# Importing libraries

In [1]:

```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
```

# Importing dataframes

In [2]:

```python
accounts_df = pd.read_csv ('full_acc/account.csv', sep = ';')
cards_df = pd.read_csv ('full_acc/card.csv', sep = ';')
clients_df = pd.read_csv ('full_acc/client.csv', sep = ';')
dispos_df = pd.read_csv ('full_acc/disp.csv', sep = ';')
loan_df = pd.read_csv ('full_acc/loan.csv', sep = ';')
district_df = pd.read_csv("full_acc/district.csv", sep=";")
order_df = pd.read_csv ('full_acc/order.csv', sep = ';')
trans_df = pd.read_csv ('full_acc/trans.csv', sep = ';')
```

```
C:\Users\abdul\AppData\Local\Temp\ipykernel_8708\2723839878.py:8: DtypeWarning: Columns (8) have mixed types. Specify dtype option on import or
set low_memory=False.
  trans_df = pd.read_csv ('full_acc/trans.csv', sep = ';')
```

# defining all necessary functions

In [3]:

```python
def date_correction (acc, col_name):
    """ Function that will re-format cells into date format. Input should be in the format of YYMMDD, e.g. 950107
    input: dataframe and column name of the dataframe
    output: updated dataframe
    """
    acc [col_name] = pd.to_datetime (acc [col_name], format = '%y%m%d', errors = 'coerce')
    return acc

def date_misinterp_cor (date): #correction of date misinterpretation when 45 treated as 2045 and not 1945
    """Function that corrects year misinterpretation when 45 treated as 2045 and not 1945
    input: date
    output: corrected date"""
    if date.year > 2000:
        date = date.replace (year = date.year - 100)
        return date
    else:
        return date

def calculate_age (born_date):
    """Age calculation as of 31.12.1998
    input: born date
    outpute: age (int)"""
    born = born_date.year
    return 1998 - born

def date_parsing (date):
    """Extracting year from a string.
    input: a value that contains date in the first 6 symbols.
    output: year"""
    date = int (str (date) [0:6])
    date2 = pd.to_datetime (date, format = '%y%m%d', errors = 'coerce')
    if date2.year > 2000:
        return date2.year - 100
    else:
        return date2.year

def year_extract (date):
    """ Function that will extract a year from date and return it
        input should be date format"""
    if date.year > 2000:
        return date.year - 100
    else:
        return date.year

def acc_row_normalize(dataframe):
    '''Normalizes the values of a given pandas.Dataframe by the total sum of each line.
    Algorithm based on https://stackoverflow.com/questions/26537878/pandas-sum-across-columns-and-divide-each-cell-from-that-value'''
    return dataframe.div(dataframe.sum(axis=1), axis=0)
```

In [4]:

```
%whos DataFrame
```

```
Variable      Type         Data/Info
------------------------------------
accounts_df   DataFrame          account_id  distric<...>\n[4500 rows x 4 columns]
cards_df      DataFrame         card_id  disp_id    <...>n\n[892 rows x 4 columns]
clients_df    DataFrame          client_id  birth_nu<...>\n[5369 rows x 3 columns]
dispos_df     DataFrame          disp_id  client_id <...>\n[5369 rows x 4 columns]
district_df   DataFrame        A1               A2  <...>n\n[77 rows x 16 columns]
loan_df       DataFrame         loan_id  account_id <...>n\n[682 rows x 7 columns]
order_df      DataFrame          order_id  account_i<...>\n[6471 rows x 6 columns]
trans_df      DataFrame             trans_id  accoun<...>056320 rows x 10 columns]
```

In [5]:

```
files = [accounts_df, cards_df, clients_df, dispos_df, loan_df, order_df, trans_df]
date_cor_files = [trans_df, accounts_df, loan_df]
```

```
files_name = ['accounts_df', 'cards_df', 'clients_df', 'dispos_df', 'loan_df', 'order_df', 'trans_df']
```

In [6]:

```
for id, item in enumerate (date_cor_files):
    date_cor_files [id] = date_correction (item, 'date')

trans_df = date_cor_files [0]
accounts_df = date_cor_files [1]
loans_df = date_cor_files [2]
```

In [7]:

```
#verifying numeric statistics and missing values in the datasets
for id, item in enumerate (files):
    print ('Dataframe name: ' + str (files_name [id]) + " with number of rows:" + str (item.shape [0]) + ' and columns:' + str (item.shape [1]) )
    display (item.describe ())
    print (item.isnull ().sum ())
    print("Duplicate Count",item.duplicated().sum())
    print ('\n')#verifying the dataframe structure
```

```
Dataframe name: accounts_df with number of rows:4500 and columns:4
```

| | account_id | district_id |
|---|---|---|
| **count** | 4500.000000 | 4500.000000 |
| **mean** | 2786.067556 | 37.310444 |
| **std** | 2313.811984 | 25.177217 |
| **min** | 1.000000 | 1.000000 |
| **25%** | 1182.750000 | 13.000000 |
| **50%** | 2368.000000 | 38.000000 |
| **75%** | 3552.250000 | 60.000000 |
| **max** | 11382.000000 | 77.000000 |

```
account_id     0
district_id    0
frequency      0
date           0
dtype: int64
Duplicate Count 0


Dataframe name: cards_df with number of rows:892 and columns:4
```

|  | card_id | disp_id |
|---|---|---|
| **count** | 892.000000 | 892.000000 |
| **mean** | 480.855381 | 3511.862108 |
| **std** | 306.933982 | 2984.373626 |
| **min** | 1.000000 | 9.000000 |
| **25%** | 229.750000 | 1387.000000 |
| **50%** | 456.500000 | 2938.500000 |
| **75%** | 684.250000 | 4459.500000 |
| **max** | 1247.000000 | 13660.000000 |

```
card_id    0
disp_id    0
type       0
issued     0
dtype: int64
Duplicate Count 0


Dataframe name: clients_df with number of rows:5369 and columns:3
```

|  | client_id | birth_number | district_id |
|---|---|---|---|
| **count** | 5369.000000 | 5369.000000 | 5369.000000 |
| **mean** | 3359.011920 | 535114.970013 | 37.310114 |
| **std** | 2832.911984 | 172895.618429 | 25.043690 |
| **min** | 1.000000 | 110820.000000 | 1.000000 |
| **25%** | 1418.000000 | 406009.000000 | 14.000000 |
| **50%** | 2839.000000 | 540829.000000 | 38.000000 |
| **75%** | 4257.000000 | 681013.000000 | 60.000000 |
| **max** | 13998.000000 | 875927.000000 | 77.000000 |

```
client_id       0
birth_number    0
district_id     0
dtype: int64
Duplicate Count 0


Dataframe name: dispos_df with number of rows:5369 and columns:4
```

| | disp_id | client_id | account_id |
|---|---|---|---|
| **count** | 5369.000000 | 5369.000000 | 5369.000000 |
| **mean** | 3337.097970 | 3359.011920 | 2767.496927 |
| **std** | 2770.418826 | 2832.911984 | 2307.843630 |
| **min** | 1.000000 | 1.000000 | 1.000000 |
| **25%** | 1418.000000 | 1418.000000 | 1178.000000 |
| **50%** | 2839.000000 | 2839.000000 | 2349.000000 |
| **75%** | 4257.000000 | 4257.000000 | 3526.000000 |
| **max** | 13690.000000 | 13998.000000 | 11382.000000 |

```
disp_id       0
client_id     0
account_id    0
type          0
dtype: int64
Duplicate Count 0


Dataframe name: loan_df with number of rows:682 and columns:7
```

|  | loan_id | account_id | amount | duration | payments |
|---|---|---|---|---|---|
| **count** | 682.000000 | 682.000000 | 682.000000 | 682.000000 | 682.000000 |
| **mean** | 6172.466276 | 5824.162757 | 151410.175953 | 36.492669 | 4190.664223 |
| **std** | 682.579279 | 3283.512681 | 113372.406310 | 17.075219 | 2215.830344 |
| **min** | 4959.000000 | 2.000000 | 4980.000000 | 12.000000 | 304.000000 |
| **25%** | 5577.500000 | 2967.000000 | 66732.000000 | 24.000000 | 2477.000000 |
| **50%** | 6176.500000 | 5738.500000 | 116928.000000 | 36.000000 | 3934.000000 |
| **75%** | 6752.500000 | 8686.000000 | 210654.000000 | 48.000000 | 5813.500000 |
| **max** | 7308.000000 | 11362.000000 | 590820.000000 | 60.000000 | 9910.000000 |

```
loan_id       0
account_id    0
date          0
amount        0
duration      0
payments      0
status        0
dtype: int64
Duplicate Count 0


Dataframe name: order_df with number of rows:6471 and columns:6
```

|  | order_id | account_id | account_to | amount |
|---|---|---|---|---|
| **count** | 6471.000000 | 6471.000000 | 6.471000e+03 | 6471.000000 |
| **mean** | 33778.197497 | 2962.302890 | 4.939904e+07 | 3280.635698 |
| **std** | 3737.681949 | 2518.503228 | 2.888356e+07 | 2714.475335 |
| **min** | 29401.000000 | 1.000000 | 3.990000e+02 | 1.000000 |
| **25%** | 31187.500000 | 1223.000000 | 2.415918e+07 | 1241.500000 |
| **50%** | 32988.000000 | 2433.000000 | 4.975606e+07 | 2596.000000 |
| **75%** | 34785.500000 | 3645.500000 | 7.400045e+07 | 4613.500000 |
| **max** | 46338.000000 | 11362.000000 | 9.999420e+07 | 14882.000000 |

```
order_id      0
account_id    0
bank_to       0
account_to    0
amount        0
k_symbol      0
dtype: int64
Duplicate Count 0


Dataframe name: trans_df with number of rows:1056320 and columns:10
```

