Customer Segmentation with Credit Card Dataset

The Project goes through 5 steps:

1. Data Cleaning
2. Exploratory Data Analysis
3. Data preprocessing
4. Dimensionality Reduction with PCA
5. K-Means Clustering and Analysis

0. Import Data & Reading

- Import needed libraries to use in loading and reading the data

```
csv_file_path = '/content/CC GENERAL.csv'

df = pd.read_csv(csv_file_path)

df
```

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMI
0	C10001	40.900749	0.818182	95.40	0.00	
1	C10002	3202.467416	0.909091	0.00	0.00	

- Checking information about the data, rows number for each feature, and will appear if there are null values, also shows the data types of our Features..

```
df.info()
```

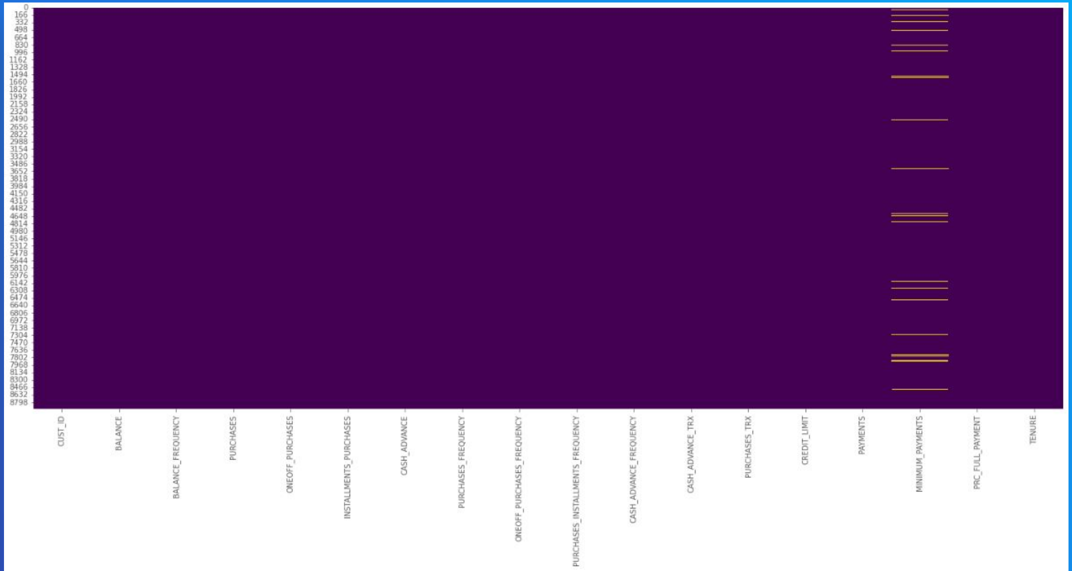
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8950 entries, 0 to 8949
Data columns (total 18 columns):
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
 0   CUST_ID                           8950 non-null   object
 1   BALANCE                           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
```

1. Data Cleaning

- To check null values, or if there is any missing data

	Data Type	Unique Values	Null Values	% null Values
MINIMUM_PAYMENTS	float64	8636	313	0.034972
CREDIT_LIMIT	float64	205	1	0.000112
CUST_ID	object	8950	0	0.000000
BALANCE	float64	8871	0	0.000000
PRC_FULL_PAYMENT	float64	47	0	0.000000
PAYMENTS	float64	8711	0	0.000000
PURCHASES_TRX	int64	173	0	0.000000
CASH_ADVANCE_TRX	int64	65	0	0.000000
CASH_ADVANCE_FREQUENCY	float64	54	0	0.000000
PURCHASES_INSTALLMENTS_FREQUENCY	float64	47	0	0.000000
ONEOFF_PURCHASES_FREQUENCY	float64	47	0	0.000000
PURCHASES_FREQUENCY	float64	47	0	0.000000
CASH_ADVANCE	float64	4323	0	0.000000
INSTALLMENTS_PURCHASES	float64	4452	0	0.000000
ONEOFF_PURCHASES	float64	4014	0	0.000000
PURCHASES	float64	6203	0	0.000000
BALANCE_FREQUENCY	float64	43	0	0.000000
TENURE	int64	7	0	0.000000

- **Visualizing Missing Values**



Dropping Unwanted Columns and Filling Null Values

- ‘CUST_ID' is a High Cardinality Feature with unique id for each customer , won't play any role in determining the cluster.

```
1  df.drop(columns='CUST_ID', inplace=True)
```

- 'CREDIT_LIMIT' has just 1 record with missing value. We can simply drop it.

```
[ ]  1  df.dropna(subset=['CREDIT_LIMIT'], inplace=True)
```

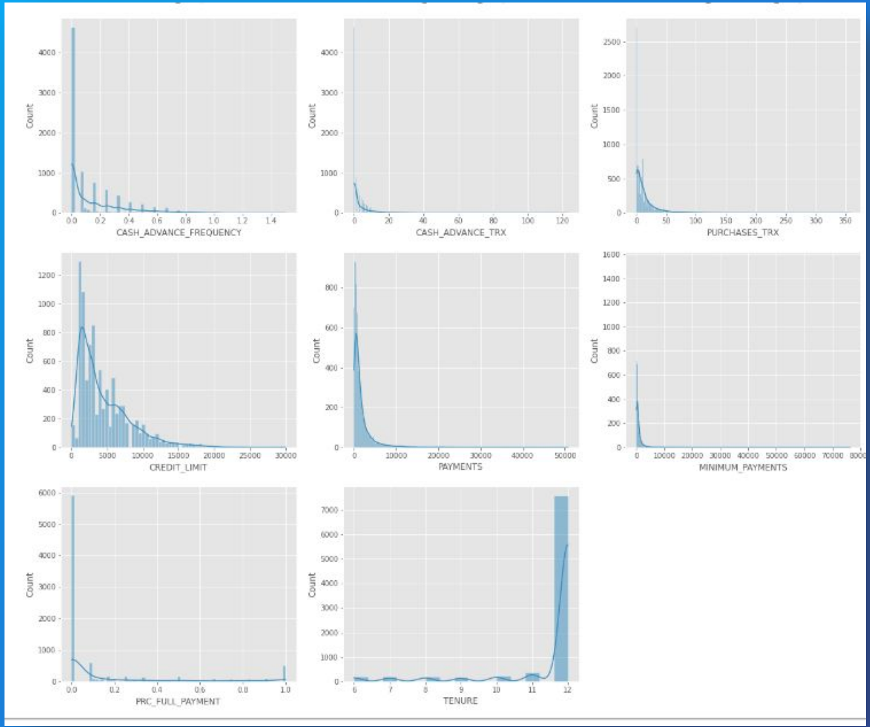
- 'MINIMUM_PAYMENTS' has 313 records with null values. We could drop it since it's only 0.3% of the columns.

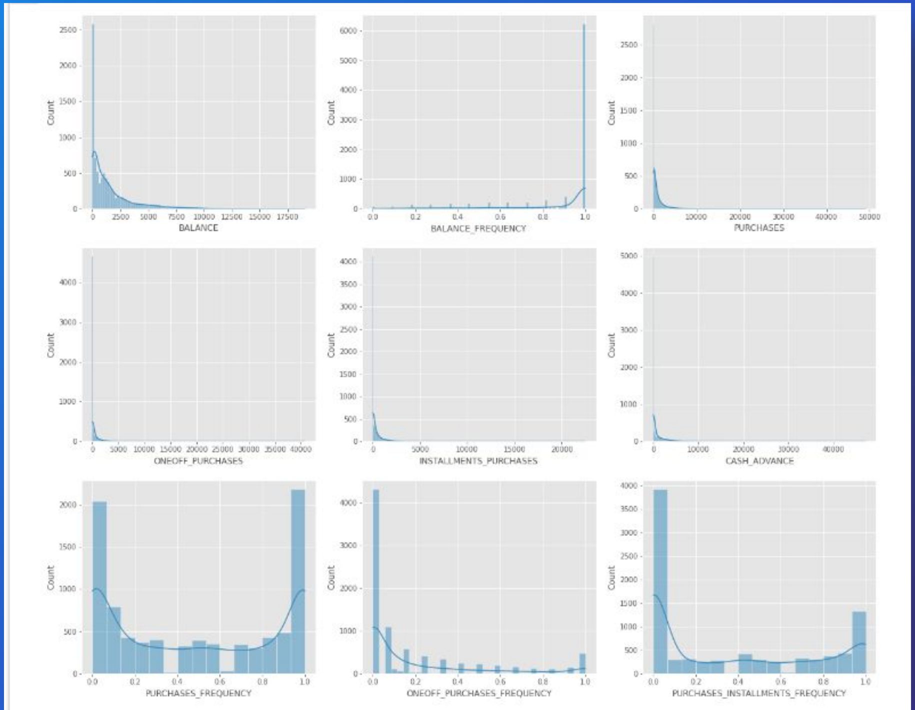
- But we could also impute it using the median value of the column, since it appear to have outliers like most of the features.

```
df['MINIMUM_PAYMENTS'].fillna( df['MINIMUM_PAYMENTS'].median(), inplace = True )
```

3. Exploratory Data Analysis

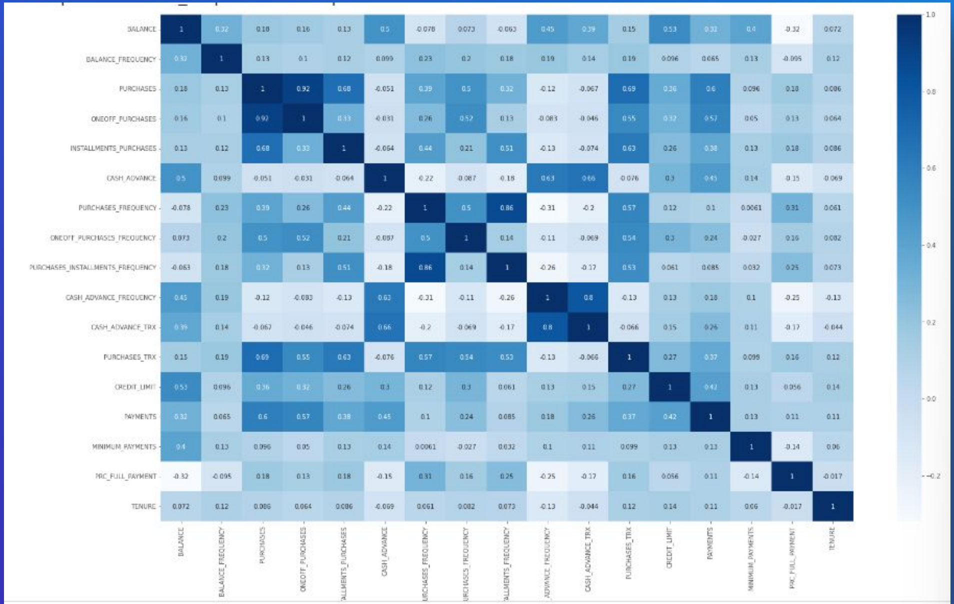
- Checking the Distribution of our features

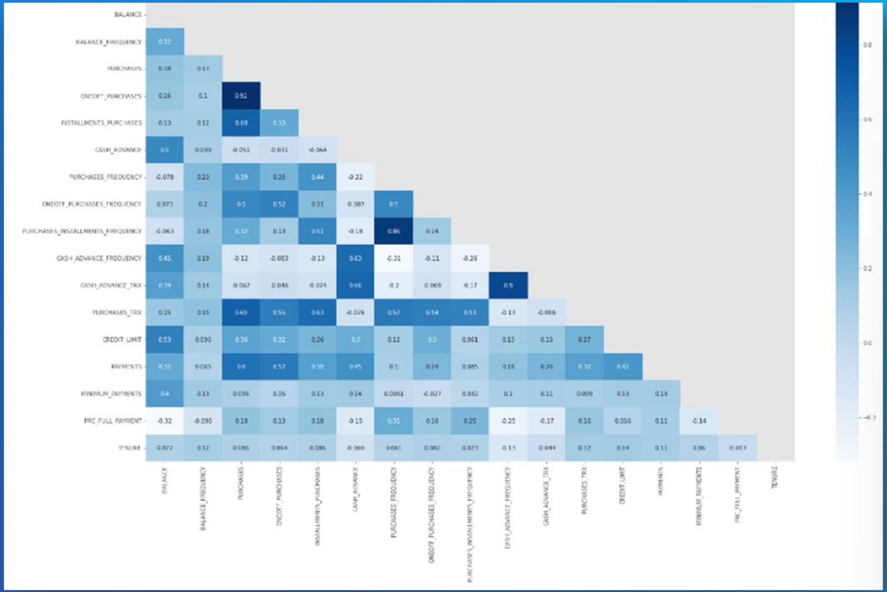




- Most of the Features are Right Skewed with Most of the values being (or close to) zero.

Looking for Correlations





- We have many Correlated Features as Most of them are just another way to represent existing ones like the frequency features.
- One way to handle this is Dimensionality Reduction with PCA.
- Latent Features can be constructed to incorporate data from multiple features.

4. Data Preprocessing

1. Removing Outliers

- We will first set all outliers as NaN, then impute the missing values.

```
for col in df.columns:

    data = df[col]

    Q1 = data.quantile(0.25)
    Q3 = data.quantile(0.75)

    IQR = Q3 - Q1

    min = Q1 - (1.5 * IQR)
    max = Q3 + (1.5 * IQR)

    outliers = ( (data < min) | (data > max) )

    df.loc[outliers, col] = np.nan

df.isna().sum()
```

```
BALANCE                              695
BALANCE_FREQUENCY                   1492
PURCHASES                            808
ONEOFF_PURCHASES                    1013
INSTALLMENTS_PURCHASES               867
CASH_ADVANCE                        1030
PURCHASES_FREQUENCY                    0
ONEOFF_PURCHASES_FREQUENCY           782
PURCHASES_INSTALLMENTS_FREQUENCY       0
CASH_ADVANCE_FREQUENCY               525
CASH_ADVANCE_TRX                     804
PURCHASES_TRX                        766
CREDIT_LIMIT                         248
PAYMENTS                             808
MINIMUM_PAYMENTS                     909
PRC_FULL_PAYMENT                    1474
TENURE                              1365
dtype: int64
```

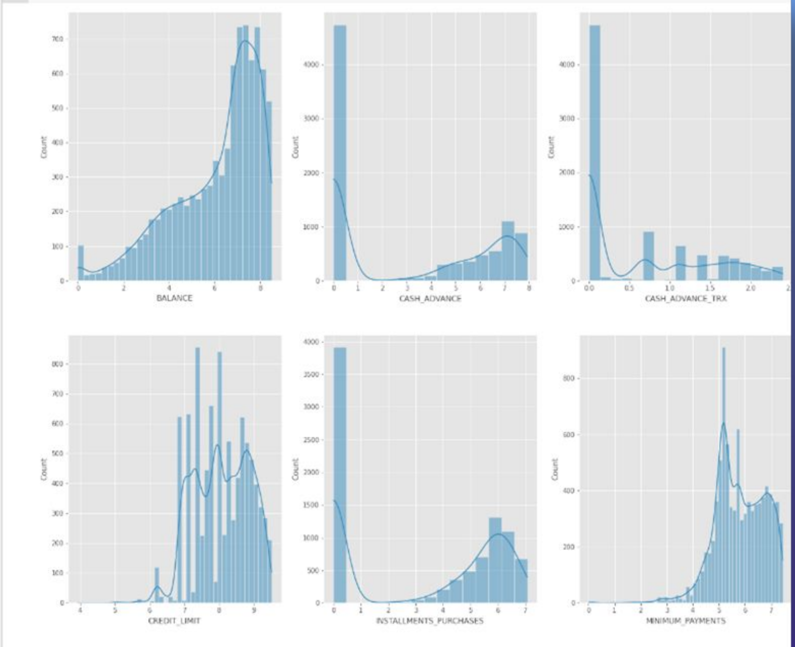
- KNN imputer: Each sample's missing values are imputed using the mean value from n_neighbors nearest neighbors found in the training set.

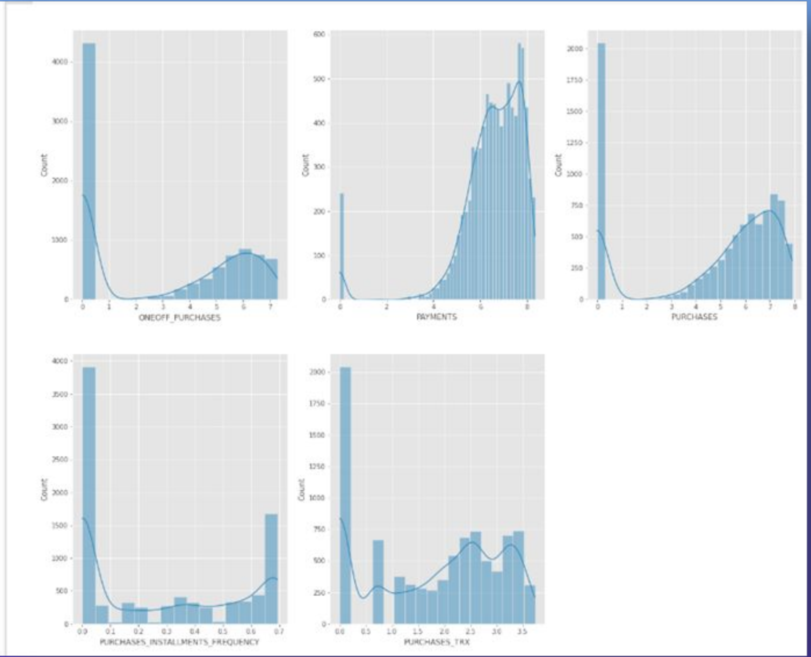
```
imputer = KNNImputer()

df = pd.DataFrame(imputer.fit_transform(df), columns=df.columns)

df.isna().sum()
```

2. Applying Log Transformation

- To handle the Skewness in our Features

- Not as Symmetric but better than the original Skewness





3. Scaling Features

- Standardizing with StandardScaler

```python
from sklearn.preprocessing import StandardScaler

Scaler = StandardScaler()
df_scaled = Scaler.fit_transform(trans_df)
```

- Changes Feature Values but keeps the same distribution

```python
df_scaled = pd.DataFrame(df_scaled, columns=df.columns)
df_scaled
```

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	ONEOFF_PURCHASES_FREQUENCY	PURCHASES_INSTALLMENTS_FREQUENCY	CASH_ADVANCE_FREQUENCY	CASH_ADVANCE_TRX	PURCHASES_TRX	CREDIT_LIMIT	PAYMENTS	MINIMUM_PAYMENTS	PRC_FULL_PAYMENT	TENURE
0	-1.218018	-2.647658	-0.075787	-0.996249	0.471989	-0.915486	-0.806649	-0.744814	-0.674357	-0.746426	-0.822942	-0.572416	-1.475368	-0.825161	-0.877686	-0.449298	0.0
1	1.025958	-1.140357	-1.713220	-0.996249	-1.103750	1.202677	-1.221928	-0.744814	-0.958359	1.029758	1.185864	-1.451115	0.980622	1.222599	1.229472	2.720734	0.0
2	0.896865	0.366944	0.670899	1.270704	-1.103750	-0.915486	1.269742	2.115946	-0.958359	-0.746426	-0.822942	0.600400	1.067726	-0.060919	0.674469	-0.449298	0.0
3	0.688170	0.065483	0.907966	0.959802	-1.103750	0.703732	-1.014290	-0.378051	-0.958359	-0.154367	0.042204	-0.896717	1.067726	-4.442181	-0.046152	-0.449298	0.0
4	0.320038	0.366944	-0.697747	-0.030681	-1.103750	-0.915486	-1.014290	-0.378051	-0.958359	-0.746426	-0.822942	-0.896717	-1.245363	-0.002050	-0.298155	-0.449298	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8944	-1.399709	0.366944	0.321575	-0.996249	0.854380	-0.915486	1.269742	-0.744814	1.192296	-0.746426	-0.822942	0.105276	-1.475368	-0.500727	-1.950854	-0.211547	0.0
8945	-1.595984	0.366944	0.332308	-0.996249	0.864709	-0.915486	1.269742	-0.744814	1.192296	-0.746426	-0.822942	0.105276	-1.475368	-0.613211	-0.046152	-0.449298	0.0
8946	-1.497837	-2.396449	0.071516	-0.996249	0.613742	-0.915486	0.854463	-0.744814	0.854123	-0.746426	-0.822942	-0.018018	-1.475368	-1.439466	-1.418046	3.116992	0.0
8947	-1.768619	-2.396449	-1.713220	-0.996249	-1.103750	0.185690	-1.221928	-0.744814	-0.958359	0.437699	0.548281	-1.451115	-2.349333	-1.731838	-1.817158	3.116992	0.0
8948	-0.085770	0.366944	0.794923	1.388633	-1.103750	0.558155	0.439186	2.189301	-0.958359	1.621817	0.548281	1.090778	-1.245363	-1.608701	-1.347565	-0.449298	0.0

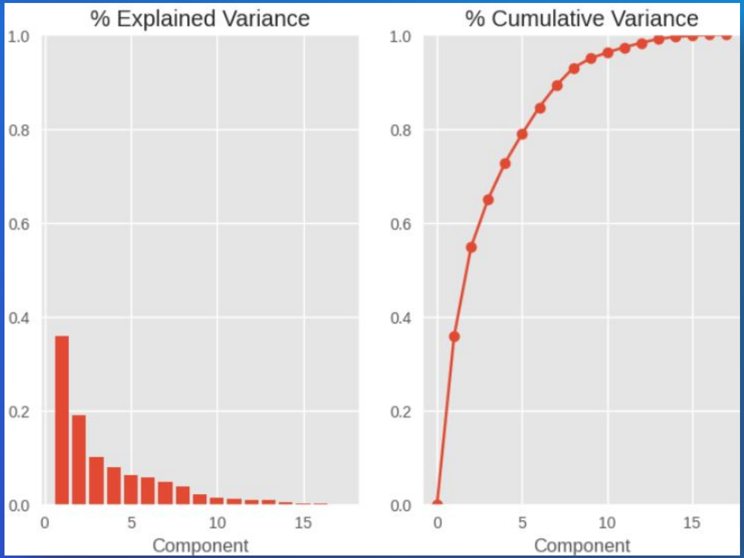
8949 rows × 17 columns

4. Dimensionality Reduction with PCA

- First, We will Make Components to all features in the data.
- Then decide how many components are needed based on the cumulative explained variance by the components.
- About 80% of the Variance in the data is explained by only 5 components.
- And About 90% of the variance in the data is explained by only 7 components.

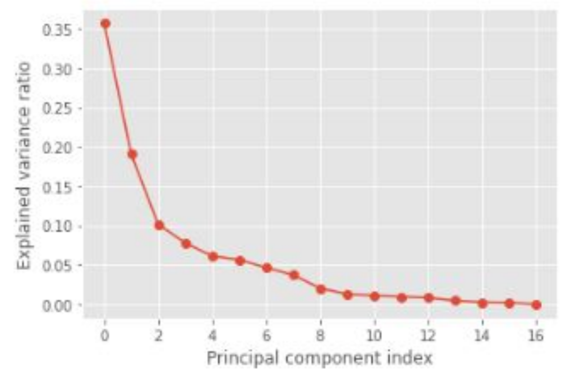
```
# Convert to dataframe

component_names = [f"PC{i+1}" for i in range(X_pca.shape[1])]

X_pca = pd.DataFrame(X_pca, columns=component_names)

X_pca.head()
```

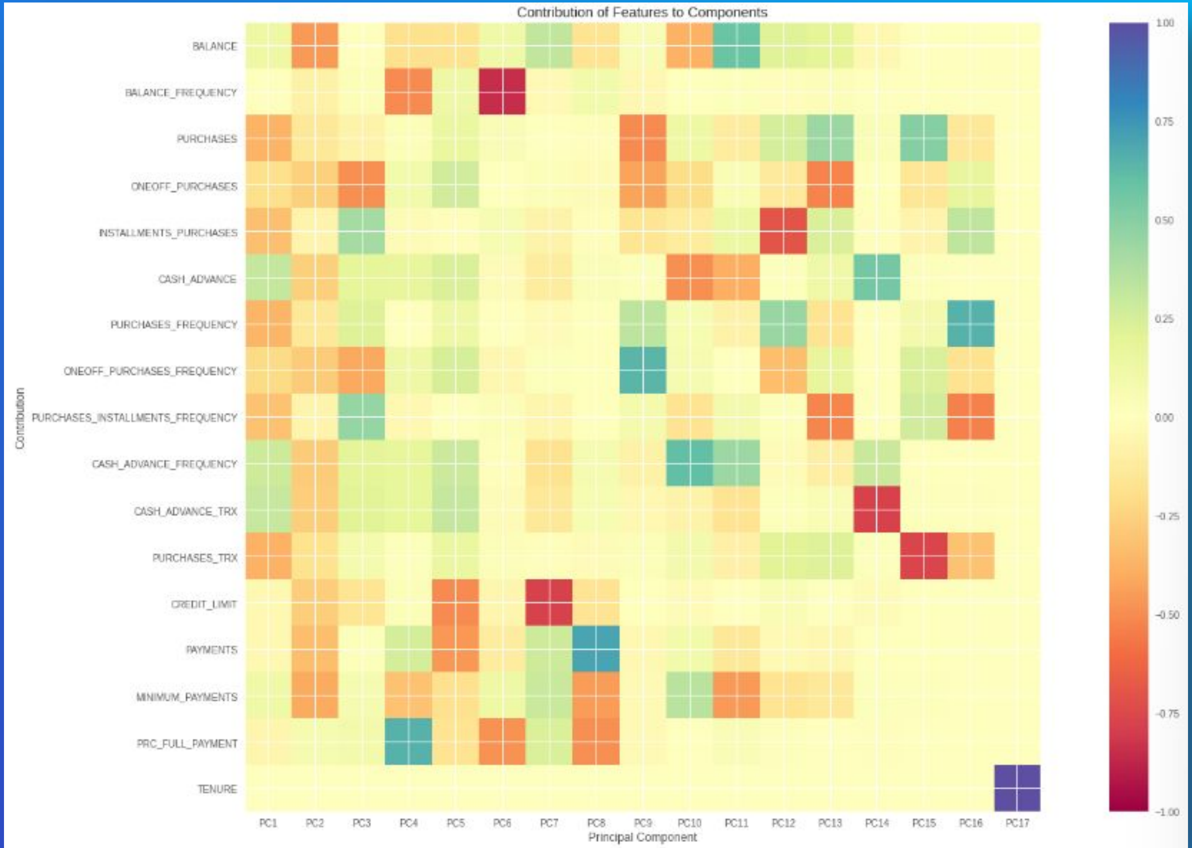
	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10	PC11	PC12	PC13	PC14	PC15	PC16	PC17
0	0.070052	3.130725	0.006426	0.663863	-0.066947	2.462336	0.613870	0.140583	-0.188902	0.281169	-0.019333	-0.617596	0.614756	-0.021266	0.062954	0.121147	0.0
1	3.672899	-1.014127	0.494287	2.506466	-2.158939	-0.484540	0.574163	-1.200474	0.254478	0.103938	-0.124555	-0.051623	-0.193422	-0.103815	-0.006647	0.022660	-0.0
2	-1.523370	-1.488558	-2.740842	-0.786773	-0.155579	-0.083975	0.046930	-0.590834	1.088823	0.304249	-0.012636	0.870089	0.153080	-0.091877	0.106218	0.578616	0.0
3	1.377642	0.858935	-1.745873	-1.311754	1.696539	0.602935	-1.823662	-3.131847	-1.126612	-1.118521	0.550070	0.737795	0.496022	0.288139	0.661877	-0.138007	-0.0
4	1.074890	1.564631	-1.307714	-0.927584	-0.375357	-0.111139	1.387989	0.467199	-0.041152	-0.110663	0.472612	0.110421	-0.121560	-0.099084	-0.048028	-0.056370	0.0



- Seems Like Purchase Features are not that Important to PCA Components compared to Cash Features.

- **Finding Optimum Number of Components**

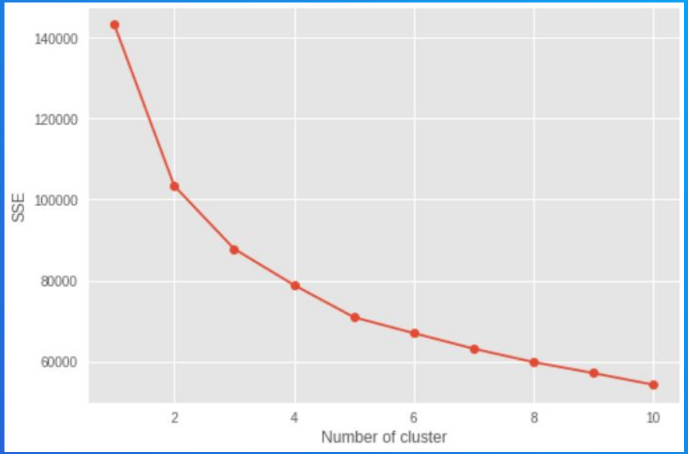




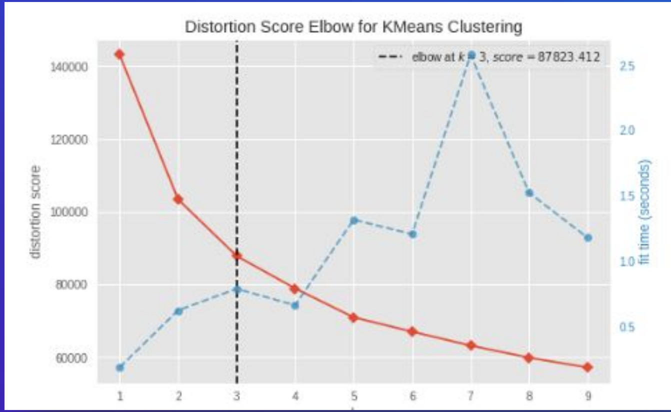
- We will go with 5 Components since it gives least number of dimensions with more explained variance.

5. K-Means Clustering

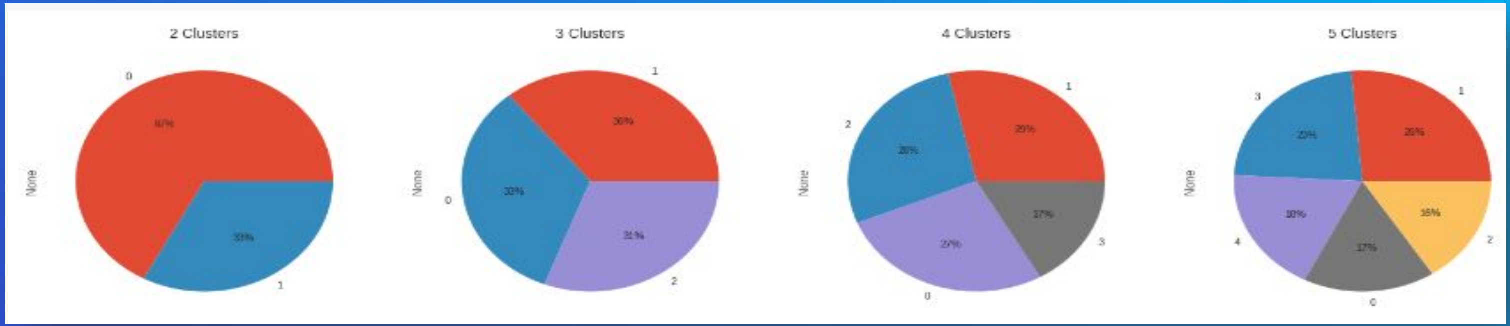
- First, we will try to find the optimum Number of Cluster



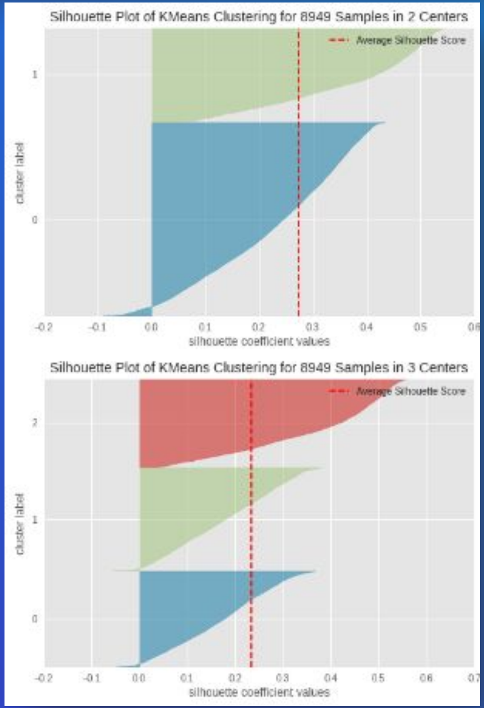
- Seems Like Customers can be grouped to 3 or 4 Clusters.

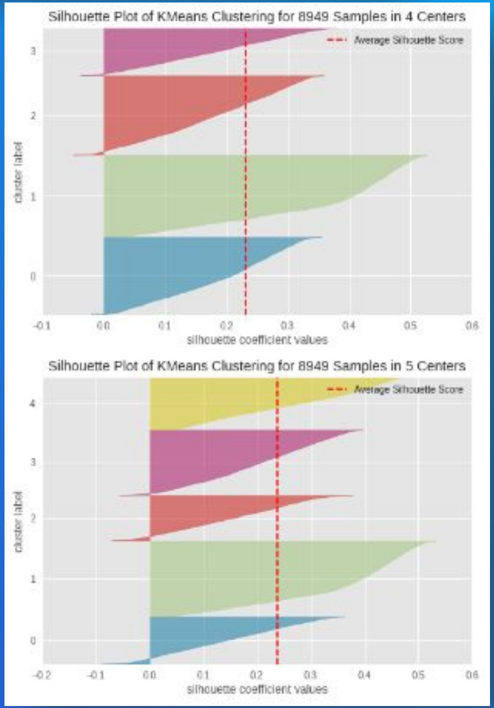


- Silhouette Score for 2 Clusters are Better.
- But we will Check the Clusters Distributions with Silhouette Diagram.



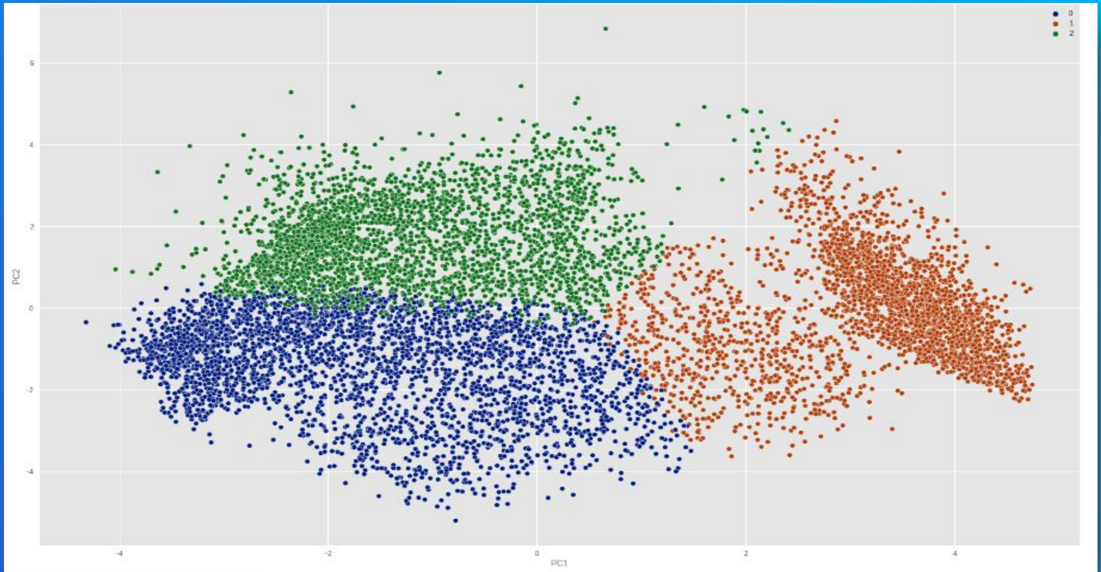
- It Appears that having 3 Clusters will result in more Equally Distributed Clusters.

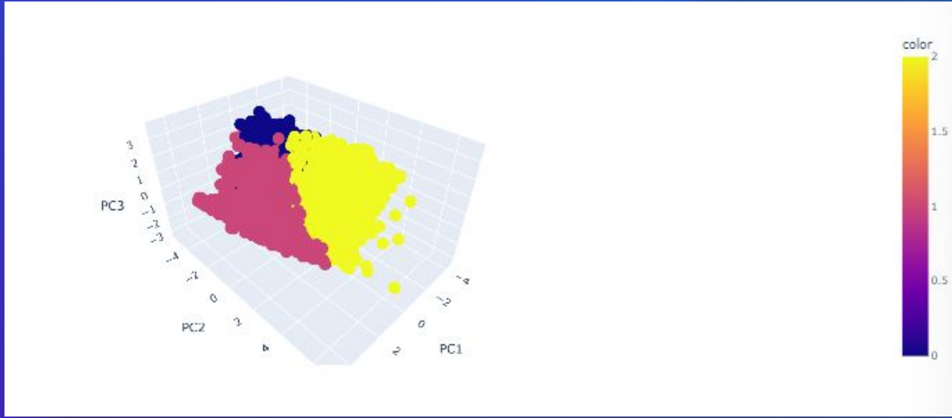




Visualizing Clusters

- With 3 Clusters there are some Overlapping in the Clusters. Explaining their smaller silhouette scores than 2 Clusters



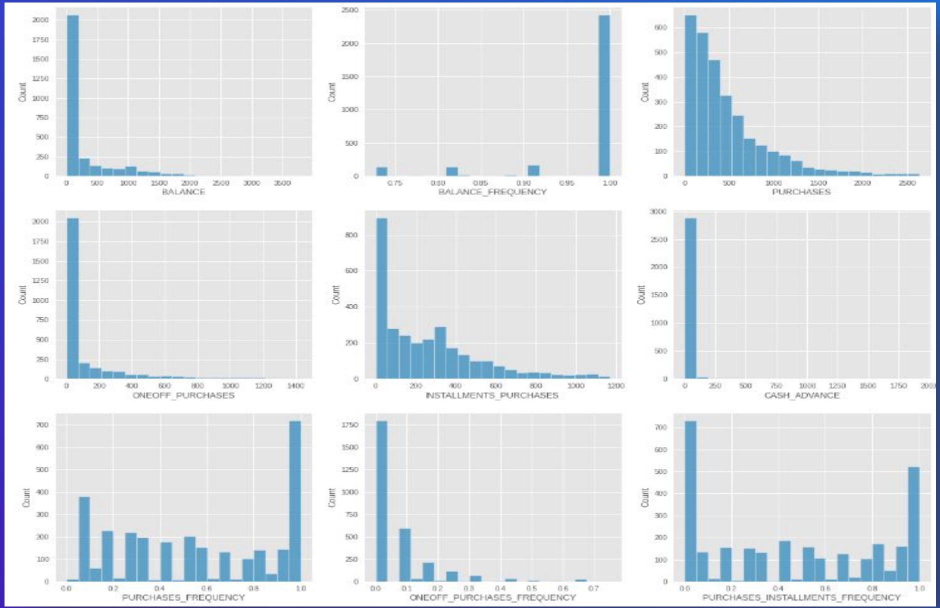


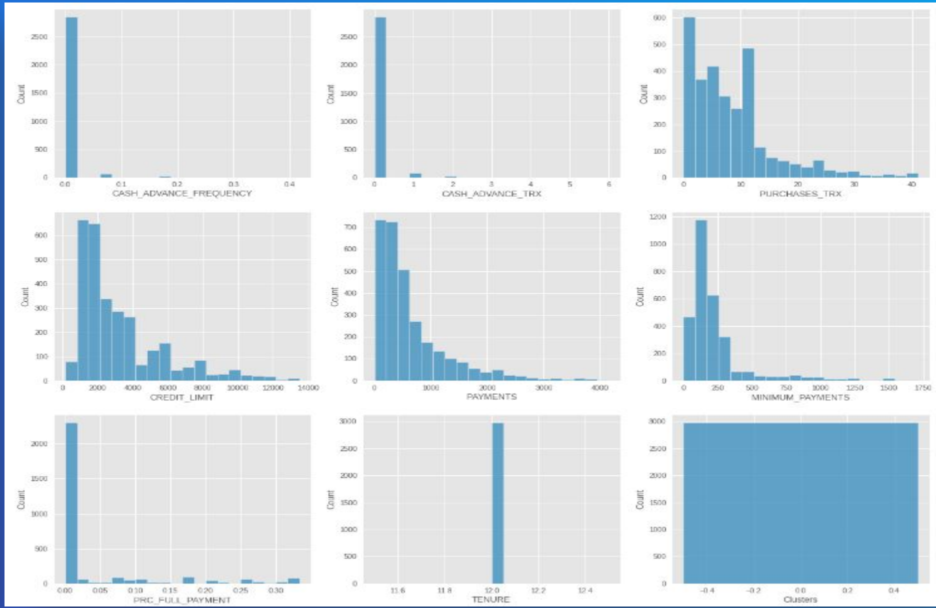
- But it’s a more reasonable way to Cluster Customer, As we will see on the cluster Analysis.

Clusters Analysis

First Cluster of Customers

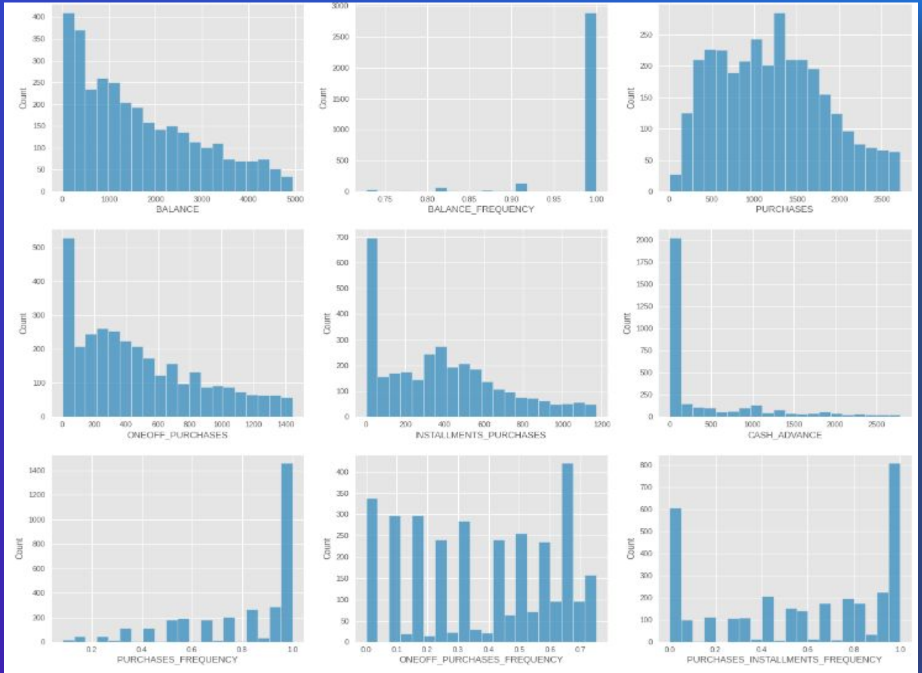
- Customers with Lower balance But Update their Balance Frequently.
- May Make Installment Purchases and doesn't Prefer Paying in Advance.
- Purchase Frequently with Low Payments and they have a Low Credit Limit.

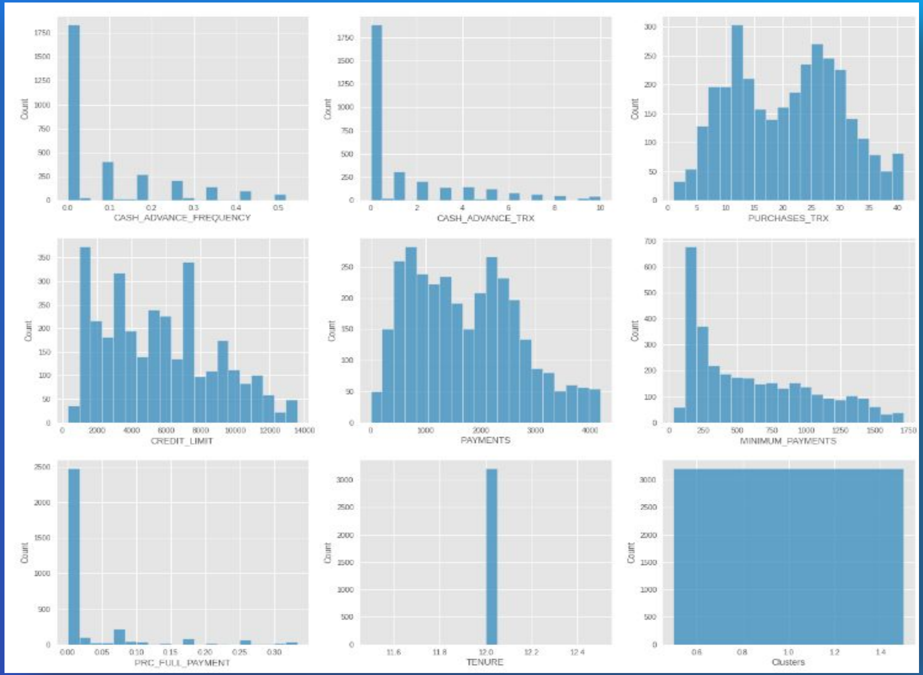




Second Cluster of Customers

- Customers with Medium Balance, Update their Balance More Frequently.
- With Medium Purchases Amount and pay more in Single Transaction, Prefer more Installment Purchases.
- Purchase More Frequently with High Payments and they have a High Credit Limit.





Third Cluster of Customers

- **Customers with Above Medium Balance.**
- **Doesn't Prefer Installment Purchases, But Prefer to Pay in advance.**
- **Doesn't Purchase Frequently but when they do it's with Medium Payments, and they have a Medium Credit Limit.**