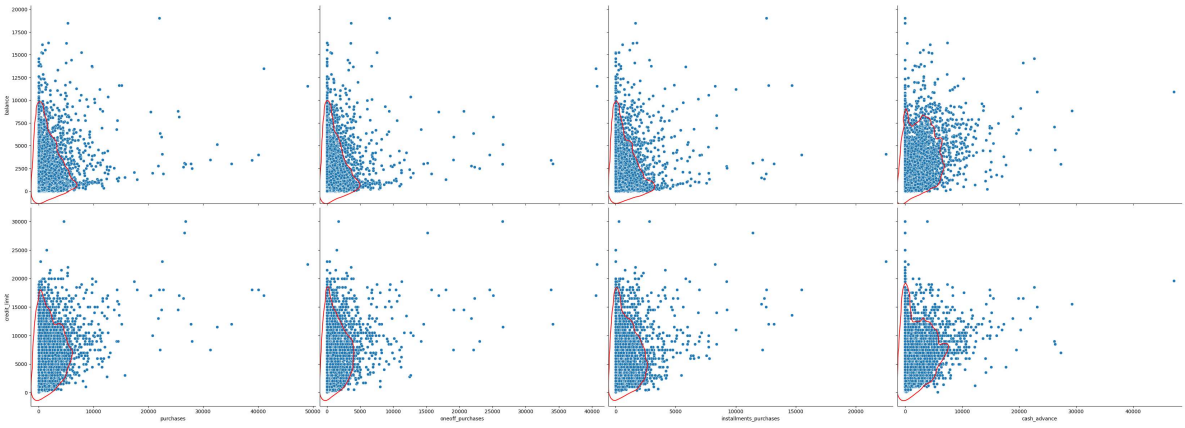


Bivariate & Multivariate Analysis

In [9]:
```
#plot relation credit limit with balance
sns.pairplot(data=df_wo_id, x_vars='credit_limit', y_vars='balance',
             height=6, aspect=1.5).map(sns.kdeplot, levels=1, color='red');
```



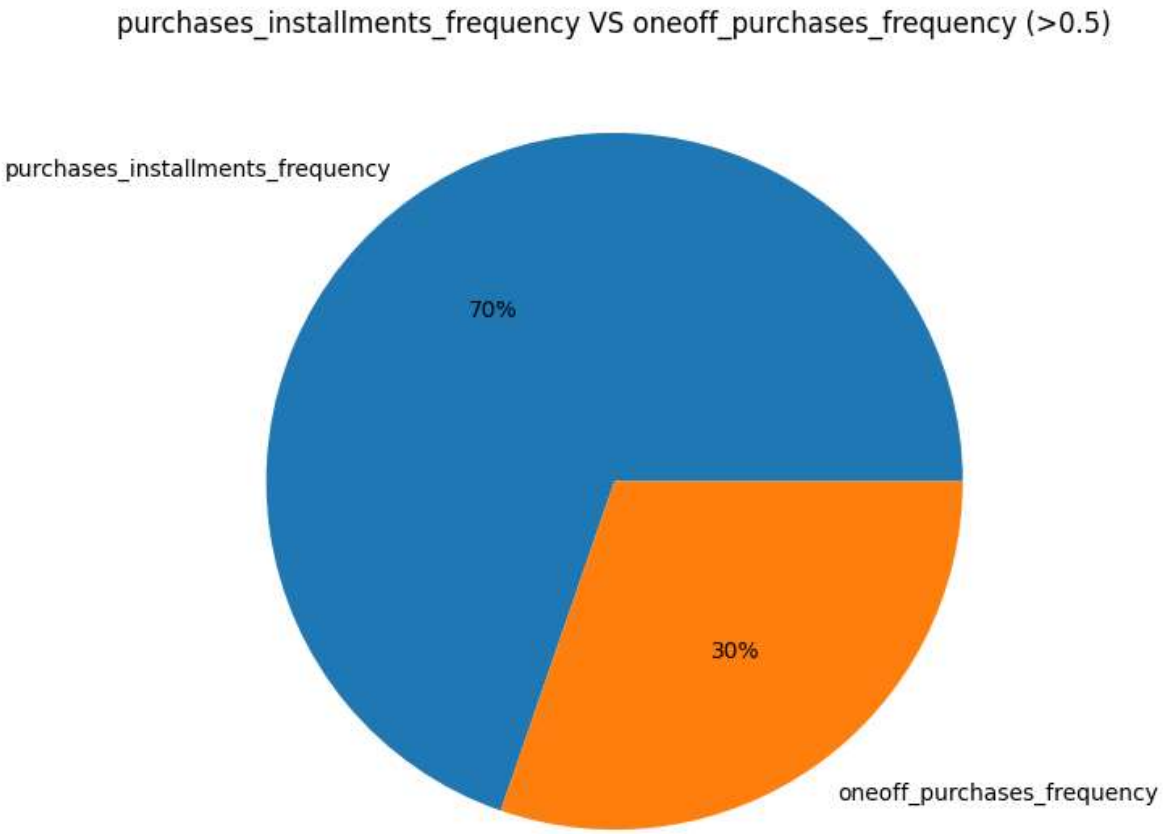
In [10]:
```
#relation plot between features
sns.pairplot(data=df_wo_id,
             y_vars=['balance', 'credit_limit'],
             x_vars=['purchases', 'oneoff_purchases', 'installments_purchases', 'c
            height=6, aspect=1.4).map(sns.kdeplot, levels=1, color='red');
```



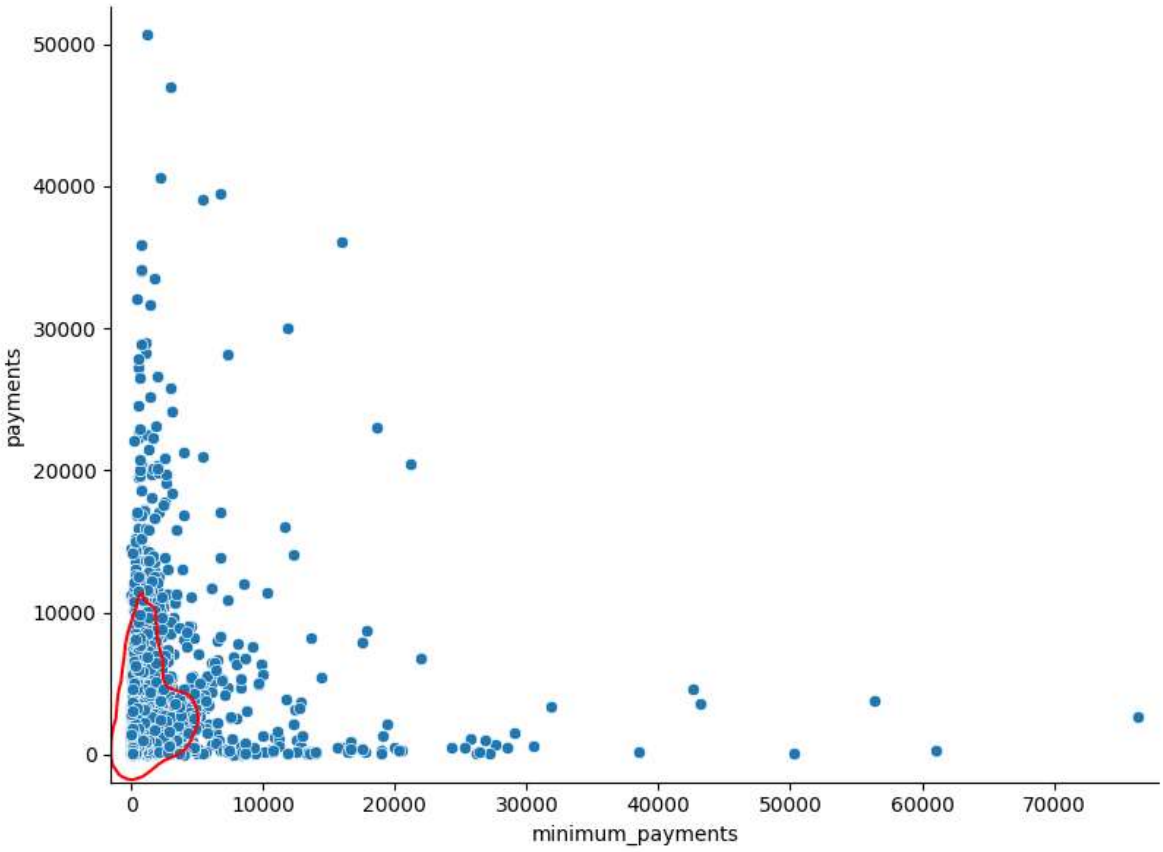
In [11]:
```
#displaying purchase frequency above 0.5
print(len(df_wo_id[df_wo_id['purchases_installments_frequency'] > .5]), 'purchases
print(len(df_wo_id[df_wo_id['oneoff_purchases_frequency'] > .5]), 'oneoff_purchase
```

```
3090 purchases_installments_frequency
1343 oneoff_purchases_frequency
```

In [12]:
```
#pie plot for installment vs oneoff
plt.figure(figsize=(7,7))
plt.pie([len(df_wo_id[df_wo_id['purchases_installments_frequency'] > .5]),
         len(df_wo_id[df_wo_id['oneoff_purchases_frequency'] > .5])],
        labels = ['purchases_installments_frequency',
                  'oneoff_purchases_frequency'],
        autopct='%.0f%%')
plt.title('purchases_installments_frequency VS oneoff_purchases_frequency (>0.5)')
```



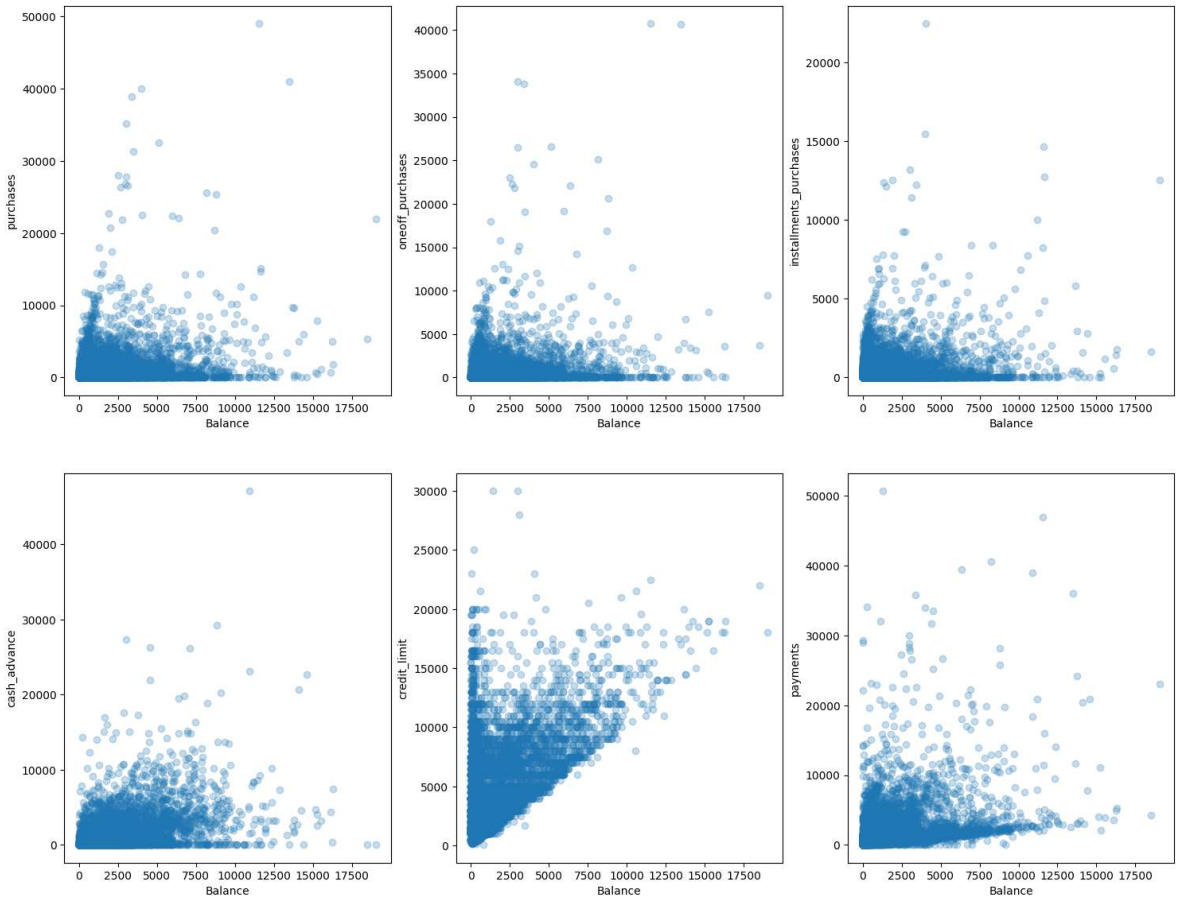
In [13]:
```
#pair plot for payments vs minimum payments
sns.pairplot(data=df_wo_id, y_vars='payments', x_vars='minimum_payments',
             height=6, aspect=1.4).map(sns.kdeplot, levels=1, color='red');
```



Bivariate Analysis

Balance

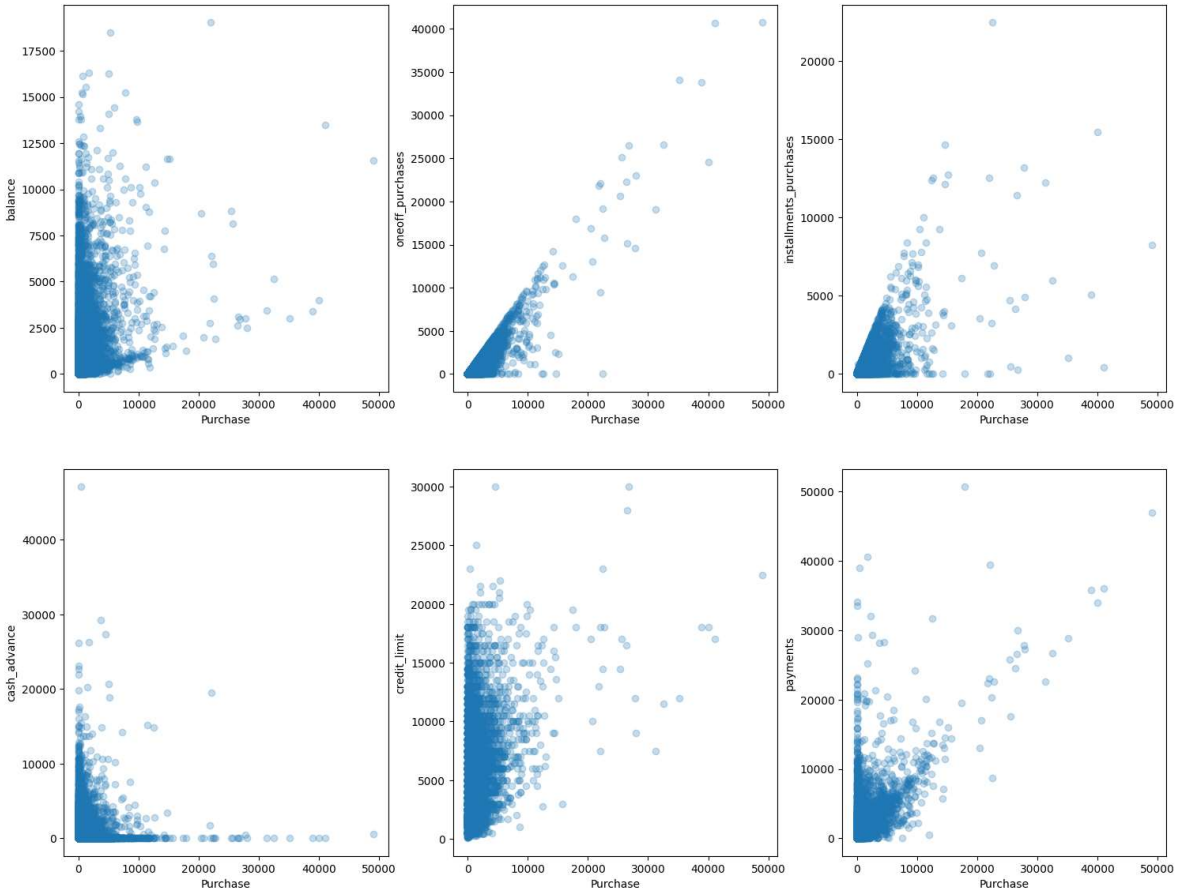
In [14]:
```
# creating plot analysis
plt.figure(figsize=(18, 14))
columns=['purchases','oneoff_purchases','installments_purchases','cash_advance','c
n=1
for x in columns:
    plt.subplot(2,3,n)
    plt.scatter(df_wo_id['balance'], df_wo_id[x], alpha=0.25)
    plt.xlabel('Balance')
    plt.ylabel(x)
    n=n+1
plt.ticklabel_format(useOffset=False, style='plain')
plt.show()
```



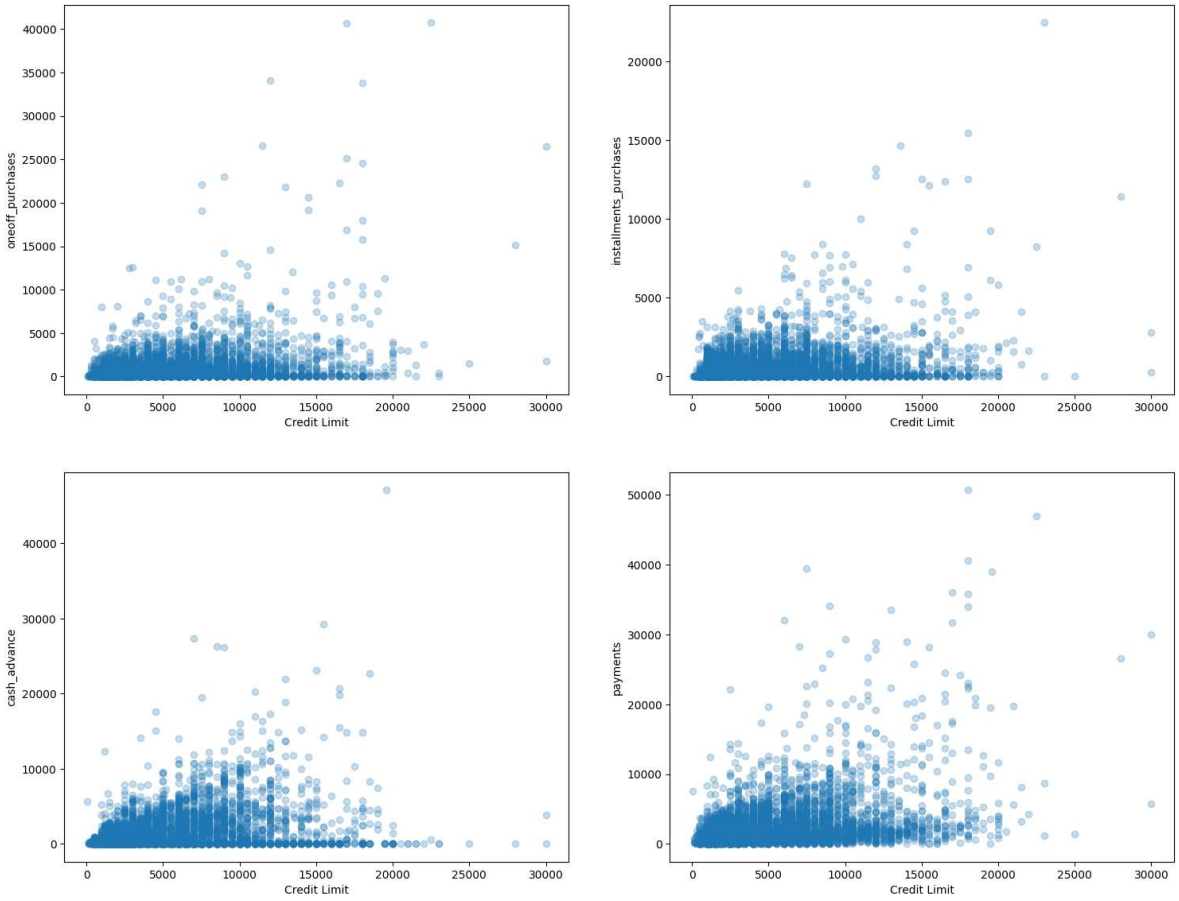
Purchase

In [15]:
```
# creating plot analysis
plt.figure(figsize=(18, 14))
columns=['balance','oneoff_purchases','installments_purchases','cash_advance','cre
n=1
for x in columns:
    plt.subplot(2,3,n)
    plt.scatter(df_wo_id['purchases'], df_wo_id[x], alpha=0.25)
    plt.xlabel('Purchase')
```

```
    plt.ylabel(x)
    n=n+1
plt.ticklabel_format(useOffset=False, style='plain')
plt.show()
```

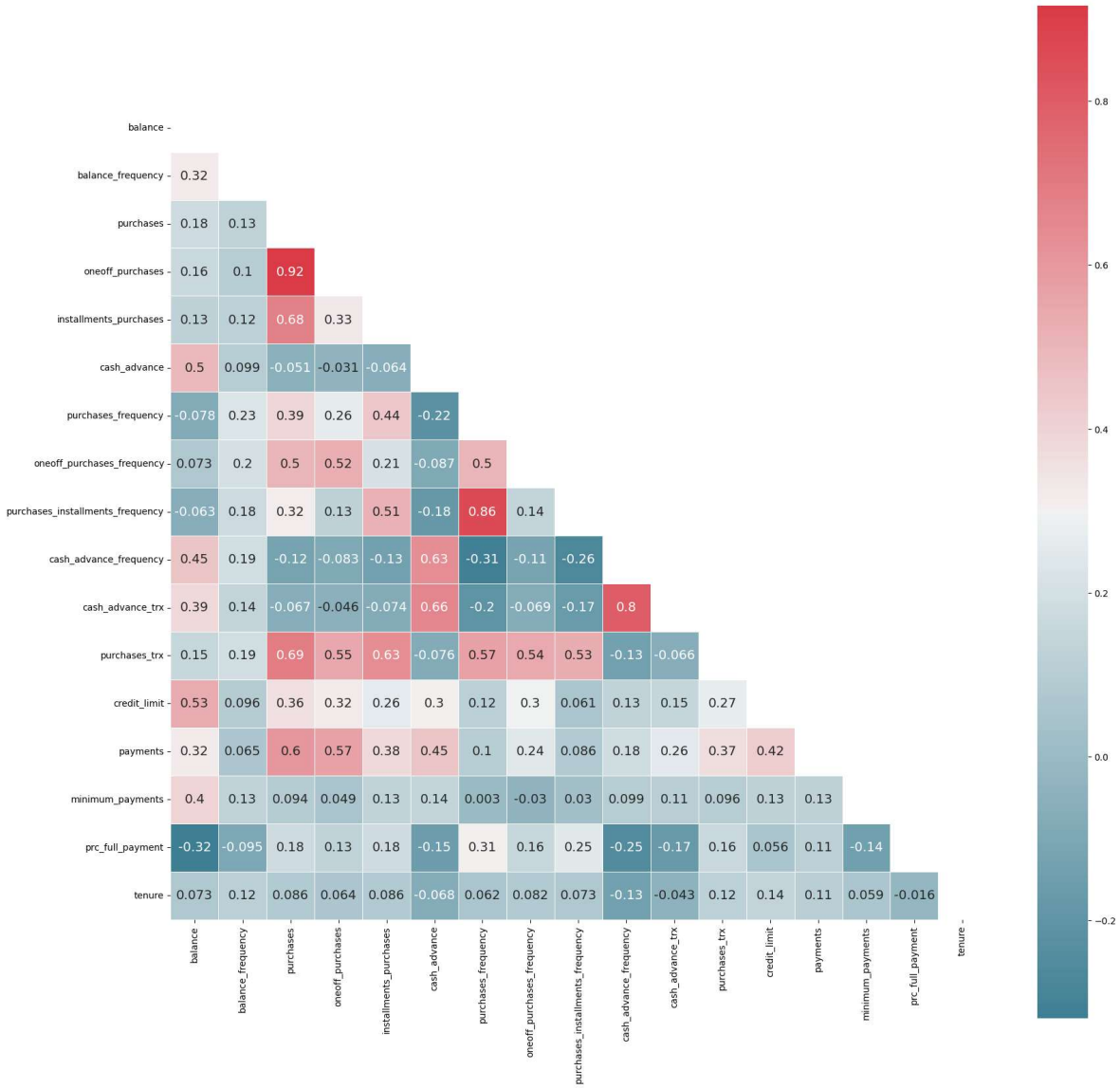


In [16]:
```
# creating plot
plt.figure(figsize=(18, 14))
columns=['oneoff_purchases','installments_purchases','cash_advance','payments']
n=1
for x in columns:
    plt.subplot(2,2,n)
    plt.scatter(df_wo_id['credit_limit'], df_wo_id[x], alpha=0.25)
    plt.xlabel('Credit Limit')
    plt.ylabel(x)
    n=n+1
plt.ticklabel_format(useOffset=False, style='plain')
plt.show()
```



Multivariate Analysis - Correlation Between Numerical Features

In [17]:
```
# calculates the correlations
correlation = df_wo_id.corr(method='pearson')

# uses the variable ax for single a Axes
fig, ax = plt.subplots()

# sets the figure size in inches
ax.figure.set_size_inches(20, 20)

# generates a mask for the upper triangle
mask = np.triu(np.ones_like(correlation, dtype=bool))

# generates a custom diverging colormap
cmap = sns.diverging_palette(220, 10, as_cmap=True)

# plots the heatmap
sns.heatmap(correlation, cmap=cmap, mask=mask, square=True, linewidths=.5,
            annot=True, annot_kws={'size':14})

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



Data Preprocessing

Outlier, Anomaly, Duplicates, Identify missing value

In [18]:
```
# check missing value
df_wo_id.info()
```

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

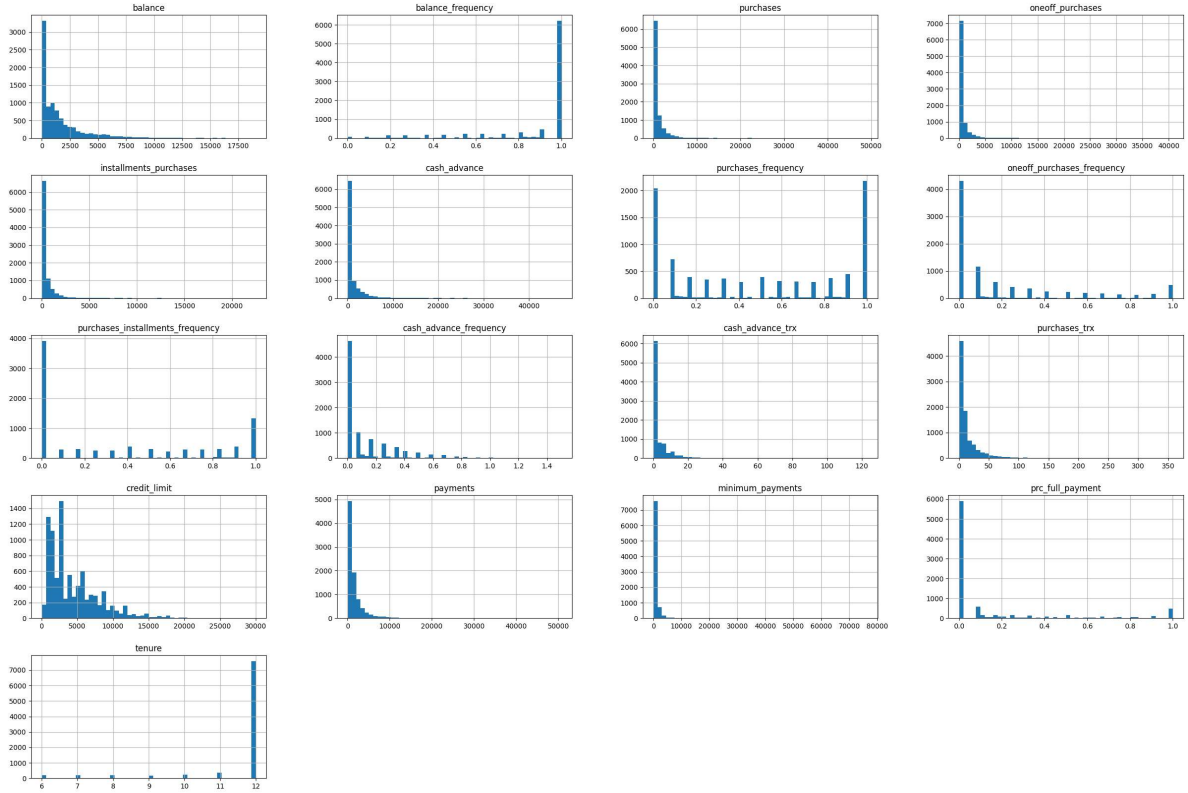
In [19]:
```
# Check detailed info for each feature
listItem = []
for col in df_wo_id.columns :
    listItem.append([col, df_wo_id[col].dtype, df_wo_id[col].isna().sum(), round((
                     df_wo_id[col].nunique(), list(df_wo_id[col].drop_duplicates().

dfDesc = pd.DataFrame(columns=['dataFeatures', 'dataType', 'missing value', 'nullP
                      data=listItem)
dfDesc
```

Out[19]:

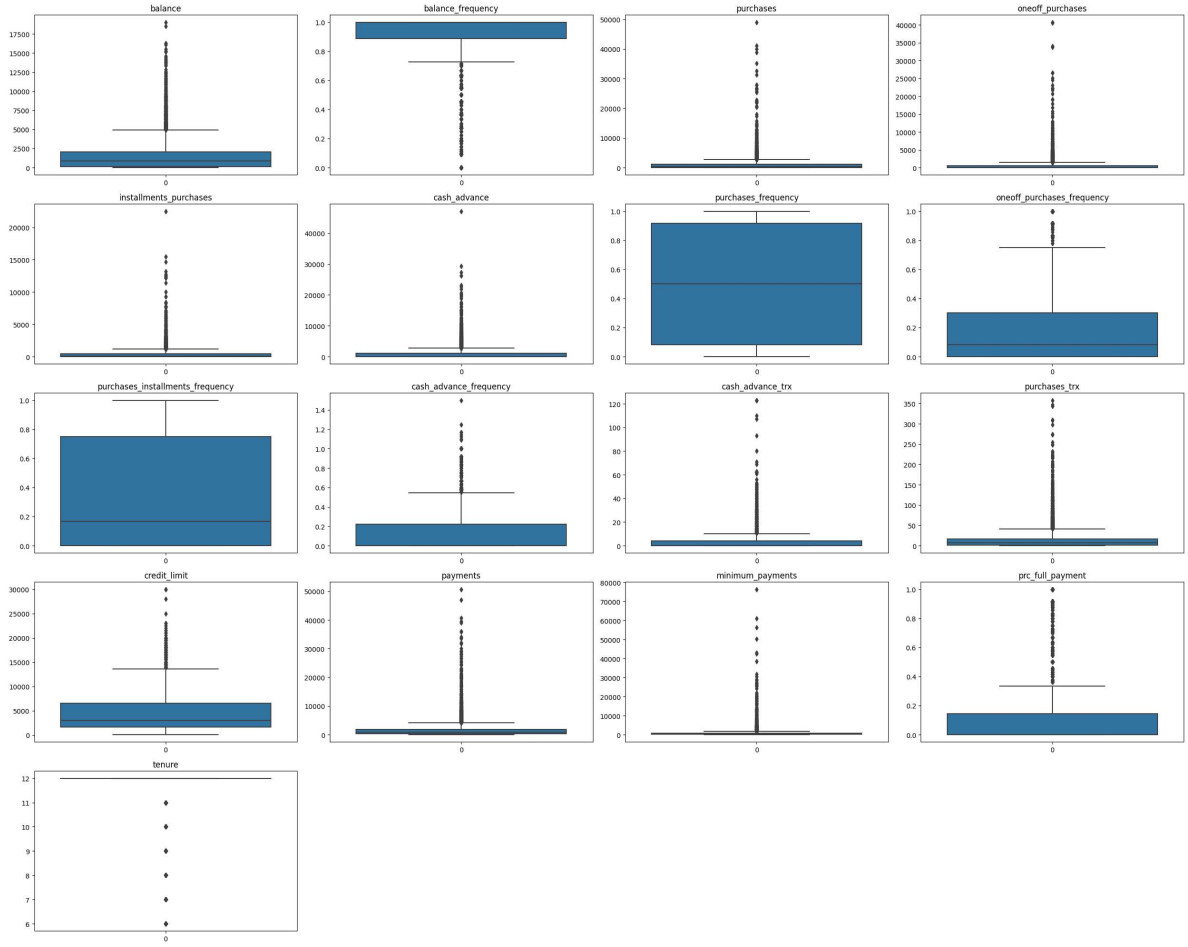
	dataFeatures	dataType	missing value	nullPct	unique	uniqueSample
0	balance	float64	0	0.00	8871	[2494.774127, 5408.754166]
1	balance_frequency	float64	0	0.00	43	[0.875, 0.9]
2	purchases	float64	0	0.00	6203	[808.0, 571.32]
3	oneoff_purchases	float64	0	0.00	4014	[3195.58, 280.6]
4	installments_purchases	float64	0	0.00	4452	[92.02, 104.95]
5	cash_advance	float64	0	0.00	4323	[1271.94319, 1470.99071]
6	purchases_frequency	float64	0	0.00	47	[0.8, 0.272727]
7	oneoff_purchases_frequency	float64	0	0.00	47	[0.416667, 0.625]
8	purchases_installments_frequency	float64	0	0.00	47	[0.444444, 0.909091]
9	cash_advance_frequency	float64	0	0.00	54	[1.125, 0.545455]
10	cash_advance_trx	int64	0	0.00	65	[17, 39]
11	purchases_trx	int64	0	0.00	173	[7, 56]
12	credit_limit	float64	1	0.01	205	[8600.0, 5750.0]
13	payments	float64	0	0.00	8711	[1686.269252, 3225.132002]
14	minimum_payments	float64	313	3.50	8636	[1938.465006, 81.829414]
15	prc_full_payment	float64	0	0.00	47	[0.3, 0.333333]
16	tenure	int64	0	0.00	7	[6, 9]

Outlier Detection

In [20]:
```
# Distribution plot to visualize data distribution

hist_b_ho = df_wo_id.hist(bins=50,figsize=(30,20))
hist_b_ho
plt.show()
```



In [21]:
```
# plot for outliers detection

plt.figure(figsize=(25,20))
for i in range(len(df_wo_id.columns)):
    plt.subplot(5,4,i+1)
    sns.boxplot(df_wo_id[df_wo_id.columns[i]])
    plt.title(df_wo_id.columns[i])

plt.tight_layout()
```



In [22]:
```
# creating function to generate IQR, lower limit, and Upper limit

def find_outlier(df, feature):
    print('Outlier ' + feature)
    q1 = df[feature].quantile(0.25)
    q3 = df[feature].quantile(0.75)
    iqr = q3 - q1
    limit = iqr*1.5
    print(f'IQR: {iqr}')

    Lower_Limit = q1 - limit
    Upper_Limit = q3 + limit
    print(f'Lower_Limit: {Lower_Limit}')
    print(f'Upper_Limit: {Upper_Limit}')
    print('_____________________________')
```

In [23]:
```
# check IQR, upper limit, and lower limit for each feature

for i in df_wo_id :
    find_outlier(df_wo_id, i)
```

```
Outlier balance
IQR: 1925.85812
Lower_Limit: -2760.5052645
Upper_Limit: 4942.9272155

_____________________________
Outlier balance_frequency
IQR: 0.11111099999999996
Lower_Limit: 0.7222225000000001
Upper_Limit: 1.1666664999999998

_____________________________
Outlier purchases
IQR: 1070.4950000000001
Lower_Limit: -1566.1075000000003
Upper_Limit: 2715.8725000000004

_____________________________
Outlier oneoff_purchases
IQR: 577.405
Lower_Limit: -866.1075
Upper_Limit: 1443.5124999999998

_____________________________
Outlier installments_purchases
IQR: 468.6375
Lower_Limit: -702.95625
Upper_Limit: 1171.59375

_____________________________
Outlier cash_advance
IQR: 1113.8211392500002
Lower_Limit: -1670.7317088750003
Upper_Limit: 2784.5528481250003

_____________________________
Outlier purchases_frequency
IQR: 0.833334
Lower_Limit: -1.166668
Upper_Limit: 2.166668

_____________________________
Outlier oneoff_purchases_frequency
IQR: 0.3
Lower_Limit: -0.44999999999999996
Upper_Limit: 0.75

_____________________________
Outlier purchases_installments_frequency
IQR: 0.75
Lower_Limit: -1.125
Upper_Limit: 1.875

_____________________________
Outlier cash_advance_frequency
IQR: 0.222222
Lower_Limit: -0.333333
Upper_Limit: 0.555555

_____________________________
Outlier cash_advance_trx
IQR: 4.0
Lower_Limit: -6.0
Upper_Limit: 10.0

_____________________________
Outlier purchases_trx
IQR: 16.0
Lower_Limit: -23.0
Upper_Limit: 41.0

_____________________________
Outlier credit_limit
```

```
IQR: 4900.0
Lower_Limit: -5750.0
Upper_Limit: 13850.0

_____________________________
Outlier payments
IQR: 1517.8581507500003
Lower_Limit: -1893.5110601250003
Upper_Limit: 4177.921542875

_____________________________
Outlier minimum_payments
IQR: 656.361752
Lower_Limit: -815.4189210000001
Upper_Limit: 1810.0280870000001

_____________________________
Outlier prc_full_payment
IQR: 0.142857
Lower_Limit: -0.21428550000000002
Upper_Limit: 0.35714250000000003

_____________________________
Outlier tenure
IQR: 0.0
Lower_Limit: 12.0
Upper_Limit: 12.0

_____________________________
```

In [24]:
```
# creating outliers columns
out1 = df_wo_id[df_wo_id['balance'] > 4942.9272155]
out2 = df_wo_id[(df_wo_id['balance_frequency'] < 0.7222225000000001) & (df['balanc
out3 = df_wo_id[df_wo_id['purchases'] > 2715.8725000000004]
out4 = df_wo_id[df_wo_id['oneoff_purchases'] > 1443.5124999999998]
out5 = df_wo_id[df_wo_id['installments_purchases'] > 1171.59375]
out6 = df_wo_id[df_wo_id['cash_advance'] > 2784.5528481250003]
out7 = df_wo_id[df_wo_id['purchases_frequency'] > 2.166668]
out8 = df_wo_id[df_wo_id['oneoff_purchases_frequency'] > 0.75]
out9 = df_wo_id[df_wo_id['purchases_installments_frequency'] > 1.875]
out10 = df_wo_id[df_wo_id['cash_advance_frequency'] > 0.555555]
out11 = df_wo_id[df_wo_id['cash_advance_trx'] > 10.0]
out12 = df_wo_id[df_wo_id['purchases_trx'] > 41.0]
out13 = df_wo_id[df_wo_id['credit_limit'] > 13850.0]
out14 = df_wo_id[df_wo_id['payments'] > 4177.921542875]
out15 = df_wo_id[df_wo_id['minimum_payments'] > 1810.0280870000001]
out16 = df_wo_id[df_wo_id['prc_full_payment'] > 0.35714250000000003]
out17 = df_wo_id[(df_wo_id['tenure'] != 12.0)]
```

In [25]:
```
# creating outliers dataframe