| | trans_id | account_id | amount | balance | account |
|---|---|---|---|---|---|
| **count** | 1.056320e+06 | 1.056320e+06 | 1.056320e+06 | 1.056320e+06 | 2.953890e+05 |
| **mean** | 1.335311e+06 | 2.936867e+03 | 5.924146e+03 | 3.851833e+04 | 4.567092e+07 |
| **std** | 1.227487e+06 | 2.477345e+03 | 9.522735e+03 | 2.211787e+04 | 3.066340e+07 |
| **min** | 1.000000e+00 | 1.000000e+00 | 0.000000e+00 | -4.112570e+04 | 0.000000e+00 |
| **25%** | 4.302628e+05 | 1.204000e+03 | 1.359000e+02 | 2.240250e+04 | 1.782858e+07 |
| **50%** | 8.585065e+05 | 2.434000e+03 | 2.100000e+03 | 3.314340e+04 | 4.575095e+07 |
| **75%** | 2.060979e+06 | 3.660000e+03 | 6.800000e+03 | 4.960362e+04 | 7.201341e+07 |
| **max** | 3.682987e+06 | 1.138200e+04 | 8.740000e+04 | 2.096370e+05 | 9.999420e+07 |

```
trans_id           0
account_id         0
date               0
type               0
operation     183114
amount             0
balance            0
k_symbol      481881
bank          782812
account       760931
dtype: int64
Duplicate Count 0
```

## Defining a dataframe that stores loan transactions and other important information

In [8]:
```
df = pd.merge(trans_df,
         loan_df,
         on="account_id",
         suffixes=["_Trans", "_Loan"],
         how="inner")
```

In [9]:

```
# length of "df", columns, memory usage
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 191556 entries, 0 to 191555
Data columns (total 16 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   trans_id      191556 non-null  int64
 1   account_id    191556 non-null  int64
 2   date_Trans    191556 non-null  datetime64[ns]
 3   type          191556 non-null  object
 4   operation     160218 non-null  object
 5   amount_Trans  191556 non-null  float64
 6   balance       191556 non-null  float64
 7   k_symbol      99109 non-null   object
 8   bank          50513 non-null   object
 9   account       62625 non-null   float64
 10  loan_id       191556 non-null  int64
 11  date_Loan     191556 non-null  datetime64[ns]
 12  amount_Loan   191556 non-null  int64
 13  duration      191556 non-null  int64
 14  payments      191556 non-null  float64
 15  status        191556 non-null  object
dtypes: datetime64[ns](2), float64(4), int64(5), object(5)
memory usage: 24.8+ MB
```

In [10]:

```
# deleting redunant columns
del df["trans_id"]
del df["account_id"]
del df["loan_id"]
del df["bank"]
del df["account"]
```

In [11]:

```
# dropping duplicated
df.drop_duplicates(inplace =True)
```

## Translating from Czech to english,

'POJISTNE': "Insurace_Payment",

'SLUZBY': "Payment_for_Statement",

'UROK': "Intrest_Credited",

'SANKC. UROK':"Sanction Intrest If Negative",

'SIPO':"Household",

'UVER':"Loan Payment",

" ": "Old Age Pension

In [12]:

```
df["k_symbol"] = df["k_symbol"].map({'POJISTNE': "Insurace_Payment",
                                     'SLUZBY': "Payment_for_Statement",
                                     'UROK': "Intrest_Credited",
                                    'SANKC. UROK':"Sanction Intrest If Negative",
                                    'SIPO':"Household",
                                    'UVER':"Loan Payment",
                                    " ":"Old Age Pension",
                                    np.nan : "NULL"})
```

In [13]:

```
df.rename(columns={"operation":"Mode of Transaction"}, inplace=True)
```

### Translating from Czech to english

VKLAD: Credit In Cash,

PREVOD Z UCTU: Collection From Another Bank

np.nan : Null

VYBER: Withdrawal In Cash

PREVOD NA UCET: Remittance To Another Bank

VYBER KARTOU: Credit Card Withdrawal

In [14]:

```
df["Mode of Transaction"] = df["Mode of Transaction"].map({"VKLAD":"Credit In Cash",
                                                           "PREVOD Z UCTU":"Collection From Another Bank",
                                                           np.nan : "Null" ,'VYBER':"Withdrawal In Cash",
                                                           'PREVOD NA UCET':"Remittance To Another Bank",
                                                           'VYBER KARTOU':"Credit Card Withdrawal"})
```

### Translating from Czech to english

PRIJEM: Credit +

VYDAJ: Withdrawn -

VYBER: NULL

In [15]:

```
df["type"] = df["type"].map({"PRIJEM":"Credit +","VYDAJ":"Withdrawn -","VYBER":"NULL"})
df["status"] = df["status"].map({'A':"Loan Paid", 'B':"Loan Not Paid", 'D':"Client In Debt", 'C':"Currently Paying Loan"})
```

In [16]:

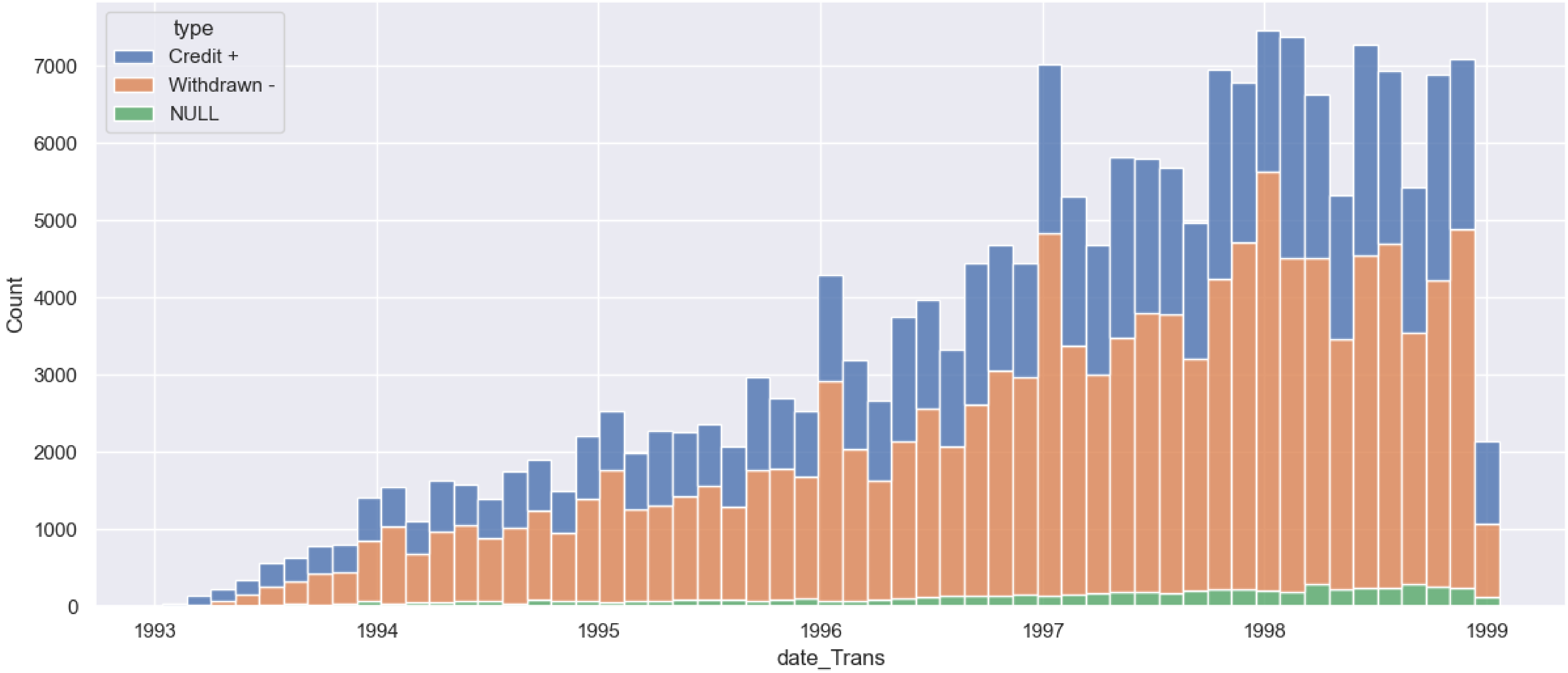
```
# percentage of null values in dataframe
round(df.isnull().sum()/df.shape[0] *100).sort_values(ascending=False)
```

Out[16]:

```
date_Trans             0.0
type                   0.0
Mode of Transaction    0.0
amount_Trans           0.0
balance                0.0
k_symbol               0.0
date_Loan              0.0
amount_Loan            0.0
duration               0.0
payments               0.0
status                 0.0
dtype: float64
```

In [17]:

```
sns.set(rc={'figure.figsize':(14.5,6)})
sns.histplot(data = df,
             x ="date_Trans",
             alpha=0.8,
             color="#55905d",
             hue="type",
             multiple="stack",
             element="bars",
             binwidth=40,
             bins=20);
```



In [18]:

```
yr = df.groupby([df.date_Trans.dt.year,df.type]).size()
```

In [19]:

```
yr = yr.groupby(level=0).apply(lambda x: 100 * x / float(x.sum()))
yr = pd.DataFrame(yr)
```

```
C:\Users\abdul\AppData\Local\Temp\ipykernel_8708\3033872491.py:1: FutureWarning: Not prepending group keys to the result index of transform-like
apply. In the future, the group keys will be included in the index, regardless of whether the applied function returns a like-indexed object.
To preserve the previous behavior, use

        >>> .groupby(..., group_keys=False)

To adopt the future behavior and silence this warning, use

        >>> .groupby(..., group_keys=True)
  yr = yr.groupby(level=0).apply(lambda x: 100 * x / float(x.sum()))
```

In [20]:

```
round(yr.sort_values(by=["date_Trans", "type"], ascending=False),1)
```

Out[20]:

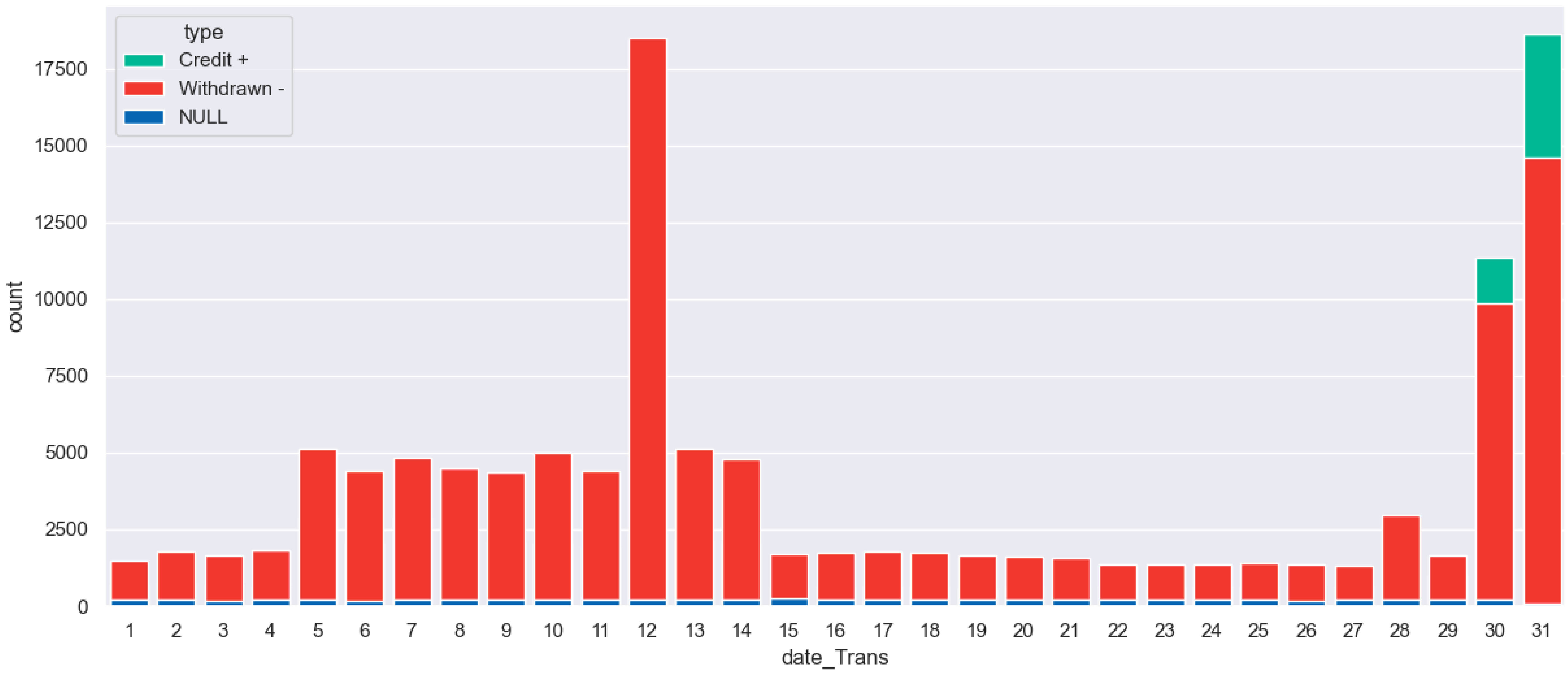
|  |  | 0 |
|---|---|---|
| date_Trans | type |  |
| 1998 | Withdrawn - | 62.3 |
|  | NULL | 3.7 |
|  | Credit + | 34.1 |
| 1997 | Withdrawn - | 61.8 |
|  | NULL | 3.2 |
|  | Credit + | 35.0 |
| 1996 | Withdrawn - | 60.3 |
|  | NULL | 2.9 |
|  | Credit + | 36.8 |
| 1995 | Withdrawn - | 60.3 |
|  | NULL | 3.1 |
|  | Credit + | 36.6 |
| 1994 | Withdrawn - | 59.7 |
|  | NULL | 3.6 |
|  | Credit + | 36.7 |
| 1993 | Withdrawn - | 46.9 |
|  | NULL | 3.3 |
|  | Credit + | 49.8 |

In [21]:

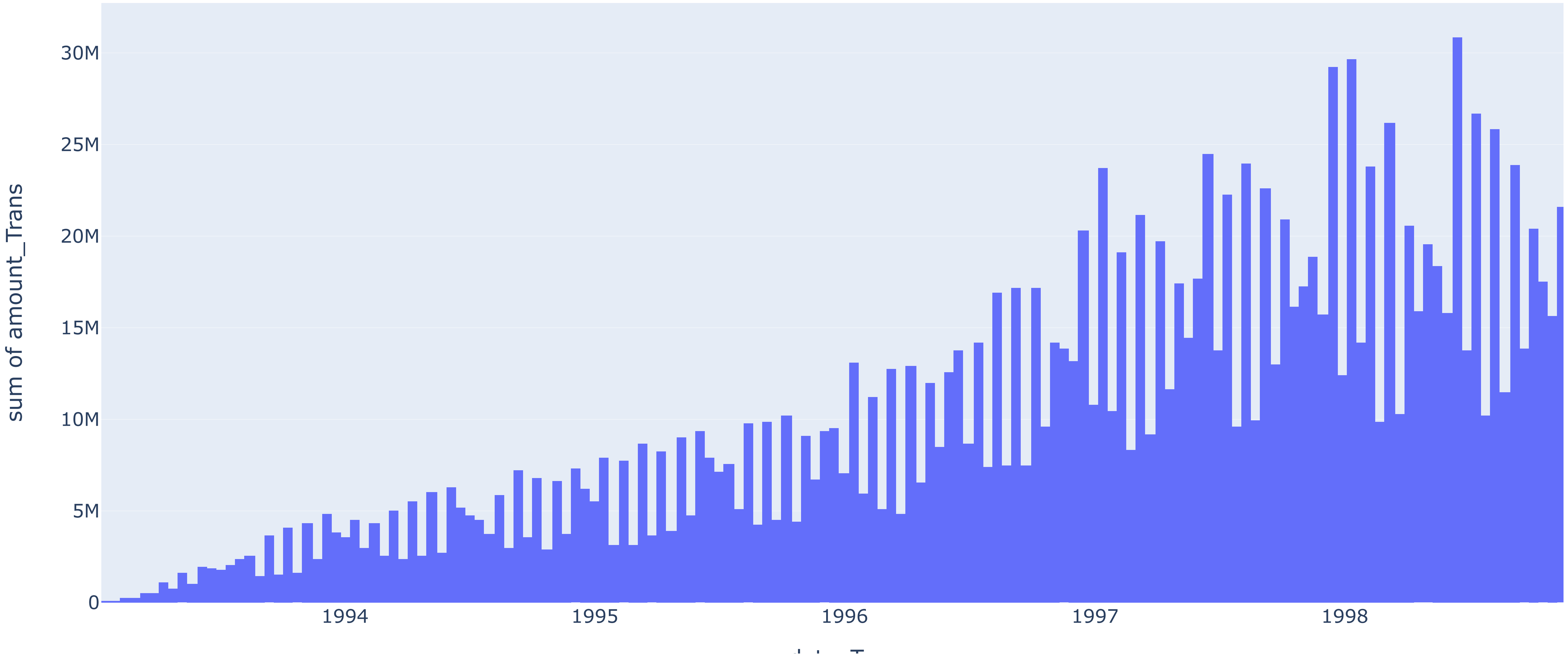
```
v = df.date_Trans.dt.day
```

In [22]:

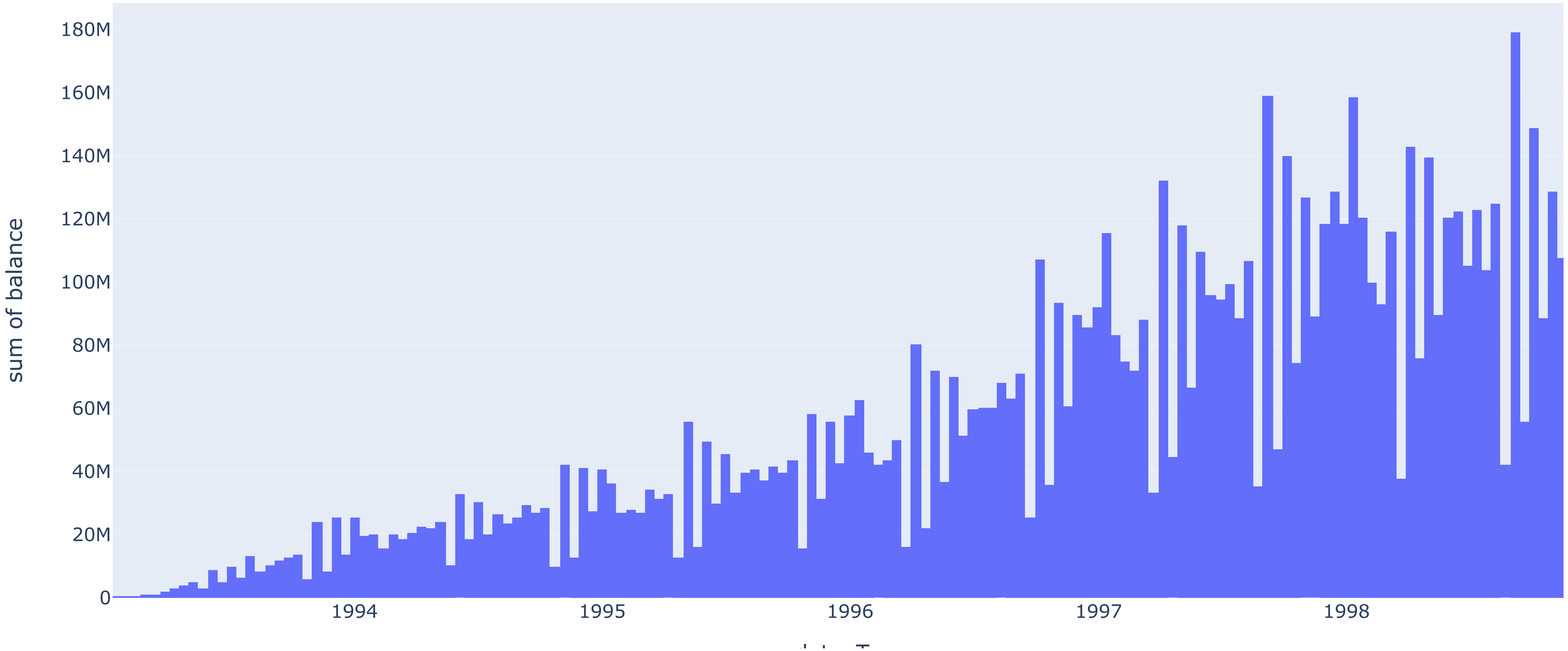
```
sns.countplot(x=v, hue=df.type, dodge=False, palette=["#00b894", "#f2372f", "#0666b2"],saturation=1)
plt.show();
```



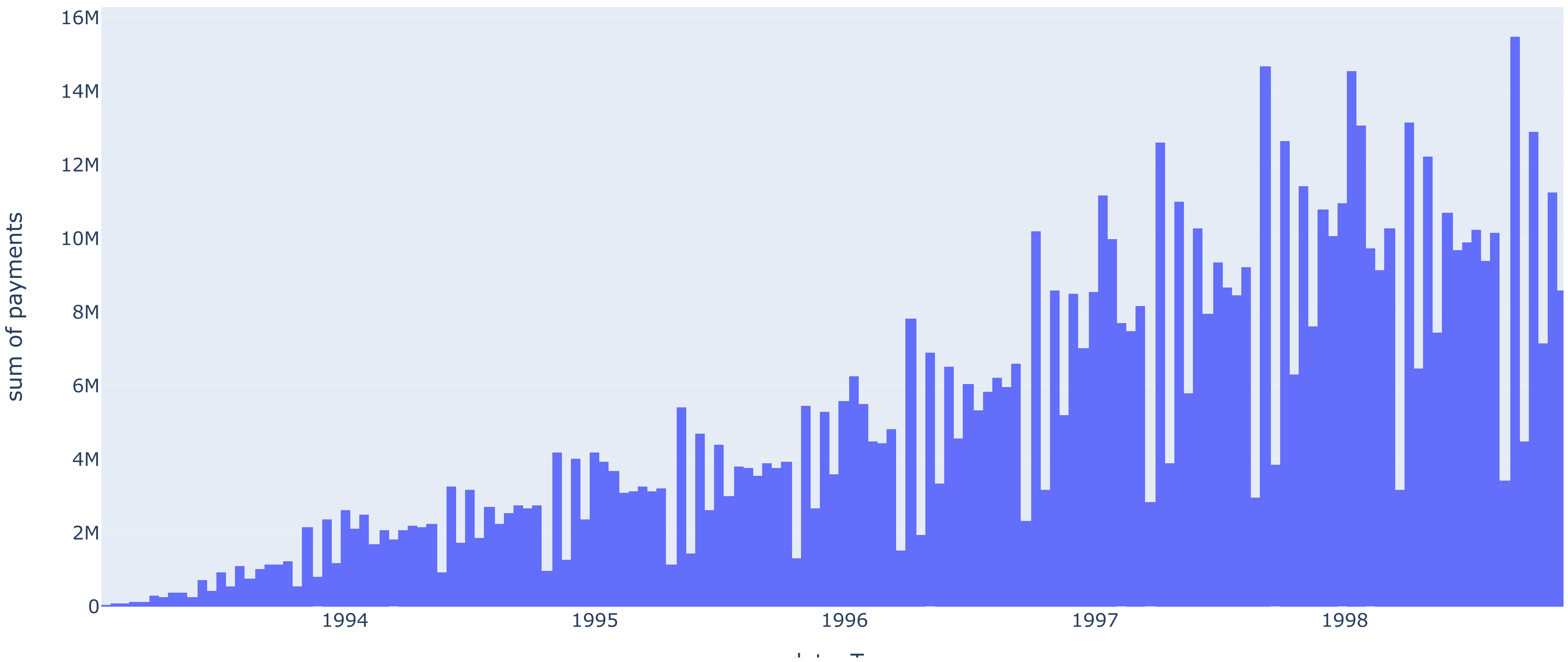
```
In [23]: px.histogram(df, x="date_Trans",y="amount_Trans")
```



```
In [24]: px.histogram(df, x="date_Trans",y="balance")
```

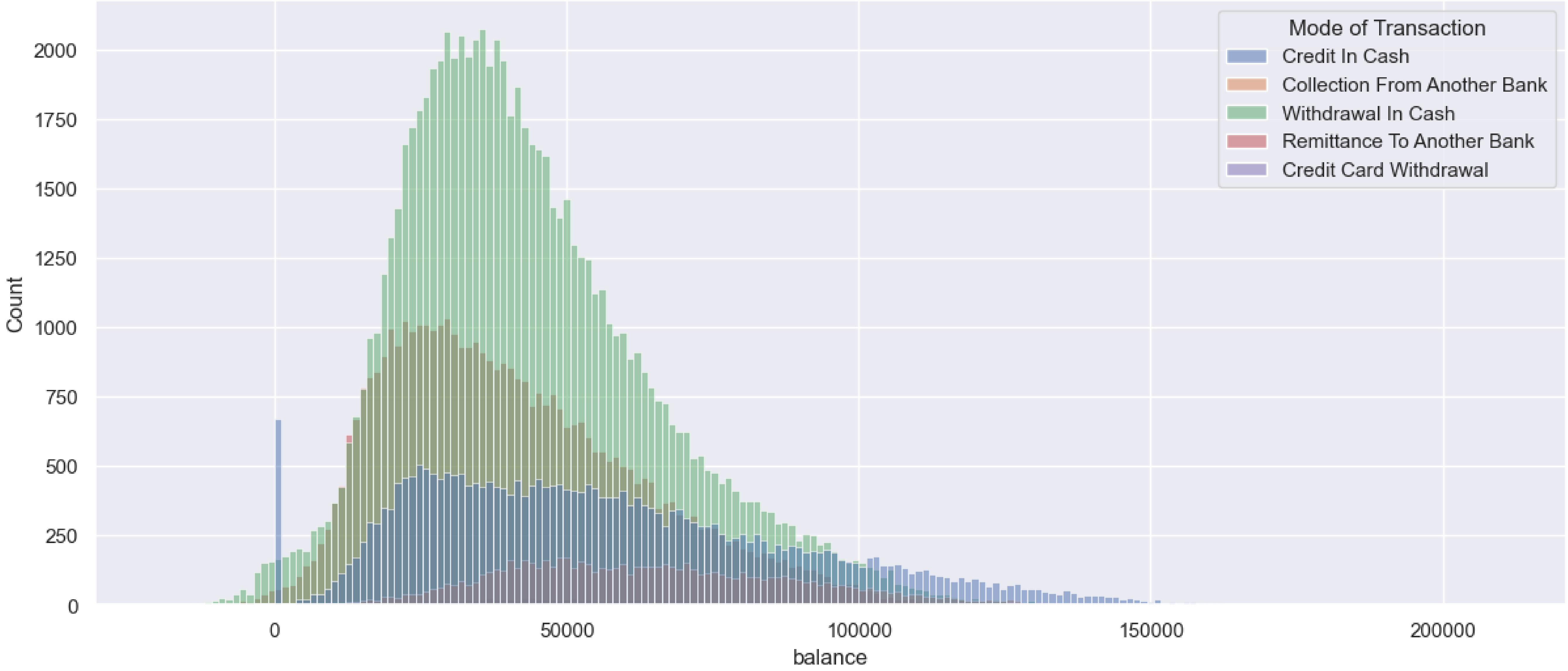


```
In [25]: px.histogram(df, x="date_Trans",y="payments")
```



```
In [26]: sns.histplot(data=df[df["Mode of Transaction"] != "Null"], x=df.balance,
                      hue="Mode of Transaction")
```

```
Out[26]: <AxesSubplot: xlabel='balance', ylabel='Count'>
```

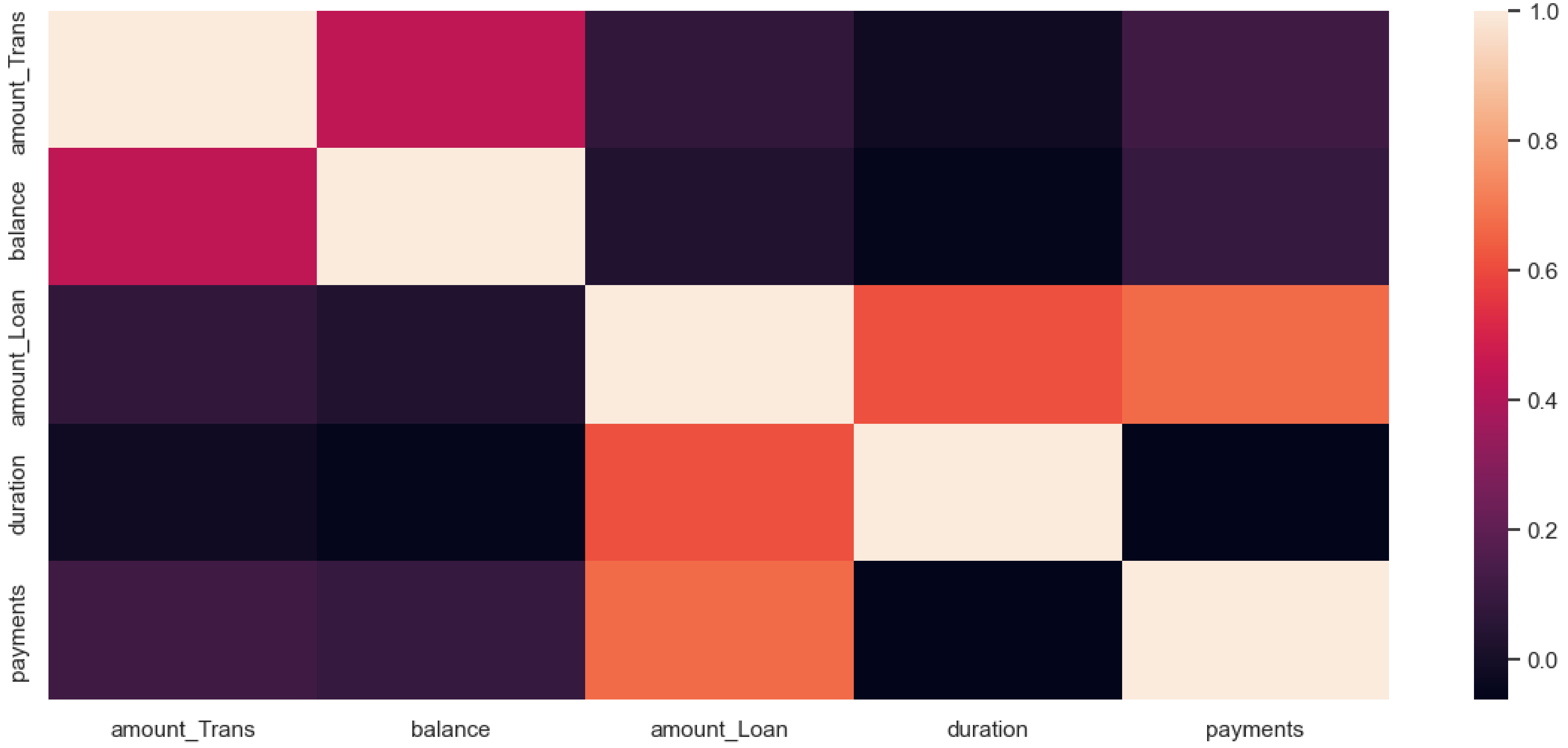


```
In [27]: sns.heatmap(df.corr())
```

```
C:\Users\abdul\AppData\Local\Temp\ipykernel_8708\58359773.py:1: FutureWarning:

The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns or s
pecify the value of numeric_only to silence this warning.
```

```
Out[27]: <AxesSubplot: >
```



In [28]:

```
nun=df.query('k_symbol!= "NULL"')
sns.lineplot(data=nun,
             x=nun.duration,
             y=nun.index,
             ci=None,
             size=nun.k_symbol,
             sizes=([7.5,6.5,5.5,4.5,3.5,2.5,1.5]),
             hue=nun.k_symbol,estimator="count")
```

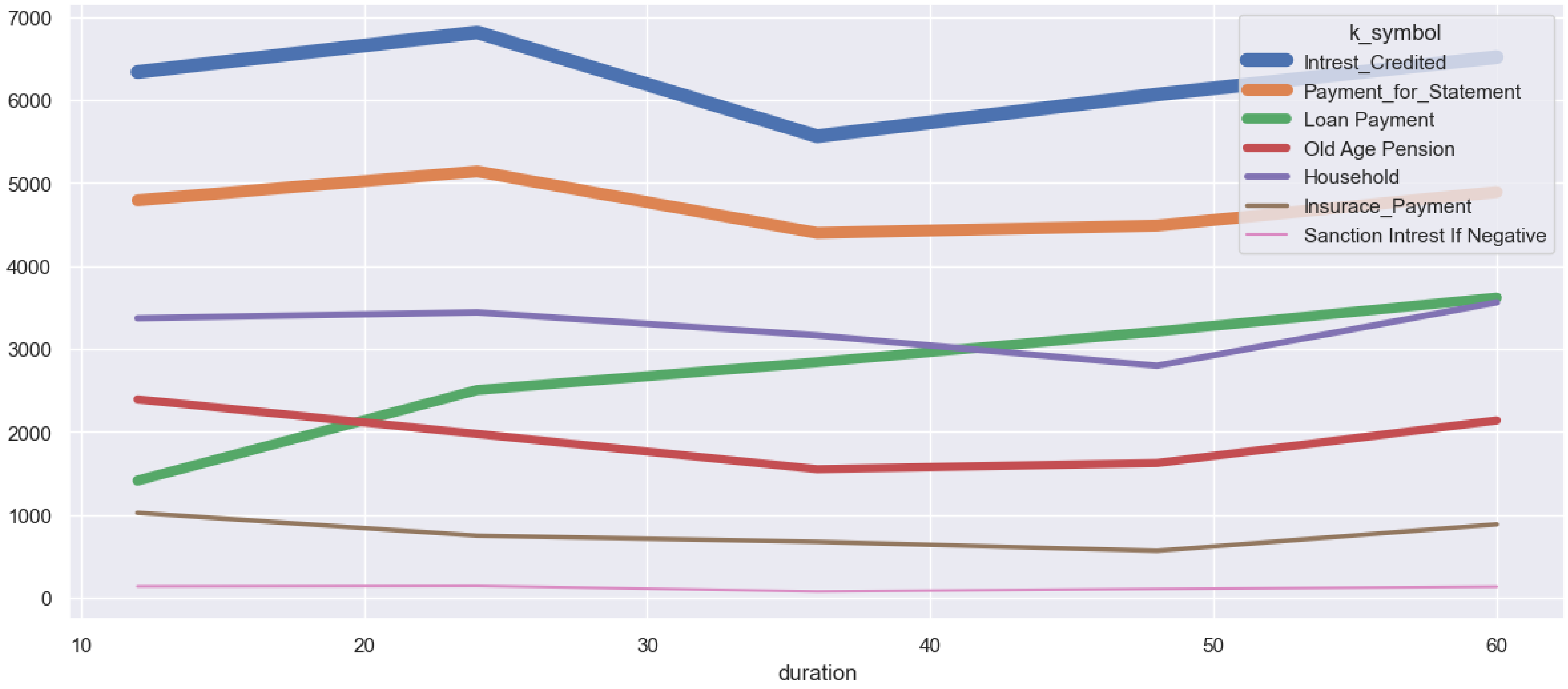
```
C:\Users\abdul\AppData\Local\Temp\ipykernel_8708\890230970.py:2: FutureWarning:



The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.
```

Out[28]:

```
<AxesSubplot: xlabel='duration'>
```

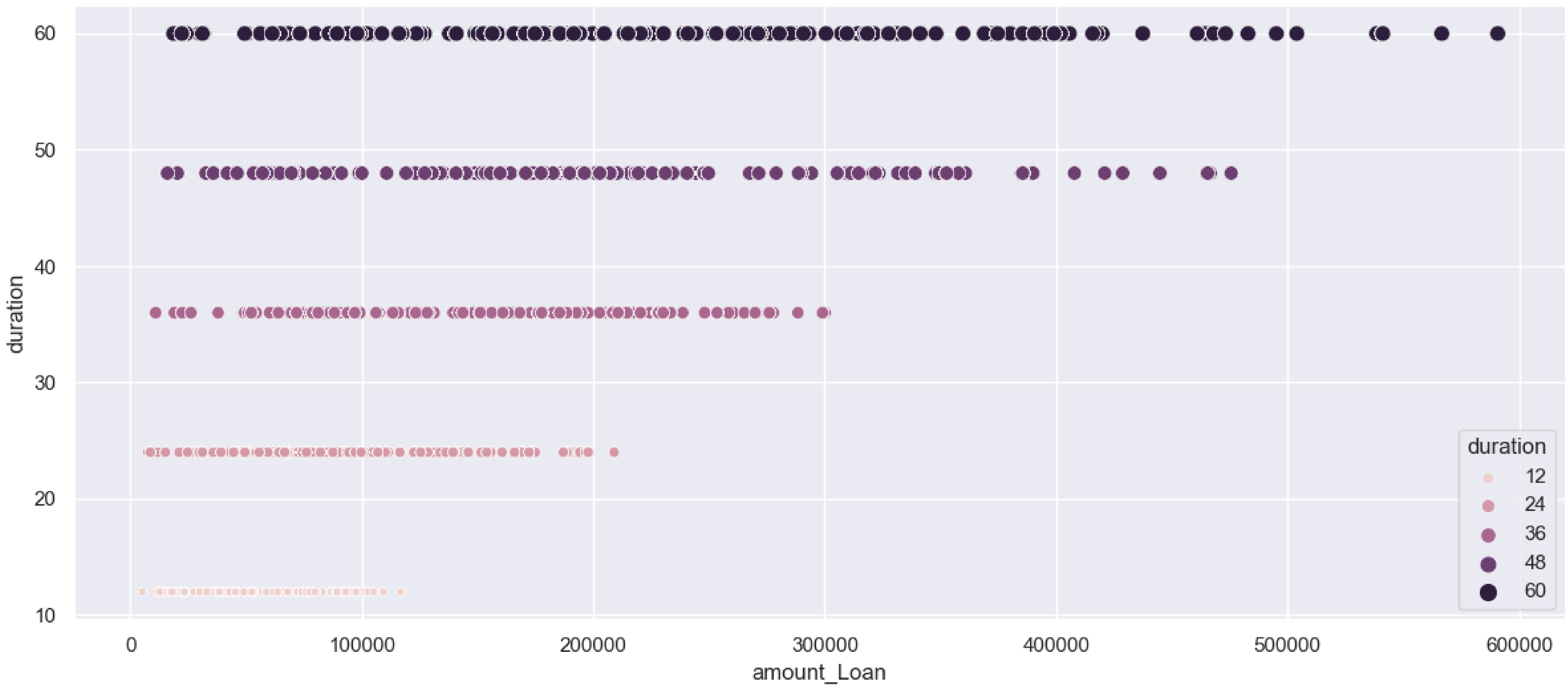


In [29]:

```
sns.scatterplot(x=df.amount_Loan, y=df.duration, size=df["duration"], hue=df["duration"])
```

Out[29]:

```
<AxesSubplot: xlabel='amount_Loan', ylabel='duration'>
```



In [30]:

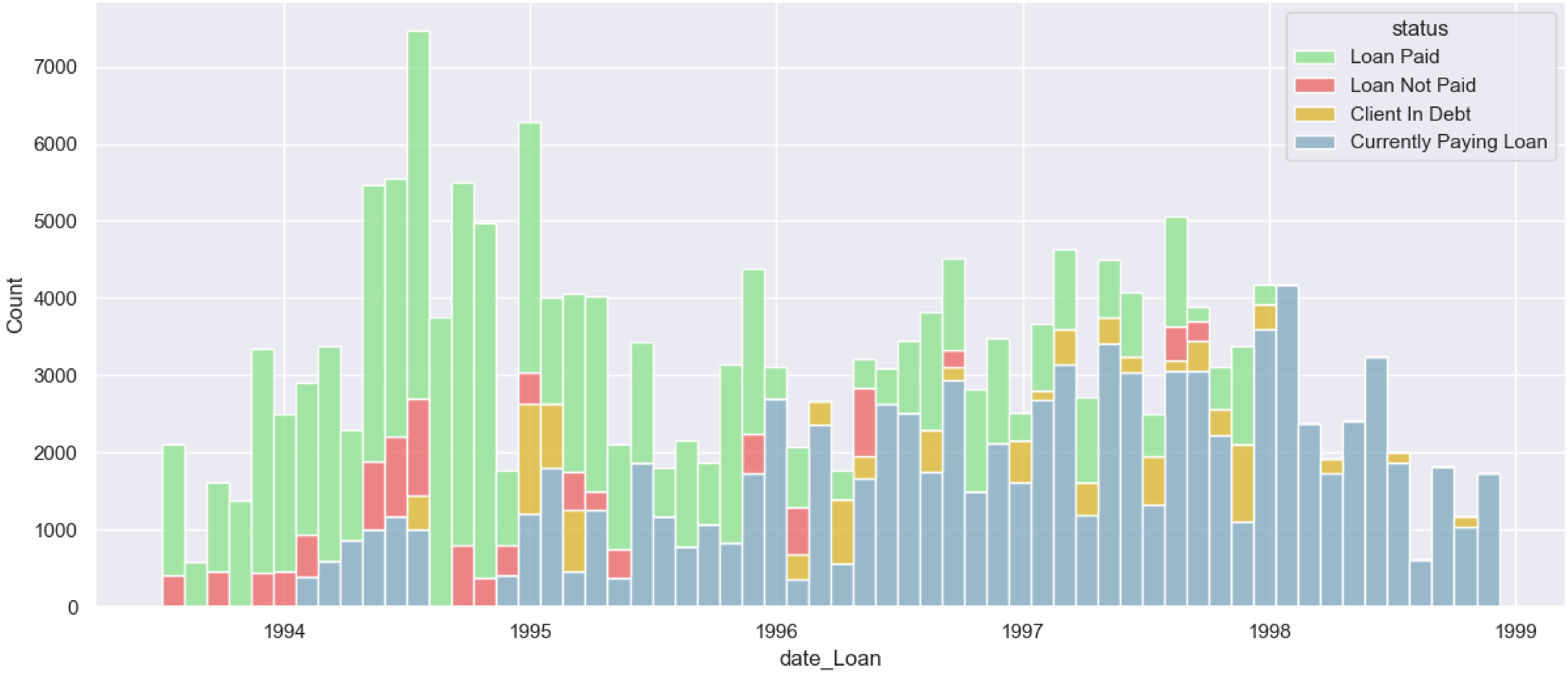
```
df["duration"].groupby(df["k_symbol"]).describe()
```

Out[30]:

| k_symbol | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **Household** | 16328.0 | 35.807447 | 17.303495 | 12.0 | 24.0 | 36.0 | 48.0 | 60.0 |
| **Insurace_Payment** | 3891.0 | 34.578258 | 18.135104 | 12.0 | 12.0 | 36.0 | 48.0 | 60.0 |
| **Intrest_Credited** | 31306.0 | 35.850125 | 17.199162 | 12.0 | 24.0 | 36.0 | 48.0 | 60.0 |
| **Loan Payment** | 13580.0 | 40.523417 | 15.919914 | 12.0 | 24.0 | 48.0 | 60.0 | 60.0 |
| **NULL** | 92354.0 | 35.020032 | 17.155613 | 12.0 | 24.0 | 36.0 | 48.0 | 60.0 |
| **Old Age Pension** | 9674.0 | 34.934464 | 17.945529 | 12.0 | 24.0 | 36.0 | 48.0 | 60.0 |
| **Payment_for_Statement** | 23706.0 | 35.770185 | 17.137143 | 12.0 | 24.0 | 36.0 | 48.0 | 60.0 |
| **Sanction Intrest If Negative** | 589.0 | 35.083192 | 17.949330 | 12.0 | 24.0 | 36.0 | 48.0 | 60.0 |

```
In [31]: sns.histplot(x=df.date_Loan, hue=df.status, multiple ='stack', palette=["#89e289","#ee6262","#dcb526","#7ea9bf"])
```

```
Out[31]: <AxesSubplot: xlabel='date_Loan', ylabel='Count'>
```



```
In [32]: nun = nun.k_symbol.value_counts()/99074 *100
```

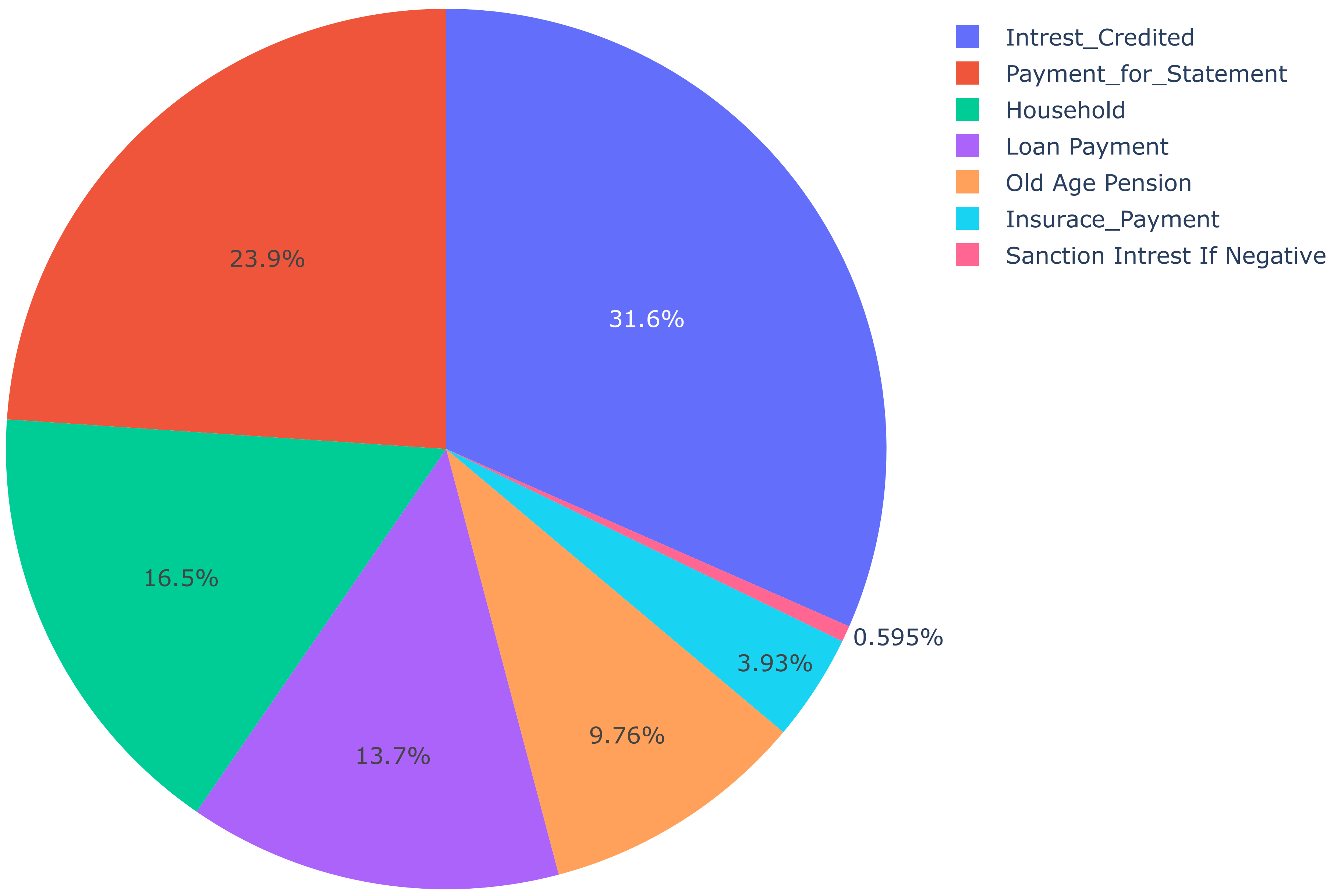
```
In [33]: pd.DataFrame(nun)
```

Out[33]:

| | k_symbol |
|---|---|
| **Intrest_Credited** | 31.598603 |
| **Payment_for_Statement** | 23.927569 |
| **Household** | 16.480610 |
| **Loan Payment** | 13.706926 |
| **Old Age Pension** | 9.764419 |
| **Insurace_Payment** | 3.927367 |
| **Sanction Intrest If Negative** | 0.594505 |

In [34]:

```
px.pie(nun, values="k_symbol", names=nun.index, width=800, height=600)
```



## Defining a dataframe that stores information on transactions related to accounts, orders, dispositions, clients, Cards

In [35]:

```
df2 = pd.merge(accounts_df,
         order_df,
         on="account_id",
         suffixes=["_Account","_Orders"],
         how="outer").merge(dispos_df,
                      on="account_id",
                            suffixes=["_Dispos","_Dispos"],
                            how="outer").merge(clients_df,
                                              on="client_id",
                                              suffixes=["_Clients","_Clients"],
                                              how="outer").merge(cards_df,
                                                                 on="disp_id",
                                                                 suffixes=["_Dispos","_Dispos"],
                                                                how="outer")
```

In [36]:

```
# dropping redunant columns

df2.drop(['district_id_Clients','account_id', 'order_id','client_id','card_id','disp_id','issued'], axis=1, inplace=True)
```

In [37]:

```
# Renaming and translating from czech to english
df2["frequency"] = df2["frequency"].map({"POPLATEK MESICNE":"Monthly Issuance",
                                         "POPLATEK TYDNE":"Weekly Issuance",
                                         "POPLATEK PO OBRATU":"Issuance after transactions"})
```

In [38]:

```
# translating and renaming from czech to english
df2["k_symbol"] = df2["k_symbol"].map({"SIPO":"Household Payment",
                                       "POJISTNE":"Insurance Payment",
                                       "UVER":"Loan Payment",
                                       "LEASING":"Leasing"})
```

In [39]:

```
# percentage of nulls
round(df2.isnull().sum()/df2.shape[0] *100)
```

Out[39]:

```
frequency        0.0
date             0.0
bank_to          9.0
account_to       9.0
amount           9.0
k_symbol        29.0
type_Dispos      0.0
birth_number     0.0
type_Dispos     83.0
dtype: float64
```

In [40]:

```
# dropping duplicates
df2.drop_duplicates(inplace = True)
```

In [41]:

```
df2.k_symbol.value_counts()
```

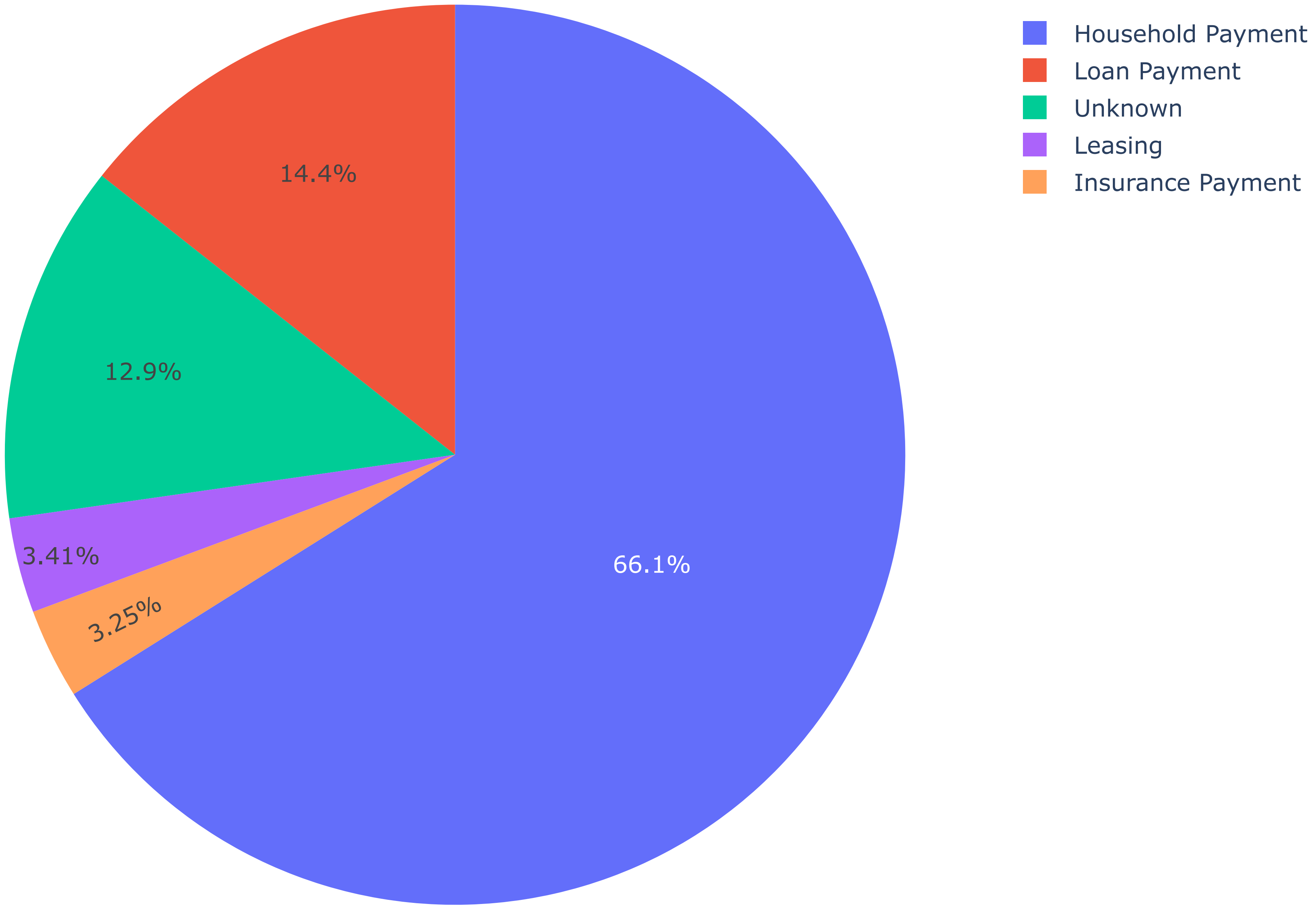
Out[41]:

```
Household Payment    4281
Loan Payment          860
Insurance Payment     640
Leasing               394
Name: k_symbol, dtype: int64
```

```
In [42]: # filling nulls
         df2.k_symbol.fillna("Unknown", inplace = True)
```

```
In [43]: nun2 = pd.DataFrame(df2.k_symbol.value_counts())
```

```
In [44]: px.pie(df2, values = df2["amount"], names=df2["k_symbol"] ,width=800, height=600)
```



```
In [45]: # amount equal to max
         df2.query("amount == amount.max()")
```

Out[45]:

| | frequency | date | bank_to | account_to | amount | k_symbol | type_Dispos | birth_number | type_Dispos |
|---|---|---|---|---|---|---|---|---|---|
| **7403** | Issuance after transactions | 1997-04-11 | ST | 30396717.0 | 14882.0 | Household Payment | OWNER | 465701 | NaN |
| **7405** | Issuance after transactions | 1997-04-11 | ST | 30396717.0 | 14882.0 | Household Payment | DISPONENT | 440719 | NaN |

In [46]:

```
# amount lesser than average
df2.query("amount < amount.mean()")
```

Out[46]:

| | frequency | date | bank_to | account_to | amount | k_symbol | type_Dispos | birth_number | type_Dispos |
|---|---|---|---|---|---|---|---|---|---|
| **3** | Monthly Issuance | 1993-01-01 | OP | 32659602.0 | 1474.0 | Unknown | OWNER | 350402 | NaN |
| **6** | Monthly Issuance | 1993-01-01 | OP | 32659602.0 | 1474.0 | Unknown | DISPONENT | 345404 | NaN |
| **8** | Monthly Issuance | 1993-01-01 | IJ | 15132719.0 | 2141.0 | Household Payment | OWNER | 450114 | NaN |
| **9** | Monthly Issuance | 1993-01-01 | UV | 96896516.0 | 1197.0 | Unknown | OWNER | 450114 | NaN |
| **10** | Monthly Issuance | 1993-01-01 | IJ | 15132719.0 | 2141.0 | Household Payment | DISPONENT | 535130 | NaN |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **8672** | Monthly Issuance | 1997-12-29 | CD | 4509616.0 | 929.0 | Unknown | OWNER | 630516 | NaN |
| **8673** | Monthly Issuance | 1997-12-29 | WX | 88365083.0 | 1017.0 | Insurance Payment | OWNER | 630516 | NaN |
| **8674** | Monthly Issuance | 1997-12-29 | IJ | 54098749.0 | 1722.0 | Household Payment | DISPONENT | 696007 | NaN |
| **8675** | Monthly Issuance | 1997-12-29 | CD | 4509616.0 | 929.0 | Unknown | DISPONENT | 696007 | NaN |
| **8676** | Monthly Issuance | 1997-12-29 | WX | 88365083.0 | 1017.0 | Insurance Payment | DISPONENT | 696007 | NaN |

4786 rows × 9 columns

In [53]:

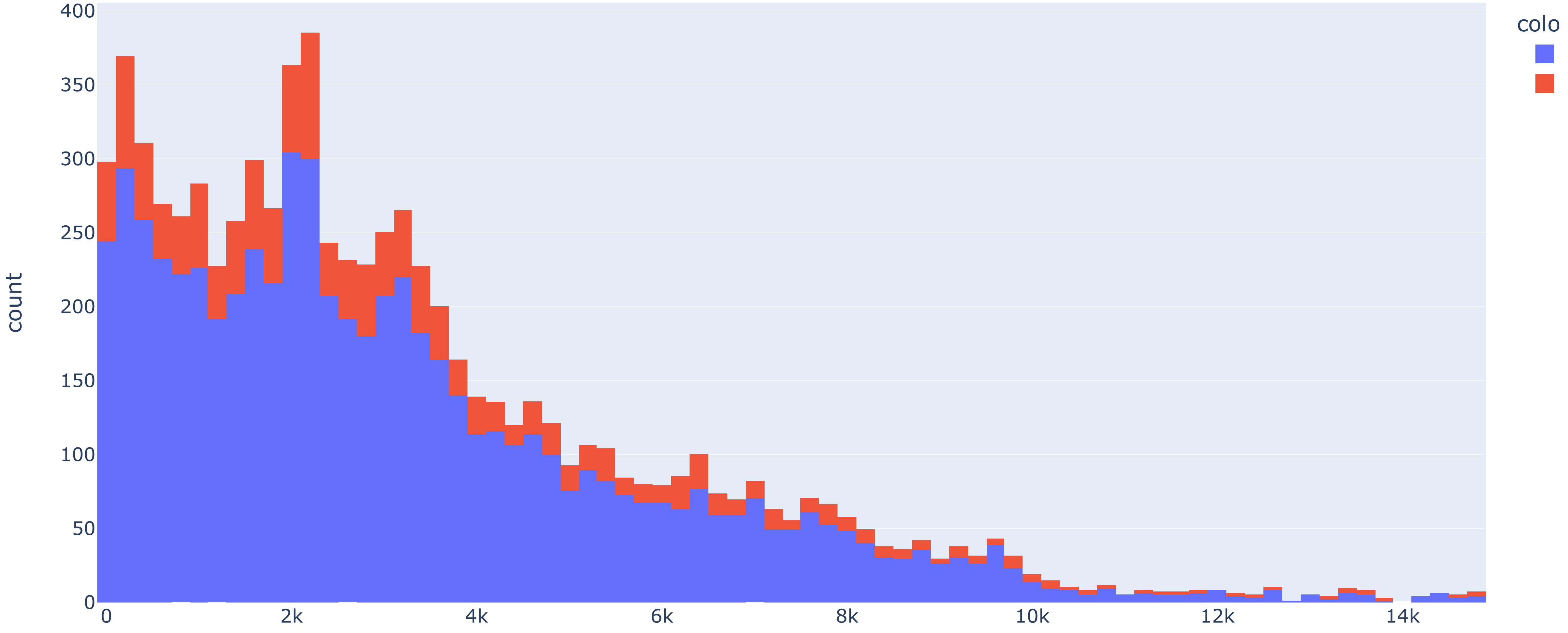
```
a = df2.iloc[::,6]
```

In [54]:

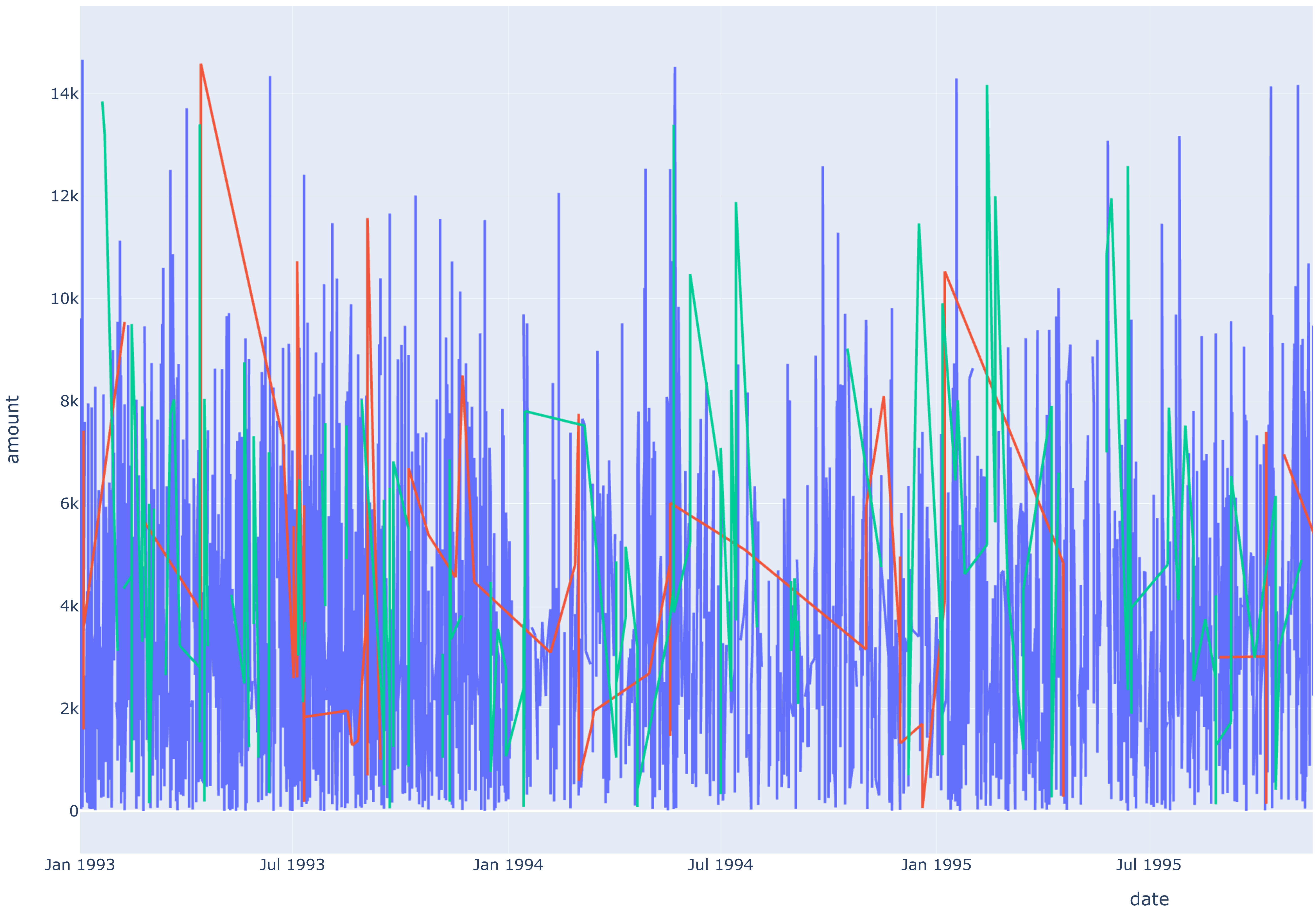
```
a = pd.DataFrame(a)
```

In [55]:

```
# distribuition of owner and disponent
px.histogram(x=df2['amount'], color=a["type_Dispos"])
```



```
In [92]: px.line(df2, x="date", y="amount", width=2000, height=800, color='frequency')
```


amount
14k
12k
10k
8k
6k
4k
2k
0
Jan 1993
Jul 1993
Jan 1994
Jul 1994
Jan 1995
Jul 1995
date

In [ ]:

In [ ]:

In [ ]: