```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
import random
import seaborn as sns
warnings.filterwarnings("ignore")
```

```python
#read the data
#dowmload dataset from https://www.kaggle.com/code/sadkoktaybicici/credit-card-data-clustering-k-mean/data

data_path = 'CC_GENERAL.csv'
df = pd.read_csv(data_path)

print('The shape of the dataset is:', df.shape)
```

```
The shape of the dataset is: (8950, 18)
```

Brief description of the data set and a summary of its attributes

The Dataset summarizes the usage behavior of 8950 active credit card holders in the last 6 months.

The data contains 18 behavior variables of each client.

- CUSTID : Credit Card Holder ID
- BALANCE : The remaining balance in their account for shopping
- BALANCE_FREQUENCY : How often the balance is updated, 0 to 1 score (1 = frequently updated, 0 = not frequently updated)
- PURCHASES : The amount of purchases made from the account
- ONEOFF_PURCHASES : The maximum amount of purchases made at one time
- INSTALLMENTS_PURCHASES : Amount of shopping in installments
- CASH_ADVANCE : Cash in advance given by the user
- PURCHASES_FREQUENCY : Rate of how often purchases are made, from 0 to 1 (1 = frequently purchased, 0 = not frequently purchased)
- ONEOFF_PURCHASES_FREQUENCY : How often purchases occur at once (1 = frequently purchased, 0 = not frequently purchased)
- PURCHASES_INSTALLMENTS_FREQUENCY : How often installment purchases are made (1=frequent, 0=not frequent)
- CASH_ADVANCE_FREQUENCY : How often the down payment is made
- CASHADVANCE_TRX : Number of Transactions with "Advanced Cash"
- PURCHASES_TRX : Number of purchases made
- CREDIT_LIMIT : Credit Card Limit for User
- PAYMENTS : Amount of payment made by the user
- MINIMUM_PAYMENTS : The minimum payment amount made by the user
- PRC_FULL_PAYMENT : Percentage of full payment paid by the user
- TENURE : The duration of the credit card service for the user

```python
df.describe()
```

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHA
count	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	
mean	1564.474828	0.877271	1003.204834	592.437371	411.067645	978.871112	
std	2081.531879	0.236904	2136.634782	1659.887917	904.338115	2097.163877	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	128.281915	0.888889	39.635000	0.000000	0.000000	0.000000	
50%	873.385231	1.000000	361.280000	38.000000	89.000000	0.000000	
75%	2054.140036	1.000000	1110.130000	577.405000	468.637500	1113.821139	
max	19043.138560	1.000000	49039.570000	40761.250000	22500.000000	47137.211760	

```python
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   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
```

Initial plan for data exploration

in this part i check the data quality, assess any issues in the data and fix it:

- null values in each column
- each column has the proper data type
- outliers
- duplicate rows
- distribution for each column (skewness)

drop id column

```python
df_copy = df.drop(columns='CUST_ID')
```

```python
df_copy.head()
```

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_
0	40.900749	0.818182	95.40	0.00	95.4	0.000000	
1	3202.467416	0.909091	0.00	0.00	0.0	6442.945483	
2	2495.148862	1.000000	773.17	773.17	0.0	0.000000	
3	1666.670542	0.636364	1499.00	1499.00	0.0	205.788017	
4	817.714335	1.000000	16.00	16.00	0.0	0.000000	

null values

```python
df_copy.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
```

```python
df_copy.isna().sum()
```

```
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
```

solution

```python
df_copy = df_copy.fillna(df_copy.median())
```

```python
df_copy.isna().sum()
```

```
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
dtype: int64
```

data types

```python
df_copy.head()
```

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_
0	40.900749	0.818182	95.40	0.00	95.4	0.000000	
1	3202.467416	0.909091	0.00	0.00	0.0	6442.945483	
2	2495.148862	1.000000	773.17	773.17	0.0	0.000000	
3	1666.670542	0.636364	1499.00	1499.00	0.0	205.788017	
4	817.714335	1.000000	16.00	16.00	0.0	0.000000	

```python
df_copy.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                      8950 non-null   float64
 13  PAYMENTS                          8950 non-null   float64
 14  MINIMUM_PAYMENTS                  8950 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
```

all data types seems correct

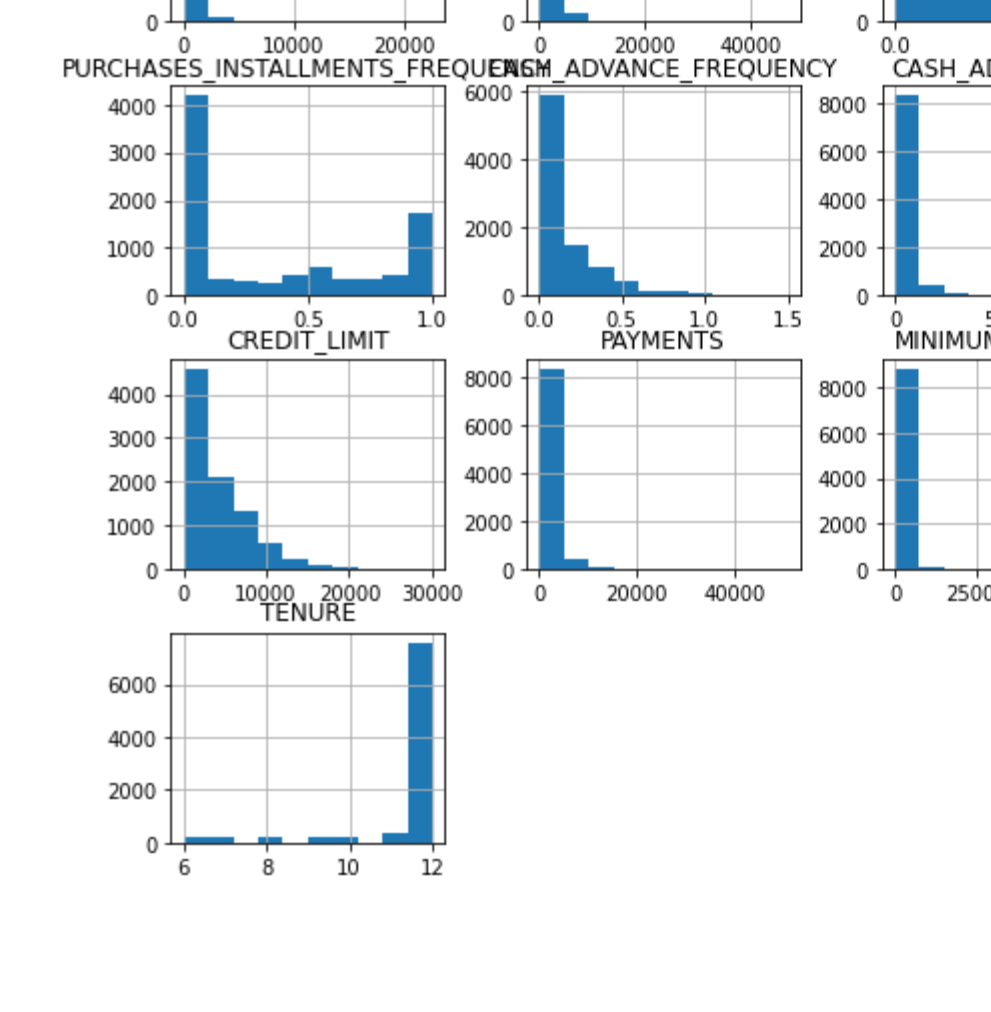
outliers

```python
for col in df_copy.columns:
    sns.boxplot(df_copy[col])
    plt.show()
    sns.histplot(df_copy[col])
    plt.show()
```

there are alot of outliers here so i can't remove them

transform data

```python
df_copy.hist(figsize=(12,12));
```



```
In [15]:
columns_to_unskew = []
columns_boxcox = []
for col in df_copy.columns:
    skew = df_copy[col].skew()
    print("skewness for ",col," is ", skew)
    if skew > 1 or skew < -1:
#         if col == 'BALANCE_FREQUENCY' or col == 'TENURE':
#             columns_boxcox.append(col)
#         else:
        columns_to_unskew.append(col)
```

```
skewness for  BALANCE  is  2.393386042571806
skewness for  BALANCE_FREQUENCY  is  -2.0232655185149078
skewness for  PURCHASES  is  8.144269064554651
skewness for  ONEOFF_PURCHASES  is  10.045082884700278
skewness for  INSTALLMENTS_PURCHASES  is  7.299119908745641
skewness for  CASH_ADVANCE  is  5.166609074074723
skewness for  PURCHASES_FREQUENCY  is  0.06016423585803591
skewness for  ONEOFF_PURCHASES_FREQUENCY  is  1.5356127835248519
skewness for  PURCHASES_INSTALLMENTS_FREQUENCY  is  0.5092011649999882
skewness for  CASH_ADVANCE_FREQUENCY  is  1.828686266477852
skewness for  CASH_ADVANCE_TRX  is  5.721298203192298
skewness for  PURCHASES_TRX  is  4.630655265932403
skewness for  CREDIT_LIMIT  is  1.5226359501884323
skewness for  PAYMENTS  is  5.907619794397562
skewness for  MINIMUM_PAYMENTS  is  13.852446498665346
skewness for  PRC_FULL_PAYMENT  is  1.942819940971858
skewness for  TENURE  is  -2.9430172876199134
```

alot of the columns are skewed

```
In [16]:
print(columns_to_unskew)
```

```
['BALANCE', 'BALANCE_FREQUENCY', 'PURCHASES', 'ONEOFF_PURCHASES', 'INSTALLMENTS_PURCHASES', 'CASH_ADVANCE', 'ON
EOFF_PURCHASES_FREQUENCY', 'CASH_ADVANCE_FREQUENCY', 'CASH_ADVANCE_TRX', 'PURCHASES_TRX', 'CREDIT_LIMIT', 'PAYM
ENTS', 'MINIMUM_PAYMENTS', 'PRC_FULL_PAYMENT', 'TENURE']
```

```
In [17]:
from scipy import stats
```

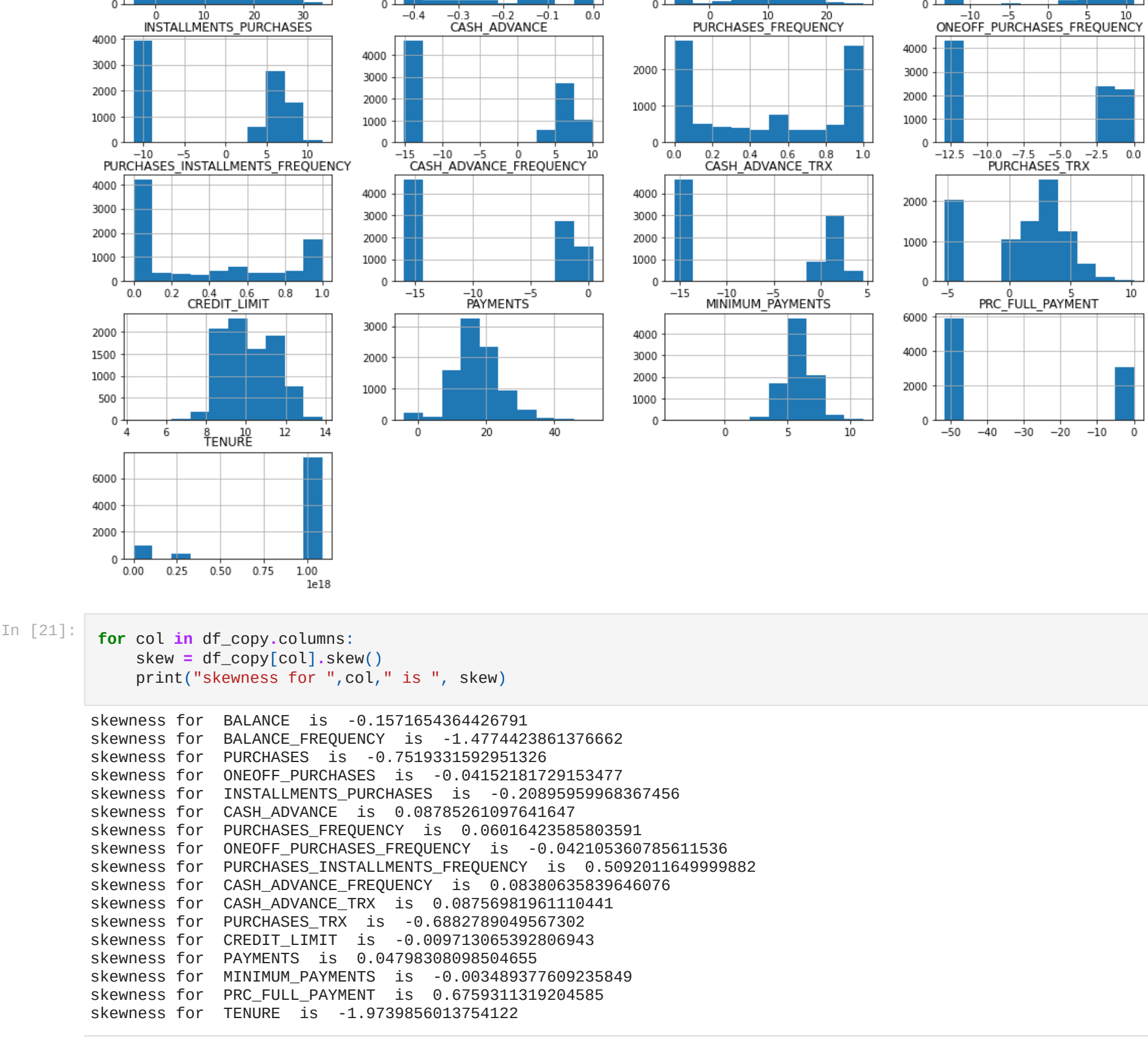
```
In [18]:
#df_copy[columns_to_unskew] = np.log(df_copy[columns_to_unskew]+0.1)
#df_copy[columns_to_unskew] = np.sqrt(df_copy[columns_to_unskew])
#df_copy[columns_to_unskew] = np.log1p(df_copy[columns_to_unskew]+0.1)
#df_copy[columns_to_unskew] = np.arcsinh(df_copy[columns_to_unskew])
for col in columns_to_unskew:
    df_copy[col], _ = stats.boxcox(df_copy[col]+1e-6) # 1e-6 to deal with zero values
```

```
In [19]:
df_copy.head()
```

Out[19]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FR
0	5.677545	-0.159621	6.468611	-13.227810	4.912375	-15.117524	
1	21.718338	-0.085292	-5.949881	-13.227810	-11.134216	8.294623	
2	20.341074	0.000001	11.218173	6.792680	-11.134216	-15.117524	
3	18.264780	-0.277124	13.048225	7.484694	-11.134216	5.148653	
4	15.012643	0.000001	3.415678	2.797098	-11.134216	-15.117524	

```
In [20]:
df_copy.hist(figsize=(18,12));
```




```
In [21]:
for col in df_copy.columns:
    skew = df_copy[col].skew()
    print("skewness for ",col," is ", skew)
```

```
skewness for  BALANCE  is  -0.1571654364426791
skewness for  BALANCE_FREQUENCY  is  -1.4774423861376662
skewness for  PURCHASES  is  -0.7519331592951326
skewness for  ONEOFF_PURCHASES  is  -0.04152181729153477
skewness for  INSTALLMENTS_PURCHASES  is  -0.20895959968367456
skewness for  CASH_ADVANCE  is  0.08785261097641647
skewness for  PURCHASES_FREQUENCY  is  0.06016423585803591
skewness for  ONEOFF_PURCHASES_FREQUENCY  is  -0.042105360785611536
skewness for  PURCHASES_INSTALLMENTS_FREQUENCY  is  0.5092011649999882
skewness for  CASH_ADVANCE_FREQUENCY  is  0.08380635839646076
skewness for  CASH_ADVANCE_TRX  is  0.08756981961110441
skewness for  PURCHASES_TRX  is  -0.6882789049567302
skewness for  CREDIT_LIMIT  is  -0.009713065392806943
skewness for  PAYMENTS  is  0.04798308098504655
skewness for  MINIMUM_PAYMENTS  is  -0.003489377609235849
skewness for  PRC_FULL_PAYMENT  is  0.6759311319204585
skewness for  TENURE  is  -1.9739856013754122
```

```
In [22]:
### some columns still have high skewness due to unbalanced categories
```

standardization and scaling

```
In [23]:
from sklearn.preprocessing import RobustScaler
scaler = RobustScaler()

df_copy[df_copy.columns] = scaler.fit_transform(df_copy)
df_copy
```

Out[23]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES
0	-0.893344	-1.553604	-0.389315	-0.857550	0.004483	0.000000	
1	0.596959	-0.830153	-2.083849	-0.857550	-0.889950	1.072663	
2	0.469002	0.000000	0.258774	0.157881	-0.889950	0.000000	
3	0.276099	-2.697255	0.508489	0.192980	-0.889950	0.928526	
4	-0.026047	0.000000	-0.805895	-0.044773	-0.889950	0.000000	
...	...	...	...	...	...	...	
8945	-0.965084	0.000000	-0.068364	-0.857550	0.077901	0.000000	
8946	-1.037474	0.000000	-0.058979	-0.857550	0.079916	0.000000	
8947	-1.001898	-1.440271	-0.276110	-0.857550	0.031451	0.000000	
8948	-1.097308	-1.440271	-2.083849	-0.857550	-0.889950	0.853768	
8949	-0.309938	-2.533616	0.386430	0.176228	-0.889950	0.907824	

8950 rows × 17 columns

duplicates

```
In [24]:
df_copy.duplicated().sum()
```

Out[24]: 0

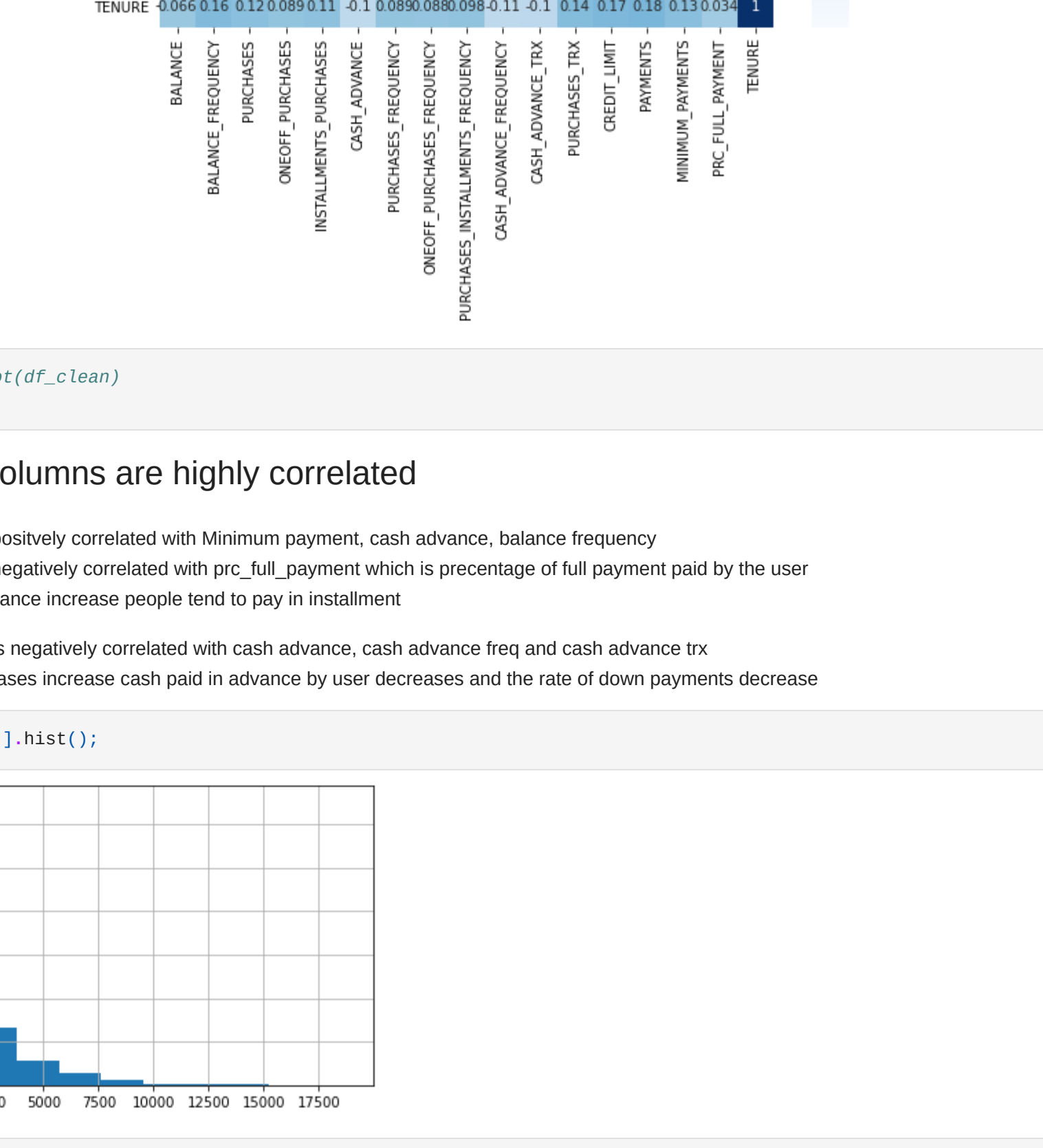
no duplicate values

saving clean data

```
In [25]:
df_clean = df_copy.copy()
df_clean.to_csv("credit_card_cleanData")
```

Exploratory Data Analysis

```
In [26]:
plt.figure(figsize=(10,10))
sns.heatmap(df_clean.corr(),cbar=True,annot=True,cmap='Blues');
```



```
In [27]:
#sns.pairplot(df_clean)
#plt.show()
```

multiple columns are highly correlated

- Balance is positvely correlated with Minimum payment, cash advance, balance frequency
- Balance is negatively correlated with prc_full_payment which is precentage of full payment paid by the user
- so when balance increase people tend to pay in installment

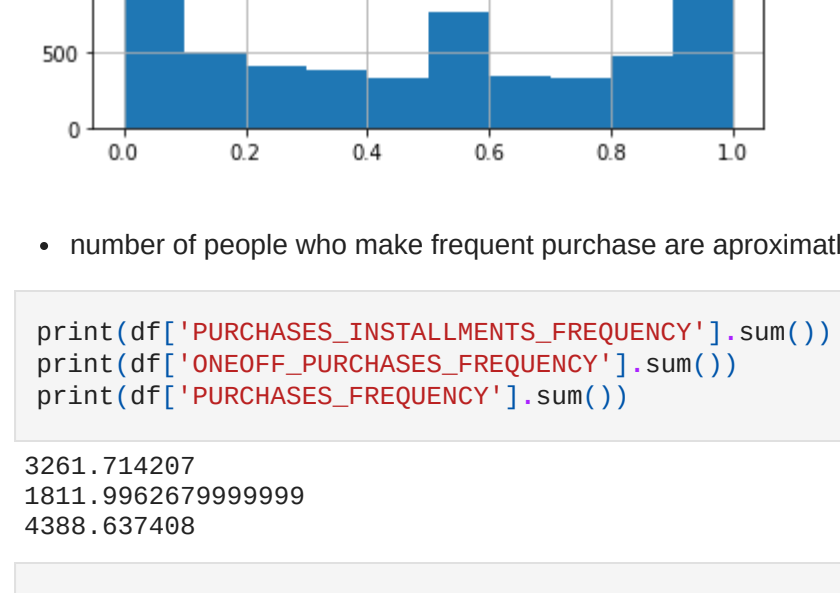
- purchases is negatively correlated with cash advance, cash advance freq and cash advance trx
- so as purchases increase cash paid in advance by user decreases and the rate of down payments decrease

```
In [28]:
df['BALANCE'].hist();
```

```
In [29]:
df['BALANCE_FREQUENCY'].hist();
```

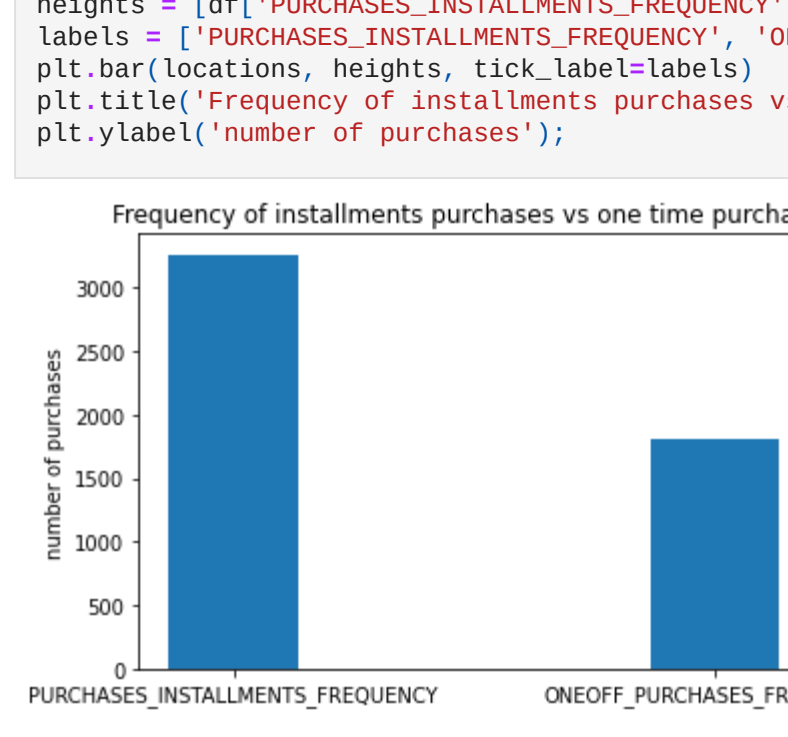
- most accounts have zero balance
- most accounts gets updated frequently so most clients are active users

```
In [30]:
df['PURCHASES'].hist();
plt.xlabel("Amount of purchases")
plt.ylabel("number of clients")
plt.title("number of clients who made a purchase with this amount")
plt.show()
```



- most people have purchase of 0

```
In [31]:
df['PURCHASES_FREQUENCY'].hist();
```

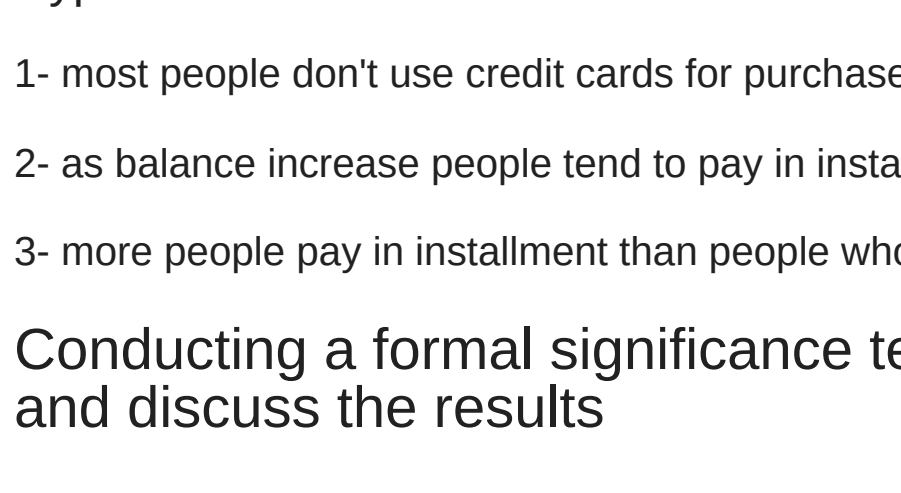


- number of people who make frequent purchase are aproximatly equal to number of people who make no purchases

```
In [32]:
print(df['PURCHASES_INSTALLMENTS_FREQUENCY'].sum())
print(df['ONEOFF_PURCHASES_FREQUENCY'].sum())
print(df['PURCHASES_FREQUENCY'].sum())
```

```
3261.714207
1811.9962679999999
4388.637408
```

```
In [33]:
locations = [1, 4]
heights = [df['PURCHASES_INSTALLMENTS_FREQUENCY'].sum(), df['ONEOFF_PURCHASES_FREQUENCY'].sum()]
labels = ['PURCHASES_INSTALLMENTS_FREQUENCY', 'ONEOFF_PURCHASES_FREQUENCY']
plt.bar(locations, heights, tick_label=labels)
plt.title('Frequency of installments purchases vs one time purchases')
plt.ylabel('number of purchases');
```



conclusion

- most users are active but number of purchase is small for most users and most users keep balance at zero
- most users tend to make installment purchases more than one time purchases

Hypothesis

1- most people don't use credit cards for purchases

2- as balance increase people tend to pay in installment

3- more people pay in installment than people who make one time purchases

Conducting a formal significance test for one of the hypotheses and discuss the results

Hypothesis three

more people pay in installment than people who make one time purchases

Ho: mean of installments purchases frequency <= mean of one time purchases frequency

Ha: mean of installments purchases frequency > mean of one time purchases frequency

```
In [34]:
installment_purchases_frequency = df['PURCHASES_INSTALLMENTS_FREQUENCY']
installment_mean = installment_purchases_frequency.mean()
print(installment_mean)
```

```
0.3644373415642452
```

```
In [35]:
oneoff_purchases_frequency = df['ONEOFF_PURCHASES_FREQUENCY']
oneoff_mean = oneoff_purchases_frequency.mean()
print(oneoff_mean)
```

```
0.20245768357542138
```

```
In [36]:
alpha=0.05
t_val2, p_value2 = stats.ttest_ind(installment_purchases_frequency, oneoff_purchases_frequency)
p_value_onetail=p_value2/2
print("t_value = {} , p_value ={} , p_value_onetail = {}".format(t_val2, p_value2, p_value_onetail))
```

```
t_value = 30.835475213423836 , p_value =1.7857215613197702e-203 , p_value_onetail = 8.928607806598851e-204
```

```
In [37]:
if p_value2 <alpha :
    print("Conclusion: Since p_value {} is less than alpha {} ". format (p_value_onetail,alpha) )
    print("Reject null hypothesis that mean of installments purchases frequency <= mean of one time purchases f
else:
    print("Conclusion: Since p_value {} is greater than alpha {} ". format (p_value_onetail,alpha))
    print("Failed to reject null hypothesis that mean of installments purchases frequency > mean of one time pu
```

```
Conclusion: Since p_value 8.928607806598851e-204 is less than alpha 0.05
Reject null hypothesis that mean of installments purchases frequency <= mean of one time purchases frequency.
```

```
In [ ]:
```

Suggestions for next steps in analyzing this data

- look more into why the balance of most segments is zero yet most clients are active
- Use clustering to check the different segments of clients and check their behaviours

the quality of this data set

- the data had alot of outliers and most of the data was heavily skewed
- most data was available and only MINIMUM_PAYMENTS had missing data

```
In [ ]:
```

```
In [ ]:
```