out_all = pd.concat([out1, out2, out3, out4, out5, out6, out7, out8, out9, out10,
out_all.drop_duplicates(inplace=True)
out_all
```

Out[25]:

	balance	balance_frequency	purchases	oneoff_purchases	installments_purchase
15	6886.213231	1.000000	1611.70	0.00	1611.7
21	6369.531318	1.000000	6359.95	5910.04	449.9
24	5368.571219	1.000000	0.00	0.00	0.0
28	7152.864372	1.000000	387.05	204.55	182.5
30	12136.219960	1.000000	3038.01	1013.20	2024.8
...	...	...	...	...	
8944	193.571722	0.833333	1012.73	1012.73	0.0
8946	19.183215	1.000000	300.00	0.00	300.0
8947	23.398673	0.833333	144.40	0.00	144.4
8948	13.457564	0.833333	0.00	0.00	0.0
8949	372.708075	0.666667	1093.25	1093.25	0.0

5238 rows × 17 columns

In [26]:
```
#check outliers percentage

print('Percentage Outlier')
len(out_all)/len(df)*100
```

```
Percentage Outlier
```

Out[26]:
```
58.52513966480447
```

Data Anomalies

In [27]:
```
#Anomaly 1
df_wo_id[df_wo_id['balance'] > df_wo_id['credit_limit']]
```

Out[27]:

	balance	balance_frequency	purchases	oneoff_purchases	installments_purchases
5	1809.828751	1.0	1333.28	0.0	1333.28
10	1293.124939	1.0	920.12	0.0	920.12
20	2016.684686	1.0	176.68	0.0	176.68
64	1923.886805	1.0	1887.64	0.0	1887.64
78	1205.716678	1.0	0.00	0.0	0.00
...	...	...	...	...	..
8589	1283.337407	1.0	1022.41	171.6	850.81
8614	1182.080141	1.0	266.68	0.0	266.68
8624	1012.089680	1.0	312.48	0.0	312.48
8724	3002.791004	1.0	2463.00	2463.0	0.00
8873	1023.883008	1.0	585.84	0.0	585.84

227 rows × 17 columns

In [28]:
```
# Anomaly 2
df_wo_id[df_wo_id['minimum_payments'] > df_wo_id['payments']]
```

Out[28]:

	balance	balance_frequency	purchases	oneoff_purchases	installments_purchases
2	2495.148862	1.000000	773.17	773.17	0.00
5	1809.828751	1.000000	1333.28	0.00	1333.28
10	1293.124939	1.000000	920.12	0.00	920.12
14	2772.772734	1.000000	0.00	0.00	0.00
15	6886.213231	1.000000	1611.70	0.00	1611.70
...	...	...	...	...	..
8933	735.652303	1.000000	619.60	255.62	363.98
8939	728.352548	1.000000	734.40	734.40	0.00
8947	23.398673	0.833333	144.40	0.00	144.40
8948	13.457564	0.833333	0.00	0.00	0.00
8949	372.708075	0.666667	1093.25	1093.25	0.00

2365 rows × 17 columns

In [29]:
```
# Anomaly 3
df_wo_id[df_wo_id['installments_purchases'] > df_wo_id['credit_limit']]
```

Out[29]:

	balance	balance_frequency	purchases	oneoff_purchases	installments_purchases
64	1923.886805	1.000000	1887.64	0.00	1887.64
180	903.297810	1.000000	2697.48	150.50	2546.98
185	168.522709	0.818182	2009.05	0.00	2009.05
231	279.608696	1.000000	3034.92	0.00	3034.92
295	635.090434	0.818182	5758.71	2252.37	3506.34
...	...	...	...	...	..
8788	133.099445	1.000000	1170.00	0.00	1170.00
8836	112.037368	1.000000	1006.69	0.00	1006.69
8837	272.624436	0.888889	1843.00	470.00	1373.00
8856	227.220411	1.000000	1387.60	288.54	1099.06
8883	931.907808	1.000000	1142.12	0.00	1142.12

118 rows × 17 columns

In [30]:
```
# Anomaly 4
df_wo_id[df_wo_id['cash_advance_frequency'] > 1]
```

Out[30]:

	balance	balance_frequency	purchases	oneoff_purchases	installments_purchases
681	5656.069801	1.000000	362.36	362.36	0.0
1626	2876.009336	1.000000	152.61	152.61	0.0
2555	5906.184924	1.000000	141.80	141.80	0.0
2608	7801.511533	1.000000	231.40	231.40	0.0
3038	3846.742530	1.000000	0.00	0.00	0.0
3253	5709.486507	0.833333	0.00	0.00	0.0
8055	1917.895730	1.000000	285.07	285.07	0.0
8365	3857.562230	1.000000	0.00	0.00	0.0

In [31]:
```
# change cash_advance_frequency above 1 to 1
cash_advance_freq_more_than_1 = df_wo_id[df_wo_id['cash_advance_frequency'] > 1].i
df_wo_id['cash_advance_frequency'].iloc[cash_advance_freq_more_than_1] = 1

#check whether there is still cash_advance_frequency above 1
df_wo_id[df_wo_id['cash_advance_frequency'] > 1].cash_advance_frequency.any()
```

Out[31]:
```
False
```

Duplicate

In [32]:
```
#check whether there is any duplicate value
df_wo_id[df_wo_id.duplicated()]
```

Out[32]:

	balance	balance_frequency	purchases	oneoff_purchases	installments_purchases	cash_a

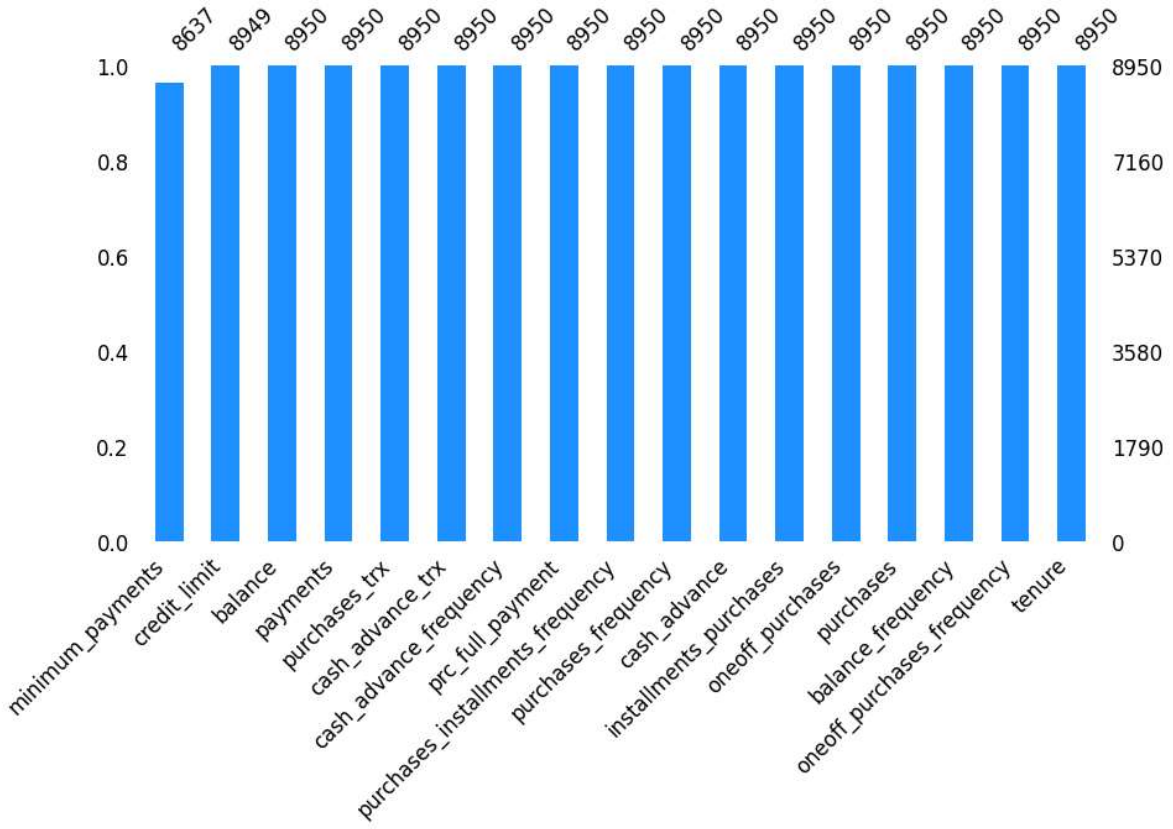
Missing Value

In [33]:
```
# check missing value
df_wo_id.isna().sum()
```

Out[33]:
```
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
```

In [34]:
```
# plot to visualize the missing value proportion
missingno.bar(df_wo_id,color="dodgerblue", sort="ascending", figsize=(10,5), fonts
```

Out[34]:
```
<Axes: >
```



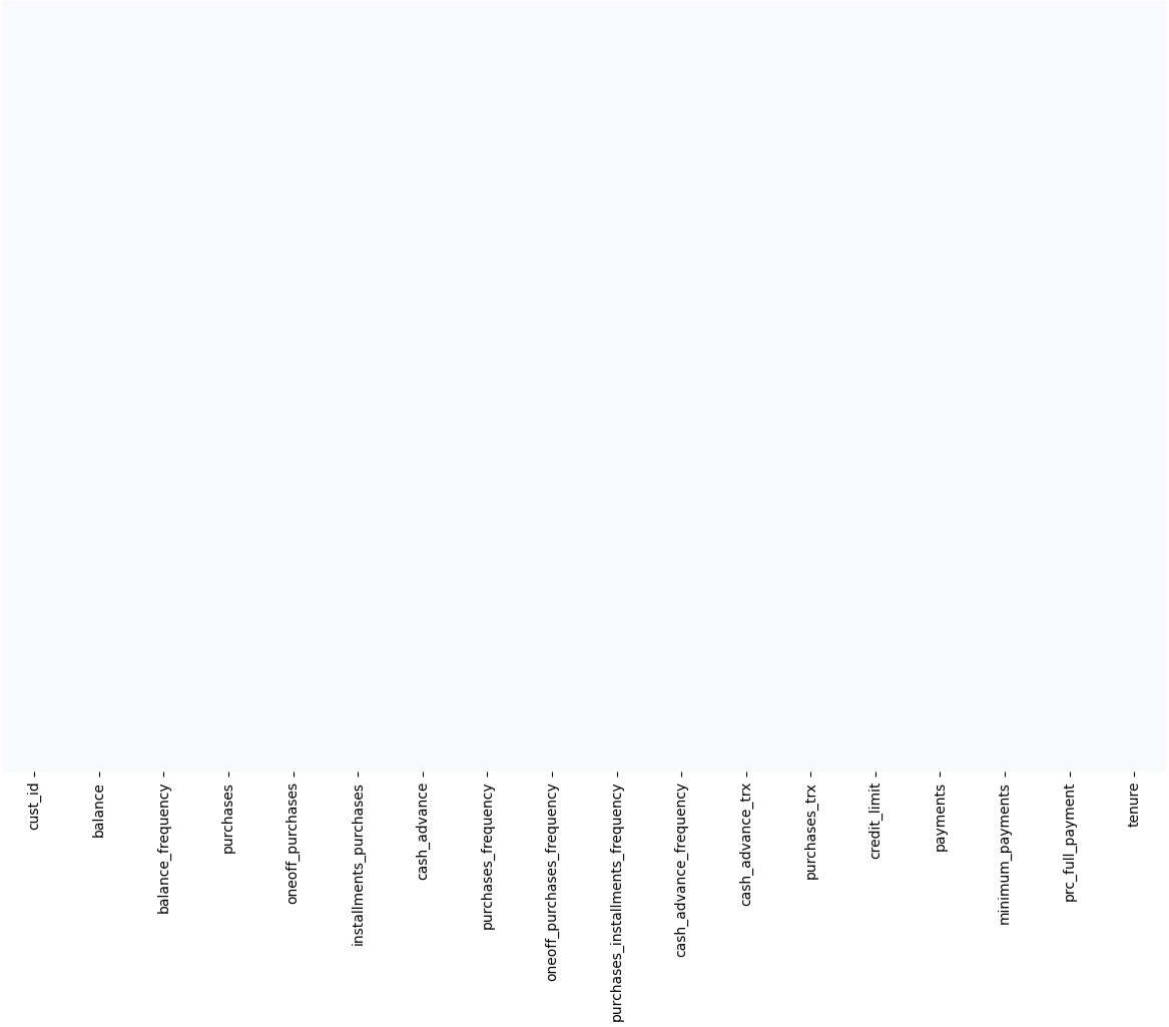
Handling Missing Value using Iterative Imputer

In [35]:
```
# import library

from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
```

In [36]:
```
# filling missing value with iterative imputer
feat_cols = [col for col in df_wo_id.columns]

itr_imputer = IterativeImputer(initial_strategy='median',
                               min_value=0, random_state=2022)

df[feat_cols] = itr_imputer.fit_transform(df[feat_cols])
```

In [37]:
```
# plot to check the missing value
plt.figure(figsize = (15,10))
sns.heatmap(df.isnull(), cmap='Blues', cbar=False, yticklabels=False, xticklabels=
```

Out[37]: <Axes: >



In [38]:
```
# checking the missing value
df.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   float64
 12  purchases_trx                     8950 non-null   float64
 13  credit_limit                      8950 non-null   float64
 14  payments                          8950 non-null   float64
 15  minimum_payments                  8950 non-null   float64
 16  prc_full_payment                  8950 non-null   float64
 17  tenure                            8950 non-null   float64
dtypes: float64(17), object(1)
memory usage: 1.2+ MB
```

In [39]:
```
# creating new variable for modeling
df_model = df.copy()
```

In [40]:
```
# droppping unused feature
df_model.drop(['cust_id', 'balance', 'purchases',
           'cash_advance', 'cash_advance_trx',
           'purchases_trx', 'credit_limit',
           'payments', 'minimum_payments', 'tenure'],
          axis=1, inplace=True, errors='ignore')
df_model.columns
```

Out[40]:
```
Index(['balance_frequency', 'oneoff_purchases', 'installments_purchases',
       'purchases_frequency', 'oneoff_purchases_frequency',
       'purchases_installments_frequency', 'cash_advance_frequency',
       'prc_full_payment'],
      dtype='object')
```

In [41]:
```
# change oneoff_purchase to oneoff_proportion
oneoff_proportion = df_wo_id['oneoff_purchases'] / df_wo_id['purchases']

# change installments_purchase to installments_proportion
installments_proportion = df_wo_id['installments_purchases'] / df_wo_id['purchases
```

In [42]:
```
# rename columns
df_model.rename(columns={'oneoff_purchases_frequency': 'oneoff_frequency',
                         'purchases_installments_frequency': 'installments_frequency',
                         'prc_full_payment': 'payments_proportion'},
                inplace=True, errors='ignore')
df_model.columns
```

Out[42]:
```
Index(['balance_frequency', 'oneoff_purchases', 'installments_purchases',
       'purchases_frequency', 'oneoff_frequency', 'installments_frequency',
       'cash_advance_frequency', 'payments_proportion'],
      dtype='object')
```

In [43]:
```
# check changed feature
df_model.head().T
```

Out[43]:

	0	1	2	3	4
balance_frequency	0.818182	0.909091	1.00	0.636364	1.000000
oneoff_purchases	0.000000	0.000000	773.17	1499.000000	16.000000
installments_purchases	95.400000	0.000000	0.00	0.000000	0.000000
purchases_frequency	0.166667	0.000000	1.00	0.083333	0.083333
oneoff_frequency	0.000000	0.000000	1.00	0.083333	0.083333
installments_frequency	0.083333	0.000000	0.00	0.000000	0.000000
cash_advance_frequency	0.000000	0.250000	0.00	0.083333	0.000000
payments_proportion	0.000000	0.222222	0.00	0.000000	0.000000

In [44]:
```
# creating function for detailed descriptive analytic
def summary_stats(df_model, n=4):
    # central tendency: mean, median
    mean = pd.DataFrame(df_model.apply(np.mean)).T
    median = pd.DataFrame(df_model.apply(np.median)).T
```

```
    # distribution: ,std, min, max, range, skew, kurtosis
    std = pd.DataFrame(df_model.apply(np.std)).T
    min_value = pd.DataFrame(df_model.apply(min)).T
    max_value = pd.DataFrame(df_model.apply(max)).T
    range_value = pd.DataFrame(df_model.apply(lambda x: x.max() - x.min())).T
    skewness = pd.DataFrame(df_model.apply(lambda x: x.skew())).T
    kurtosis = pd.DataFrame(df_model.apply(lambda x: x.kurtosis())).T

    # concatenates
    summary_stats = pd.concat([min_value, max_value, range_value, mean, median, st
    summary_stats.columns = ['attributes','min','max', 'range','mean','median', 's

    return round(summary_stats, n)
```

In [45]:
```
# descriptive analytic for modeling feature
summary_stats(df_model)
```

Out[45]:

	attributes	min	max	range	mean	median	std	skewnes
0	balance_frequency	0.0	1.00	1.00	0.8773	1.0000	0.2369	-2.023
1	oneoff_purchases	0.0	40761.25	40761.25	592.4374	38.0000	1659.7952	10.045
2	installments_purchases	0.0	22500.00	22500.00	411.0676	89.0000	904.2876	7.299
3	purchases_frequency	0.0	1.00	1.00	0.4904	0.5000	0.4013	0.060
4	oneoff_frequency	0.0	1.00	1.00	0.2025	0.0833	0.2983	1.535
5	installments_frequency	0.0	1.00	1.00	0.3644	0.1667	0.3974	0.509
6	cash_advance_frequency	0.0	1.50	1.50	0.1351	0.0000	0.2001	1.828
7	payments_proportion	0.0	1.00	1.00	0.1537	0.0000	0.2925	1.942

Check New Feature

In [46]:
```
# Knowing proportion anomalies (nan, inf, more_than_1)
def proportion_anomalies():
    nan_oneoff_proportion = np.isnan(oneoff_proportion).sum()
    nan_installments_proportion = np.isnan(installments_proportion).sum()
    inf_oneoff_proportion = np.isinf(oneoff_proportion).sum()
    inf_installments_proportion = np.isinf(installments_proportion).sum()
    more_than_1_oneoff_proportion = len(oneoff_proportion[oneoff_proportion > 1])
    more_than_1_installments_proportion = len(installments_proportion[installments

    proportion_anomalies = pd.DataFrame({'nan': [nan_oneoff_proportion, nan_instal
                                         'inf': [inf_oneoff_proportion, inf_install
                                         'more_than_1': [more_than_1_oneoff_proport
                                         index=['oneoff_proportion', 'installments_

    return proportion_anomalies
```

In [47]:
```
proportion_anomalies()
```

Out[47]:

	nan	inf	more_than_1
oneoff_proportion	2044	0	3
installments_proportion	2042	2	14

In [48]:
```
df_wo_id[np.isnan(installments_proportion)].loc[:, ['purchases', 'installments_pur
```

Out[48]:
```
purchases  installments_purchases
0.0        0.0                     2042
Name: count, dtype: int64
```

In [49]:
```
df_wo_id[np.isnan(oneoff_proportion)].iloc[:, 2:4].value_counts()
```

Out[49]:
```
purchases  oneoff_purchases
0.0        0.0                2044
Name: count, dtype: int64
```

In [50]:
```
# Fill NaN value with zero value
oneoff_proportion.fillna(0, inplace=True)
installments_proportion.fillna(0, inplace=True)

# check anomalies
proportion_anomalies()
```

Out[50]:

	nan	inf	more_than_1
oneoff_proportion	0	0	3
installments_proportion	0	2	14

In [51]:
```
df_wo_id.iloc[installments_proportion[installments_proportion == np.inf].index].il
```

Out[51]:

	purchases	oneoff_purchases	installments_purchases
4682	0.0	0.0	20.00
5737	0.0	0.0	66.95

In [52]:
```
inf_to_1 = installments_proportion[installments_proportion == np.inf].index
installments_proportion.iloc[inf_to_1] = 1

proportion_anomalies()
```

Out[52]:

	nan	inf	more_than_1
oneoff_proportion	0	0	3
installments_proportion	0	0	12

In [53]:
```
oneoff_more_than_1 = oneoff_proportion[oneoff_proportion > 1].index
oneoff_proportion.iloc[oneoff_more_than_1] = 1

installments_more_than_1 = installments_proportion[installments_proportion > 1].in
installments_proportion.iloc[installments_more_than_1] = 1

proportion_anomalies()
```

Out[53]:

	nan	inf	more_than_1
oneoff_proportion	0	0	0
installments_proportion	0	0	0

In [54]:
```
# change the oneoff_purchase with oneoff_proportion and installment_purchase with
df_model.oneoff_purchases = oneoff_proportion
df_model.installments_purchases = installments_proportion
```

In [55]:
```
df_model.rename(columns={'oneoff_purchases': 'oneoff_proportion',
                         'installments_purchases': 'installments_proportion'},
                inplace=True)
df_model.head()
```

Out[55]:

	balance_frequency	oneoff_proportion	installments_proportion	purchases_frequency	or
0	0.818182	0.0	1.0	0.166667	
1	0.909091	0.0	0.0	0.000000	
2	1.000000	1.0	0.0	1.000000	
3	0.636364	1.0	0.0	0.083333	
4	1.000000	1.0	0.0	0.083333	

In [56]:
```
# descriptive analytic for modeling feature
summary_stats(df_model)
```

Out[56]:

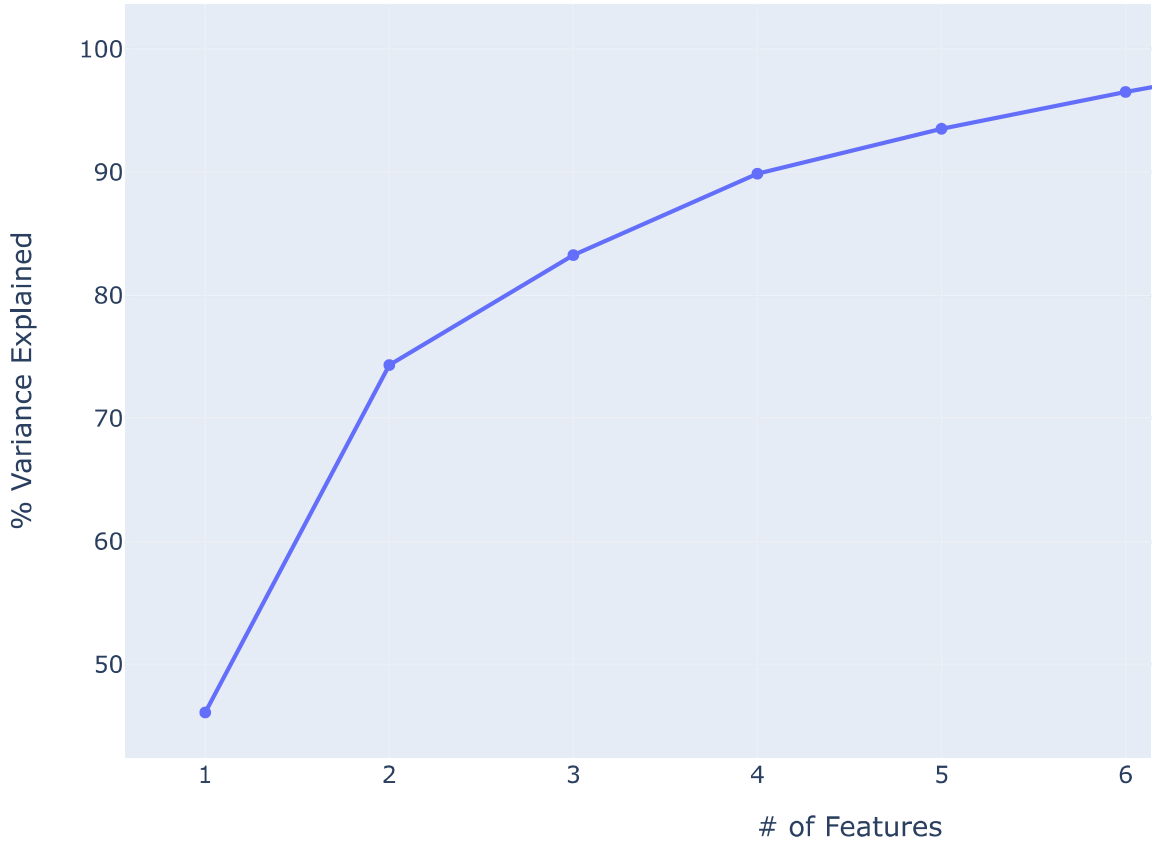
	attributes	min	max	range	mean	median	std	skewness	kurtosis
0	balance_frequency	0.0	1.0	1.0	0.8773	1.0000	0.2369	-2.0233	3.0924
1	oneoff_proportion	0.0	1.0	1.0	0.3795	0.1114	0.4256	0.4578	-1.5581
2	installments_proportion	0.0	1.0	1.0	0.3923	0.1851	0.4290	0.4509	-1.5629
3	purchases_frequency	0.0	1.0	1.0	0.4904	0.5000	0.4013	0.0602	-1.6386
4	oneoff_frequency	0.0	1.0	1.0	0.2025	0.0833	0.2983	1.5356	1.1618
5	installments_frequency	0.0	1.0	1.0	0.3644	0.1667	0.3974	0.5092	-1.3986
6	cash_advance_frequency	0.0	1.5	1.5	0.1351	0.0000	0.2001	1.8287	3.3347
7	payments_proportion	0.0	1.0	1.0	0.1537	0.0000	0.2925	1.9428	2.4324

In [57]:
```
# Model Init
number_of_cluster = np.arange(2,11) # Number of Clusters
np.random.seed(2022) # Set global randomseed for sklearn models
```

Principal Component Analysis

In [58]:
```
#running PCA with full components
pca_all = PCA(n_components=len(df_model.columns), random_state = 2022)
```

```
pca_all.fit(df_model)
variance = pca_all.explained_variance_ratio_
var = np.cumsum(variance)*100

#plot for information extracted
fig = px.line(x=np.arange(len(df_model.columns))+1, y=var, markers=True)
fig.update_xaxes(title_text='# of Features')
fig.update_yaxes(title_text='% Variance Explained')
fig.update_layout(width=900)
```



In [59]:
```
#total information extracted after pca
print('information extracted {} %'.format(np.sum(variance[:2])*100/np.sum(variance
```

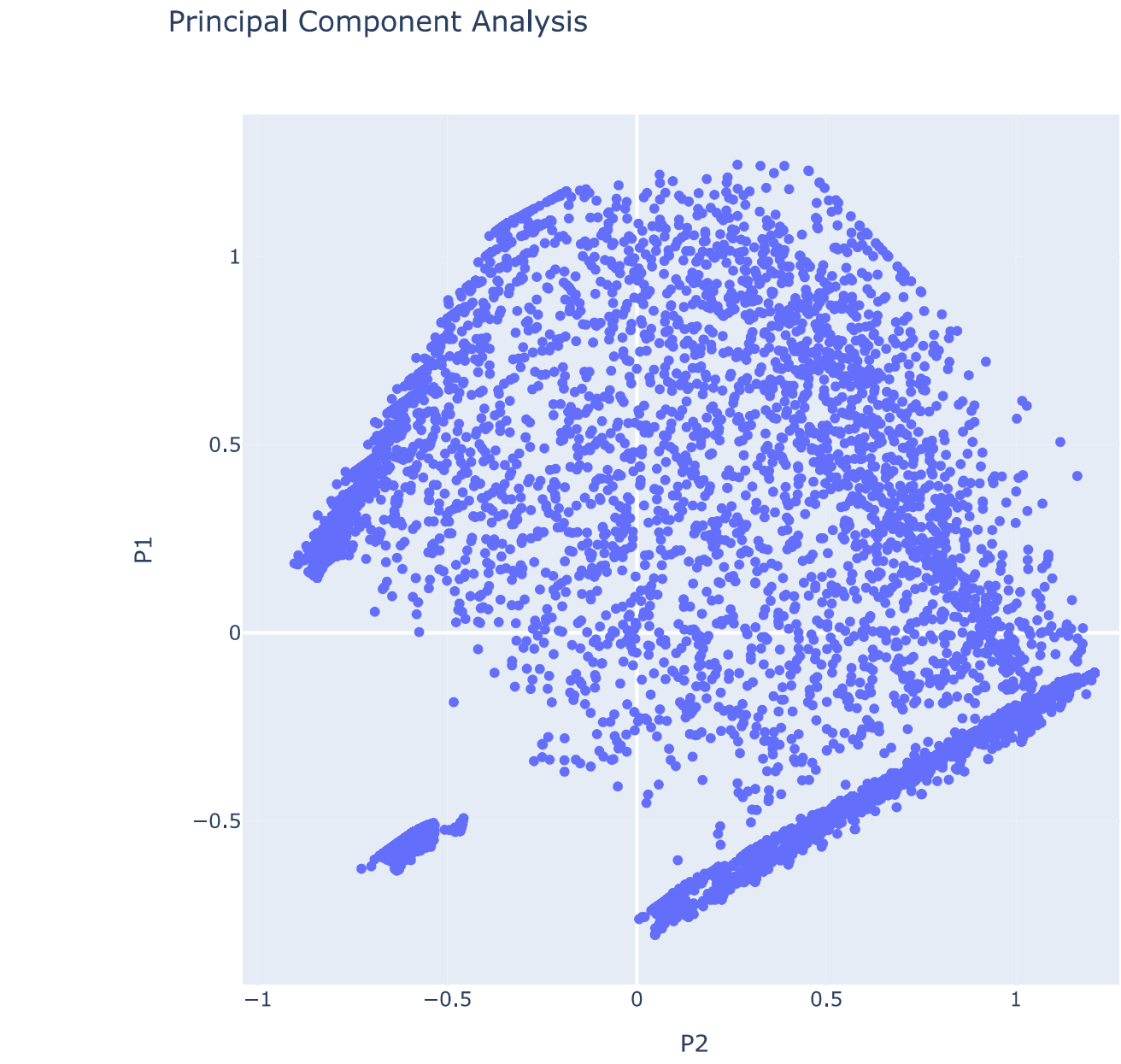
```
information extracted 74.33796026869159 %
```

In [60]:
```
# Reducing the dimensions of the data
pca_n = PCA(n_components = 2, random_state = 2022)
X_principal = pca_n.fit_transform(df_model)
X_principal = pd.DataFrame(X_principal)
X_principal.columns = ['P1', 'P2']

summary_stats(X_principal)
```

Out[60]:

	attributes	min	max	range	mean	median	std	skewness	kurtosis
0	P1	-0.9035	1.2076	2.111	0.0	-0.0803	0.6634	0.2954	-1.424
1	P2	-0.8021	1.2440	2.046	0.0	-0.1210	0.5193	0.4634	-0.921

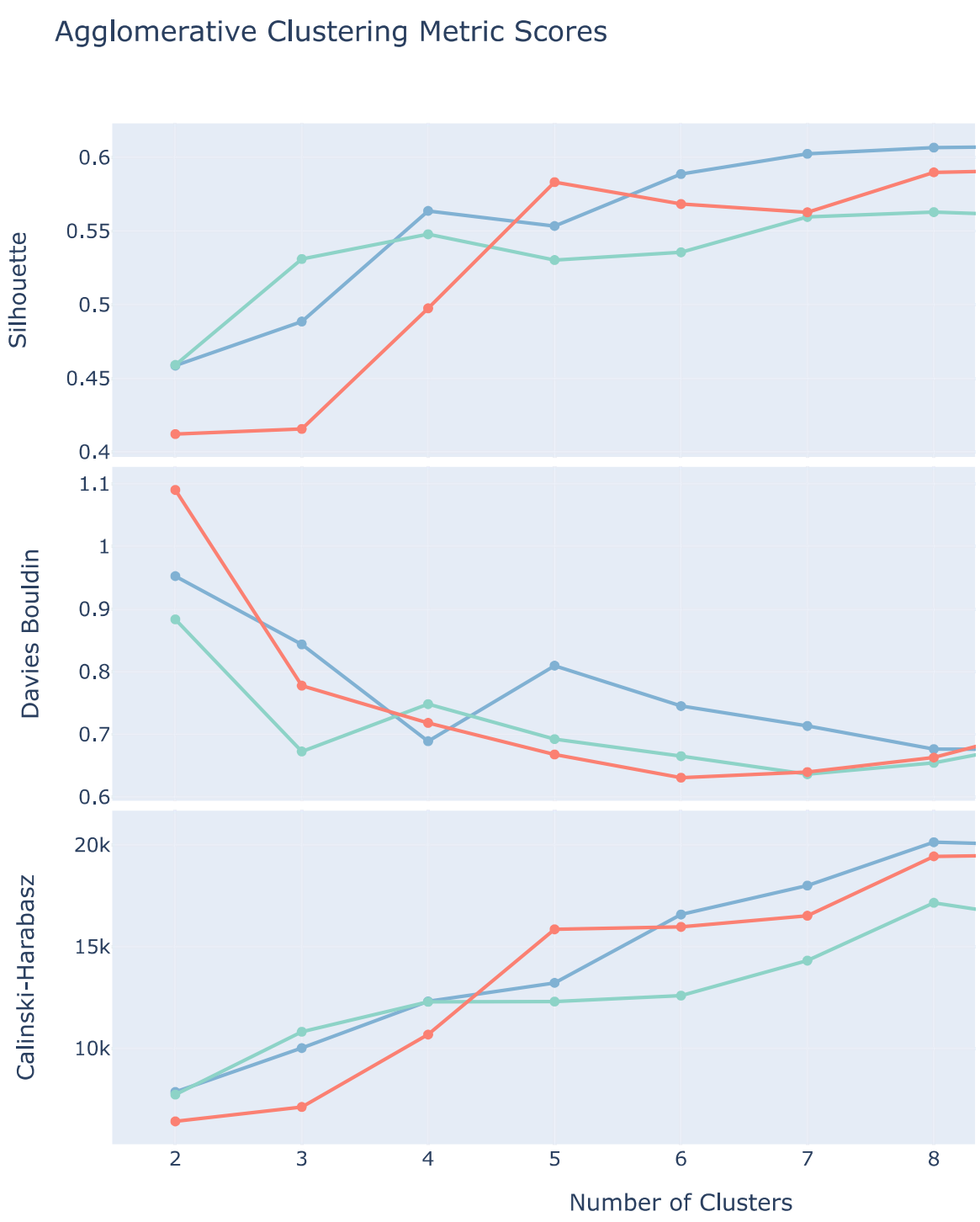
In [61]:
```
#plot for dimensions after pca
fig = go.Figure(go.Scatter(
    x=X_principal['P1'], y=X_principal['P2'], mode='markers'))
fig.update_xaxes(title_text='P2')
fig.update_yaxes(title_text='P1')
fig.update_layout(height=700, width=700,
                  title_text='Principal Component Analysis')
```



In [62]:
```
# Create Accumulator for matric
ward_s_scores = []
ward_db_scores = []
ward_calinski_scores = []
average_s_scores = []
average_db_scores = []
average_calinski_scores = []
complete_s_scores = []
complete_db_scores = []
complete_calinski_scores = []
```

```
agglo_metrics = {'ward': [ward_s_scores, ward_db_scores, ward_calinski_scores, 'rg
                 'average': [average_s_scores, average_db_scores, average_calinski
                 'complete': [complete_s_scores, complete_db_scores, complete_cali

for i, j in product(agglo_metrics, number_of_cluster):
    agglo = AgglomerativeClustering(linkage=i, n_clusters=j)
    agglo.fit(X_principal)
    agglo_metrics[i][0].append(silhouette_score(
        X_principal, agglo.labels_))
    agglo_metrics[i][1].append(davies_bouldin_score(
        X_principal, agglo.labels_))
    agglo_metrics[i][2].append(calinski_harabasz_score(
        X_principal, agglo.labels_))
```

In [63]:
```
#silhouette score between each method
fig = make_subplots(rows=3, cols=1, shared_xaxes=True, vertical_spacing=.01)

for i in agglo_metrics:
    for j, k in zip(range(1, 4), ['Silhouette', 'Davies Bouldin', 'Calinski-Haraba
        fig.append_trace(go.Scatter(x=list(number_of_cluster), y=agglo_metrics[i][
                                    legendgroup=i, line_color=agglo_metrics[i][-1]
                                    showlegend=False if j != 1 else True), row=j,
        fig.update_yaxes(title_text=k, row=j, col=1)

fig.update_xaxes(title_text='Number of Clusters', row=3)
fig.update_layout(height=800, width=900,
                  legend_title_text='Metrics',
                  title_text='Agglomerative Clustering Metric Scores')
```



In [64]:
```
#comparing result
compare_agg = pd.DataFrame({'Method' : ['Ward', 'Average', 'Complete'],
                  'n Cluster' :['7', '4', '7'],
                  'Silhouete Score' : [ward_s_scores[5], average_s_scores[2], comp
                  'Davies Score' : [ward_db_scores[5], average_db_scores[2], compl
                  'Calinski Score' : [ward_calinski_scores[5], average_calinski_sc
compare_agg
```

Out[64]:

	Method	n Cluster	Silhouete Score	Davies Score	Calinski Score
0	Ward	7	0.602435	0.713352	18002.677338
1	Average	4	0.547800	0.748187	12285.553170
2	Complete	7	0.562750	0.639673	16521.121399

In [65]:
```
#creating new dataframe for agglomerative clustering
df_dend = X_principal.copy()
```

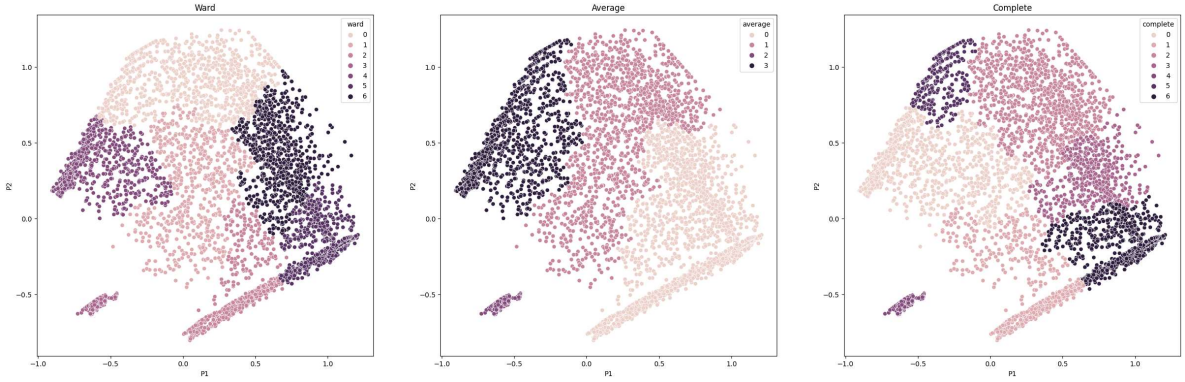
In [66]:
```
#adding new columns for ward
agg_ward = AgglomerativeClustering(n_clusters=7, linkage='ward')
df_dend['ward'] = agg_ward.fit_predict(X_principal)

#adding new columns for average
agg_average = AgglomerativeClustering(n_clusters=4, linkage='average')
df_dend['average'] = agg_average.fit_predict(X_principal)

#adding new columns for complete
agg_complete = AgglomerativeClustering(n_clusters=7, linkage='complete')
df_dend['complete'] = agg_complete.fit_predict(X_principal)

#showing dataframe
df_dend
```

Out[66]:

	P1	P2	ward	average	complete
0	0.120873	-0.686723	2	0	1
1	-0.574777	-0.531535	3	2	4
2	-0.340687	1.088999	0	3	5
3	-0.827379	0.203195	4	3	0
4	-0.805437	0.244772	4	3	0
...	...	...	...	...	...
8945	1.037319	-0.172448	5	0	6
8946	0.960066	-0.214203	5	0	6
8947	0.816581	-0.303066	5	0	6
8948	-0.565310	-0.530952	3	2	4
8949	-0.554672	0.726226	0	3	0

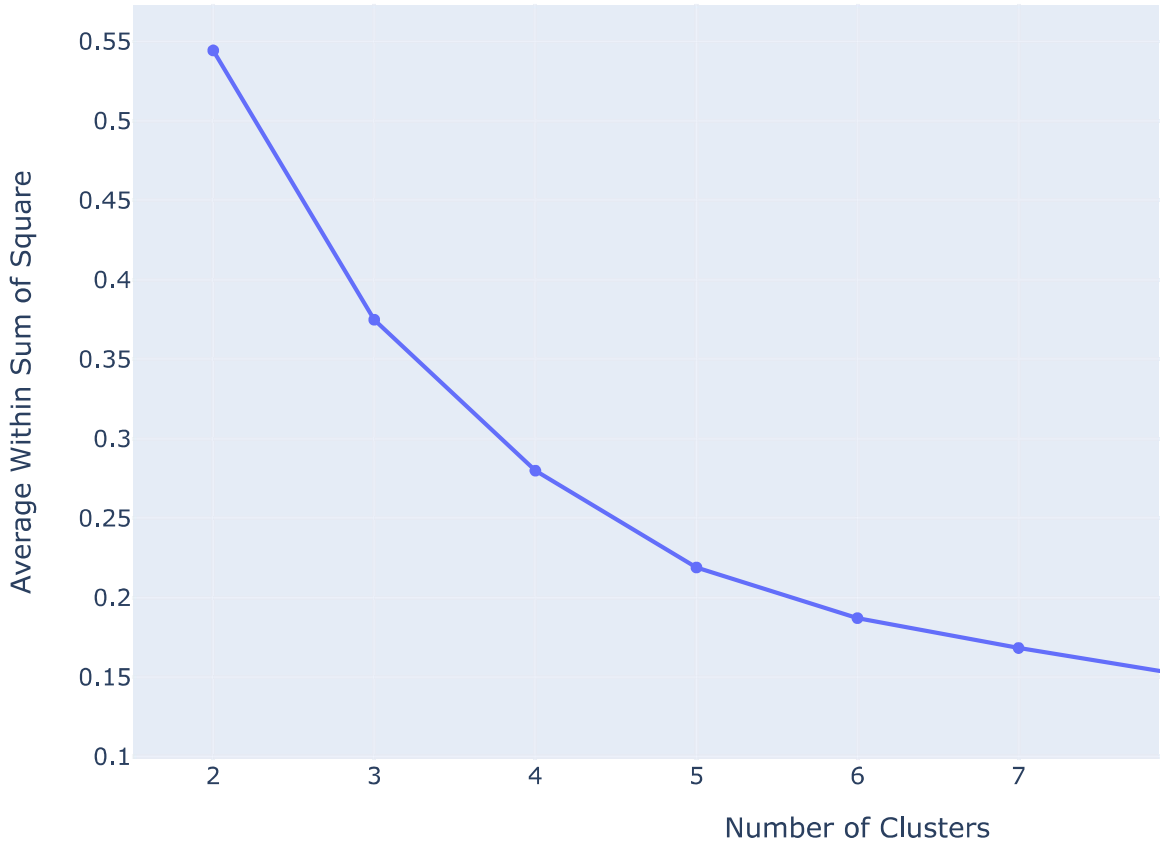
8950 rows × 5 columns

In [67]:
```
#creating visualization to compare the result from the best score for each agglome
num, hue = 0, ['ward', 'average', 'complete']
plt.figure(figsize=(30, 9))
for i in hue:
    num += 1
    plt.subplot(1, 3, num)
    sns.scatterplot(x='P1', y='P2', hue=i, data=df_dend)
    plt.title(i.title())
```



K-Means

In [68]:
```
#creating list for scoring
avg_withinSS = []

#create loop to run the algorithm
for i in number_of_cluster:
    kmeans = KMeans(n_clusters=i, random_state = 2022)
    kmeans.fit(X_principal)
    centroids = kmeans.cluster_centers_

    D_k = cdist(X_principal, centroids, 'euclidean')
    cIdx = np.argmin(D_k, axis=1)
    dist = np.min(D_k, axis=1)
    avg_withinSS.append(sum(dist)/X_principal.shape[0])

#creating plot for elbow method visualization
fig = px.line(x=number_of_cluster, y=avg_withinSS, markers=True)
fig.update_xaxes(title_text='Number of Clusters')
fig.update_yaxes(title_text='Average Within Sum of Square')
fig.update_layout(width=900)
```



In [69]:
```
# Create Accumulator for matric
kmeans_s_scores = []
kmeans_db_scores = []
kmeans_calinski_scores = []

#looping for modeling
for i in number_of_cluster:
    kmeans = KMeans(n_clusters=i, random_state = 2022)
    kmeans.fit(X_principal)
    kmeans_labels = kmeans.predict(X_principal)
    kmeans_s_scores.append(silhouette_score(
        X_principal, kmeans_labels, metric='euclidean'))
    kmeans_db_scores.append(davies_bouldin_score(X_principal, kmeans_labels))
    kmeans_calinski_scores.append(
        calinski_harabasz_score(X_principal, kmeans_labels))
```

In [70]:
```
#creating list for metrics
kmeans_std_s_scores = []
kmeans_std_db_scores = []
kmeans_std_calinski_scores = []

#parameter for checking the cluster stability
n_sets = 8
sets = np.array_split(X_principal, n_sets)

# calculates the scores and store in their respective list
for element in sets:
    kmeans = KMeans(n_clusters=4, random_state = 2022)
```

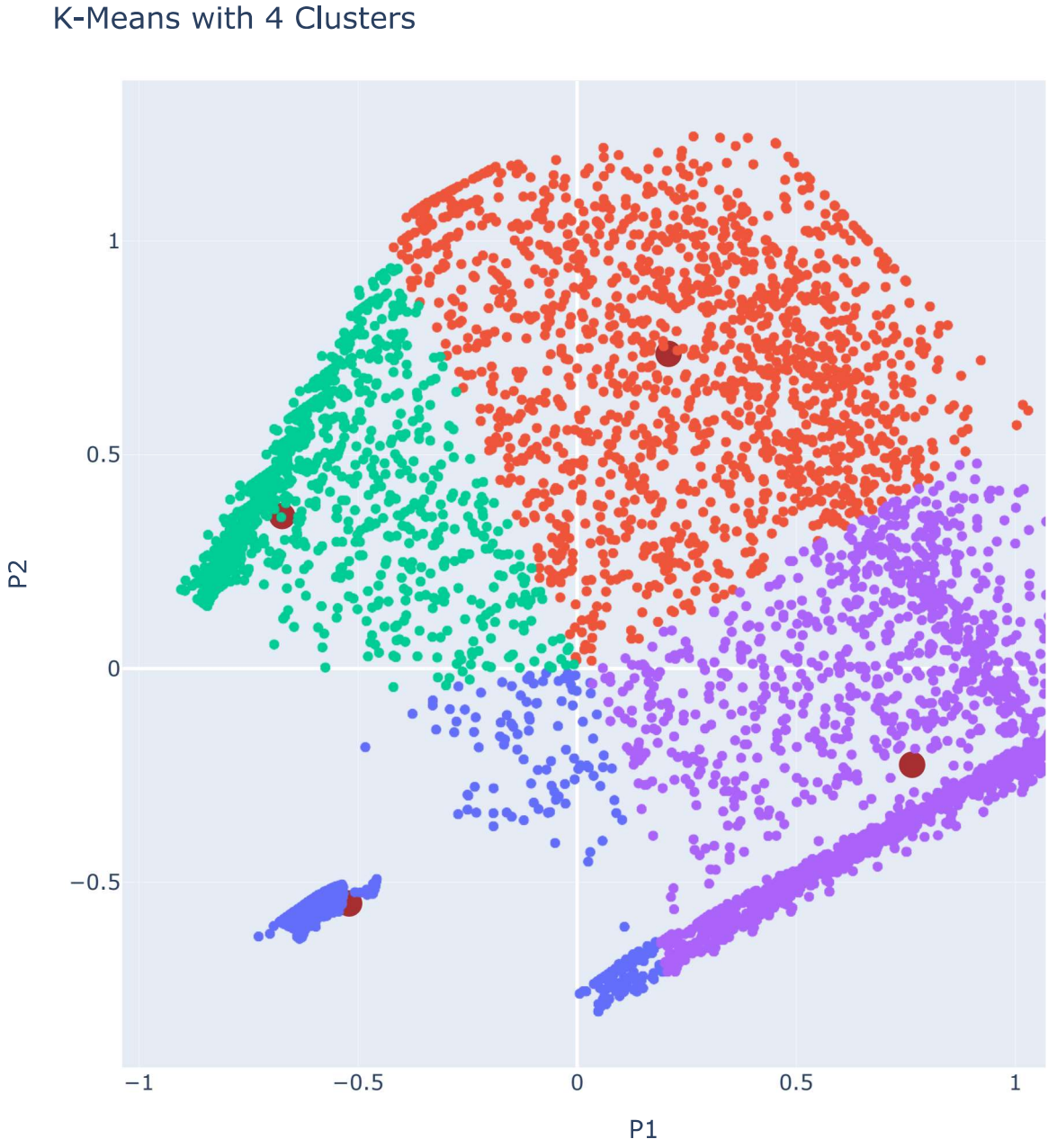
```
    kmeans.fit(element)
    kmeans_std_s_scores.append(silhouette_score(
        element, kmeans.labels_, metric='euclidean'))
    kmeans_std_db_scores.append(davies_bouldin_score(element, kmeans.labels_))
    kmeans_std_calinski_scores.append(
        calinski_harabasz_score(element, kmeans.labels_))

#check the cluster stability
print(f'Standard deviation Kmeans with {4} clusters:\nSilouette: {np.std(kmeans_st
```

```
Standard deviation Kmeans with 4 clusters:
Silouette: 0.01589437717278682
Davies Bouldin: 0.014988175775226476
Calinski Harabasz: 256.53120335065495
```

In [71]:
```
#choosing 4 cluster as the best number of cluster
kmeans = KMeans(n_clusters=4, random_state = 2022)
kmeans.fit(X_principal)
```

Out[71]:
```
KMeans(n_clusters=4, random_state=2022)
```

In [72]:
```
#plot to visualize the result
fig = px.scatter(X_principal, x='P1', y='P2',
                 color=list(map(lambda x: str(x), kmeans.labels_)))
fig.add_trace(go.Scatter(x=kmeans.cluster_centers_[:, 0], y=kmeans.cluster_centers
                         marker_color='rgba(152, 0, 0, .8)', marker_size=15, name=

fig.update_xaxes(title_text='P1')
fig.update_yaxes(title_text='P2')
fig.update_layout(height=700, width=800,
                  legend_title_text='Clusters',
                  title_text='K-Means with 4 Clusters')
```



K-Medoids

In [ ]:
```
#creating list for metrics
manhattan_avg_withinSS = []
euclidean_avg_withinSS = []
cosine_avg_withinSS = []

#creating list for metrics
kmedoids_avg_withinSS = {'cityblock': manhattan_avg_withinSS,
                         'euclidean': euclidean_avg_withinSS,
                         'cosine': cosine_avg_withinSS}

#Looping for running k-medoids
for i, j in product(kmedoids_avg_withinSS, number_of_cluster):
    kmedoids = KMedoids(metric=i, n_clusters=j, random_state = 2022)
    kmedoids.fit(X_principal)
```

```
    D_k = cdist(X_principal, kmedoids.cluster_centers_, i)
    cIdx = np.argmin(D_k, axis=1)
    dist = np.min(D_k, axis=1)
    kmedoids_avg_withinSS[i].append(sum(dist)/X_principal.shape[0])

#creating plot to visualize elbow method scoring
fig = go.Figure()
for i in kmedoids_avg_withinSS:
    fig.add_trace(go.Scatter(x=list(number_of_cluster),
                  y=kmedoids_avg_withinSS[i],
                  name=i if i != 'cityblock' else 'manhattan'))

fig.update_xaxes(title_text='Number of Clusters')
fig.update_yaxes(title_text='Average Within Sum of Square')
fig.update_layout(width=900, legend_title_text='Metrics')
```

In [ ]:
```
# Create Accumulator for matric
manhattan_s_scores = []
manhattan_db_scores = []
manhattan_calinski_scores = []
euclidean_s_scores = []
euclidean_db_scores = []
euclidean_calinski_scores = []
cosine_s_scores = []
cosine_db_scores = []
cosine_calinski_scores = []

kmedoids_metrics = {'manhattan': [manhattan_s_scores, manhattan_db_scores, manhatt
                    'euclidean': [euclidean_s_scores, euclidean_db_scores, euclide
                    'cosine': [cosine_s_scores, cosine_db_scores, cosine_calinski_

#looping to run k-medoids by each distance metrics
for i, j in product(kmedoids_metrics, number_of_cluster):
    kmedoids = KMedoids(metric=i, n_clusters=j, random_state = 2022)
    kmedoids.fit(X_principal)
    kmedoids_metrics[i][0].append(silhouette_score(
        X_principal, kmedoids.labels_, metric=i))
    kmedoids_metrics[i][1].append(davies_bouldin_score(
        X_principal, kmedoids.labels_))
    kmedoids_metrics[i][2].append(calinski_harabasz_score(
        X_principal, kmedoids.labels_))
```

In [ ]:
```
#creating plot to compare the result
fig = make_subplots(rows=3, cols=1, shared_xaxes=True, vertical_spacing=.01)

for i in kmedoids_metrics:
    for j, k in zip(range(1, 4), ['Silhouette', 'Davies Bouldin', 'Calinski-Haraba
        fig.append_trace(go.Scatter(x=list(number_of_cluster), y=kmedoids_metrics[
                                    legendgroup=i, line_color=kmedoids_metrics[i][
                                    showlegend=False if j != 1 else True), row=j,
        fig.update_yaxes(title_text=k, row=j, col=1)

fig.update_xaxes(title_text='Number of Clusters', row=3)
fig.update_layout(height=800, width=900,
                  legend_title_text='Metrics',
                  title_text='Metric Scores',
                  legend_traceorder='reversed')
```

In [ ]:
```
#using 4 as number of clusters
kmedoids_manhattan = KMedoids(metric="manhattan", n_clusters=4, random_state = 202
kmedoids_euclidean = KMedoids(metric="euclidean", n_clusters=4, random_state = 202
kmedoids_cosine = KMedoids(metric="cosine", n_clusters=4, random_state = 2022)

#running the algorithm
kmedoids_manhattan.fit(X_principal)
kmedoids_euclidean.fit(X_principal)
kmedoids_cosine.fit(X_principal);
```

In [ ]:
```
#creating plot to compare the result by each distance parameter
kmedoids_graph = {'Manhattan': kmedoids_manhattan,
                  'Euclidean': kmedoids_euclidean,
                  'Cosine': kmedoids_cosine}

plt.figure(figsize=(24, 8), constrained_layout=True)
for i, j in zip(kmedoids_graph, range(1,4)):
    plt.subplot(1,3,j)
    sns.scatterplot(x='P1', y='P2', data=X_principal,
                    hue= kmedoids_graph[i].labels_)
    sns.scatterplot(data = None,
                    x = kmedoids_graph[i].cluster_centers_[:,0],
                    y = kmedoids_graph[i].cluster_centers_[:,1])
    plt.title(i)
```

Compare Model

In [ ]:
```
df_compare = pd.DataFrame({'Model' : ['Agglomerative (Average)', 'K-Means', 'K-Med
              'N Cluster' : [4, 4, 4],
              'Silhouette Score' : [average_s_scores[2], kmeans_s_scores[2], cosin
              'Devies Bouldin Score' : [average_db_scores[2], kmeans_db_scores[2],
              'Calinski Harabasz Score' : [average_calinski_scores[2], kmeans_cali
             })
df_compare
```

Best Model

In [ ]:
```
#choosing best model for clustering
model_fix = KMedoids(metric="cosine", n_clusters=4, random_state = 2022)
model_fix.fit(X_principal)
model_fix_centers = model_fix.cluster_centers_
model_fix_labels = model_fix.predict(X_principal)
model_fix_centers
```

In [ ]:
```
#showing cluster
np.unique(model_fix_labels)
```

In [ ]:
```
#add the cluster to dataframe
df_cluster = pd.concat([df, pd.DataFrame({'cluster' :model_fix_labels})], axis = 1
df_cluster.head()
```

In [ ]:
```
#cluster population
df_cluster['cluster'].value_counts()
```

Visualization

In [ ]:
```
#running PCA
pca_n = PCA(n_components = 2, random_state = 2022)
X_principal = pca_n.fit_transform(df_model)
df_X_principal = pd.DataFrame(X_principal, columns = ['Principal Component 1', 'Pr
df_X_principal.head(2)
```

In [ ]:
```
#combine pca and cluster
finalDf = pd.concat([df_X_principal, pd.DataFrame({'cluster':model_fix_labels})],
finalDf.head()
```

In [ ]:
```
#plot for visualization
fig = go.Figure()

def cluster(cluster, color):
    fig.add_trace(go.Scatter(
        x=finalDf[finalDf.cluster == cluster]['Principal Component 1'],
        y=finalDf[finalDf.cluster == cluster]['Principal Component 2'],
        mode='markers',
        name=f'Cluster {cluster}',
        marker_color=color))

for C in list(finalDf.cluster.unique()):
    if C == 0:
        cluster(0, 'red')
    elif C == 1:
        cluster(1, 'green')
    elif C == 2:
        cluster(2, 'blue')
    elif C == 3:
        cluster(3, 'purple')

fig.add_trace(go.Scatter(x=model_fix_centers[:, 0], y=model_fix_centers[:, 1], mod
                         marker_color='rgba(152, 0, 0, .8)', marker_size=15, name=

fig.update_xaxes(title_text='Principal Component 1')
fig.update_yaxes(title_text='Principal Component 2')
fig.update_layout(height=900, width=1100,
                  legend_title_text='Clusters',
                  title_text='K-Medoids with 4 Clusters')
```

Cluster Analysis

In [ ]:
```
#creating new df for cluster analysis
df_cluster = pd.concat([df_cluster, df_model[['oneoff_proportion',
                 'installments_proportion', 'payments_proportion']]], axis=1)
```

In [ ]:
```
#separating the dataframe by cluster
cluster_0 = df_cluster[df_cluster['cluster'] == 0]
cluster_1 = df_cluster[df_cluster['cluster'] == 1]
cluster_2 = df_cluster[df_cluster['cluster'] == 2]
cluster_3 = df_cluster[df_cluster['cluster'] == 3]
```

In [ ]:
```
#summary for each cluster
need_columns = ['balance', 'purchases', 'cash_advance', 'credit_limit', 'payments'
d={}
for i in need_columns:
    d[i] = pd.DataFrame(df_cluster.groupby('cluster', axis = 0)[i].describe()[['me

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

In [ ]:
```
#plot comparison balance from each cluster
plt.figure(figsize=(21,5))
plt.subplot(1,4,1)
sns.histplot(cluster_0['balance'], color = 'red')
plt.title('cluster 0', size = 16)
plt.subplot(1,4,2)
sns.histplot(cluster_1['balance'], color='green' )
plt.title('cluster 1', size = 16)
plt.subplot(1,4,3)
sns.histplot(cluster_2['balance'], color='blue')
plt.title('cluster 2', size = 16)
plt.subplot(1,4,4)
sns.histplot(cluster_3['balance'], color='purple')
plt.title('cluster 3', size = 16)
plt.show()
```

In [ ]:
```
#plot comparison purchase from each cluster
plt.figure(figsize=(21,5))
plt.subplot(1,4,1)
sns.histplot(cluster_0['purchases'], color = 'red')
plt.title('cluster 0', size = 16)
plt.subplot(1,4,2)
sns.histplot(cluster_1['purchases'], color='green')
plt.title('cluster 1', size = 16)
plt.subplot(1,4,3)
sns.histplot(cluster_2['purchases'], color='blue')
plt.title('cluster 2', size = 16)
plt.subplot(1,4,4)
sns.histplot(cluster_3['purchases'], color='purple')
plt.title('cluster 3', size = 16)
plt.show()
```

In [ ]:
```
#Check Detail Purchases
plt.figure(figsize=(21,8))
ax = pd.DataFrame({'Cluster' : ['Cluster 0', 'Cluster 0', 'Cluster 1', 'Cluster 1'
                   'Purchases' : [cluster_0['oneoff_purchases'].sum(),
                                  cluster_0['installments_purchases'].sum(),
                                  cluster_1['oneoff_purchases'].sum(),
                                  cluster_1['installments_purchases'].sum(),
                                  cluster_2['oneoff_purchases'].sum(),
                                  cluster_2['installments_purchases'].sum(),
                                  cluster_3['oneoff_purchases'].sum(),
                                  cluster_3['installments_purchases'].sum()],
                   'Purchases Method' : ['one off', 'installments', 'one off', 'i
sns.barplot(x='Cluster', y='Purchases', data=ax, hue = 'Purchases Method' )
plt.ylabel('Total Purchases (Million)')
plt.xlabel('Cluster')
plt.title('Purchases Persentage', size = 20)
```

In [ ]:
```
#plot comparison cash advance from each cluster
plt.figure(figsize=(21,5))
plt.subplot(1,4,1)
```

```
sns.histplot(cluster_0['cash_advance'], color = 'red')
plt.title('cluster 0', size = 16)
plt.subplot(1,4,2)
sns.histplot(cluster_1['cash_advance'], color='green')
plt.title('cluster 1', size = 16)
plt.subplot(1,4,3)
sns.histplot(cluster_2['cash_advance'], color='blue')
plt.title('cluster 2', size = 16)
plt.subplot(1,4,4)
sns.histplot(cluster_3['cash_advance'], color='purple')
plt.title('cluster 3', size = 16)
plt.show()
```

In [ ]:
```
#plot comparison credit limit from each cluster
plt.figure(figsize=(21,5))
plt.subplot(1,4,1)
sns.histplot(cluster_0['credit_limit'], color = 'red')
plt.title('cluster 0', size = 16)
plt.subplot(1,4,2)
sns.histplot(cluster_1['credit_limit'], color='green')
plt.title('cluster 1', size = 16)
plt.subplot(1,4,3)
sns.histplot(cluster_2['credit_limit'], color='blue')
plt.title('cluster 2', size = 16)
plt.subplot(1,4,4)
sns.histplot(cluster_3['credit_limit'], color='purple')
plt.title('cluster 3', size = 16)
plt.show()
```

In [ ]:
```
##plot comparison payments from each cluster
plt.figure(figsize=(21,5))
plt.subplot(1,4,1)
sns.histplot(cluster_0['payments'], color = 'red')
plt.title('cluster 0', size = 16)
plt.subplot(1,4,2)
sns.histplot(cluster_1['payments'], color='green')
plt.title('cluster 1', size = 16)
plt.subplot(1,4,3)
sns.histplot(cluster_2['payments'], color='blue')
plt.title('cluster 2', size = 16)
plt.subplot(1,4,4)
sns.histplot(cluster_3['payments'], color='purple')
plt.title('cluster 3', size = 16)
plt.show()
```

In [ ]:
```
#plot comparison purchase from each cluster
plt.figure(figsize=(10,7))
sns.scatterplot(data = df_cluster, x = 'purchases', y = 'balance', hue = 'cluster'
plt.show()
plt.figure(figsize=(25,10))
plt.subplot(2,4,1)
sns.scatterplot(cluster_0['purchases'], cluster_0['balance'], color='red')
plt.title('cluster 0', size = 16)
plt.subplot(2,4,2)
sns.scatterplot(cluster_1['purchases'], cluster_1['balance'], color='green')
plt.title('cluster 1', size = 16)
plt.subplot(2,4,3)
sns.scatterplot(cluster_2['purchases'], cluster_2['balance'], color='blue')
plt.title('cluster 2', size = 16)
plt.subplot(2,4,4)
```

```
sns.scatterplot(cluster_3['purchases'], cluster_3['balance'], color='purple')
plt.title('cluster 3', size = 16)
plt.show()
```

In [ ]:
```
#plot comparison one off purchase from each cluster
plt.figure(figsize=(21,5))
plt.subplot(1,4,1)
sns.histplot(cluster_0['oneoff_purchases_frequency'], color = 'red')
plt.title('cluster 0', size = 16)
plt.subplot(1,4,2)
sns.histplot(cluster_1['oneoff_purchases_frequency'], color='green')
plt.title('cluster 1', size = 16)
plt.subplot(1,4,3)
sns.histplot(cluster_2['oneoff_purchases_frequency'], color='blue')
plt.title('cluster 2', size = 16)
plt.subplot(1,4,4)
sns.histplot(cluster_3['oneoff_purchases_frequency'], color='purple')
plt.title('cluster 3', size = 16)
plt.show()
```

In [ ]:
```
#plot comparison installment purchase from each cluster
plt.figure(figsize=(21,5))
plt.subplot(1,4,1)
sns.histplot(cluster_0['purchases_installments_frequency'], color = 'red')
plt.title('cluster 0', size = 16)
plt.subplot(1,4,2)
sns.histplot(cluster_1['purchases_installments_frequency'], color='green')
plt.title('cluster 1', size = 16)
plt.subplot(1,4,3)
sns.histplot(cluster_2['purchases_installments_frequency'], color='blue')
plt.title('cluster 2', size = 16)
plt.subplot(1,4,4)
sns.histplot(cluster_3['purchases_installments_frequency'], color='purple')
plt.title('cluster 3', size = 16)
plt.show()
```

In [ ]:
```
#plot comparison cash advance frequency from each cluster
plt.figure(figsize=(21,5))
plt.subplot(1,4,1)
sns.histplot(cluster_0['cash_advance_frequency'], color = 'red')
plt.title('cluster 0', size = 16)
plt.subplot(1,4,2)
sns.histplot(cluster_1['cash_advance_frequency'], color='green')
plt.title('cluster 1', size = 16)
plt.subplot(1,4,3)
sns.histplot(cluster_2['cash_advance_frequency'], color='blue')
plt.title('cluster 2', size = 16)
plt.subplot(1,4,4)
sns.histplot(cluster_3['cash_advance_frequency'], color='purple')
plt.title('cluster 3', size = 16)
plt.show()
```

In [ ]:
```
#3d plot for oneoff purchase frequency, purchase installment frequency, and cash a
fig = go.Figure()

def cluster3d(cluster, color):
    fig.add_trace(go.Scatter3d(
        x=df_cluster[df_cluster.cluster == cluster]['oneoff_purchases_frequency'],
        y=df_cluster[df_cluster.cluster == cluster]['purchases_installments_freque
        z=df_cluster[df_cluster.cluster == cluster]['cash_advance_frequency'],
```

```
        mode='markers',
        name=f'Cluster {cluster}',
        marker_color=color,
        marker_size=5))


for C in list(df_cluster.cluster.unique()):
    if C == 0:
        cluster3d(0, 'red')
    elif C == 1:
        cluster3d(1, 'green')
    elif C == 2:
        cluster3d(2, 'blue')
    elif C == 3:
        cluster3d(3, 'purple')

fig.update_layout(height=900, width=1100,
                  scene=dict(
                      xaxis_title='oneoff_purchases_frequency',
                      yaxis_title='purchases_installments_frequency',
                      zaxis_title='cash_advance_frequency'),
                  title_text='A 3D Projection Of Data In The Reduced Dimension')
```

In [ ]: