In [1]:
```
import gc
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:
```
data = pd.read_csv('Data_problem _1.csv')
```

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

Out[4]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PUR
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 [5]:
```
data.tail()
```

Out[5]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_P
8945	C19186	28.493517	1.000000	291.12	0.00	
8946	C19187	19.183215	1.000000	300.00	0.00	
8947	C19188	23.398673	0.833333	144.40	0.00	
8948	C19189	13.457564	0.833333	0.00	0.00	
8949	C19190	372.708075	0.666667	1093.25	1093.25	

In [6]:
```
data.isna().sum()
```

Out[6]:
```
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
```

In [7]:
```
data.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
```

In [8]:
```
data.describe()
```

Out[8]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHA
count	8950.000000	8950.000000	8950.000000	8950.000000	8950.00(
mean	1564.474828	0.877271	1003.204834	592.437371	411.06
std	2081.531879	0.236904	2136.634782	1659.887917	904.33
min	0.000000	0.000000	0.000000	0.000000	0.00(
25%	128.281915	0.888889	39.635000	0.000000	0.00(
50%	873.385231	1.000000	361.280000	38.000000	89.00(
75%	2054.140036	1.000000	1110.130000	577.405000	468.63
max	19043.138560	1.000000	49039.570000	40761.250000	22500.00(

In [9]:
```
data.MINIMUM_PAYMENTS.head()
```

Out[9]:
```
0     139.509787
1    1072.340217
2     627.284787
3            NaN
4     244.791237
Name: MINIMUM_PAYMENTS, dtype: float64
```

In [10]:
```
# replaced the nan values with '0'
data.MINIMUM_PAYMENTS.fillna(0,inplace = True)
```

In [11]:
```
data.MINIMUM_PAYMENTS.isna().sum()
```

Out[11]:
```
0
```

In [12]:
```
data.CREDIT_LIMIT.fillna(0,inplace = True)
```

In [13]:
```
# check the payment with minimum payment.
x = list(zip(data.MINIMUM_PAYMENTS,data.PAYMENTS))
for val in x:
    if val[1] == 0:
        print(val)
```

```
(0.0, 0.0)
(0.0, 0.0)
(0.0, 0.0)
(0.0, 0.0)
(0.0, 0.0)
(0.0, 0.0)
(0.0, 0.0)
(0.0, 0.0)
(0.0, 0.0)
(0.0, 0.0)
(0.0, 0.0)
(0.0, 0.0)
(0.0, 0.0)
(0.0, 0.0)
(0.0, 0.0)
(0.0, 0.0)
(0.0, 0.0)
(0.0, 0.0)
(0.0, 0.0)
(0.0, 0.0)
(0.0, 0.0)
```

From above output we can see that the value replaced with 0 in MINIMUM_PAYMENTS whose PAYMENT is also 0.

In [14]:
```
# check for values in  MINIMUM_PAYMENTS is not match with the PAYMENT.
x = list(zip(data.MINIMUM_PAYMENTS,data.PAYMENTS))
for val in x:
    if val[1]==0:
        if val[0] != val[1]:
            print('Payment not Match with minimum payment')
            print(val)
```

In [15]:
```
data.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                      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   int64
dtypes: float64(14), int64(3), object(1)
memory usage: 1.2+ MB
```

In [16]:
```
data.sample(10)
```

Out[16]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_
359	C10371	1543.287931	1.000000	0.00	0.00	
5599	C15755	3128.613320	1.000000	3875.71	2244.32	
5229	C15376	1155.338824	0.909091	5744.03	4055.80	
8572	C18806	28.838076	1.000000	484.68	0.00	
6204	C16377	2522.938460	1.000000	394.48	221.20	
5765	C15926	408.724408	1.000000	89.39	89.39	
3955	C14066	816.765152	1.000000	7967.04	7839.22	
8106	C18326	641.317325	1.000000	1099.56	0.00	
2605	C12681	4805.164203	1.000000	0.00	0.00	
5547	C15703	3127.073601	1.000000	4468.44	4432.46	

In [17]:
```
# Monthly average purchase and cash advance amount
```

In [18]:
```
data['Monthly_average_purchase'] = data.PURCHASES / data.TENURE
```

In [19]:
```
data['cash_advance_amount'] = data.CASH_ADVANCE /  data.TENURE
```

In [20]:
```
# Purchases by type (one-off, installments)
```

In [21]:
```
def type_(data):
    if data.ONEOFF_PURCHASES == 0 and data.INSTALLMENTS_PURCHASES == 0 :
        return 'None'
    elif data.ONEOFF_PURCHASES > 0 and data.INSTALLMENTS_PURCHASES > 0 :
        return 'Both'
    elif data.ONEOFF_PURCHASES == 0 and data.INSTALLMENTS_PURCHASES > 0 :
        return 'installments'
    elif data.ONEOFF_PURCHASES > 0 and data.INSTALLMENTS_PURCHASES == 0 :
        return 'one-off'
```

In [22]:
```
data['purchase_type'] = data.apply(type_, axis = 1)
```

In [23]:
```
data.head(5)
```

Out[23]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PUR
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	

5 rows × 21 columns

In [24]:
```
data['purchase_type'].head(5)
```

Out[24]:
```
0    installments
1            None
2         one-off
3         one-off
4         one-off
Name: purchase_type, dtype: object
```

In [25]:
```
gc.collect()
```

Out[25]:
```
0
```

In [26]:
```
# Limit usage
data['limit_usage'] = data.BALANCE / data.CREDIT_LIMIT
```

In [27]:
```
data['limit_usage'].fillna(0,inplace=True)
```

In [28]:
```
data['limit_usage'].isnull().sum()
```

Out[28]:
```
0
```

In [29]:
```
data['pay/min_pay'] = data.PAYMENTS / data.MINIMUM_PAYMENTS
```

In [30]:
```
data['pay/min_pay'].isna().sum()
```

Out[30]:
```
240
```

In [31]:
```
data[data==np.inf] = np.nan
```

In [32]:
```
data['pay/min_pay'].fillna(0,inplace= True)
```

In [33]:
```
data.head()
```

Out[33]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PUR
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	

5 rows × 23 columns

In [34]:
```
data.drop('CUST_ID',axis = 1,inplace = True)
```

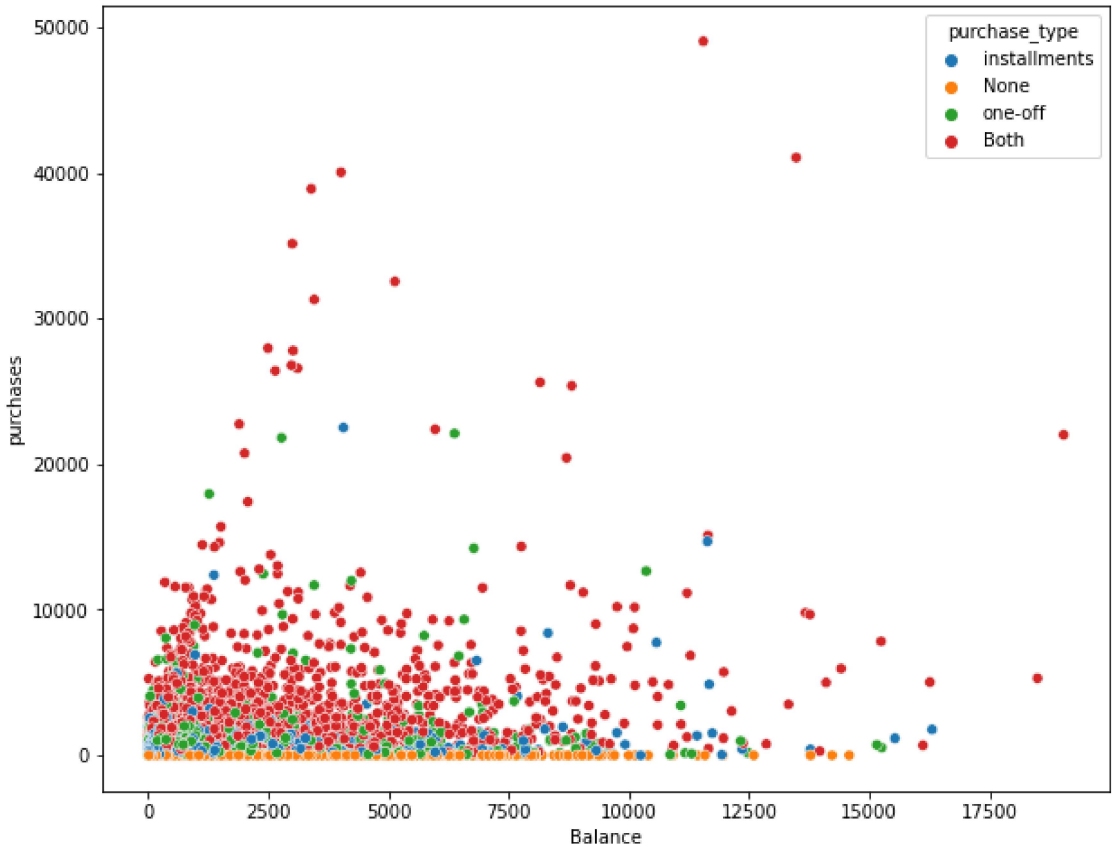
In [35]:
```
data.head()
```

Out[35]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES
0	40.900749	0.818182	95.40	0.00	95.4
1	3202.467416	0.909091	0.00	0.00	0.0
2	2495.148862	1.000000	773.17	773.17	0.0
3	1666.670542	0.636364	1499.00	1499.00	0.0
4	817.714335	1.000000	16.00	16.00	0.0

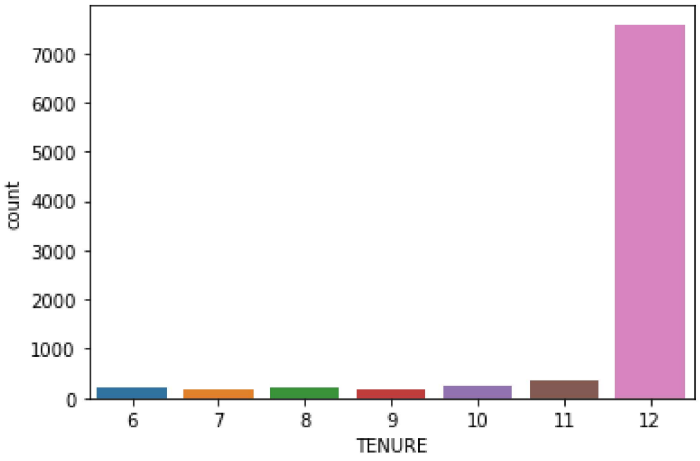
5 rows × 22 columns

In [36]:
```
plt.figure(figsize=(10,8))
sns.scatterplot(data=data, x="BALANCE", y="PURCHASES", hue="purchase_type")
plt.xlabel('Balance');
plt.ylabel('purchases');
```

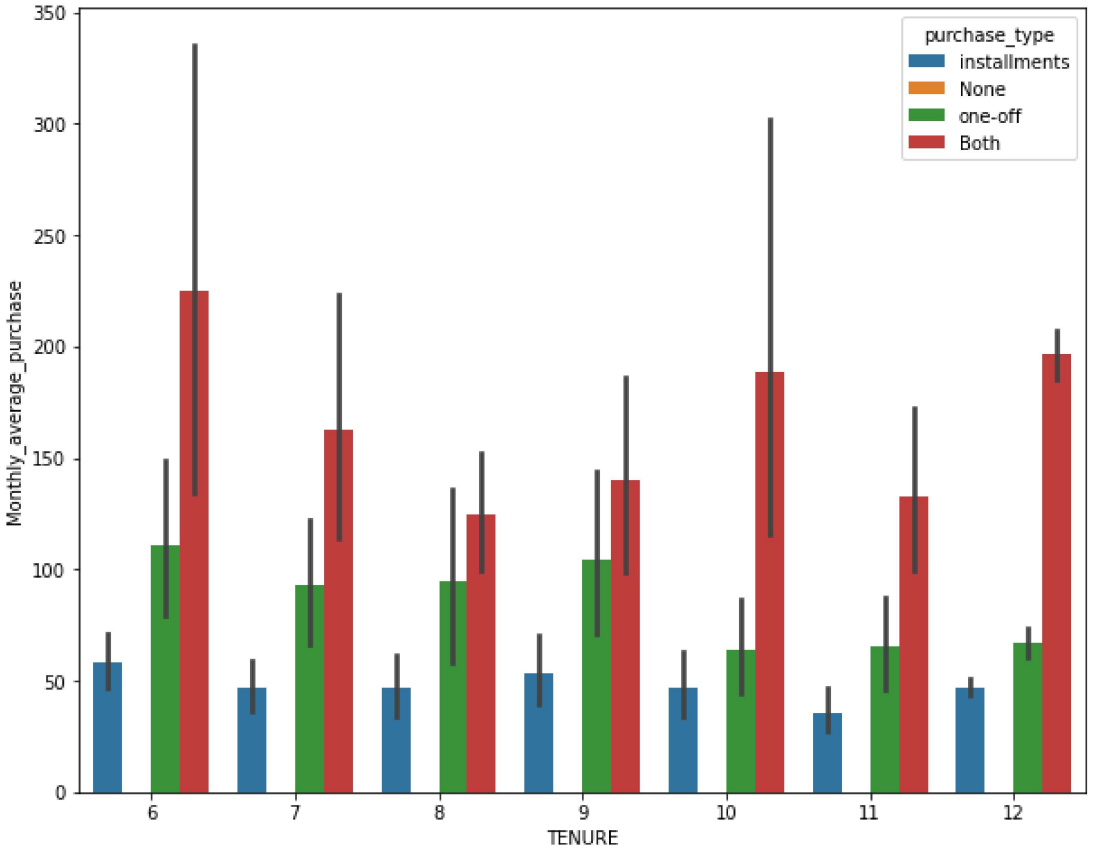


From above we can say that most of the pruchases are 'one-off'

In [37]:
```
sns.countplot(data.TENURE);
```

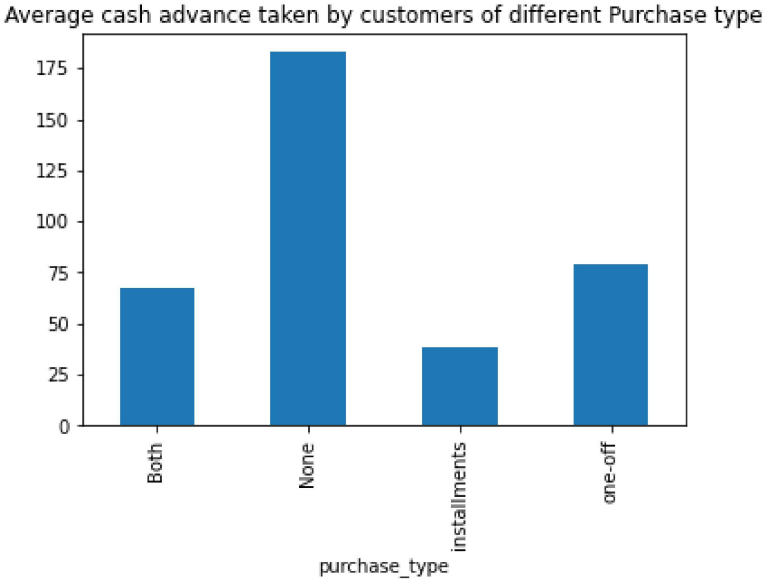


In [38]:
```
plt.figure(figsize=(10,8))
sns.barplot(data=data, x="TENURE", y="Monthly_average_purchase",hue = 'purchase_type');
```

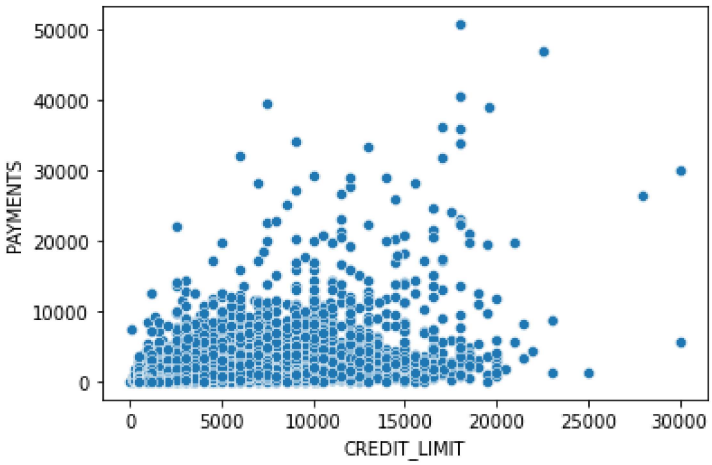


From above graph we can say that most of the persons doing the both installment & one-off purchase & less persons are go for the installment purchase.

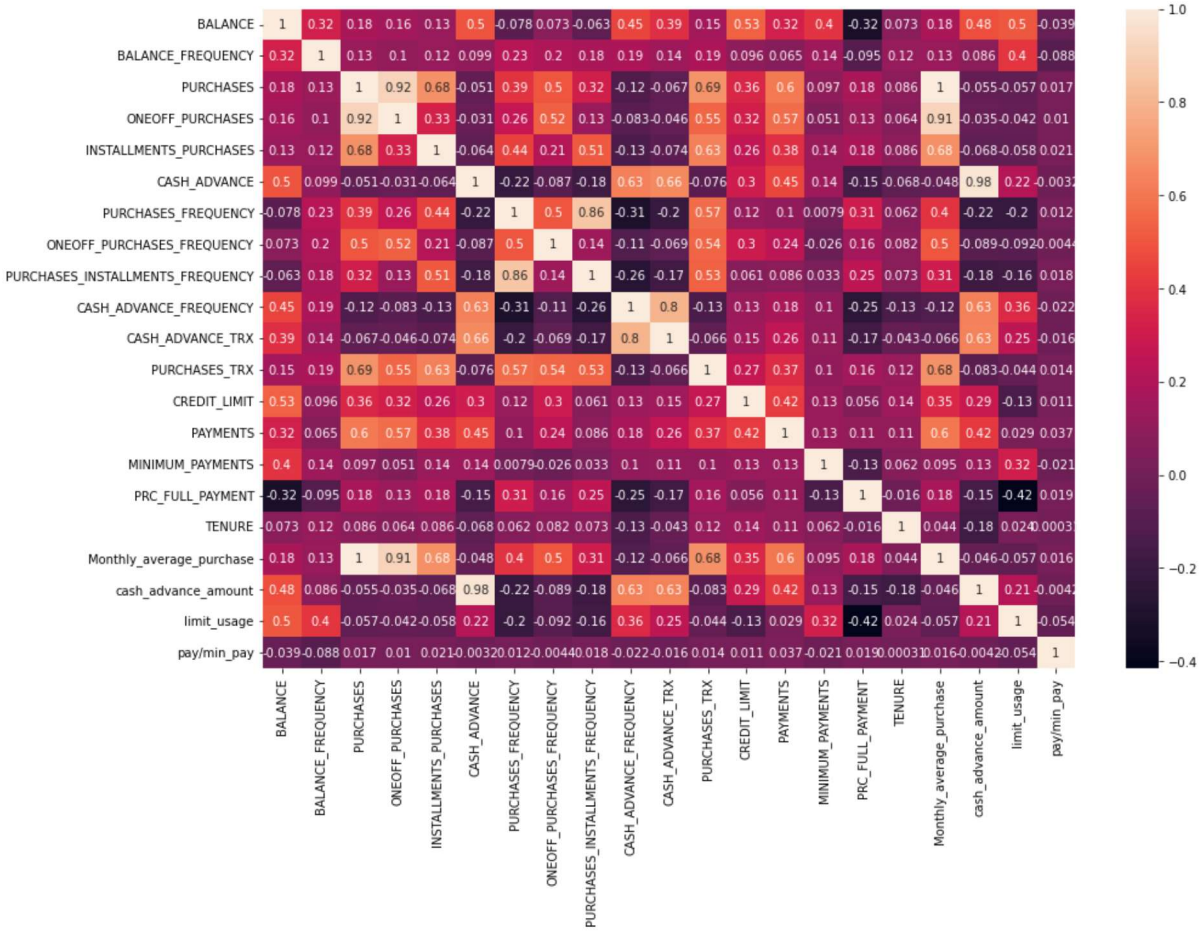
In [39]:
```
data.groupby('purchase_type').apply(lambda x: np.mean(x['cash_advance_amount'])).plot.ba

plt.title('Average cash advance taken by customers of different Purchase type');
```



In [40]:
```
sns.scatterplot(x=data["CREDIT_LIMIT"],y=data["PAYMENTS"]);
```



In [41]:
```
plt.figure(figsize=(15,10))
sns.heatmap(data.corr(),annot=True);
```



In [42]:
```
gc.collect()
```

Out[42]: 34024

In [43]:
```
# Handling the Categorical Data
```

In [44]:
```
df_cat = data.select_dtypes('object')
```

In [45]:
```
df_cat.columns
```

Out[45]:
```
Index(['purchase_type'], dtype='object')
```

In [46]:
```
from sklearn.preprocessing import LabelEncoder,MinMaxScaler,normalize
```

In [47]:
```
le = LabelEncoder()
data['purchase_type']=le.fit_transform(data.purchase_type)
```

In [48]:
```
le.classes_
```

Out[48]:
```
array(['Both', 'None', 'installments', 'one-off'], dtype=object)
```

In [49]:
```
data.head()
```

Out[49]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES
0	40.900749	0.818182	95.40	0.00	95.4
1	3202.467416	0.909091	0.00	0.00	0.0
2	2495.148862	1.000000	773.17	773.17	0.0
3	1666.670542	0.636364	1499.00	1499.00	0.0
4	817.714335	1.000000	16.00	16.00	0.0

5 rows × 22 columns

In [50]:
```
# Data Normalization
```

In [51]:
```
scaler = MinMaxScaler()
data[data.columns] = scaler.fit_transform(data[data.columns])
```

In [52]:
```
data.head()
```

Out[52]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	C
0	0.002148	0.818182	0.001945	0.000000	0.00424	
1	0.168169	0.909091	0.000000	0.000000	0.00000	
2	0.131026	1.000000	0.015766	0.018968	0.00000	
3	0.087521	0.636364	0.030567	0.036775	0.00000	
4	0.042940	1.000000	0.000326	0.000393	0.00000	

5 rows × 22 columns

In [53]:
```
gc.collect()
```

Out[53]:
```
0
```

In [54]:
```
from sklearn.decomposition import PCA
```

In [55]:
```
data.shape
```

Out[55]:
```
(8950, 22)
```

In [56]:
```
data.isna().sum()
```

Out[56]:
```
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                        0
PAYMENTS                            0
MINIMUM_PAYMENTS                    0
PRC_FULL_PAYMENT                    0
TENURE                              0
Monthly_average_purchase            0
cash_advance_amount                 0
purchase_type                       0
limit_usage                         1
pay/min_pay                         0
dtype: int64
```

In [57]:
```
data['limit_usage'].fillna(0,inplace=True)
```

In [58]:
```
var_ratio = {}
for  i in range(2,22):
    pca = PCA(n_components = i)
    df_pca = pca.fit(data)
    var_ratio[i] = sum(df_pca.explained_variance_ratio_)
```

In [59]:
```
var_ratio
```

Out[59]:
```
{2: 0.5943582791586859,
 3: 0.7086206713105055,
 4: 0.8062878913754342,
 5: 0.8689887780726891,
 6: 0.926058680640039,
 7: 0.9551751946001337,
 8: 0.9703852454260097,
 9: 0.9798863956910121,
 10: 0.9854659101807528,
 11: 0.990463791618813,
 12: 0.9935758604415539,
 13: 0.9953789134323677,
 14: 0.996462428538838,
 15: 0.9974859071167271,
 16: 0.9984187979397232,
 17: 0.9992728078823109,
 18: 0.9996369019712589,
 19: 0.9999459849160676,
 20: 0.9999896970961601,
 21: 0.9999999744213612}
```

In [60]:
```
# from above output 6 components giving the greater than 90% variance ratio, Hence n_com
```

In [61]:
```
pca = PCA(n_components = 6)
df_pca = pca.fit_transform(data)
```

In [62]:
```
df_pca.shape
```

Out[62]:
```
(8950, 6)
```

In [63]:
```
pc_df = pd.DataFrame(df_pca)
pc_df.columns = ['P1', 'P2','P3','P4','P5','P6']
pc_df.head(4)
```

Out[63]:

	P1	P2	P3	P4	P5	P6
0	-0.526425	0.061188	-0.094427	-0.060926	-0.072864	0.101752
1	-0.539443	-0.284750	-0.036165	0.201628	-0.191588	-0.097068
2	0.117733	0.433241	0.965672	-0.430280	0.102859	0.120548
3	-0.731507	0.311239	0.048312	-0.100654	-0.024749	0.204918

In [64]:
```
gc.collect()
```

Out[64]:
```
0
```

In [65]:
```
from sklearn.preprocessing import Normalizer

df = Normalizer().fit_transform(pc_df.values)
```

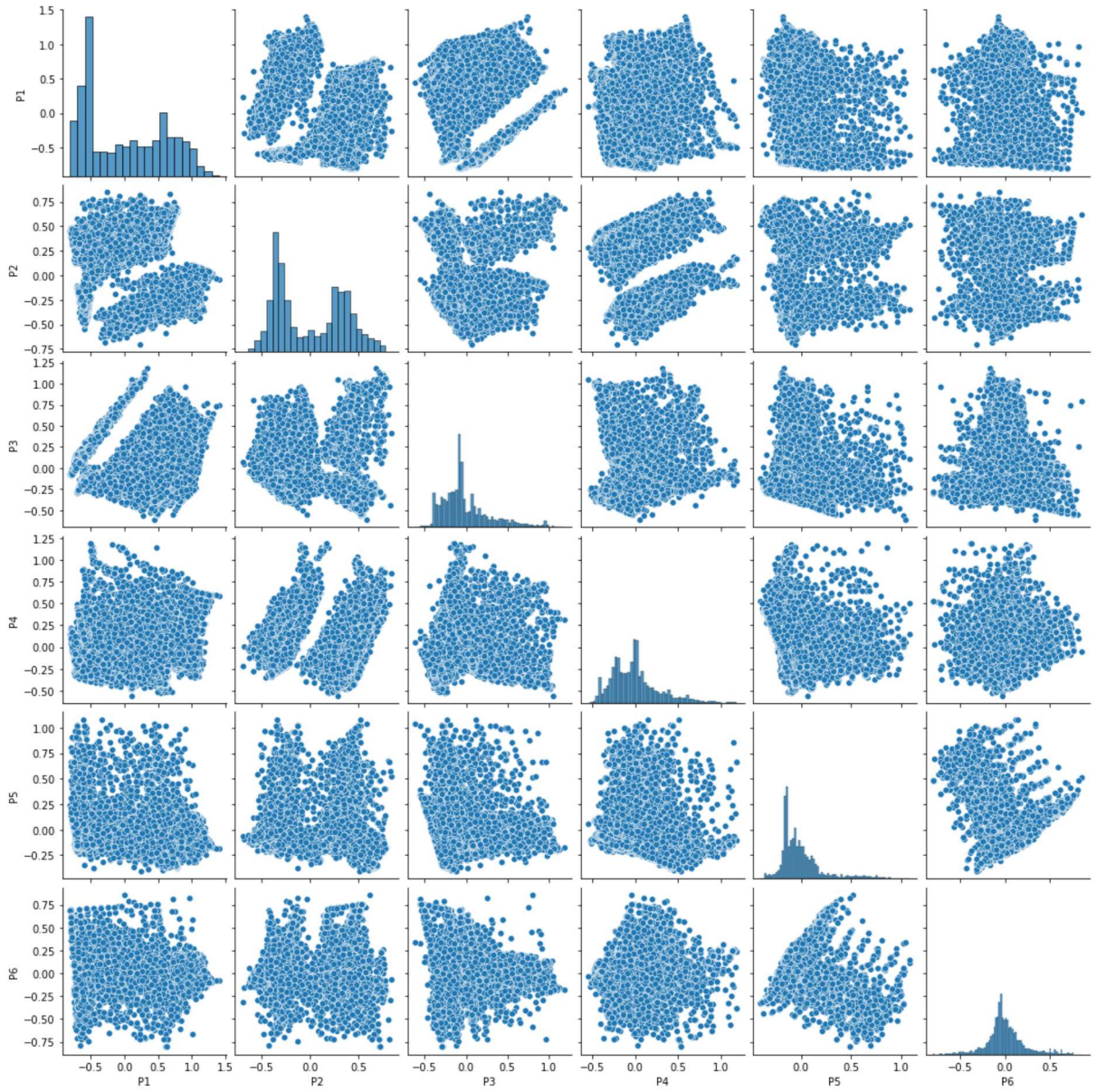
In [66]:
```
data_new = pd.DataFrame(df)
```

In [67]:
```
data_new.head()
```

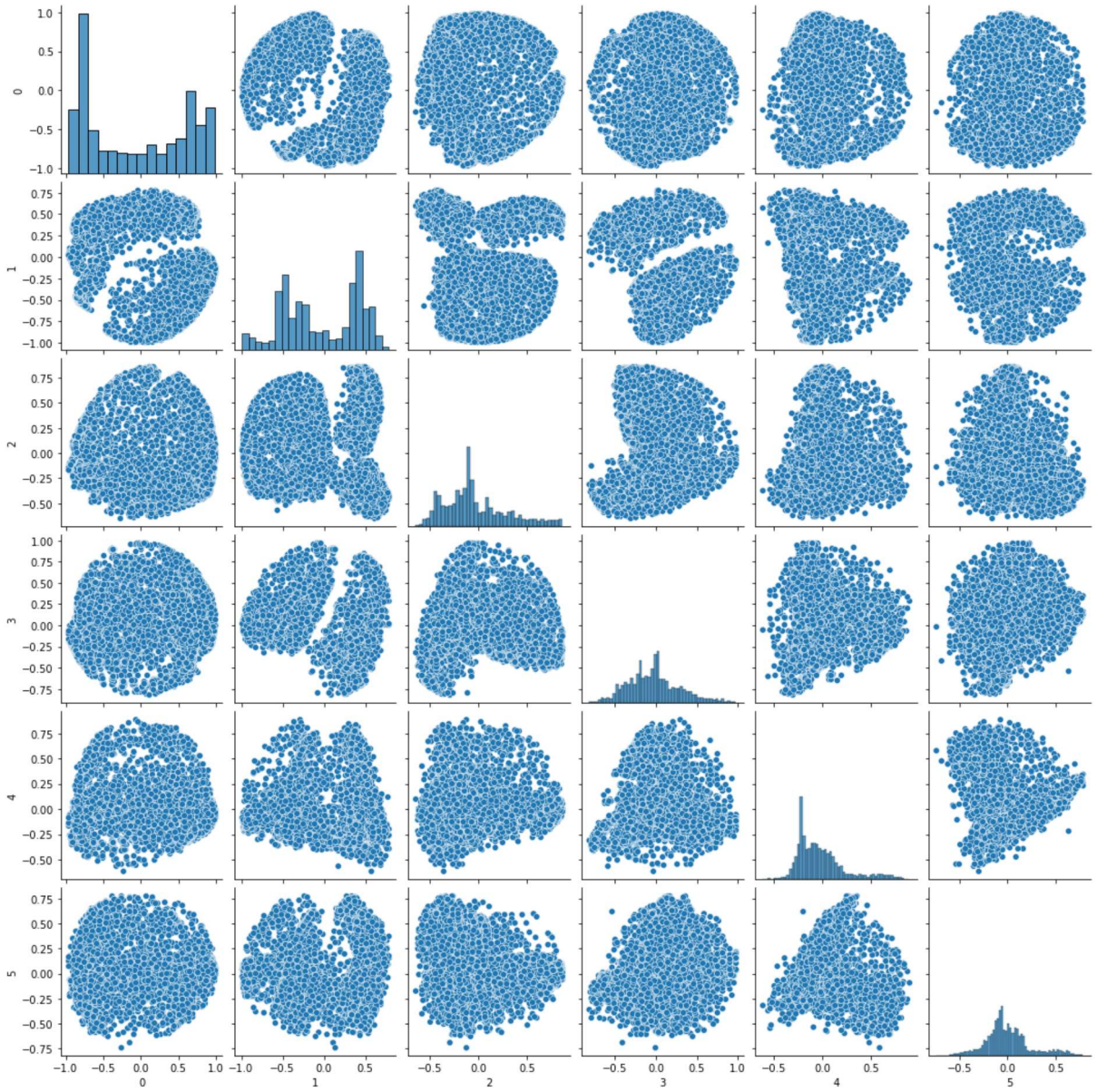
Out[67]:

	0	1	2	3	4	5
0	-0.946773	0.110046	-0.169826	-0.109575	-0.131046	0.183001
1	-0.795217	-0.419763	-0.053312	0.297229	-0.282428	-0.143093
2	0.101542	0.373659	0.832868	-0.371106	0.088713	0.103969
3	-0.882525	0.375493	0.058286	-0.121434	-0.029858	0.247223
4	-0.869351	0.338755	0.090589	-0.280436	-0.197469	-0.060246

In [68]:
```
sns.pairplot(pd.DataFrame(pc_df));
```



In [69]:
```
sns.pairplot(pd.DataFrame(data_new));
```



KMean Clustering

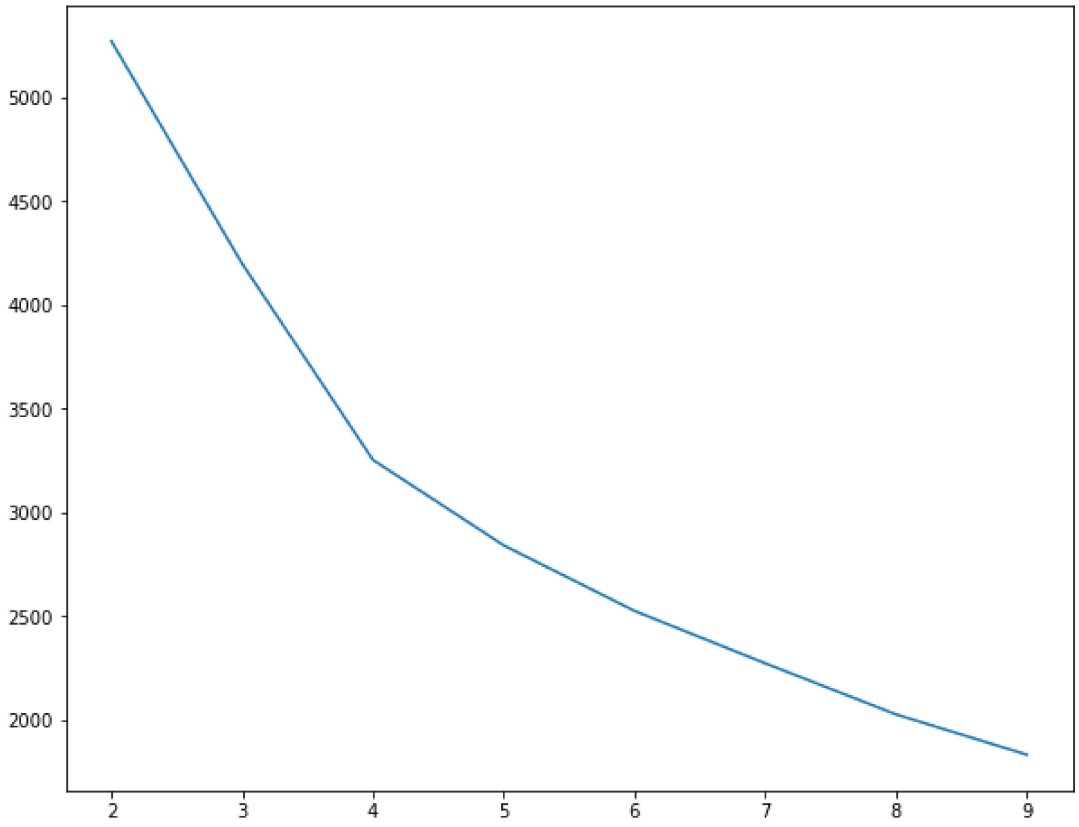
In [70]:
```
gc.collect()
```

Out[70]:
```
124580
```

In [71]:
```
from sklearn.cluster import KMeans
```

In [72]:
```
sum_of_sq_error = []
k_rng = range(2,10)
for k in k_rng:
    km = KMeans(n_clusters = k).fit(data_new)
    sum_of_sq_error.append(km.inertia_)
```

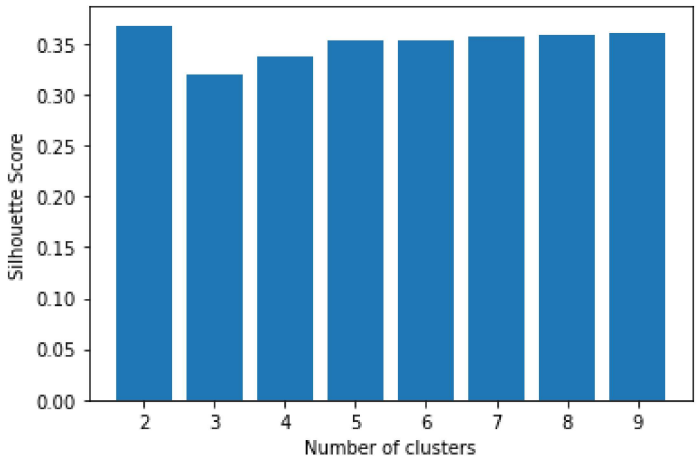
In [73]:
```
plt.figure(figsize=(10,8));
plt.plot(k_rng,sum_of_sq_error);
```



In [74]:
```
from sklearn.metrics import silhouette_score
```

In [75]:
```
silhouette_scores = []
for n in range(2, 10):
    silhouette_scores.append(
        silhouette_score(pc_df, KMeans(n_clusters = n).fit_predict(data_new)))
```

In [76]:
```
k = range(2, 10)
plt.bar(k, silhouette_scores) ;
plt.xlabel('Number of clusters') ;
plt.ylabel('Silhouette Score') ;
```

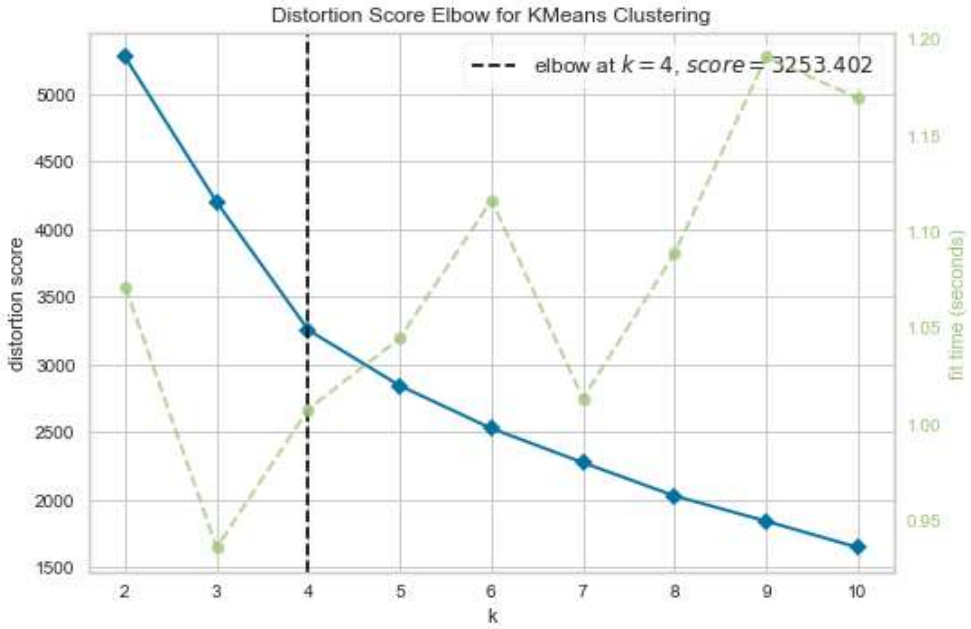


In [77]:
```
max(silhouette_scores)# n_clusters = 2
```

Out[77]:
```
0.36793743856896394
```

In [78]:
```
from yellowbrick.cluster import kelbow_visualizer
```

In [79]:
```
elbow = kelbow_visualizer(KMeans(), data_new)
kmeans = KMeans(n_clusters=elbow.elbow_value_, max_iter=1000)
label_pred = kmeans.fit_predict(data_new)
print("The best number of cluster is: ", elbow.elbow_value_)
```




```
The best number of cluster is:  4
```

In [ ]:
```

```

In [80]:
```
km=KMeans(n_clusters=4,random_state = 42)
y_pred_4=km.fit_predict(data_new)
y_pred_4
```

Out[80]:
```
array([3, 0, 3, ..., 2, 0, 3])
```

In [81]:
```
km.labels_
```

Out[81]:
```
array([3, 0, 3, ..., 2, 0, 3])
```

In [82]:
```
pd.DataFrame(y_pred_4).value_counts()
```

Out[82]:
```
1    2497
3    2405
0    2387
2    1661
dtype: int64
```

In [83]:
```
len(label_pred)
```

Out[83]:
```
8950
```

In [84]:
```
pc_df.shape
```

Out[84]:
```
(8950, 6)
```

In [85]:
```
df_1 = data_new.copy()
```

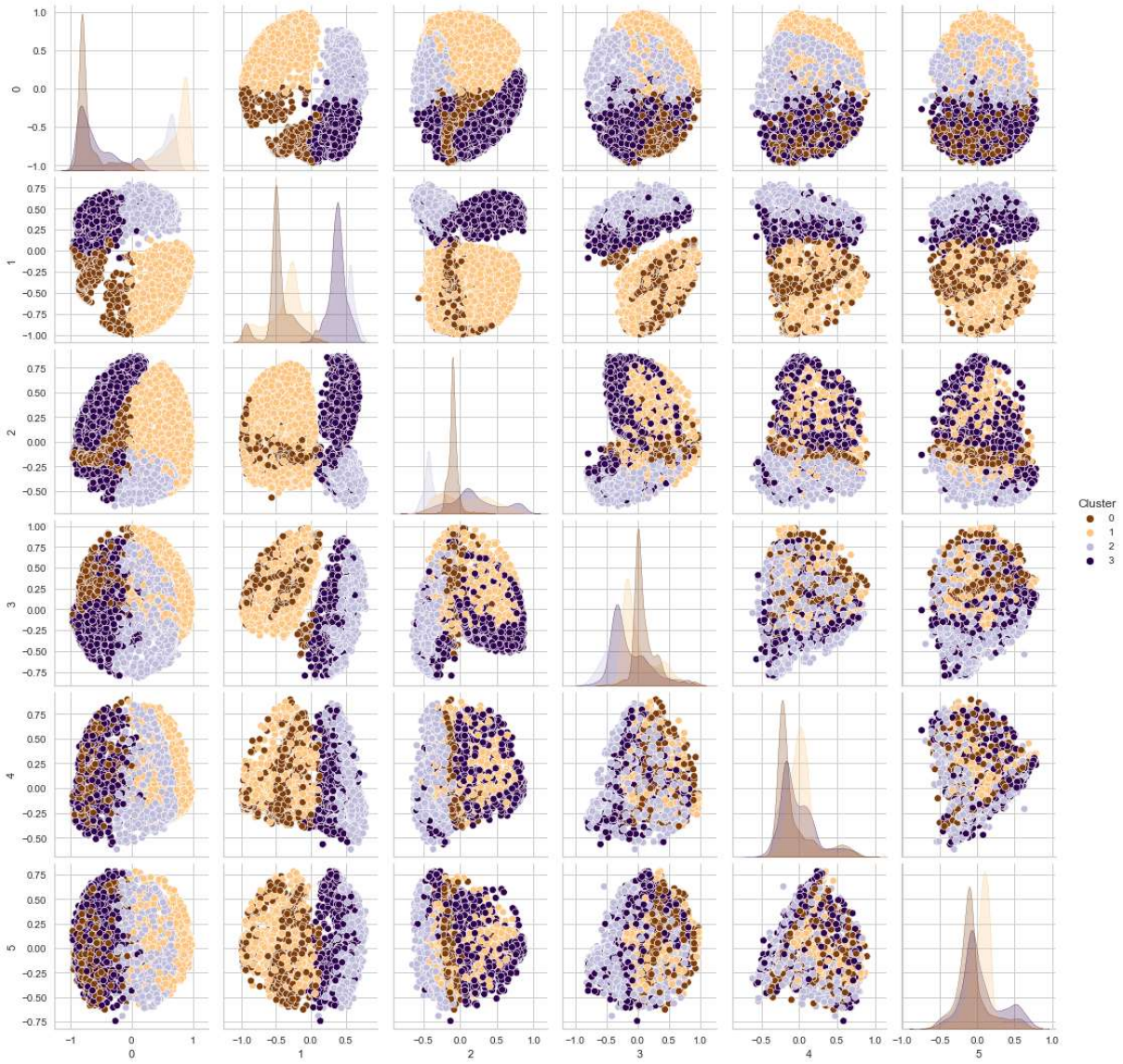
In [86]:
```
df_1['Cluster'] = y_pred_4
```

In [87]:
```
df_1.head()
```

Out[87]:

	0	1	2	3	4	5	Cluster
0	-0.946773	0.110046	-0.169826	-0.109575	-0.131046	0.183001	3
1	-0.795217	-0.419763	-0.053312	0.297229	-0.282428	-0.143093	0
2	0.101542	0.373659	0.832868	-0.371106	0.088713	0.103969	3
3	-0.882525	0.375493	0.058286	-0.121434	-0.029858	0.247223	3
4	-0.869351	0.338755	0.090589	-0.280436	-0.197469	-0.060246	3

In [88]:
```
sns.pairplot(df_1,hue="Cluster",palette = 'PuOr');
```



In [89]:
```
gc.collect()
```

Out[89]:
```
123840
```

In [90]:
```
# by silhouette_scores n_clusters = 2
```

In [91]:
```
km2=KMeans(n_clusters=2,random_state = 42)
y_pred_2=km2.fit_predict(data_new)
y_pred_2
```

Out[91]:
```
array([0, 0, 1, ..., 1, 0, 0])
```

In [92]:
```
df_2 = data_new.copy()
```

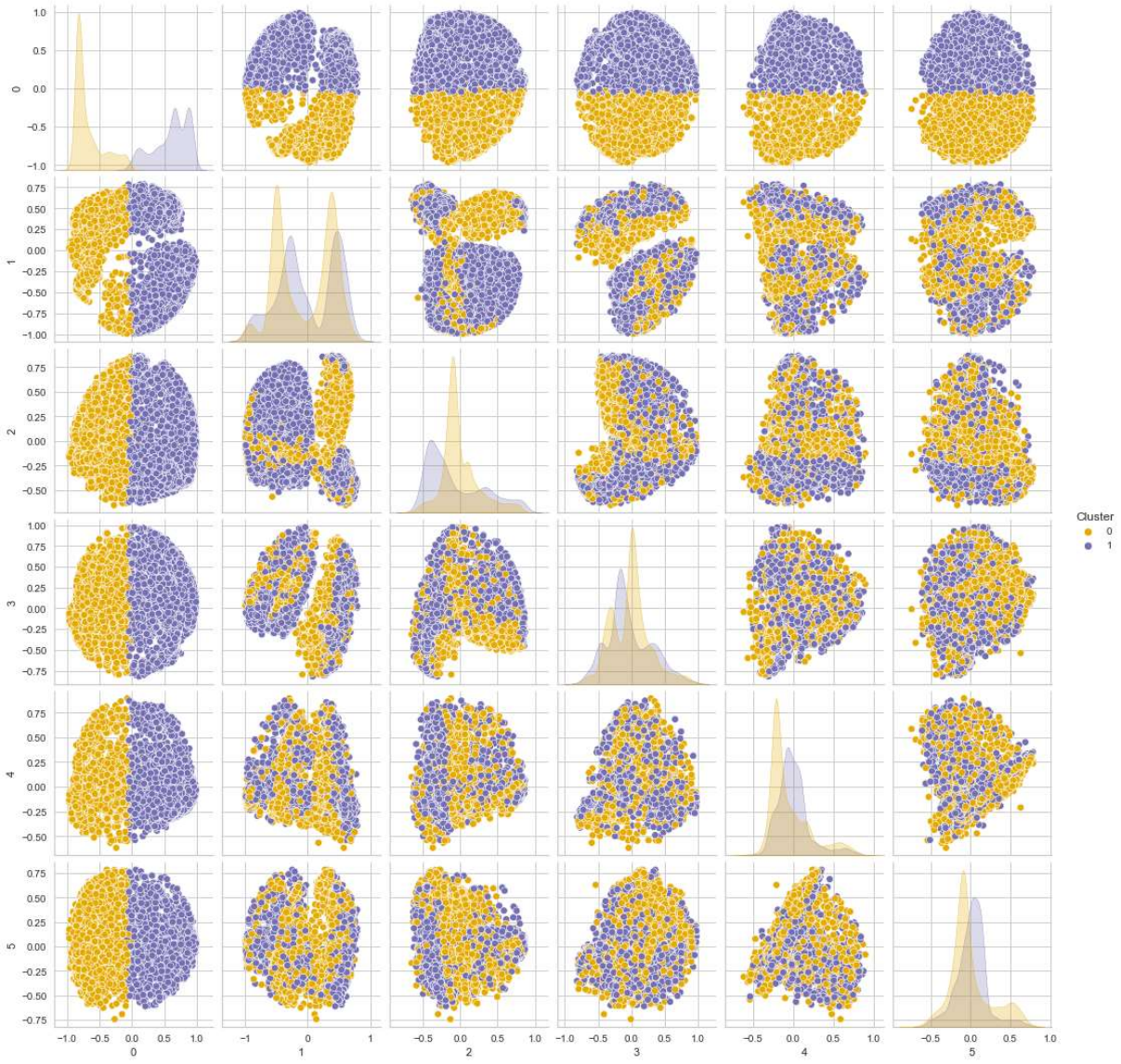
In [93]:
```
df_2['Cluster'] = y_pred_2
```

In [94]:
```
df_2.head()
```

Out[94]:

	0	1	2	3	4	5	Cluster
0	-0.946773	0.110046	-0.169826	-0.109575	-0.131046	0.183001	0
1	-0.795217	-0.419763	-0.053312	0.297229	-0.282428	-0.143093	0
2	0.101542	0.373659	0.832868	-0.371106	0.088713	0.103969	1
3	-0.882525	0.375493	0.058286	-0.121434	-0.029858	0.247223	0
4	-0.869351	0.338755	0.090589	-0.280436	-0.197469	-0.060246	0

In [95]:
```
sns.pairplot(df_2,hue="Cluster",palette = 'Dark2_r');
```



In [96]:
```
data_cl = data.copy()
```

In [97]:
```
data_cl['Cluster'] = y_pred_4
```

In [98]:
```
data_cl.head()
```

Out[98]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	C
0	0.002148	0.818182	0.001945	0.000000	0.00424	
1	0.168169	0.909091	0.000000	0.000000	0.00000	
2	0.131026	1.000000	0.015766	0.018968	0.00000	
3	0.087521	0.636364	0.030567	0.036775	0.00000	
4	0.042940	1.000000	0.000326	0.000393	0.00000	

5 rows × 23 columns

In [99]:
```
data_cl.Cluster.unique()
```

Out[99]:
```
array([3, 0, 2, 1])
```

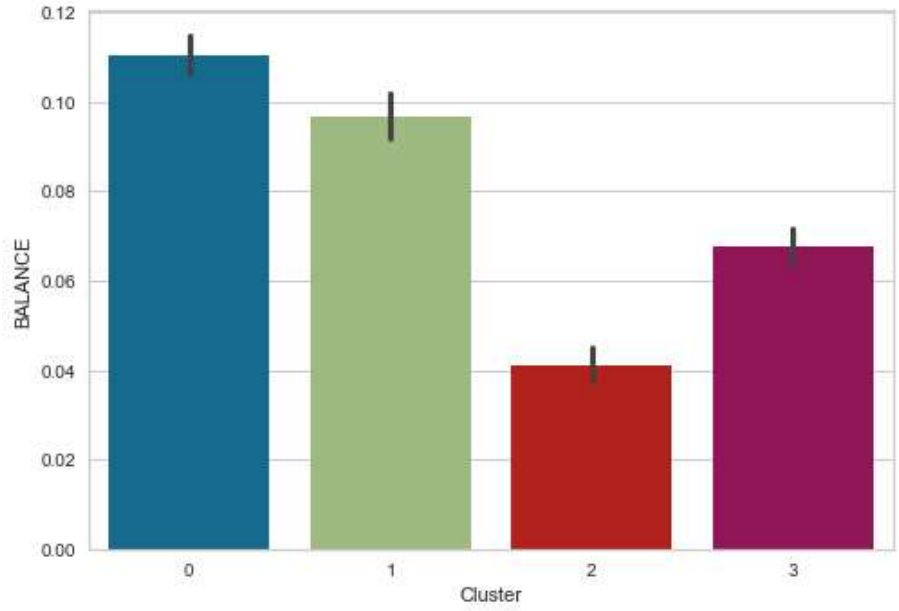
In [100]:
```
data_cl.describe()
```

Out[100]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHAS
count	8950.000000	8950.000000	8950.000000	8950.000000	8950.000(
mean	0.082154	0.877271	0.020457	0.014534	0.018
std	0.109306	0.236904	0.043570	0.040722	0.040
min	0.000000	0.000000	0.000000	0.000000	0.000(
25%	0.006736	0.888889	0.000808	0.000000	0.000(
50%	0.045864	1.000000	0.007367	0.000932	0.003
75%	0.107868	1.000000	0.022637	0.014166	0.020
max	1.000000	1.000000	1.000000	1.000000	1.000(

8 rows × 23 columns

In [101]:
```
sns.barplot(data= data_cl, y='BALANCE',x='Cluster');
```



we can see that Monthly average balance is highest in the 0th cluster & lowest in the 2nd cluster.

In [102]:
```
data_cl.shape
```

Out[102]:
```
(8950, 23)
```

In [103]:
```
xx = ['PURCHASES','ONEOFF_PURCHASES','INSTALLMENTS_PURCHASES','CASH_ADVANCE','limit_usag
```

In [104]:
```
cluster_d=data_cl.groupby('Cluster').apply(lambda x: x[xx].mean()).T
cluster_d
```

Out[104]:

Cluster	0	1	2	3
PURCHASES	0.001579	0.049926	0.013443	0.013442
ONEOFF_PURCHASES	0.001194	0.036467	0.000000	0.015041
INSTALLMENTS_PURCHASES	0.001280	0.042758	0.029353	0.002053
CASH_ADVANCE	0.039452	0.016085	0.007185	0.016461
limit_usage	0.034775	0.022056	0.017171	0.021687
Monthly_average_purchase	0.001610	0.050903	0.014145	0.014256

In [105]:
```
len(cluster_d.columns)
```

Out[105]:
```
4
```

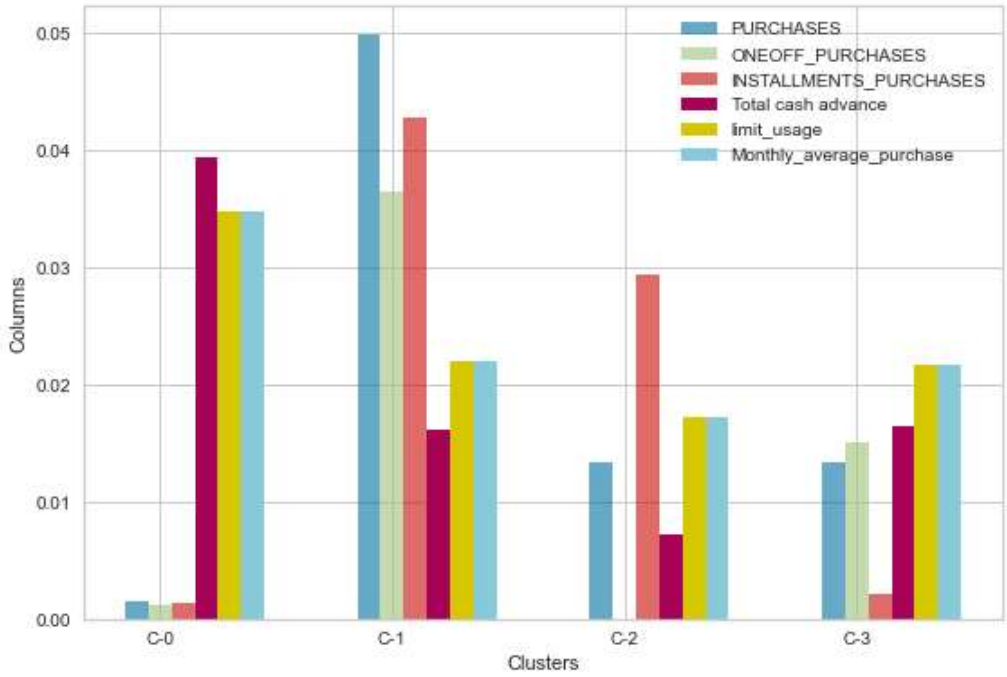
In [106]:
```
cluster_d.loc['CASH_ADVANCE']
```

Out[106]:
```
Cluster
0    0.039452
1    0.016085
2    0.007185
3    0.016461
Name: CASH_ADVANCE, dtype: float64
```

In [107]:
```
col =sns.color_palette()
col
```

Out[107]:

In [108]:
```
n= 4

cl1=cluster_d.loc['PURCHASES']
cl2=cluster_d.loc['ONEOFF_PURCHASES']
cl3=cluster_d.loc['INSTALLMENTS_PURCHASES']
cl4=cluster_d.loc['CASH_ADVANCE']
cl5=cluster_d.loc['limit_usage']
cl6=cluster_d.loc['Monthly_average_purchase']

fig, ax = plt.subplots()

index = np.arange(len(cluster_d.columns))
bar_width = 0.1

opacity = 0.6

b1 =ax.bar(index,cl1,bar_width, color=col[0],label='PURCHASES',alpha=opacity)
b2= ax.bar(index + bar_width, cl2, bar_width,color=col[1],label='ONEOFF_PURCHASES',alpha
b3= ax.bar(index + 2*bar_width, cl3, bar_width,color=col[2],label='INSTALLMENTS_PURCHASE
b4 = ax.bar(index + 3*bar_width, cl4, bar_width,label='Total cash advance',color=col[3])
b5 = ax.bar(index + 4*bar_width, cl5, bar_width,label='limit_usage',color=col[4])
b6 = ax.bar(index + 5*bar_width, cl5, bar_width,label='Monthly_average_purchase',color=c

plt.xlabel('Clusters')
plt.ylabel('Columns')
plt.xticks(index + bar_width, ('C-0', 'C-1', 'C-2', 'C-3'))

plt.tight_layout()
plt.legend();
```



Insights with 4 Clusters

- Cluster 0 is the group of customers having very low Total purchase amount spent during last 12 months, having lowest total amount of one-off purchases & total amount of installment purchases & having highest total cash-advance amount, having highest limit usage(balance to credit limit ratio).Also having highest Monthly_average_purchase.
- Cluster 1 is the group of customers having highest Total purchase amount spent during last 12 months & highest total amount of installment purchases and highest one-off purchase.
- Cluster 2 customers are doing very less or no One_Off transactions and lowest total cash-advance amount, having highest limit usage(balance to credit limit ratio) & lowest Monthly_average_purchase.

- Cluster 3 customers is the group of customer having low total amount of installment purchases& 2nd highest one-off purchase amount.

#####

we can provide less interest rate on purchase transaction to **group 0**,because they are taking the highest cash on advance & having poor one-off & installment transiction.

Group 1 is performing best among all. we can give some rewards or some more offers, so they can purchase more.

we can give some cash on advanceto the **group 2**, because they having the lowest total cash-advance amount

In [109]:
```
import pandas_profiling as pp
```

In [110]:
```
profile = pp.ProfileReport(data_cl)
```

In [111]:
```
profile
```

Overview | Variables | Correlations | Missing values | Sample

Dataset info

Number of variables	23
Number of observations	8950
Missing cells	0 (0.0%)
Duplicate rows	0 (0.0%)
Total size in memory	1.5 MiB
Average record size in memory	180.0 B

Variables types

NUM	21
CAT	2

Toggle Reproduction Information

Toggle Warnings

Warnings

CASH_ADVANCE has 4628 (51.7%) zeros	Zeros
cash_advance_amount has 4628 (51.7%) zeros	Zeros
CASH_ADVANCE_FREQUENCY has 4628 (51.7%) zeros	Zeros
CASH_ADVANCE_TRX has 4628 (51.7%) zeros	Zeros
INSTALLMENTS_PURCHASES has 3916 (43.8%) zeros	Zeros
MINIMUM_PAYMENTS has 313 (3.5%) zeros	Zeros
Monthly_average_purchase has 2044 (22.8%) zeros	Zeros
ONEOFF_PURCHASES has 4302 (48.1%) zeros	Zeros
ONEOFF_PURCHASES_FREQUENCY has 4302 (48.1%) zeros	Zeros
pay/min_pay is highly skewed (γ1 = 43.00880362)	Skewed
pay/min_pay has 313 (3.5%) zeros	Zeros
PAYMENTS has 240 (2.7%) zeros	Zeros
PRC_FULL_PAYMENT has 5903 (66.0%) zeros	Zeros
PURCHASES has 2044 (22.8%) zeros	Zeros
PURCHASES_FREQUENCY has 2043 (22.8%) zeros	Zeros
PURCHASES_INSTALLMENTS_FREQUENCY has 3915 (43.7%) zeros	Zeros
PURCHASES_TRX has 2044 (22.8%) zeros	Zeros
TENURE has 204 (2.3%) zeros	Zeros
cash_advance_amount is highly correlated with CASH_ADVANCE	High Correlation

CASH_ADVANCE is highly correlated with cash_advance_amount	High Correlation
ONEOFF_PURCHASES is highly correlated with Monthly_average_purchase and 1 other fields (Monthly_average_purchase, PURCHASES)	High Correlation
Monthly_average_purchase is highly correlated with ONEOFF_PURCHASES and 1 other fields (ONEOFF_PURCHASES, PURCHASES)	High Correlation
PURCHASES is highly correlated with Monthly_average_purchase and 1 other fields (Monthly_average_purchase, ONEOFF_PURCHASES)	High Correlation

Report generated with pandas-profiling (https://github.com/pandas-profiling/pandas-profiling).

Out[111]:

In [ ]: