

Unsupervised Anomaly Detection in Time Series of Bank Transactions

1. Data Cleaning & Exploratory Data Analysis

```
# Data and Stats Packages
import pandas as pd
import numpy as np
import re
import datetime as dt

# Visualization Packages
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns

# Anomaly Detection Packages
from sklearn import preprocessing
from sklearn.ensemble import IsolationForest
from sklearn.svm import OneClassSVM
from sklearn.covariance import EllipticEnvelope
from statsmodels.tsa.seasonal import seasonal_decompose

import warnings
warnings.filterwarnings("ignore")

/usr/local/lib/python3.7/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: import pandas.util.testing as tm

bank = pd.read_excel('Sample_bank_transaction_data.xlsx')
bank_raw = bank.copy()
```

```
bank.head()
```

	Account No	DATE	TRANSACTION DETAILS	CHQ.NO.	VALUE DATE	WITHDRAWAL AMT	DEPOSIT AMT	BALANCE AMT
0	409000611074'	2017-06-29	TRF FROM Indiaforensic SERVICES	NaN	2017-06-29	NaN	1000000.0	1000000.0
1	409000611074'	2017-07-05	TRF FROM Indiaforensic SERVICES	NaN	2017-07-05	NaN	1000000.0	2000000.0
		2017-	FDRI /INTFRNAI		2017-			

```
bank.shape
```

```
(116201, 9)
```

```
bank.describe()
```

	CHQ.NO.	WITHDRAWAL AMT	DEPOSIT AMT	BALANCE AMT
count	905.000000	5.354900e+04	6.265200e+04	1.162010e+05
mean	791614.503867	4.489190e+06	3.806586e+06	-1.404852e+09
std	151205.932910	1.084850e+07	8.683093e+06	5.348202e+08
min	1.000000	1.000000e-02	1.000000e-02	-2.045201e+09
25%	704231.000000	3.000000e+03	9.900000e+04	-1.690383e+09
50%	873812.000000	4.708300e+04	4.265000e+05	-1.661395e+09
75%	874167.000000	5.000000e+06	4.746411e+06	-1.236888e+09
max	874525.000000	4.594475e+08	5.448000e+08	8.500000e+06

```
bank.columns
```

```
Index(['Account No', 'DATE', 'TRANSACTION DETAILS', 'CHQ.NO.', 'VALUE DATE', 'WITHDRAWAL AMT', 'DEPOSIT AMT', 'BALANCE AMT', '.'],  
      dtype='object')
```

```
bank.rename(columns={'Account No': 'account_no', 'DATE':'date', 'TRANSACTION D  
'WITHDRAWAL AMT':'withdrawal_amt', 'DEPOSIT AMT':'deposit_amt', 'BALANC  
bank.head(3)
```

	account_no	date	details	chq_no	value_date	withdrawal_amt	dep
0	409000611074'	2017- 06-29	TRF FROM Indiaforensic SERVICES	NaN	2017-06-29		NaN
1	409000611074'	2017- -- --	TRF FROM Indiaforensic	NaN	2017-07-05		NaN

We have a time series data without anomaly labeling. Let's take a closer look at the features.

col):

ck search for characteristics of a column in a data frame.

e
f interest

```
Frame(df[col].value_counts(dropna=False))  
ign(percent=(df[col].value_counts(dropna=False, normalize=True).values)*100)  
aFrame({'Column_Name': [col], 'Null_Rows': [df[col].isna().sum()], 'Column_Type':  
val}
```

▼ i. account_no

```
info , val = first_look(bank, bank.columns[0])
```

info

	Column_Name	Null_Rows	Column_Type
column_info	account_no	0	object

```
val
```

	account_no	percent
1196428'	48779	41.978124
409000362497'	29840	25.679641
409000438620'	13454	11.578214
1196711'	10536	9.067048
409000493210'	6014	5.175515
409000438611'	4588	3.948331
409000611074'	1093	0.940612
409000493201'	1044	0.898443
409000425051'	802	0.690183
409000405747'	51	0.043889

Since **account_no** is a unique indicator of customers (in our case we have 10 customers who have rather unbalanced transaction frequencies), we should keep the data type as string and for future investigation purposes we will remove the ' (last character).

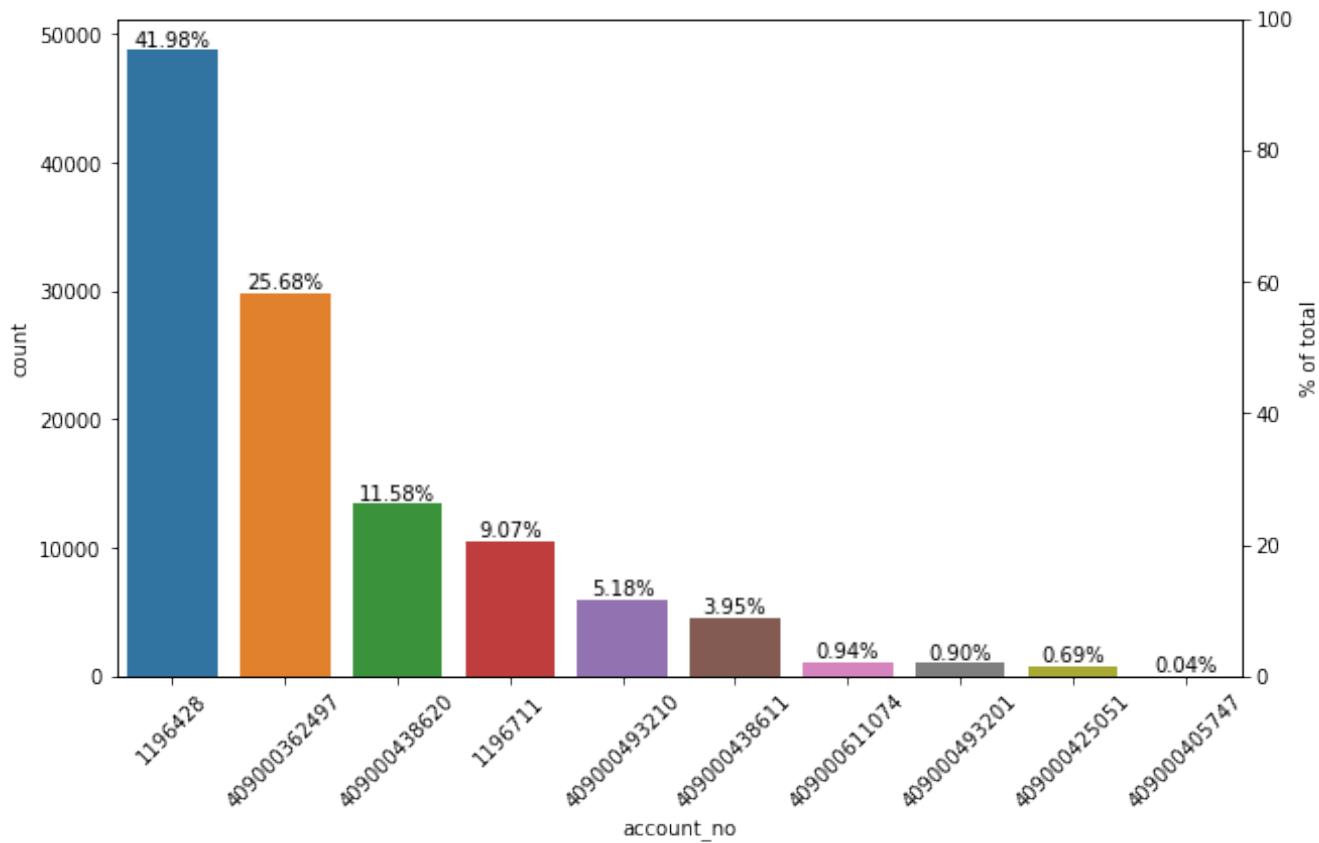
```
bank['account_no'] = bank.account_no.str.extract(r'(\d+)''')
```

```
plt.figure(figsize=(10,6))
plt.xticks(rotation=45)
fig = sns.countplot(data=bank, x='account_no', order=bank.account_no.value_counts()
fig2 = fig.twinx()

fig.yaxis.set_label_position('left')
fig2.yaxis.set_label_position('right')

fig2.set_ylabel('% of total')

for p in fig.patches:
    x=p.get_bbox().get_points()[:,0]
    y=p.get_bbox().get_points()[1,1]
    fig.annotate('{:.2f}%'.format(100.*y/len(bank)), (x.mean(), y),
                 ha='center', va='bottom') # set the alignment of the text
fig2.grid(None)
fig2.set_ylim(0,100)
fig2.grid()
plt.show()
```



▼ ii. date

```
info , val = first_look(bank, bank.columns[1])
```

```
info
```

Column_Name	Null_Rows	Column_Type
column_info	date	0 datetime64[ns]

```
bank.date.apply(lambda x: x.year).value_counts()
```

2018	35534
2016	30372
2017	29113
2015	15658
2019	5524

Name: date, dtype: int64

```
((bank.date).dt.dayofweek).value_counts()
```

0	22662
4	22058
1	21561
3	20446
2	19421
5	9830
6	223

Name: date, dtype: int64

```
conditions = [((bank.date).dt.dayofweek > 4), ((bank.date).dt.dayofweek < 5)]
values = [1, 0]
bank['weekend_date'] = np.select(conditions, values)
```

▼ iii. details

```
info , val = first_look(bank, bank.columns[2])
```

```
info
```

Column_Name	Null_Rows	Column_Type
column_info	details	2499 object

```
bank[bank.details.isna()].account_no.value_counts()
```

```
409000362497    2484  
1196711        15  
Name: account_no, dtype: int64
```

```
bank[bank.details.isna()].date.apply(lambda x: x.year).value_counts()
```

```
2015    1259  
2016    1240  
Name: date, dtype: int64
```

```
bank.details.sample(10)
```

```
80267      FDRL/INTERNAL FUND TRANSFE  
28273      RTGS/ICICH15230026766/Indfor  
88478      IMPS I/W 121015 2C  
102573     IRCTC CORPORATE OFFICE AC  
18803      Indiaforensic AEPS NPCI WDL SET  
5282       Indiaforensic RUPAY POS ACQ 1504  
41605      CASHDEP/NEW-DELHI/  
31023      RTGS/SBINH15329725245/Indfor  
19523     AEPS Indiaforensic INCOME 030118  
23768      CR823214152456/RMCPL18519/2  
Name: details, dtype: object
```

This function has the potential to provide information about anomalies. It could help us find new features. For starters, let's define a column that indicates whether a transaction has no proper details value (Null or nonsense values).

```
conditions = [(bank.details.isna())]  
values = [1]  
bank['details_null_penalty'] = np.select(conditions, values)
```

```
conditions = [(bank.details.str.extract(r'(\w+')[0].isna())]  
values = [1]  
bank['details_numbers_penalty'] = np.select(conditions, values)
```

```
bank.details_null_penalty.value_counts()
```

```
0      113702  
1      2499  
Name: details_null_penalty, dtype: int64
```

```
bank.details_numbers_penalty.value_counts()  
0    112050  
1     4151  
Name: details_numbers_penalty, dtype: int64
```

This penalty is not a red flag by itself, but can be an additional checkpoint for the anomaly detection results. Let's leave the **details** column at this point and move on to the next feature.

► iv. chq_no

```
[ ] ↳ 9 cells hidden
```

► v. value_date

```
[ ] ↳ 8 cells hidden
```

▼ vi. withdrawal_amt

```
info , val = first_look(bank, bank.columns[5])
```

```
info
```

	Column_Name	Null_Rows	Column_Type
column_info	withdrawal_amt	62652	float64

```
bank.withdrawal_amt.describe()
```

```
count      5.354900e+04  
mean       4.489190e+06  
std        1.084850e+07  
min        1.000000e-02  
25%        3.000000e+03  
50%        4.708300e+04  
75%        5.000000e+06  
max        4.594475e+08  
Name: withdrawal_amt, dtype: float64
```

```
bank[~bank.withdrawal_amt.isna()].count()
```

```
account_no          53549  
date               53549  
details            53549  
chq_no             905  
value_date         53549  
withdrawal_amt     53549  
deposit_amt        0  
balance_amt        53549  
.                  53549  
weekend_date       53549  
details_null_penalty 53549  
details_numbers_penalty 53549  
cheque_penalty    53549  
day_diff           53549  
weekend_value_date 53549  
dtype: int64
```

```
bank[bank.withdrawal_amt.isna()].count()
```

```
account_no          62652  
date               62652  
details            60153  
chq_no             0  
value_date         62652  
withdrawal_amt     0  
deposit_amt        62652  
balance_amt        62652  
.                  62652  
weekend_date       62652  
details_null_penalty 62652  
details_numbers_penalty 62652  
cheque_penalty    62652  
day_diff           62652  
weekend_value_date 62652  
dtype: int64
```

```
pd.DataFrame(bank.groupby('account_no').withdrawal_amt.count()).sort_values('withdrawal_amt', ascending=False)
```

account_no	withdrawal_amt
1196428	16687
409000362497	15477
409000438620	8391
409000493210	5227
1196711	5025
409000438611	1333
409000611074	778
409000493201	575
409000405747	28
409000425051	28

```
conditions = [((bank.balance_amt+bank.withdrawal_amt)==0), ((bank.balance_amt+bank.withdrawal_amt)>0)]
values = [0, 100*(bank.withdrawal_amt/(bank.balance_amt+bank.withdrawal_amt))]
bank['withdrawal_over_balance'] = np.select(conditions, values)
```

```
bank['withdrawal_amt'] = bank.withdrawal_amt.fillna(0)
```

```
bank['withdrawal_over_balance'] = bank.withdrawal_over_balance.fillna(0)
```

```
bank[bank.withdrawal_amt!=0].head()
```

	account_no	date	details	chq_no	value_date	withdrawal_amt	deposit
10	409000611074	2017-08-16	INDO GIBL Indiaforensic STL01071	NaN	2017-08-16	133900.0	
11	409000611074	2017-08-16	INDO GIBL Indiaforensic STL02071	NaN	2017-08-16	18000.0	
12	409000611074	2017-08-16	INDO GIBL Indiaforensic STL03071	NaN	2017-08-16	5000.0	
13	409000611074	2017-08-16	INDO GIBL Indiaforensic STL04071	NaN	2017-08-16	195800.0	
14	409000611074	2017-08-16	INDO GIBL Indiaforensic STL05071	NaN	2017-08-16	81600.0	

▼ vii. deposit_amt

```
info , val = first_look(bank, bank.columns[6])
```

info

	Column_Name	Null_Rows	Column_Type
column_info	deposit_amt	53549	float64

```
bank.deposit_amt.describe()
```

```
count      6.265200e+04
mean      3.806586e+06
std       8.683093e+06
min       1.000000e-02
25%       9.900000e+04
50%       4.265000e+05
75%       4.746411e+06
max       5.448000e+08
Name: deposit_amt, dtype: float64
```

```
bank[~bank.deposit_amt.isna()].count()
```

```
account_no           62652
date                 62652
details              60153
chq_no               0
value_date           62652
withdrawal_amt       62652
deposit_amt          62652
balance_amt          62652
.
weekend_date         62652
details_null_penalty 62652
details_numbers_penalty 62652
cheque_penalty       62652
day_diff              62652
weekend_value_date   62652
withdrawal_over_balance 62652
dtype: int64
```

```
bank[bank.deposit_amt.isna()].count()
```

```
account_no      53549  
date           53549  
details        53549  
chq_no          905  
value_date     53549  
withdrawal_amt 53549  
deposit_amt       0  
balance_amt     53549  
.               53549  
weekend_date    53549  
details_null_penalty 53549  
details_numbers_penalty 53549  
cheque_penalty   53549  
day_diff         53549  
weekend_value_date 53549  
withdrawal_over_balance 53549  
dtype: int64
```

```
pd.DataFrame(bank.groupby('account_no').deposit_amt.count()).sort_values('depc
```

	deposit_amt
account_no	
1196428	32092
409000362497	14363
1196711	5511
409000438620	5063
409000438611	3255
409000493210	787
409000425051	774
409000493201	469
409000611074	315
409000405747	23

```
bank[bank.deposit_amt.isna() & bank.withdrawal_amt.isna()].count()
```

```
account_no          0  
date               0  
details            0  
chq_no             0  
value_date         0  
withdrawal_amt     0  
deposit_amt        0  
balance_amt        0  
.                 0  
weekend_date       0  
details_null_penalty 0  
details_numbers_penalty 0  
cheque_penalty     0  
day_diff           0  
weekend_value_date 0  
withdrawal_over_balance 0  
dtype: int64
```

```
conditions = [((bank.balance_amt-bank.deposit_amt)==0), ((bank.balance_amt-ban  
values = [0, 100*(bank.deposit_amt/(bank.balance_amt-bank.deposit_amt))]  
bank['deposit_over_balance'] = np.select(conditions, values)  
bank['deposit_over_balance'] = bank.deposit_over_balance.fillna(0)
```

```
bank['deposit_amt'] = bank.deposit_amt.fillna(0)
```

► viii. balance_amt

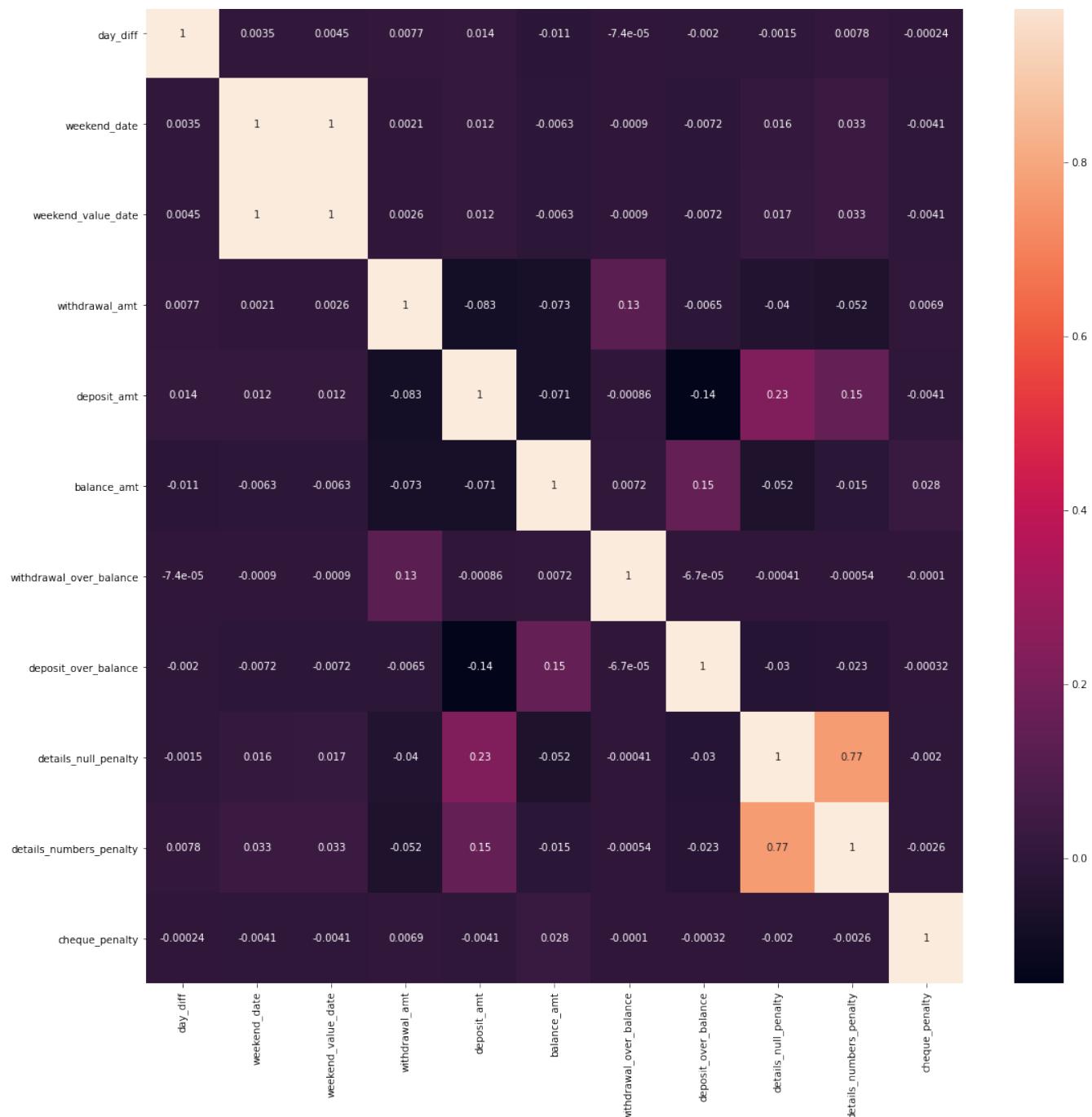
```
[ ] ↳ 5 cells hidden
```

► ix. Dropping Empty Column

```
[ ] ↳ 7 cells hidden
```

▼ 2. Data Visualization

```
plt.figure(figsize=(18, 18))  
sns.heatmap(bank.corr(), annot=True)  
plt.yticks(rotation=0)  
plt.show()
```

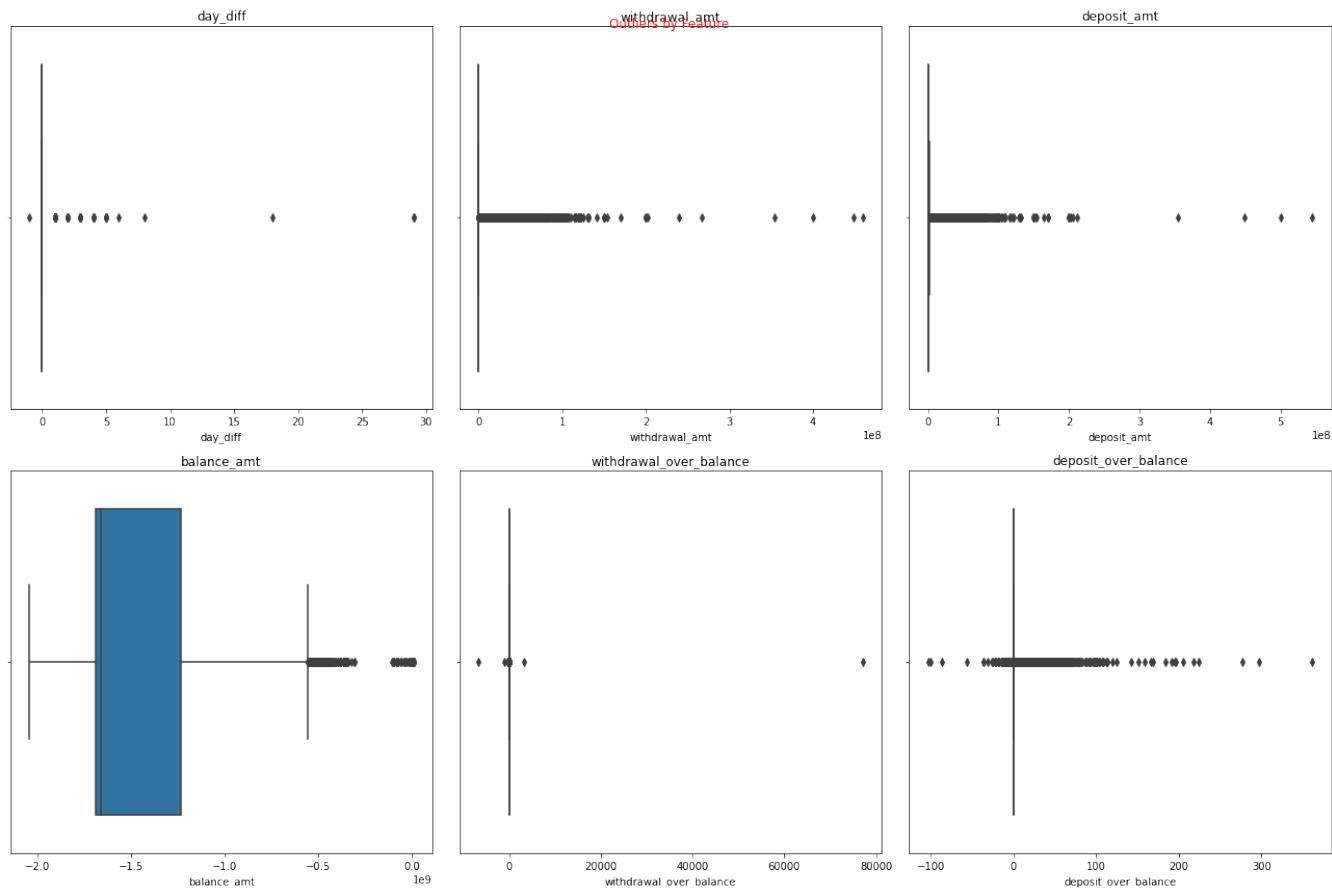


```
features = ['day_diff', 'withdrawal_amt', 'deposit_amt', 'balance_amt', 'withd
```

```
fig, axes = plt.subplots(2,3, sharex=False, figsize=(18, 12))
fig.suptitle('Outliers by Feature', color='r' )

for i in range(2):
    for j in range(3):
        sns.boxplot(ax=axes[i,j], x= features[3*i + j], data = bank)
        axes[i,j].set_title(f'{features[3*i + j]}')

plt.tight_layout()
plt.show()
```



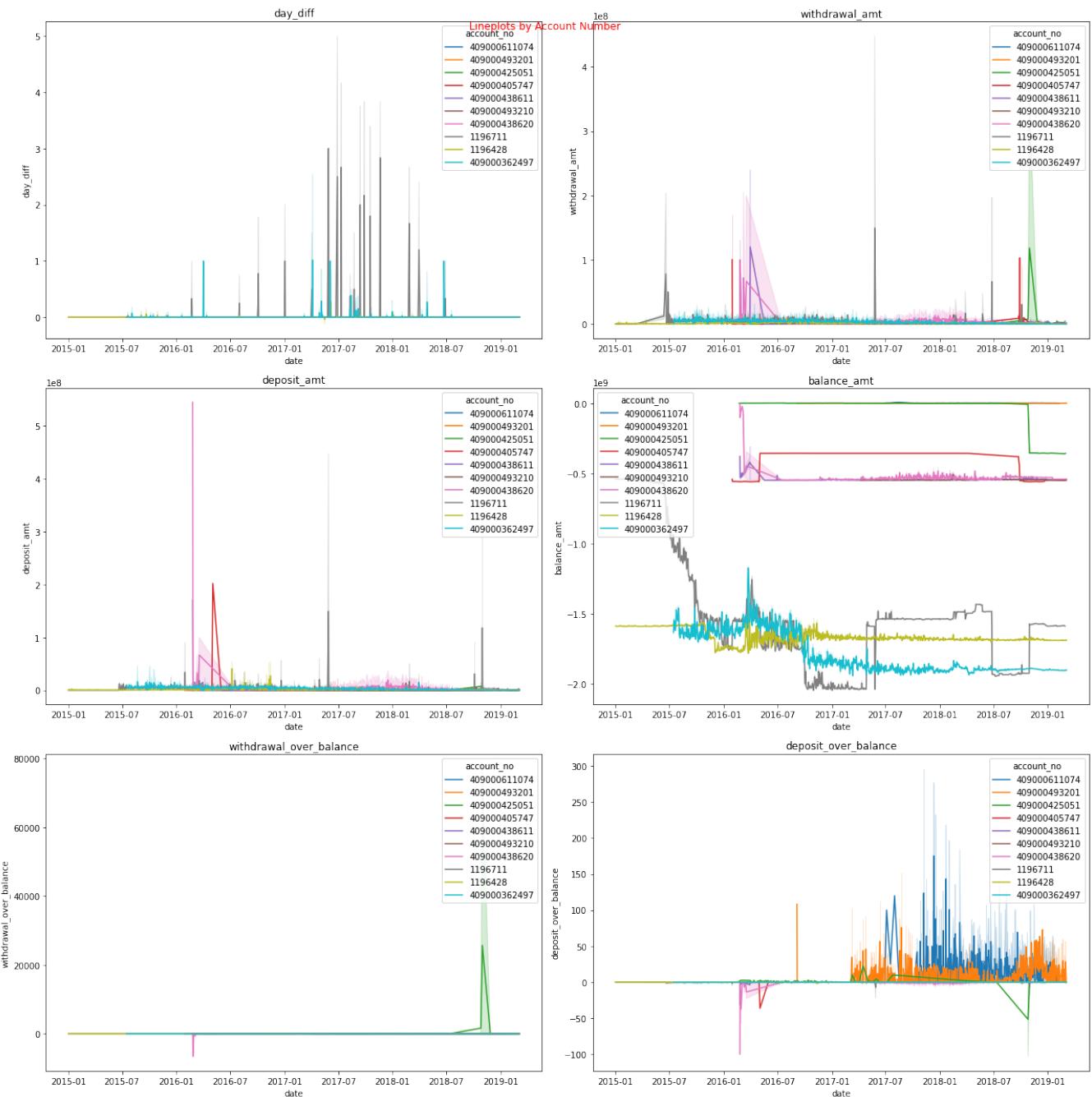
```

features = ['day_diff', 'withdrawal_amt', 'deposit_amt', 'balance_amt', 'withd
fig, axes = plt.subplots(3,2, sharex=False, figsize=(18, 18))
fig.suptitle('Lineplots by Account Number', color='r' )

for i in range(3):
    for j in range(2):
        sns.lineplot(ax=axes[i,j], y= features[2*i + j], x = 'date', data = ba
        axes[i,j].set_title(f'{features[2*i + j]}')

plt.tight_layout()
plt.show()

```



▼ 3. Anomaly Detection Whole Dataset (Account No. as a Feature)

Before getting started with the anomaly detection we need to convert categorical variable into dummy/indicator variables.

```
bank_premodel = bank.copy()

bank = pd.get_dummies(bank, columns = ['account_no'])

bank.columns

Index(['date', 'value_date', 'day_diff', 'weekend_date', 'weekend_value_da
    'withdrawal_amt', 'deposit_amt', 'balance_amt',
    'withdrawal_over_balance', 'deposit_over_balance',
    'details_null_penalty', 'details_numbers_penalty', 'cheque_penalty',
    'account_no_1196428', 'account_no_1196711', 'account_no_40900036249',
    'account_no_409000405747', 'account_no_409000425051',
    'account_no_409000438611', 'account_no_409000438620',
    'account_no_409000493201', 'account_no_409000493210',
    'account_no_409000611074'],
      dtype='object')
```

```
bank.head(3)
```

	date	value_date	day_diff	weekend_date	weekend_value_date	withdrawal_
0	2017-06-29	2017-06-29	0	0	0	0
1	2017-07-05	2017-07-05	0	0	0	0
2	2017-07-18	2017-07-18	0	0	0	0

```
(bank.weekend_date + bank.weekend_value_date).value_counts()
```

```
0    106140
2     10047
1      14
dtype: int64
```

▼ i. Isolation Forest Anomaly Detection

- Isolation Forest returns the anomaly score of each sample using the IsolationForest algorithm.
- The IsolationForest ‘isolates’ observations by randomly selecting a feature and then randomly selecting a split value between the maximum and minimum values of the selected feature.
- Since recursive partitioning can be represented by a tree structure, the number of splittings required to isolate a sample is equivalent to the path length from the root node to the terminating node.
- This path length, averaged over a forest of such random trees, is a measure of normality and our decision function.
- Random partitioning produces noticeably shorter paths for anomalies. Hence, when a forest of random trees collectively produce shorter path lengths for particular samples, they are highly likely to be anomalies.

link : [\[learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html\]\(https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html\)](https://scikit-</p>
</div>
<div data-bbox=)

```

from sklearn.preprocessing import StandardScaler
bank_iso = bank.drop(columns=['date', 'value_date'], axis=1)
scaler = StandardScaler()
bank_iso_scaled = pd.DataFrame(scaler.fit_transform(bank_iso), columns=bank_iso.columns)

bank_main = bank.copy()
outliers_fraction = .007
model = IsolationForest(contamination=outliers_fraction)
model.fit(bank_iso_scaled)
bank_main['anomaly_isolation_forest'] = pd.Series(model.predict(bank_iso_scaled))

bank_main['anomaly_isolation_forest'].value_counts()

0    115398
1      803
Name: anomaly_isolation_forest, dtype: int64

features = ['account_no_1196428', 'account_no_1196711', 'account_no_4090003624',
            'account_no_409000438611', 'account_no_409000438620', 'account_no_40900045747']
fig, axes = plt.subplots(5,2, sharex=False, figsize=(18, 18))
fig.suptitle('Anomalies by Account Number', color='r' )

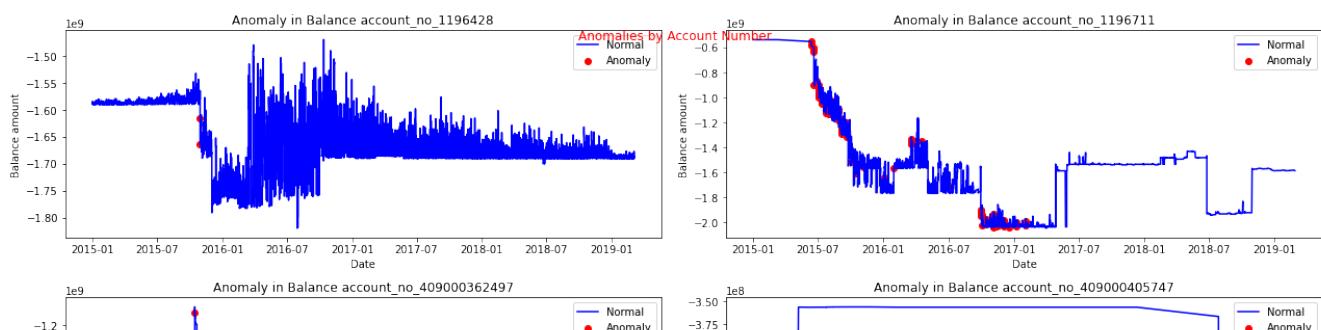
for i in range(5):
    for j in range(2):

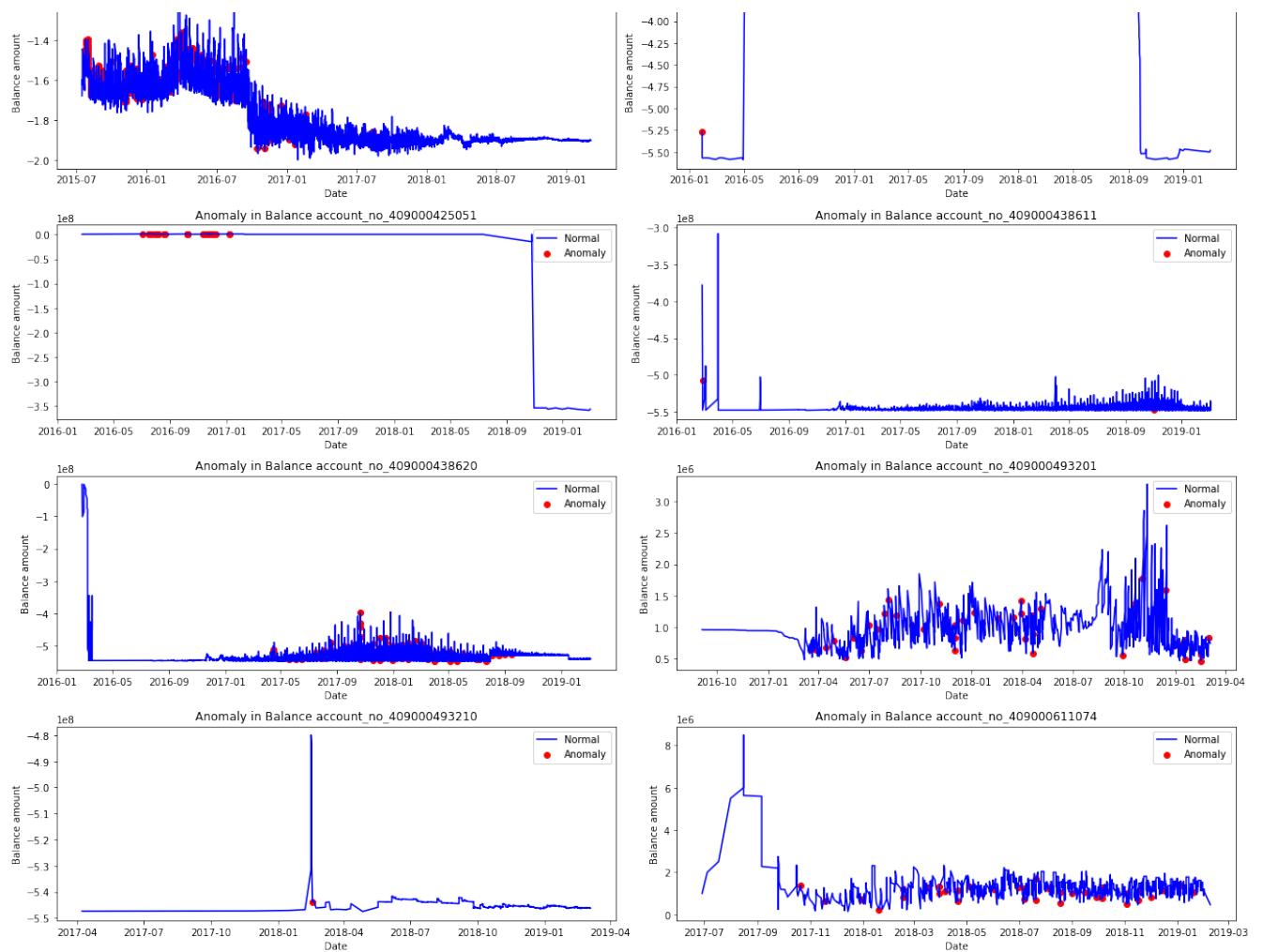
        df = bank_main[bank_main[features[2*i + j]]==1]
        a= df.loc[df['anomaly_isolation_forest']==1, ['date', 'balance_amt']]

        axes[i,j].plot(df['date'], df['balance_amt'], color='blue', label='Normal')
        axes[i,j].scatter(a['date'], a['balance_amt'], color='red', label='Anomaly')
        axes[i,j].set_title(f'Anomaly in Balance {features[2*i + j]}')
        axes[i,j].set_xlabel('Date')
        axes[i,j].set_ylabel('Balance amount')
        axes[i,j].legend()

plt.tight_layout()
plt.show()

```

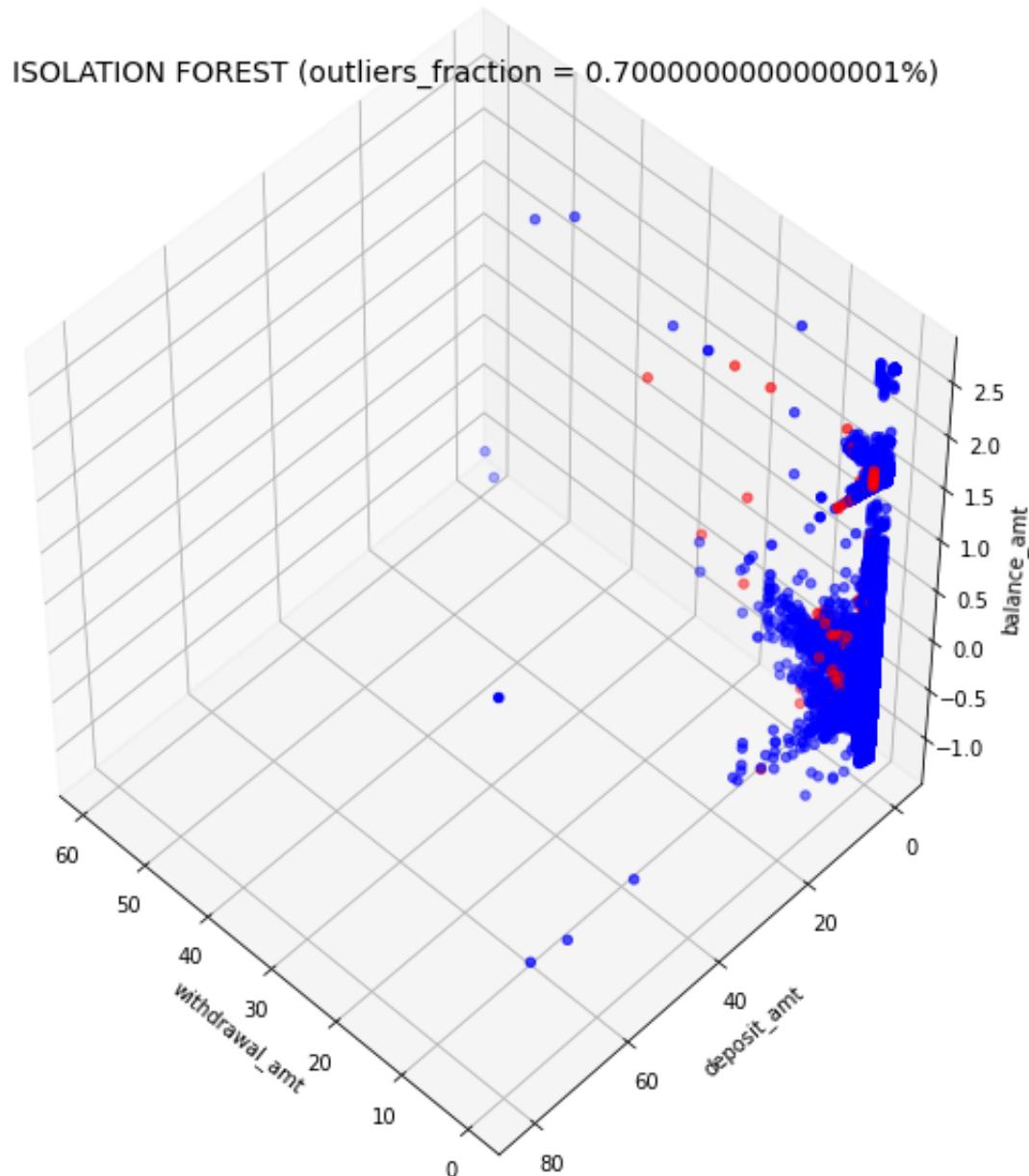




```
from mpl_toolkits.mplot3d import Axes3D
labels = bank_main.anomaly_isolation_forest
colors = {0:'blue', 1:'red'}
```

```
fig = plt.figure(1, figsize=(8,8))
ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)
ax.scatter(bank_iso_scaled.iloc[:,3], bank_iso_scaled.iloc[:,4], bank_iso_scaled.iloc[:,0], color=colors)
ax.set_title(f'ISOLATION FOREST (outliers_fraction = {100*outliers_fraction}%)')
ax.set_xlabel('withdrawal_amt')
ax.set_ylabel('deposit_amt')
ax.set_zlabel('balance_amt')
```

Text(0.5, 0, 'balance_amt')



▼ ii. OneClassSVM Anomaly Detection

```

bank_svm = bank.drop(columns=['date', 'value_date'], axis=1)
scaler = StandardScaler()
bank_svm_scaled = pd.DataFrame(scaler.fit_transform(bank_svm), columns=bank_svm.columns)

outliers_fraction = .005
model = OneClassSVM(nu=outliers_fraction, kernel='rbf', gamma=.01)
model.fit(bank_svm_scaled)
bank_main['anomaly_one_class_svm'] = pd.Series(model.predict(bank_svm_scaled))

bank_main['anomaly_one_class_svm'].value_counts()

0    115589
1      612
Name: anomaly_one_class_svm, dtype: int64

features = ['account_no_1196428', 'account_no_1196711', 'account_no_4090003624',
            'account_no_409000438611', 'account_no_409000438620', 'account_no_409000438621']
fig, axes = plt.subplots(5,2, sharex=False, figsize=(18, 18))
fig.suptitle('Anomalies by Account Number', color='r' )

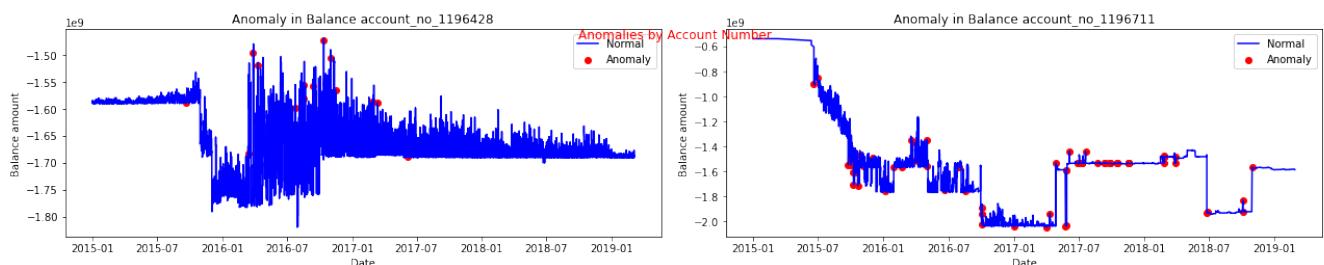
for i in range(5):
    for j in range(2):

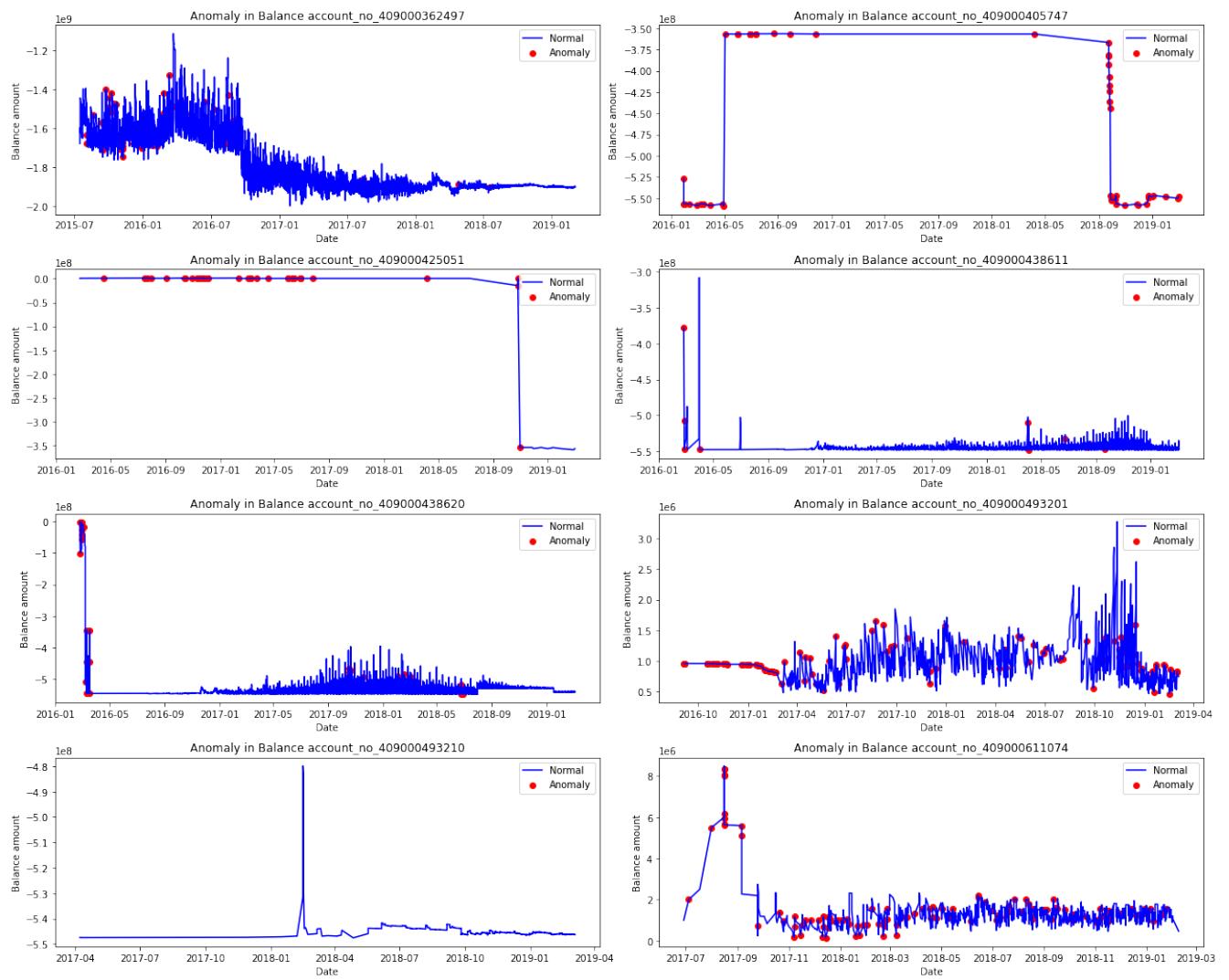
        df = bank_main[bank_main[features[2*i + j]]==1]
        a= df.loc[df['anomaly_one_class_svm']==1, ['date', 'balance_amt']]

        axes[i,j].plot(df['date'], df['balance_amt'], color='blue', label='Normal')
        axes[i,j].scatter(a['date'], a['balance_amt'], color='red', label='Anomaly')
        axes[i,j].set_title(f'Anomaly in Balance {features[2*i + j]}')
        axes[i,j].set_xlabel('Date')
        axes[i,j].set_ylabel('Balance amount')
        axes[i,j].legend()

plt.tight_layout()
plt.show()

```

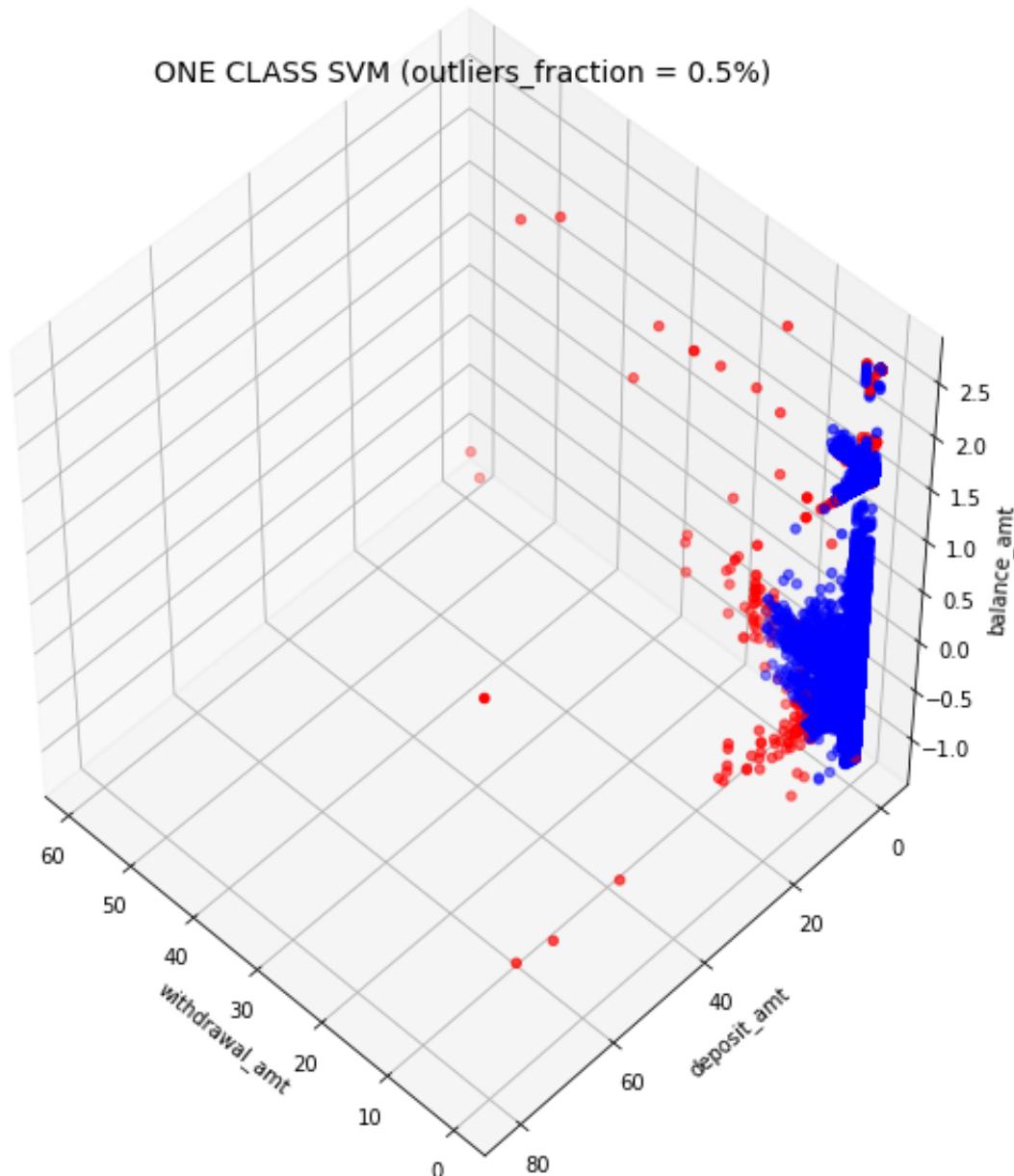




```
labels = bank_main.anomaly_one_class_svm
colors = {0:'blue', 1:'red'}
```

```
fig = plt.figure(1, figsize=(8,8))
ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)
ax.scatter(bank_svm_scaled.iloc[:,3], bank_svm_scaled.iloc[:,4], bank_svm_scaled.iloc[:,5], c=colors[labels])
ax.set_title(f'ONE CLASS SVM (outliers_fraction = {100*outliers_fraction}%)')
ax.set_xlabel('withdrawal_amt')
ax.set_ylabel('deposit_amt')
ax.set_zlabel('balance_amt')

Text(0.5, 0, 'balance_amt')
```



► iii. Gaussian Distribution Anomaly Detection (Elliptic Envelope)

```
[ ] ↳ 4 cells hidden
```

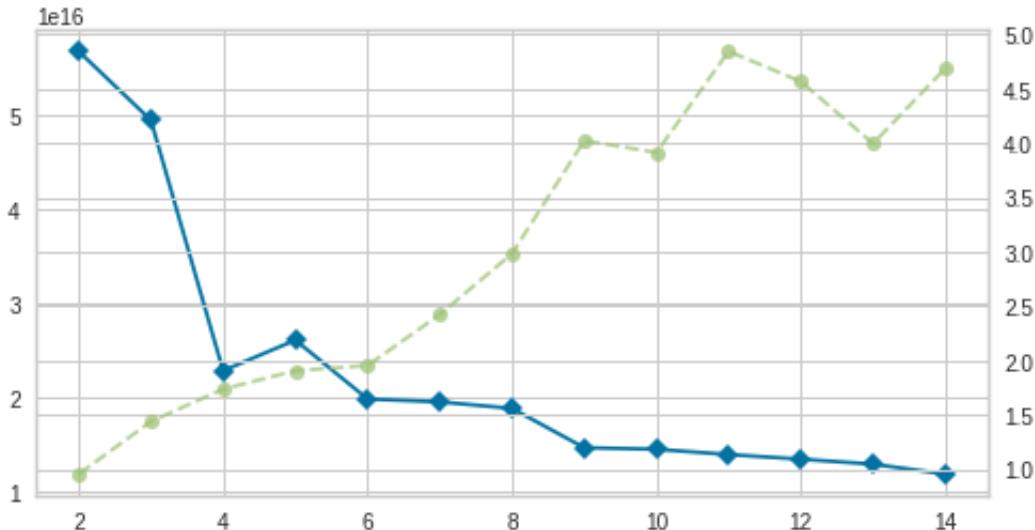
▼ v. Clustering-Based Anomaly Detection

```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from yellowbrick.cluster import KElbowVisualizer

bank_clstr = bank.drop(columns=['date', 'value_date'], axis=1)
scaler = StandardScaler()
bank_scaled = scaler.fit_transform(bank_clstr)

# Instantiate the clustering model and visualizer
visualizer = KElbowVisualizer(KMeans(), k=(2,15))
plt.figure(figsize=(8,4)).patch.set_facecolor('xkcd:white')
visualizer.fit(bank_clstr)      # Fit the data to the visualizer
# visualizer.show()           # Finalize and render the figure

KElbowVisualizer(ax=<matplotlib.axes._subplots.AxesSubplot object at 0x7f26
```



```
kmeans = KMeans(n_clusters=5)
kmeans.fit(bank_clstr)

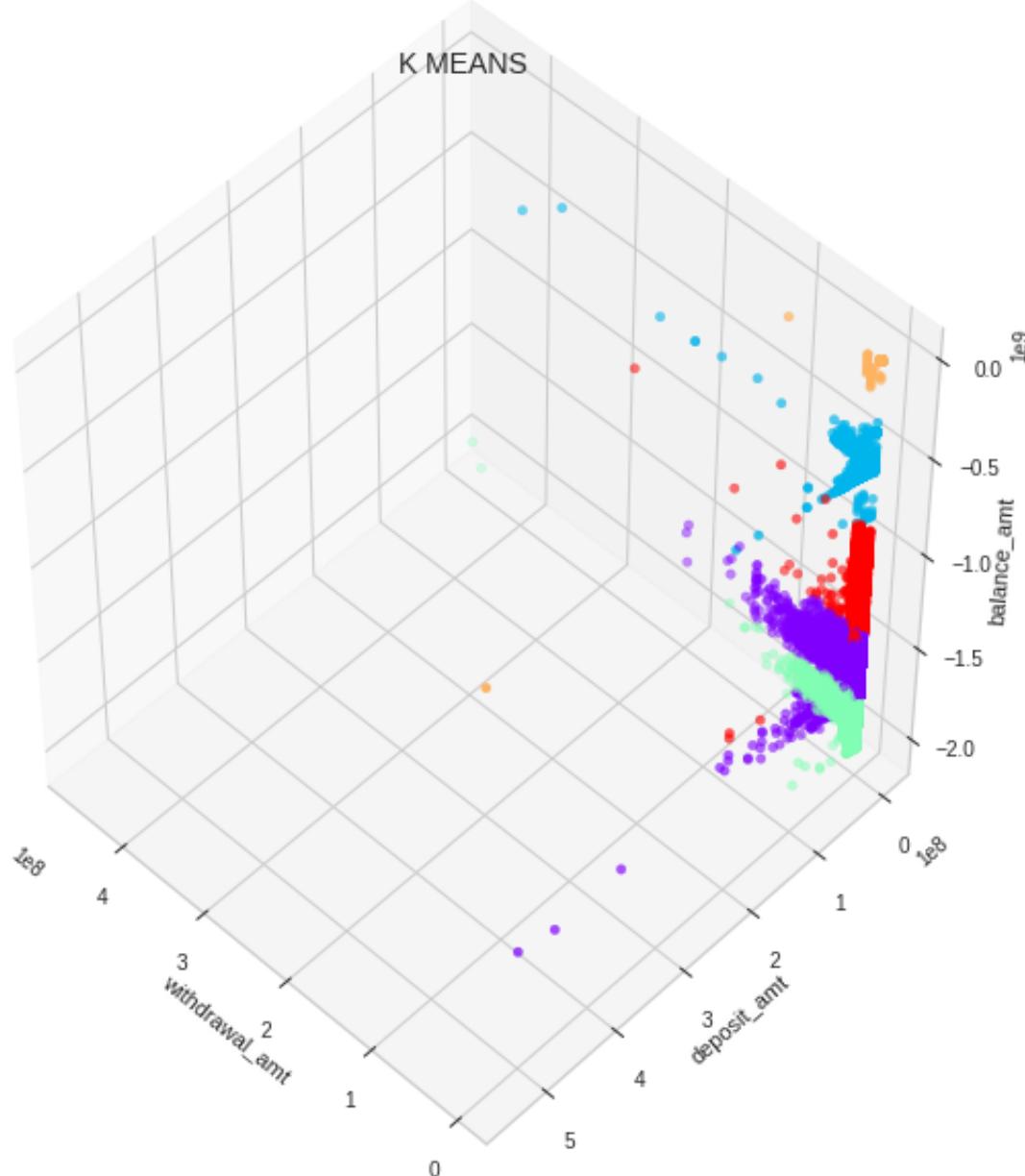
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=5, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

Let's visualise labeled clusters.

```
labels = kmeans.labels_

fig = plt.figure(1, figsize=(8,8))
ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)
ax.scatter(bank_clstr.iloc[:,3], bank_clstr.iloc[:,4], bank_clstr.iloc[:,5], c
ax.set_title('K MEANS', fontsize=14)
ax.set_xlabel('withdrawal_amt')
ax.set_ylabel('deposit_amt')
ax.set_zlabel('balance_amt')

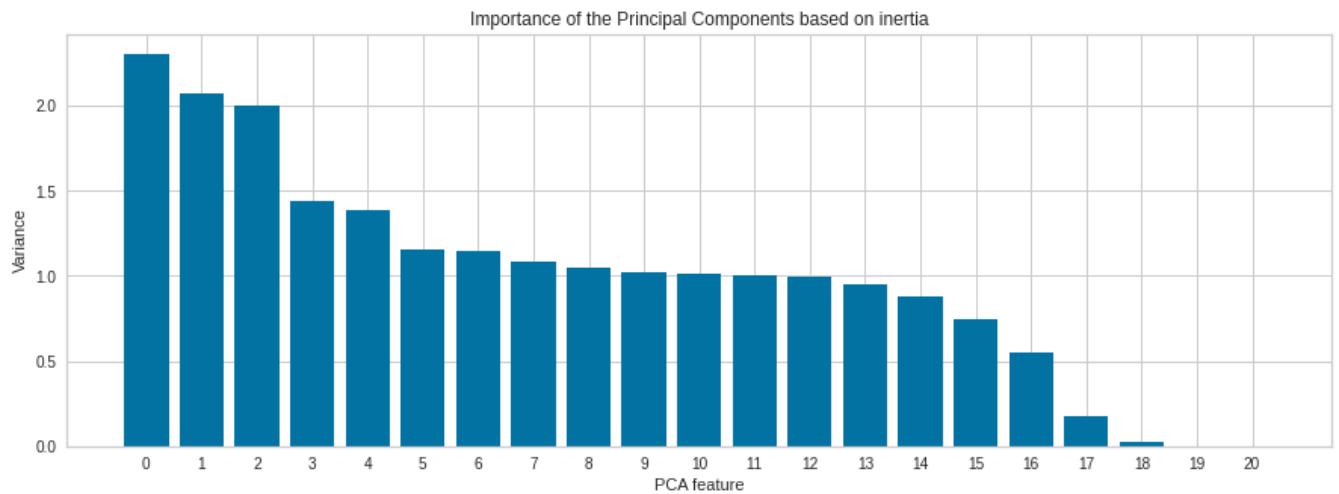
Text(0.5, 0, 'balance_amt')
```



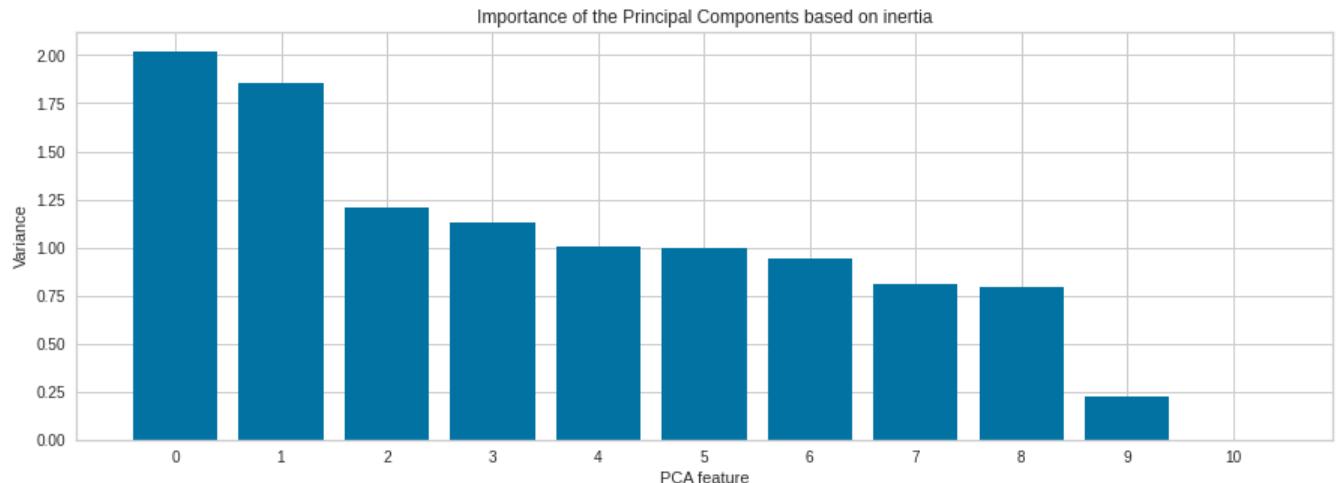
To overcome the curse of dimensionality, we need to figure out which components we want to keep.

```
# Standardize/scale the dataset and apply PCA
from sklearn.decomposition import PCA
from sklearn.pipeline import make_pipeline
# Extract the names of the numerical columns
x = bank_clstr
scaler = StandardScaler()
pca = PCA()
pipeline = make_pipeline(scaler, pca)
pipeline.fit(x)

# Plot the principal components against their inertia
features = range(pca.n_components_)
_ = plt.figure(figsize=(15, 5))
_ = plt.bar(features, pca.explained_variance_)
_ = plt.xlabel('PCA feature')
_ = plt.ylabel('Variance')
_ = plt.xticks(features)
_ = plt.title("Importance of the Principal Components based on inertia")
plt.show()
```



```
bank_gen = bank_clstr[bank_clstr.columns[:11]]  
  
# Extract the names of the numerical columns  
x = bank_gen  
scaler = StandardScaler()  
pca = PCA()  
pipeline = make_pipeline(scaler, pca)  
pipeline.fit(x)  
# Plot the principal components against their inertia  
features = range(pca.n_components_)  
_ = plt.figure(figsize=(15, 5))  
_ = plt.bar(features, pca.explained_variance_)  
_ = plt.xlabel('PCA feature')  
_ = plt.ylabel('Variance')  
_ = plt.xticks(features)  
_ = plt.title("Importance of the Principal Components based on inertia")  
plt.show()
```



It turns out that even a data frame without customer columns PCA does not help too much in our case.

In K-Means Clustering approach to anomaly detection we calculate the distance between each point and its nearest centroid and the biggest distances are considered as anomaly. We are going to take following steps:

- We assume a proportion of data as outliers (outliers_fraction: proportion of the outliers present in our data set) and calculate number_of_outliers using outliers_fraction.
- We set a threshold as the minimum distance of these outliers.
- After labeling (0:normal, 1:anomaly) anomalies we will visualize them.

```
def getDistanceByPoint(data, model):
    """
    Function that calculates the distance between a point and centroid of a cluster
    returns the distances in pandas series
    """
    distance = []
    for i in range(0,len(data)):
        Xa = np.array(data.loc[i])
        Xb = model.cluster_centers_[model.labels_[i]-1]
        distance.append(np.linalg.norm(Xa-Xb))
    return pd.Series(distance, index=data.index)

bank_scaled = pd.DataFrame(bank_scaled, columns=bank_clstr.columns)
kmeans = KMeans(n_clusters=5)
kmeans.fit(bank_scaled)
labels = kmeans.predict(bank_scaled)
unique_elements, counts_elements = np.unique(labels, return_counts=True)
clusters = np.asarray((unique_elements, counts_elements))

# Assume that 0.5% of the entire data set are anomalies
outliers_fraction = 0.005
# get the distance between each point and its nearest centroid. The biggest distance
distance = getDistanceByPoint(bank_scaled, kmeans)
# number of observations that equate to the 0.5% of the entire data set
number_of_outliers = int(outliers_fraction*len(distance))
# Take the minimum of the largest 0.5% of the distances as the threshold
threshold = distance.nlargest(number_of_outliers).min()
# anomaly1 contain the anomaly result of the above method Cluster (0:normal, 1
bank_scaled['anomaly_kmeans'] = (distance >= threshold).astype(int)
```

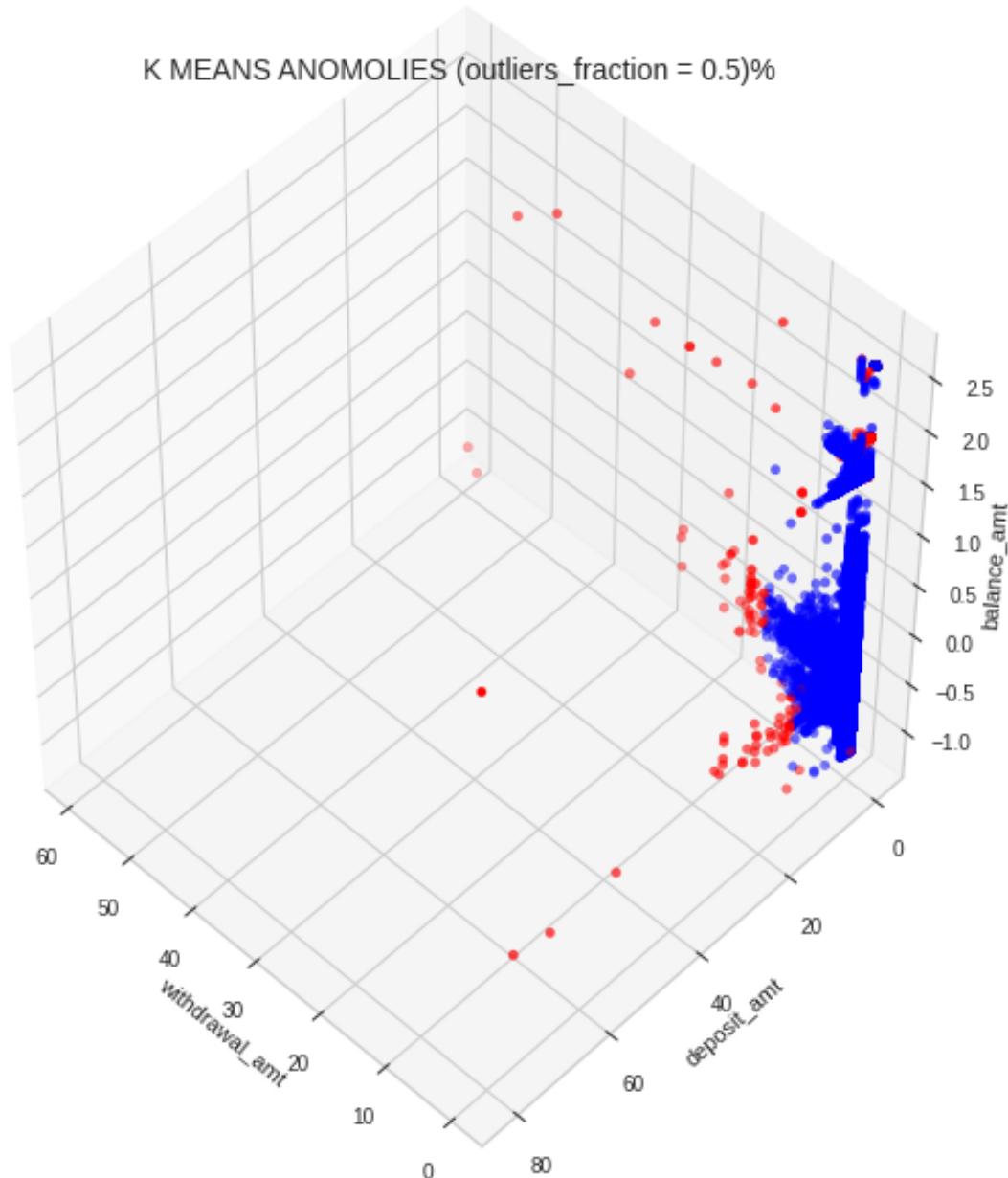
```
bank_scaled.anomaly_kmeans.value_counts()
```

```
0    115620  
1     581  
Name: anomaly_kmeans, dtype: int64
```

```
labels = bank_scaled.anomaly_kmeans
colors = {0:'blue', 1:'red'}
```

```
fig = plt.figure(1, figsize=(8,8))
ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)
ax.scatter(bank_scaled.iloc[:,3], bank_scaled.iloc[:,4], bank_scaled.iloc[:,5])
ax.set_title(f'K MEANS ANOMOLIES (outliers_fraction = {100*outliers_fraction})')
ax.set_xlabel('withdrawal_amt')
ax.set_ylabel('deposit_amt')
ax.set_zlabel('balance_amt')

Text(0.5, 0, 'balance_amt')
```



```
bank_main['anomaly_kmeans'] = bank_scaled.anomaly_kmeans
```

```

# bank_main = bank_clstr.copy()
# bank_main['date'], bank_main['value_date'], bank_main['anomaly_kmeans'] = ba

features = ['account_no_1196428', 'account_no_1196711', 'account_no_4090003624'
            'account_no_409000438611', 'account_no_409000438620', 'account_no_40900045747'
fig, axes = plt.subplots(5,2, sharex=False, figsize=(18, 18))
fig.suptitle('Anomalies by Account Number', color='r' )

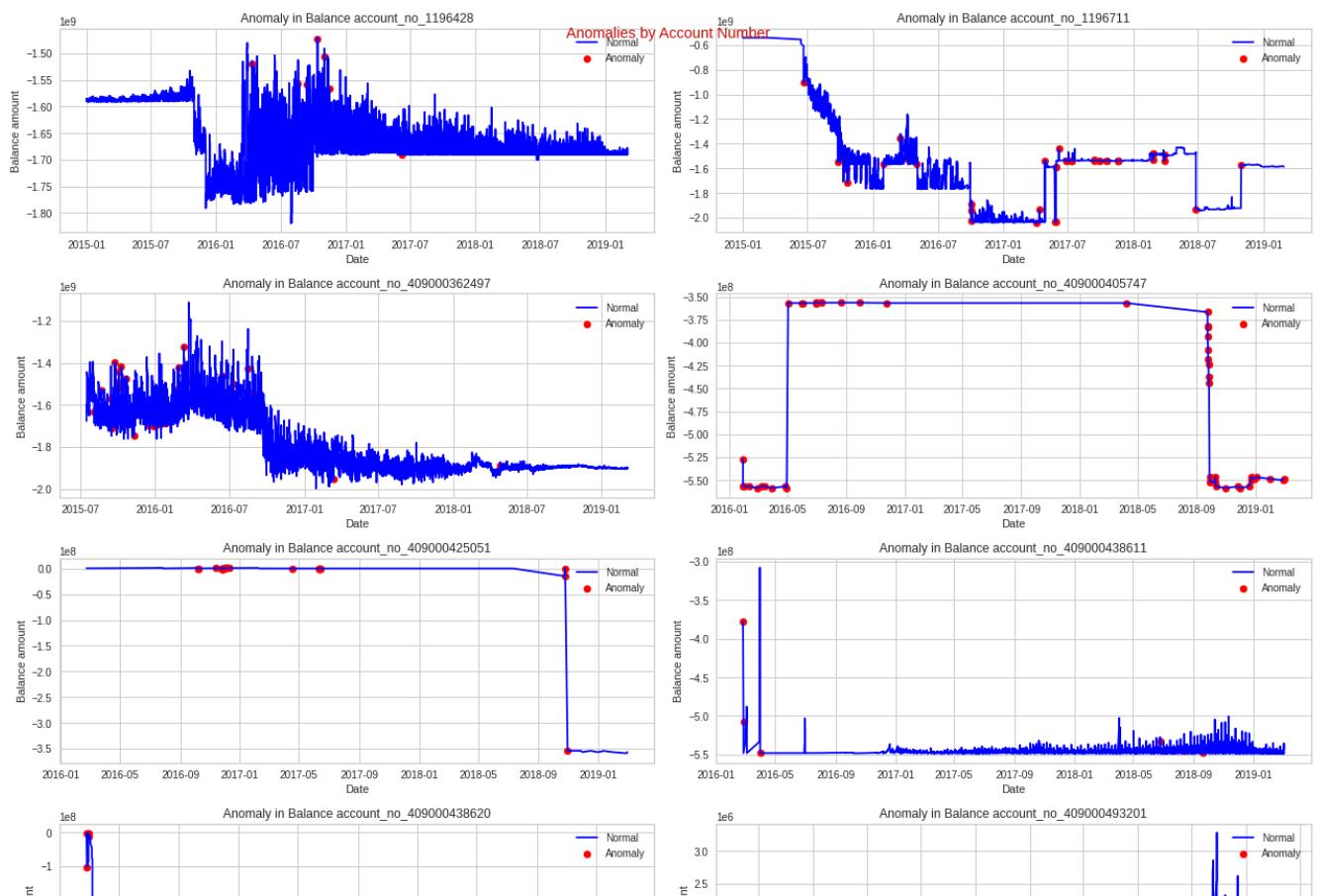
for i in range(5):
    for j in range(2):

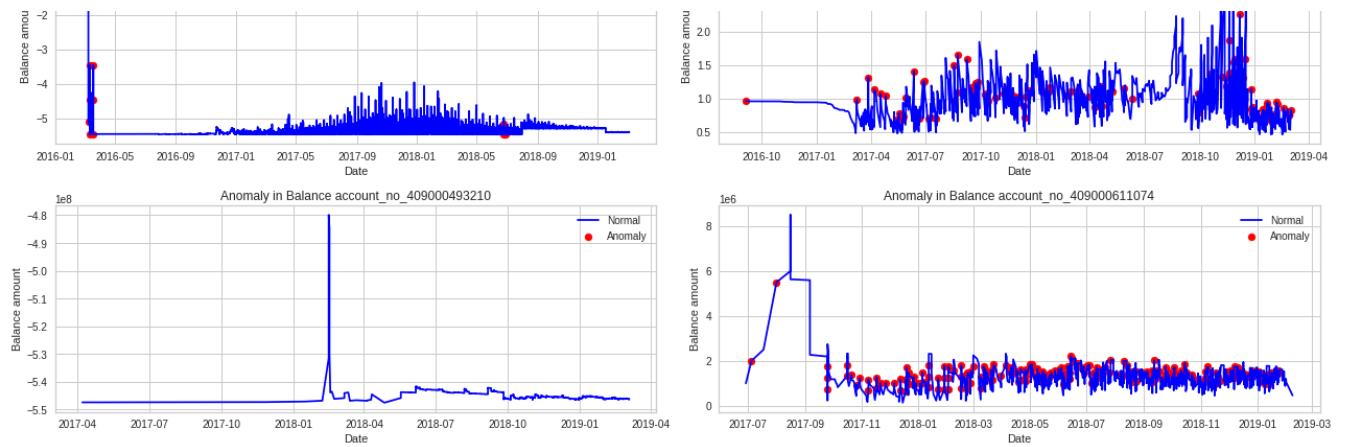
        df = bank_main[bank_main[features[2*i + j]]==1]
        a= df.loc[df['anomaly_kmeans']==1, ['date', 'balance_amt']]

        axes[i,j].plot(df['date'], df['balance_amt'], color='blue', label='Normal')
        axes[i,j].scatter(a['date'], a['balance_amt'], color='red', label='Anomaly')
        axes[i,j].set_title(f'Anomaly in Balance {features[2*i + j]}')
        axes[i,j].set_xlabel('Date')
        axes[i,j].set_ylabel('Balance amount')
        axes[i,j].legend()

plt.tight_layout()
plt.show()

```





Let's follow the same process, **but this time customer by customer**. To do this, we will split the bank data frame and run the clustering approach for each individual data frame.

```
bank_main.columns
```

```
Index(['date', 'value_date', 'day_diff', 'weekend_date', 'weekend_value_da  
    'withdrawal_amt', 'deposit_amt', 'balance_amt',  
    'withdrawal_over_balance', 'deposit_over_balance',  
    'details_null_penalty', 'details_numbers_penalty', 'cheque_penalty',  
    'account_no_1196428', 'account_no_1196711', 'account_no_40900036249',  
    'account_no_409000405747', 'account_no_409000425051',  
    'account_no_409000438611', 'account_no_409000438620',  
    'account_no_409000493201', 'account_no_409000493210',  
    'account_no_409000611074', 'anomaly_isolation_forest',  
    'anomaly_one_class_svm', 'anomaly_gaussian_elliptic_envelope',  
    'anomaly_kmeans'],  
    dtype='object')
```

```
bank_main['anomaly_sum_1'] = bank_main.anomaly_kmeans + bank_main.anomaly_isol  
bank_main.anomaly_sum_1.value_counts()
```

```
0    114443  
1     1110  
2      494  
3      136  
4      18  
Name: anomaly_sum_1, dtype: int64
```

▼ 4. Anomaly Detection by Account Number

In this section, we will analyze each account number independently.

```
accounts = ['1196428', '409000362497', '409000438620', '1196711', '40900049321  
acc1, acc2, acc3, acc4, acc5, acc6, acc7, acc8, acc9, acc10 = bank_premodel[bank_premodel.account_no == accounts[1]].drop('account_no', axis=1),  
bank_premodel[bank_premodel.account_no == accounts[3]].drop('account_no', axis=1),  
bank_premodel[bank_premodel.account_no == accounts[5]].drop('account_no', axis=1),  
bank_premodel[bank_premodel.account_no == accounts[7]].drop('account_no', axis=1),  
bank_premodel[bank_premodel.account_no == accounts[9]].drop('account_no', axis=1),  
df_list = [acc1, acc2, acc3, acc4, acc5, acc6, acc7, acc8, acc9, acc10]
```

► i. Isolation Forest Anomaly Detection

```
[ ] ↳ 75 cells hidden
```

► ii. OneClassSVM Anomaly Detection

```
[ ] ↳ 75 cells hidden
```

► iii. Gaussian Distribution Anomaly Detection (Elliptic Envelope)

```
[ ] ↳ 72 cells hidden
```

▼ iv. KMeans Clustering Anomaly Detection

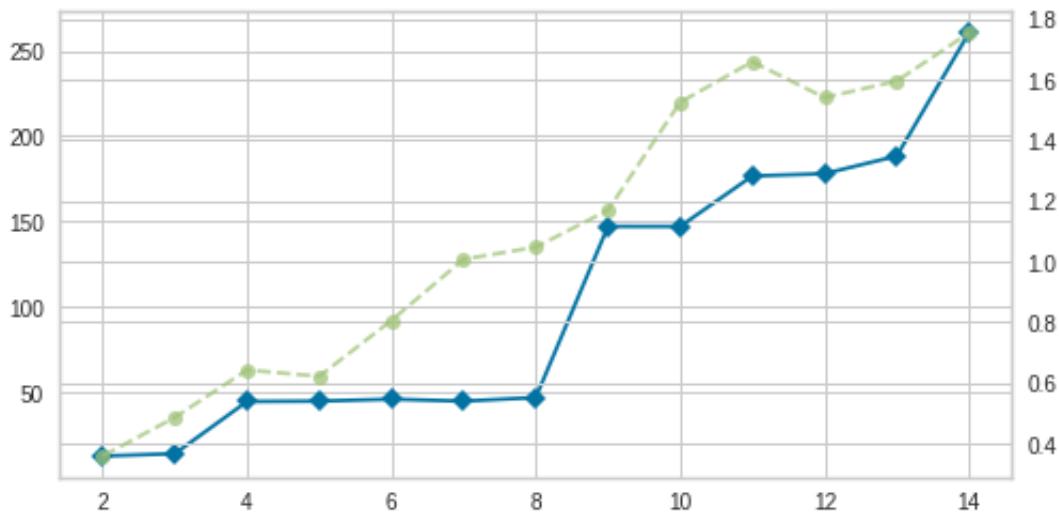
▼ (I) Customer account_no_1196428

```
acc1_kmeans = acc1.copy()
scaler = StandardScaler()
acc1_kmeans_scaled = pd.DataFrame(scaler.fit_transform(acc1_kmeans[acc1.columns])
acc1_kmeans_scaled.set_index(acc1_kmeans.index, drop=True, inplace=True)
acc1_kmeans_scaled.head(3)
```

	day_diff	weekend_date	weekend_value_date	withdrawal_amt	deposit_a
37582	-0.009703	-0.327498	-0.327574	-0.363895	-0.0352
37583	-0.009703	-0.327498	-0.327574	-0.363895	-0.1051
37584	-0.009703	-0.327498	-0.327574	-0.363895	-0.2422

```
# Instantiate the clustering model and visualizer
visualizer = KElbowVisualizer(KMeans(), k=(2,15))
plt.figure(figsize=(8,4)).patch.set_facecolor('xkcd:white')
visualizer.fit(acc1_kmeans_scaled)          # Fit the data to the visualizer
# visualizer.show()           # Finalize and render the figure
```

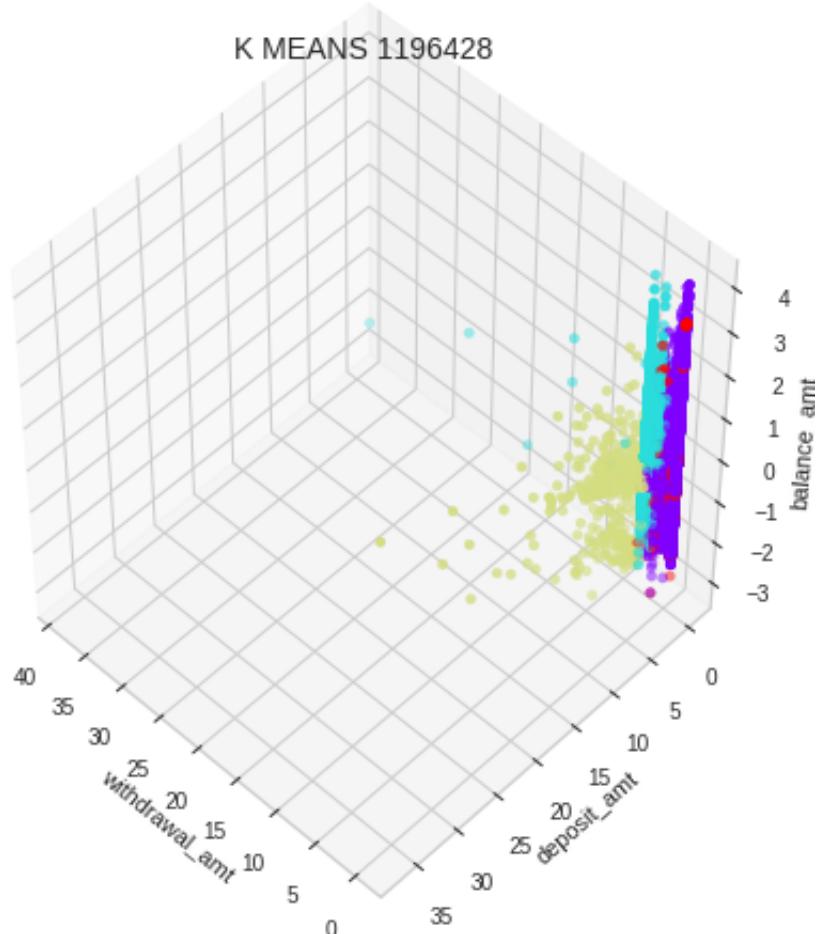
KElbowVisualizer(ax=<matplotlib.axes._subplots.AxesSubplot object at 0x7f26
k=None, metric=None, model=None, timings=True)



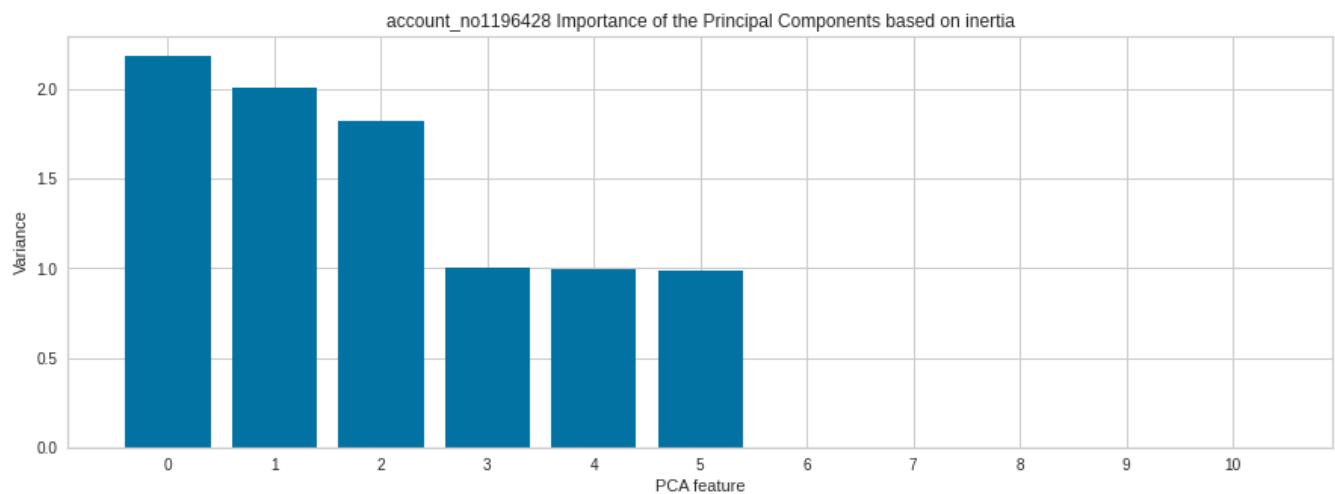
```
kmeans = KMeans(n_clusters=4)
kmeans.fit(acc1_kmeans_scaled)
labels = kmeans.labels_
```

```
fig = plt.figure(1, figsize=(6,6))
ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)
ax.scatter(acc1_kmeans_scaled.iloc[:,3], acc1_kmeans_scaled.iloc[:,4], acc1_kmeans_scaled.iloc[:,0])
ax.set_title(f'K MEANS {accounts[0]}', fontsize=14)
ax.set_xlabel('withdrawal_amt')
ax.set_ylabel('deposit_amt')
ax.set_zlabel('balance_amt')

Text(0.5, 0, 'balance_amt')
```



```
x = acc1_kmeans_scaled
pca = PCA()
pipeline = make_pipeline(pca)
pipeline.fit(x)
# Plot the principal components against their inertia
features = range(pca.n_components_)
_ = plt.figure(figsize=(15, 5))
_ = plt.bar(features, pca.explained_variance_)
_ = plt.xlabel('PCA feature')
_ = plt.ylabel('Variance')
_ = plt.xticks(features)
_ = plt.title(f"account_no{accounts[0]} Importance of the Principal Components")
plt.show()
```



```
acc1_pca = pd.DataFrame(PCA(n_components=5).fit_transform(acc1_kmeans_scaled),
```

```
kmeans = KMeans(n_clusters=4)
kmeans.fit(acc1_pca)
labels = kmeans.predict(acc1_pca)
unique_elements, counts_elements = np.unique(labels, return_counts=True)
clusters = np.asarray((unique_elements, counts_elements))

# Assume that 0.5% of the entire data set are anomalies
outliers_fraction = 0.003
# get the distance between each point and its nearest centroid. The biggest di
distance = getDistanceByPoint(acc1_pca, kmeans)
# number of observations that equate to the 0.5% of the entire data set
number_of_outliers = int(outliers_fraction*len(distance))
# Take the minimum of the largest 0.5% of the distances as the threshold
threshold = distance.nlargest(number_of_outliers).min()
# anomaly1 contain the anomaly result of the above method Cluster (0:normal, 1
acc1_pca['anomaly_kmeans'] = (distance >= threshold).astype(int)

acc1_pca.index = acc1.index
acc1_pca.anomaly_kmeans.value_counts()

0    48633
1     146
Name: anomaly_kmeans, dtype: int64

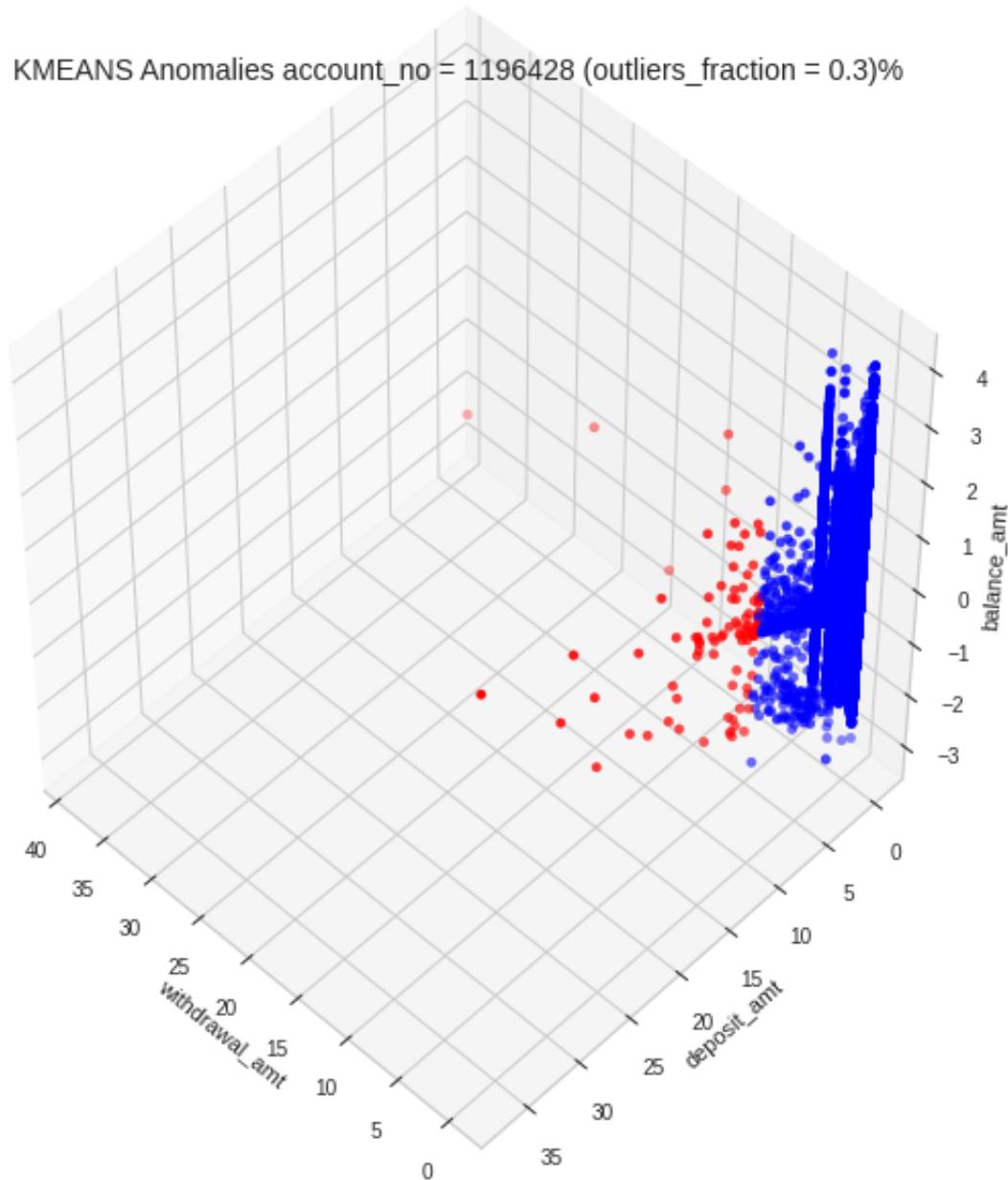
acc1_kmeans_scaled['anomaly_kmeans'] = acc1_pca.anomaly_kmeans
bank_main['km_1'] = 0
bank_main.loc[acc1_kmeans_scaled.index,'km_1'] = acc1_kmeans_scaled['anomaly_k
bank_main['km_1'].value_counts()

0    116055
1     146
Name: km_1, dtype: int64
```

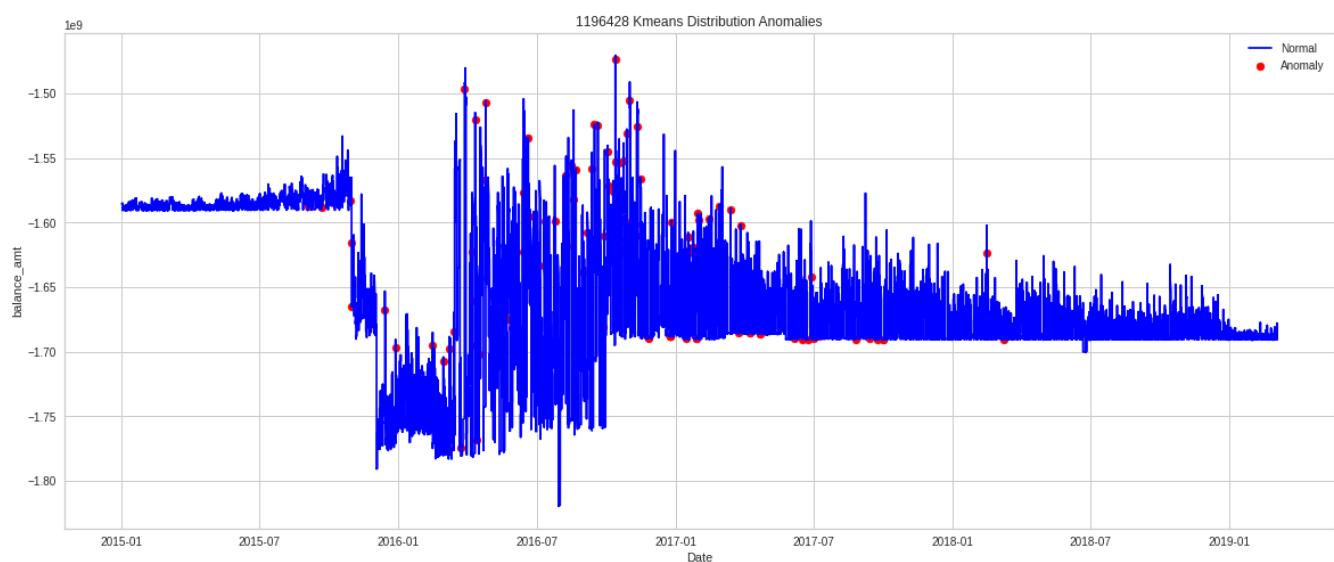
```
labels = acc1_kmeans_scaled.anomaly_kmeans
colors = {0:'blue', 1:'red'}
```

```
fig = plt.figure(1, figsize=(8,8))
ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)
ax.scatter(acc1_kmeans_scaled.iloc[:,3], acc1_kmeans_scaled.iloc[:,4], acc1_kmeans_scaled.iloc[:,5], c=colors[labels])
ax.set_title(f'KMEANS Anomalies account_no = {accounts[0]} (outliers_fraction = {outliers_fraction})')
ax.set_xlabel('withdrawal_amt')
ax.set_ylabel('deposit_amt')
ax.set_zlabel('balance_amt')

Text(0.5, 0, 'balance_amt')
```



```
acc1_kmeans['anomaly_kmeans'] = acc1_kmeans_scaled['anomaly_kmeans']
df = acc1_kmeans.copy()
a= df.loc[df['anomaly_kmeans']==1, ['date', 'balance_amt']]
plt.figure(figsize=(20,8))
plt.plot(df['date'], df['balance_amt'], color='blue', label='Normal')
plt.scatter(a['date'], a['balance_amt'], color='red', label='Anomaly')
plt.title(f'{accounts[0]} Kmeans Distribution Anomalies')
plt.xlabel('Date')
plt.ylabel('balance_amt')
plt.legend()
plt.show()
```



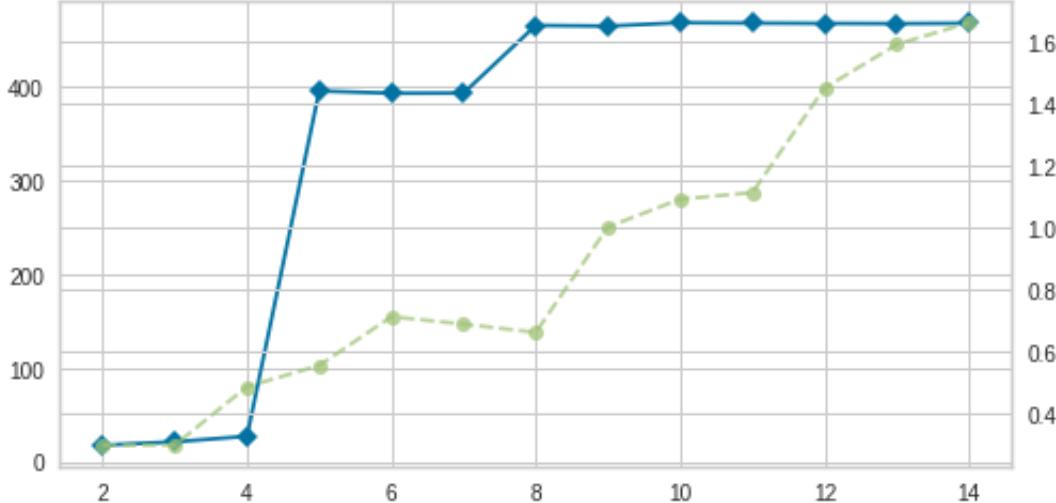
▼ (II) Customer account_no_409000362497

```
acc2_kmeans = acc2.copy()
scaler = StandardScaler()
acc2_kmeans_scaled = pd.DataFrame(scaler.fit_transform(acc2_kmeans[acc2.columns])
acc2_kmeans_scaled.set_index(acc2_kmeans.index, drop=True, inplace=True)
acc2_kmeans_scaled.head(3)
```

	day_diff	weekend_date	weekend_value_date	withdrawal_amt	deposit_a
86361	-0.020357	-0.286669		-0.286463	-0.297246
86362	-0.020357	-0.286669		-0.286463	-0.297246
86363	-0.020357	-0.286669		-0.286463	-0.297246

```
# Instantiate the clustering model and visualizer
visualizer = KElbowVisualizer(KMeans(), k=(2,15))
plt.figure(figsize=(8,4)).patch.set_facecolor('xkcd:white')
visualizer.fit(acc2_kmeans_scaled)          # Fit the data to the visualizer
# visualizer.show()                      # Finalize and render the figure
```

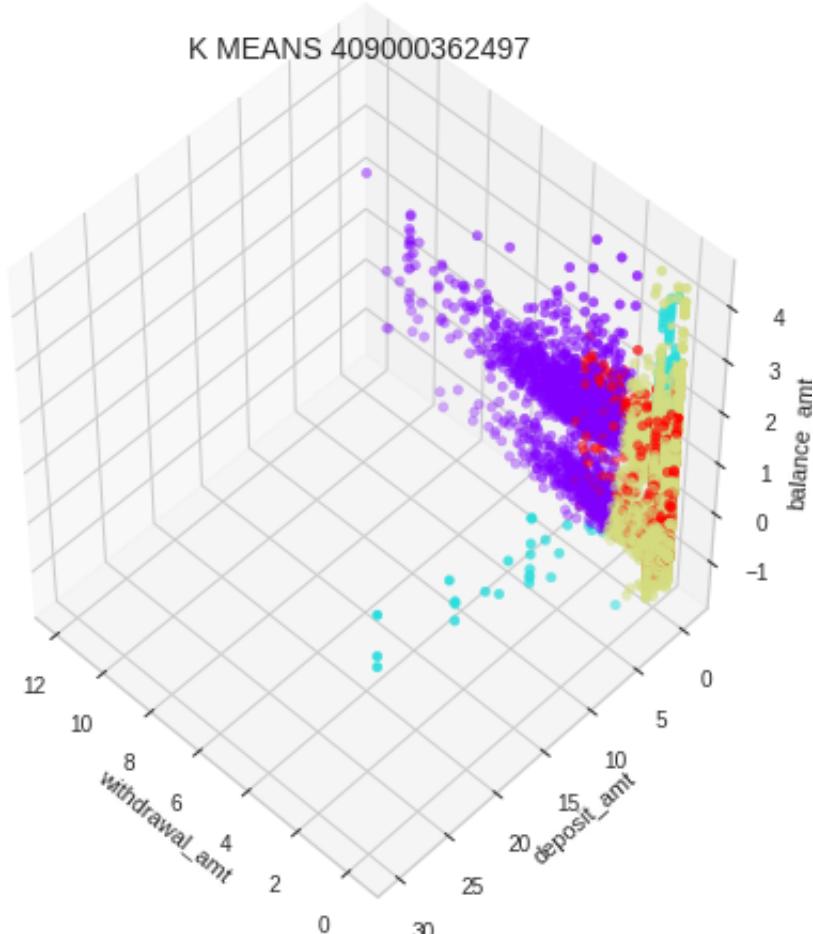
KElbowVisualizer(ax=<matplotlib.axes._subplots.AxesSubplot object at 0x7f25
k=None, metric=None, model=None, timings=True)



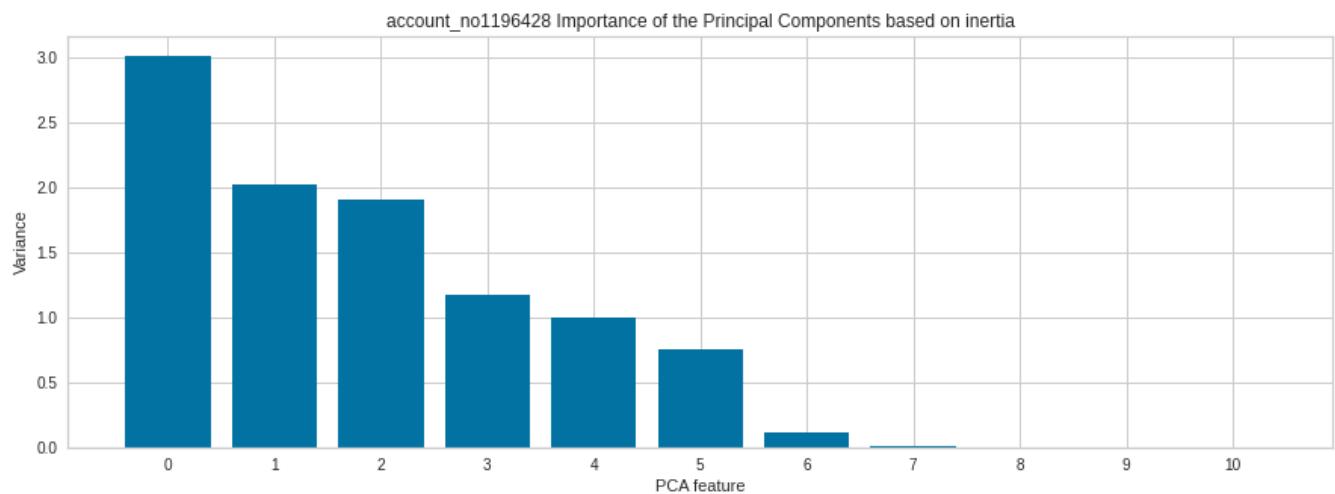
```
kmeans = KMeans(n_clusters=4)
kmeans.fit(acc2_kmeans_scaled)
labels = kmeans.labels_
```

```
fig = plt.figure(1, figsize=(6,6))
ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)
ax.scatter(acc2_kmeans_scaled.iloc[:,3], acc2_kmeans_scaled.iloc[:,4], acc2_kmeans_scaled.iloc[:,5])
ax.set_title(f'K MEANS {accounts[1]}', fontsize=14)
ax.set_xlabel('withdrawal_amt')
ax.set_ylabel('deposit_amt')
ax.set_zlabel('balance_amt')

Text(0.5, 0, 'balance_amt')
```



```
x = acc2_kmeans_scaled
pca = PCA()
pipeline = make_pipeline(pca)
pipeline.fit(x)
# Plot the principal components against their inertia
features = range(pca.n_components_)
_ = plt.figure(figsize=(15, 5))
_ = plt.bar(features, pca.explained_variance_)
_ = plt.xlabel('PCA feature')
_ = plt.ylabel('Variance')
_ = plt.xticks(features)
_ = plt.title(f"account_no{accounts[0]} Importance of the Principal Components")
plt.show()
```



```
acc2_pca = pd.DataFrame(PCA(n_components=6).fit_transform(acc2_kmeans_scaled),
```

```
kmeans = KMeans(n_clusters=4)
kmeans.fit(acc2_pca)
labels = kmeans.predict(acc2_pca)
unique_elements, counts_elements = np.unique(labels, return_counts=True)
clusters = np.asarray((unique_elements, counts_elements))

outliers_fraction = 0.003
distance = getDistanceByPoint(acc2_pca, kmeans)
number_of_outliers = int(outliers_fraction*len(distance))
threshold = distance.nlargest(number_of_outliers).min()
acc2_pca['anomaly_kmeans'] = (distance >= threshold).astype(int)

acc2_pca.index = acc2.index
acc2_pca.anomaly_kmeans.value_counts()

0    29751
1      89
Name: anomaly_kmeans, dtype: int64

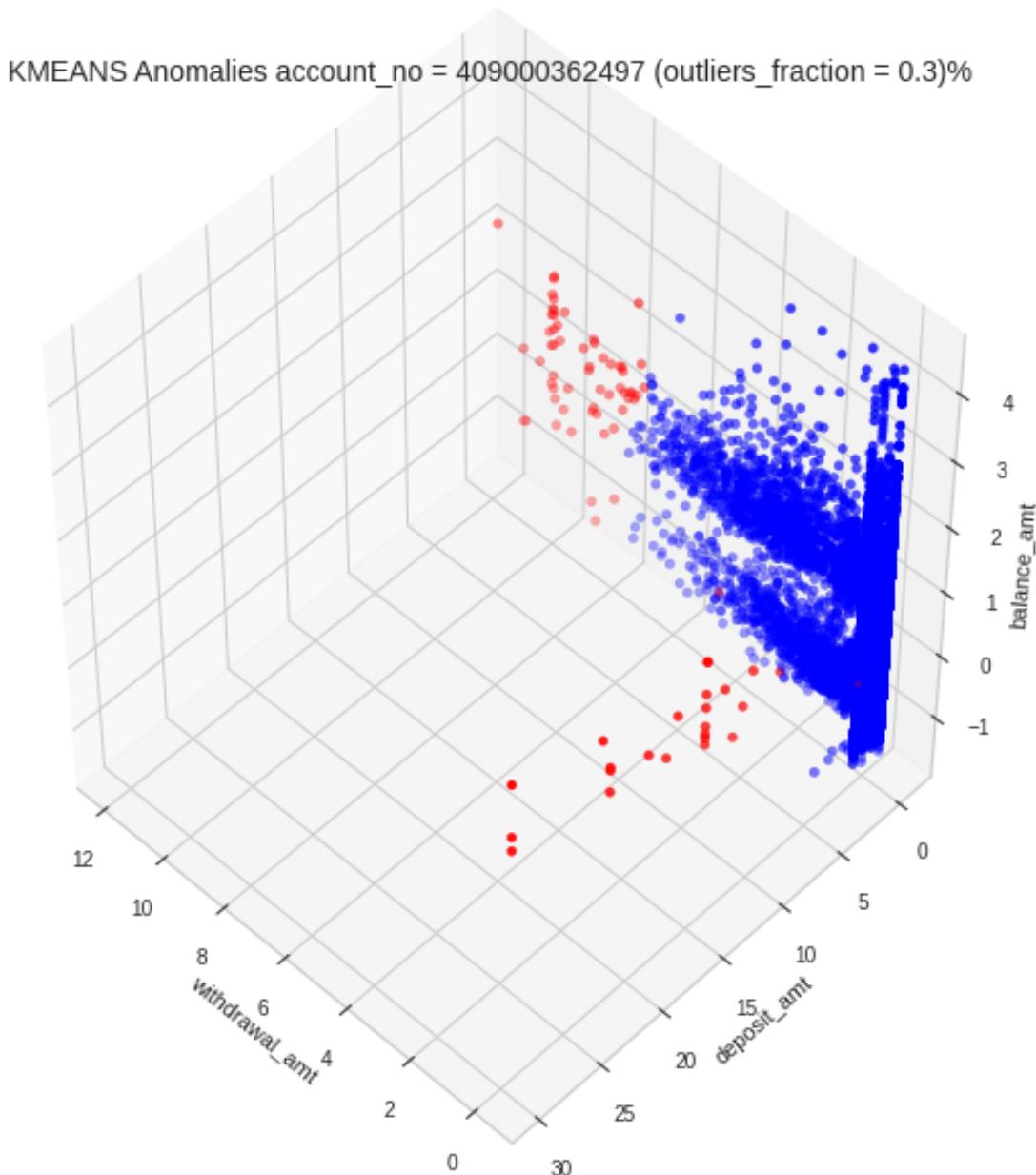
acc2_kmeans_scaled['anomaly_kmeans'] = acc2_pca.anomaly_kmeans
bank_main['km_2'] = 0
bank_main.loc[acc2_kmeans_scaled.index,'km_2'] = acc2_kmeans_scaled['anomaly_kmeans']
bank_main['km_2'].value_counts()

0    116112
1      89
Name: km_2, dtype: int64
```

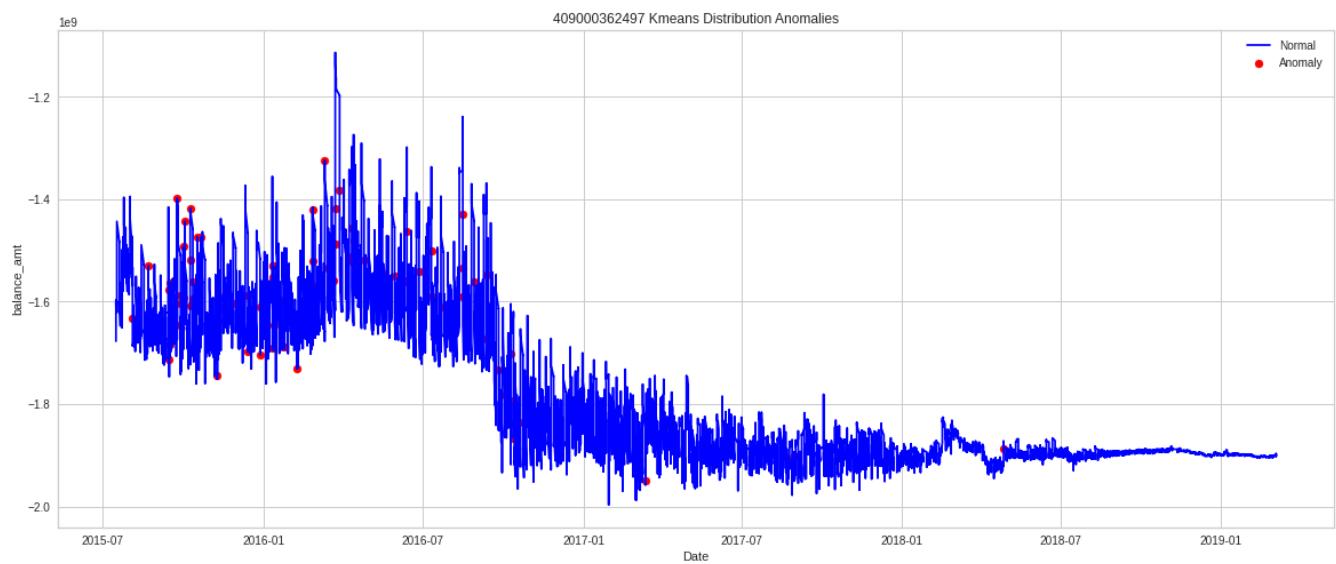
```
labels = acc2_kmeans_scaled.anomaly_kmeans
colors = {0:'blue', 1:'red'}
```

```
fig = plt.figure(1, figsize=(8,8))
ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)
ax.scatter(acc2_kmeans_scaled.iloc[:,3], acc2_kmeans_scaled.iloc[:,4], acc2_kmeans_scaled.iloc[:,5], c=colors[labels])
ax.set_title(f'KMEANS Anomalies account_no = {accounts[1]} (outliers_fraction = 0.3)%')
ax.set_xlabel('withdrawal_amt')
ax.set_ylabel('deposit_amt')
ax.set_zlabel('balance_amt')

Text(0.5, 0, 'balance_amt')
```



```
acc2_kmeans['anomaly_kmeans'] = acc2_kmeans_scaled['anomaly_kmeans']
df = acc2_kmeans.copy()
a= df.loc[df['anomaly_kmeans']==1, ['date', 'balance_amt']]
plt.figure(figsize=(20,8))
plt.plot(df['date'], df['balance_amt'], color='blue', label='Normal')
plt.scatter(a['date'], a['balance_amt'], color='red', label='Anomaly')
plt.title(f'{accounts[1]} Kmeans Distribution Anomalies')
plt.xlabel('Date')
plt.ylabel('balance_amt')
plt.legend()
plt.show()
```



accounts[2]

'409000438620'

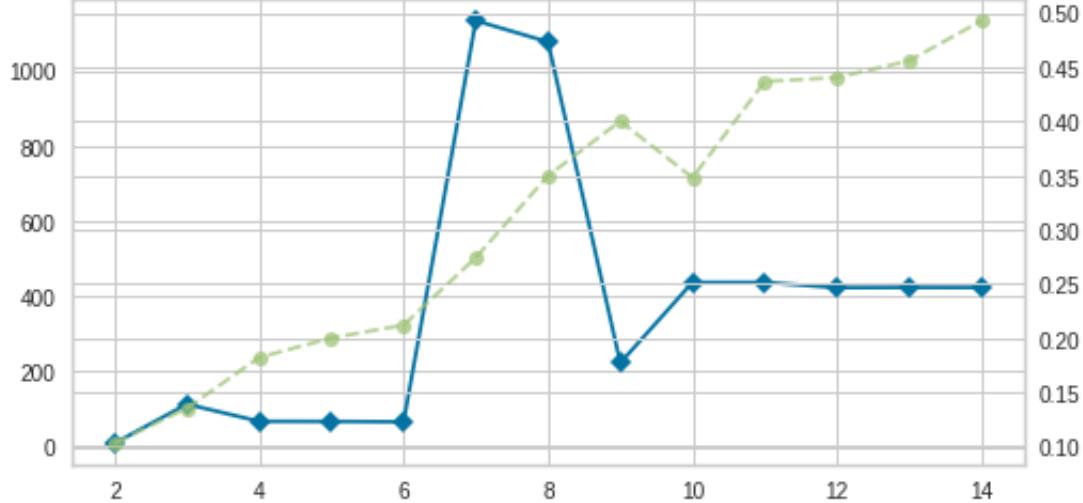
▼ (III) Customer account_no_409000438620

```
acc3_kmeans = acc3.copy()
scaler = StandardScaler()
acc3_kmeans_scaled = pd.DataFrame(scaler.fit_transform(acc3_kmeans[acc3.columns])
acc3_kmeans_scaled.set_index(acc3_kmeans.index, drop=True, inplace=True)
acc3_kmeans_scaled.head(3)
```

	day_diff	weekend_date	weekend_value_date	withdrawal_amt	deposit_a
13592	0.0	-0.310493		-0.310493	-0.199596
13593	0.0	-0.310493		-0.310493	15.416558
13594	0.0	-0.310493		-0.310493	-0.199596

```
# Instantiate the clustering model and visualizer
visualizer = KElbowVisualizer(KMeans(), k=(2,15))
plt.figure(figsize=(8,4)).patch.set_facecolor('xkcd:white')
visualizer.fit(acc3_kmeans_scaled)          # Fit the data to the visualizer
# visualizer.show()                      # Finalize and render the figure
```

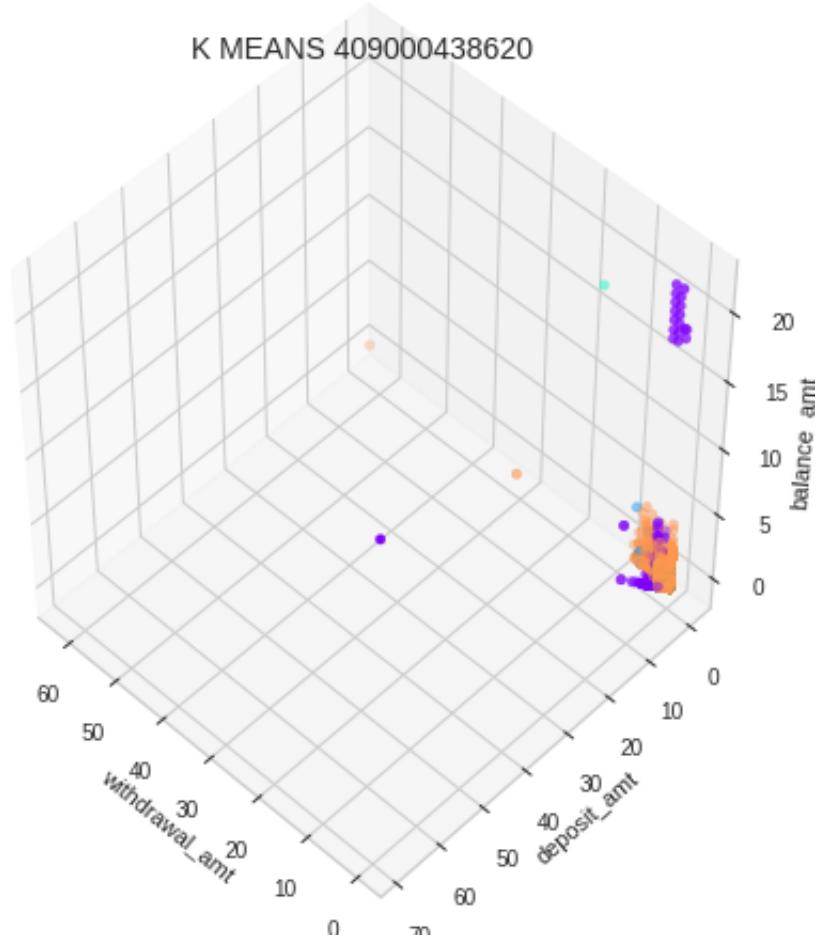
KElbowVisualizer(ax=<matplotlib.axes._subplots.AxesSubplot object at 0x7f25
k=None, metric=None, model=None, timings=True)



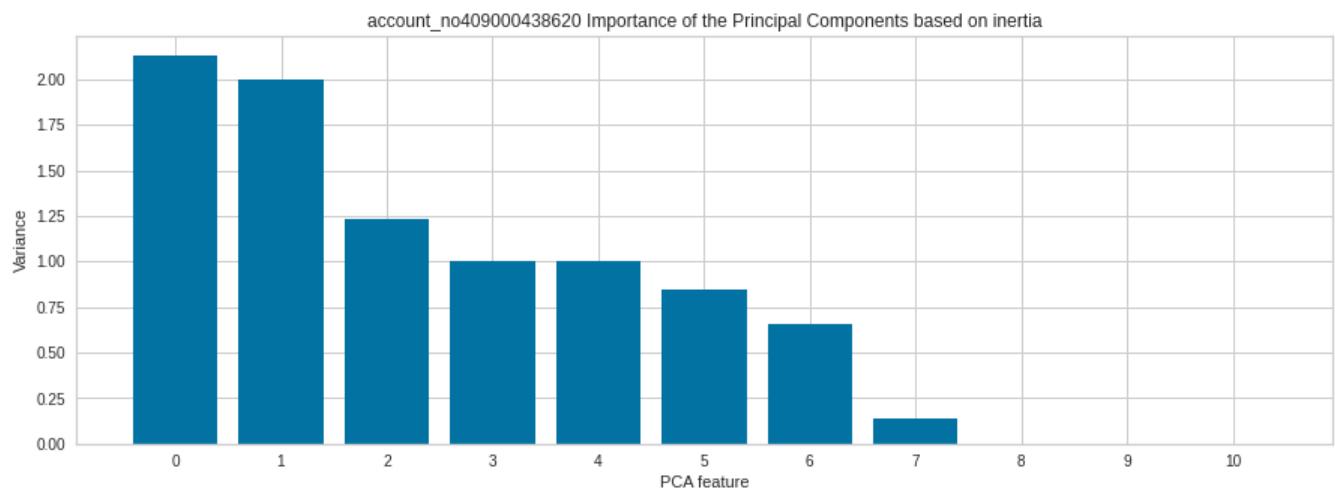
```
kmeans = KMeans(n_clusters=6)
kmeans.fit(acc3_kmeans_scaled)
labels = kmeans.labels_
```

```
fig = plt.figure(1, figsize=(6,6))
ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)
ax.scatter(acc3_kmeans_scaled.iloc[:,3], acc3_kmeans_scaled.iloc[:,4], acc3_kmeans_scaled.iloc[:,5])
ax.set_title(f'K MEANS {accounts[2]}', fontsize=14)
ax.set_xlabel('withdrawal_amt')
ax.set_ylabel('deposit_amt')
ax.set_zlabel('balance_amt')

Text(0.5, 0, 'balance_amt')
```



```
x = acc3_kmeans_scaled
pca = PCA()
pipeline = make_pipeline(pca)
pipeline.fit(x)
# Plot the principal components against their inertia
features = range(pca.n_components_)
_ = plt.figure(figsize=(15, 5))
_ = plt.bar(features, pca.explained_variance_)
_ = plt.xlabel('PCA feature')
_ = plt.ylabel('Variance')
_ = plt.xticks(features)
_ = plt.title(f"account_no{accounts[2]} Importance of the Principal Components")
plt.show()
```



```
acc3_pca = pd.DataFrame(PCA(n_components=7).fit_transform(acc3_kmeans_scaled),
```

```
kmeans = KMeans(n_clusters=6)
kmeans.fit(acc3_pca)
labels = kmeans.predict(acc3_pca)

unique_elements, counts_elements = np.unique(labels, return_counts=True)
clusters = np.asarray((unique_elements, counts_elements))

outliers_fraction = 0.003
distance = getDistanceByPoint(acc3_pca, kmeans)
number_of_outliers = int(outliers_fraction*len(distance))
threshold = distance.nlargest(number_of_outliers).min()
acc3_pca['anomaly_kmeans'] = (distance >= threshold).astype(int)

acc3_pca.index = acc3.index
acc3_pca.anomaly_kmeans.value_counts()

0    13414
1      40
Name: anomaly_kmeans, dtype: int64

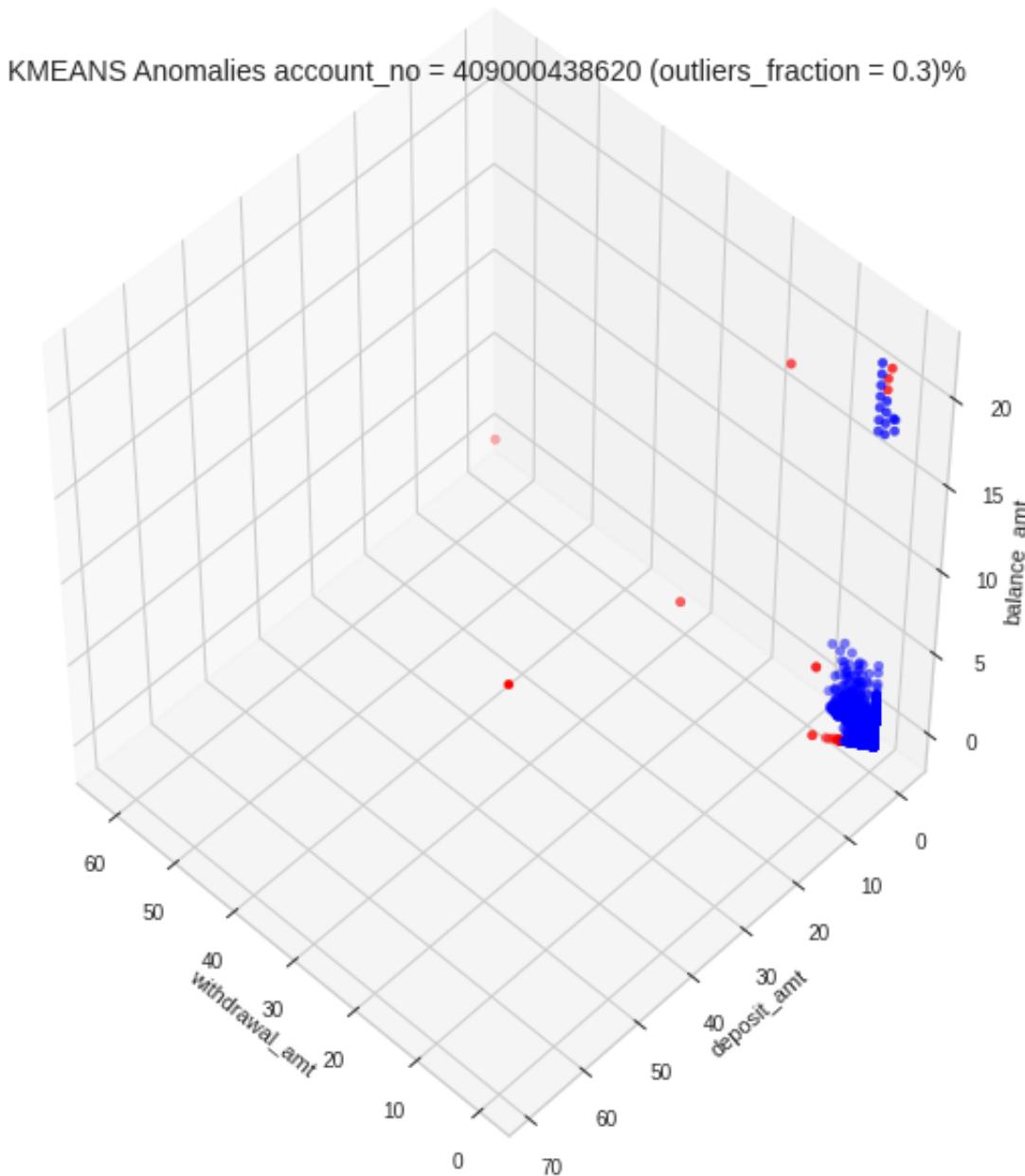
acc3_kmeans_scaled['anomaly_kmeans'] = acc3_pca.anomaly_kmeans
bank_main['km_3'] = 0
bank_main.loc[acc3_kmeans_scaled.index, 'km_3'] = acc3_kmeans_scaled['anomaly_kmeans']
bank_main['km_3'].value_counts()

0    116161
1      40
Name: km_3, dtype: int64
```

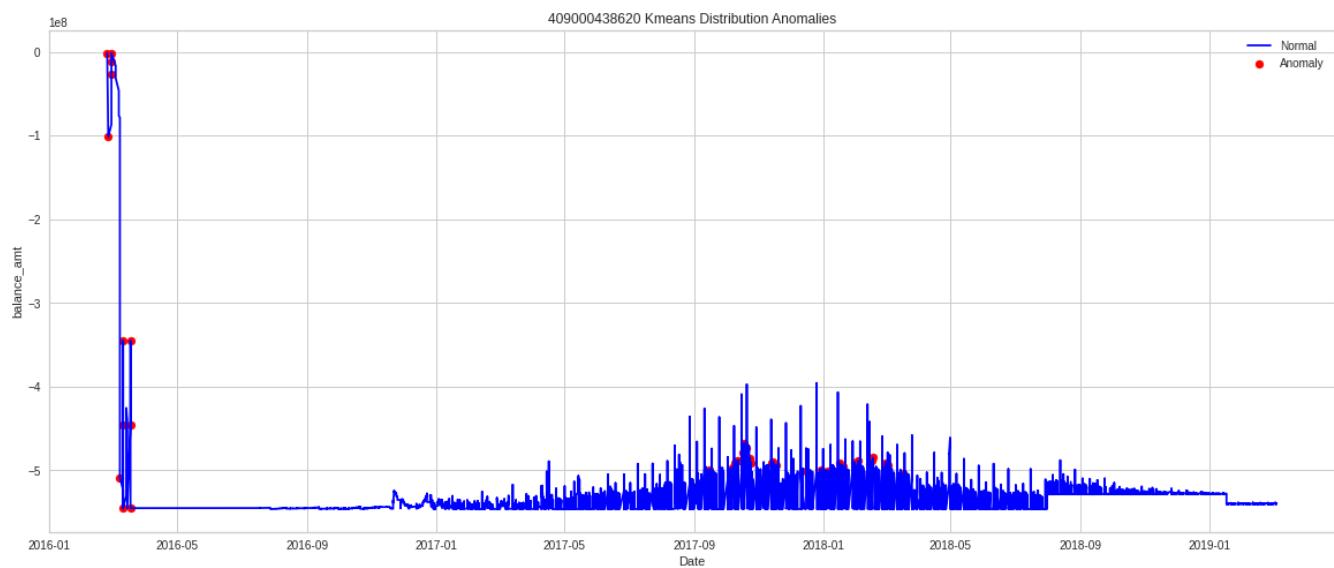
```
labels = acc3_kmeans_scaled.anomaly_kmeans
colors = {0:'blue', 1:'red'}
```

```
fig = plt.figure(1, figsize=(8,8))
ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)
ax.scatter(acc3_kmeans_scaled.iloc[:,3], acc3_kmeans_scaled.iloc[:,4], acc3_kmeans_scaled.iloc[:,5], c=colors[labels])
ax.set_title(f'KMEANS Anomalies account_no = {accounts[2]} (outliers_fraction = 0.3)%')
ax.set_xlabel('withdrawal_amt')
ax.set_ylabel('deposit_amt')
ax.set_zlabel('balance_amt')

Text(0.5, 0, 'balance_amt')
```



```
acc3_kmeans['anomaly_kmeans'] = acc3_kmeans_scaled['anomaly_kmeans']
df = acc3_kmeans.copy()
a= df.loc[df['anomaly_kmeans']==1, ['date', 'balance_amt']]
plt.figure(figsize=(20,8))
plt.plot(df['date'], df['balance_amt'], color='blue', label='Normal')
plt.scatter(a['date'], a['balance_amt'], color='red', label='Anomaly')
plt.title(f'{accounts[2]} Kmeans Distribution Anomalies')
plt.xlabel('Date')
plt.ylabel('balance_amt')
plt.legend()
plt.show()
```



accounts[3]

'1196711'

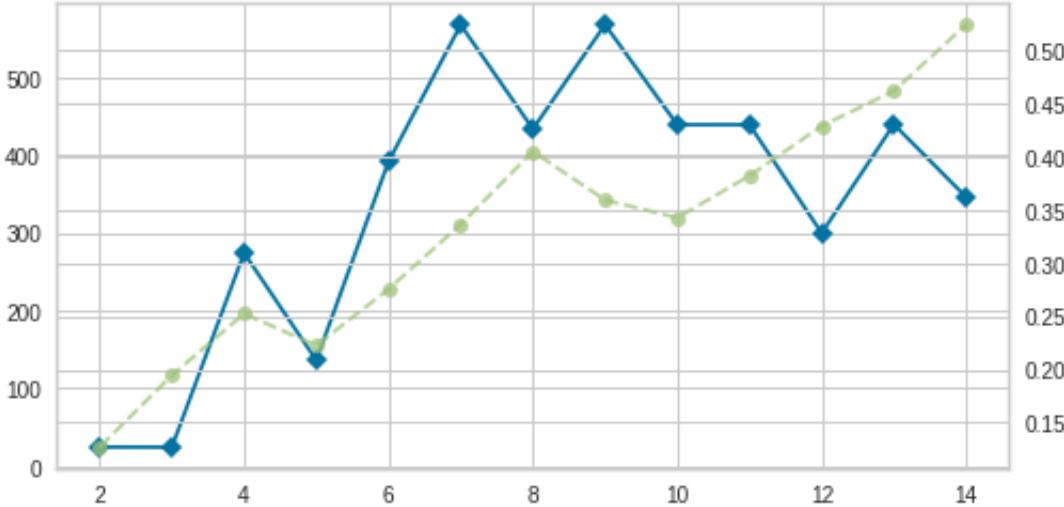
▼ (IV) Customer account_no_1196711

```
acc4_kmeans = acc4.copy()
scaler = StandardScaler()
acc4_kmeans_scaled = pd.DataFrame(scaler.fit_transform(acc4_kmeans[acc4.columns])
acc4_kmeans_scaled.set_index(acc4_kmeans.index, drop=True, inplace=True)
acc4_kmeans_scaled.head(3)
```

	day_diff	weekend_date	weekend_value_date	withdrawal_amt	deposit_a
27046	-0.06111	-0.308943		-0.309496	-0.415321
27047	-0.06111	-0.308943		-0.309496	-0.415321
27048	-0.06111	-0.308943		-0.309496	-0.415321

```
# Instantiate the clustering model and visualizer
visualizer = KElbowVisualizer(KMeans(), k=(2,15))
plt.figure(figsize=(8,4)).patch.set_facecolor('xkcd:white')
visualizer.fit(acc4_kmeans_scaled)          # Fit the data to the visualizer
# visualizer.show()                      # Finalize and render the figure
```

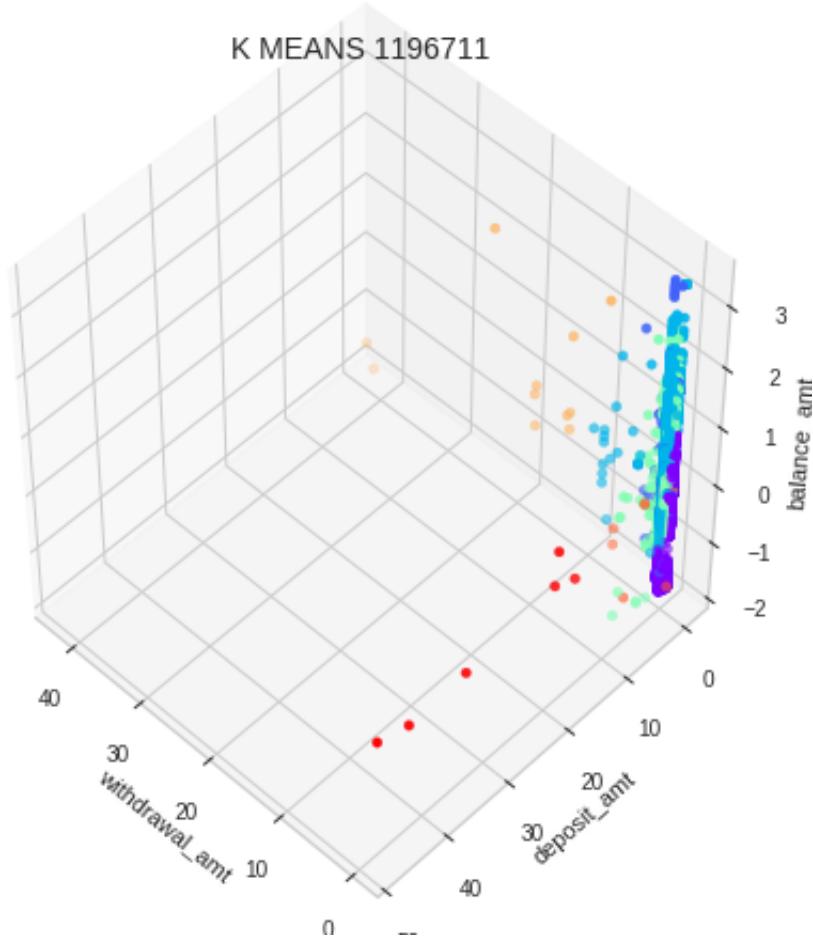
KElbowVisualizer(ax=<matplotlib.axes._subplots.AxesSubplot object at 0x7f25
k=None, metric=None, model=None, timings=True)



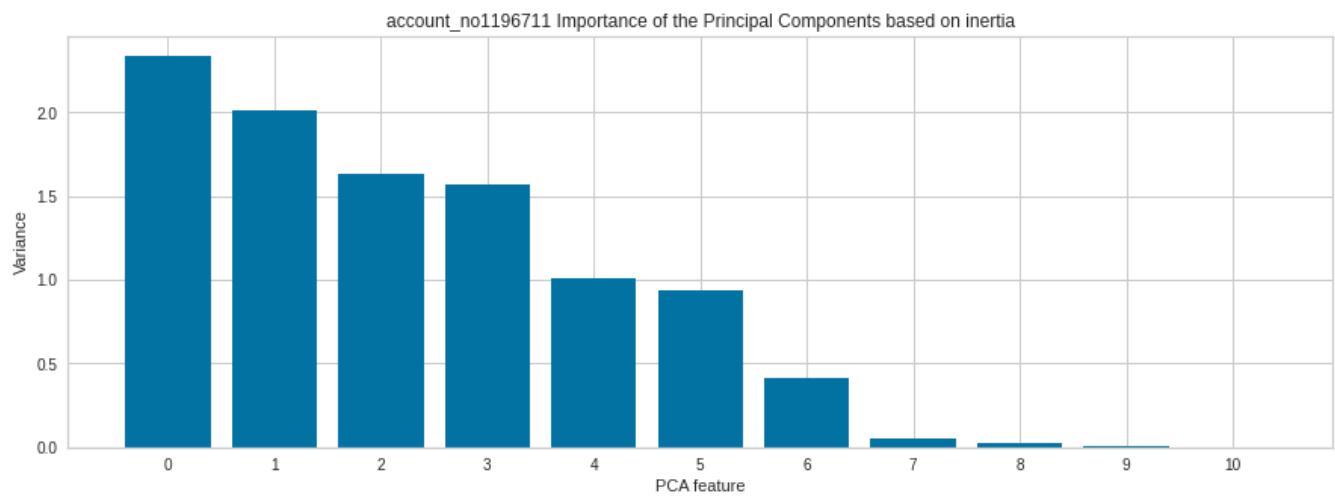
```
kmeans = KMeans(n_clusters=9)
kmeans.fit(acc4_kmeans_scaled)
labels = kmeans.labels_
```

```
fig = plt.figure(1, figsize=(6,6))
ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)
ax.scatter(acc4_kmeans_scaled.iloc[:,3], acc4_kmeans_scaled.iloc[:,4], acc4_kmeans_scaled.iloc[:,5])
ax.set_title(f'K MEANS {accounts[3]}', fontsize=14)
ax.set_xlabel('withdrawal_amt')
ax.set_ylabel('deposit_amt')
ax.set_zlabel('balance_amt')

Text(0.5, 0, 'balance_amt')
```



```
x = acc4_kmeans_scaled
pca = PCA()
pipeline = make_pipeline(pca)
pipeline.fit(x)
# Plot the principal components against their inertia
features = range(pca.n_components_)
_ = plt.figure(figsize=(15, 5))
_ = plt.bar(features, pca.explained_variance_)
_ = plt.xlabel('PCA feature')
_ = plt.ylabel('Variance')
_ = plt.xticks(features)
_ = plt.title(f"account_no{accounts[3]} Importance of the Principal Components")
plt.show()
```



```
acc4_pca = pd.DataFrame(PCA(n_components=7).fit_transform(acc4_kmeans_scaled),
```

```
kmeans = KMeans(n_clusters=9)
kmeans.fit(acc4_pca)
labels = kmeans.predict(acc4_pca)

unique_elements, counts_elements = np.unique(labels, return_counts=True)
clusters = np.asarray((unique_elements, counts_elements))

outliers_fraction = 0.005
distance = getDistanceByPoint(acc4_pca, kmeans)

number_of_outliers = int(outliers_fraction*len(distance))
threshold = distance.nlargest(number_of_outliers).min()
acc4_pca['anomaly_kmeans'] = (distance >= threshold).astype(int)

acc4_pca.index = acc4.index
acc4_pca.anomaly_kmeans.value_counts()

0    10484
1      52
Name: anomaly_kmeans, dtype: int64

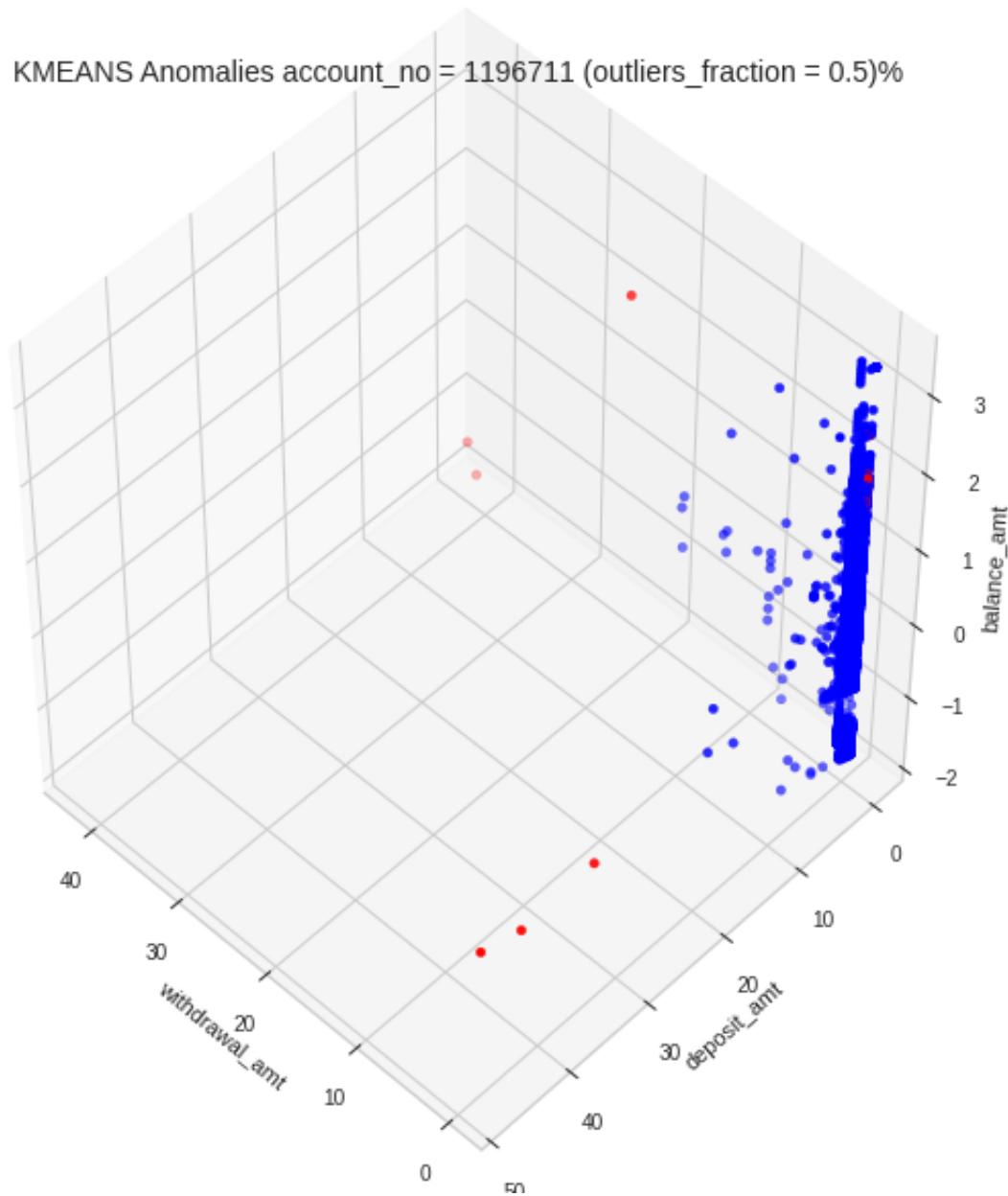
acc4_kmeans_scaled['anomaly_kmeans'] = acc4_pca.anomaly_kmeans
bank_main['km_4'] = 0
bank_main.loc[acc4_kmeans_scaled.index, 'km_4'] = acc4_kmeans_scaled['anomaly_kmeans']
bank_main['km_4'].value_counts()

0    116149
1      52
Name: km_4, dtype: int64
```

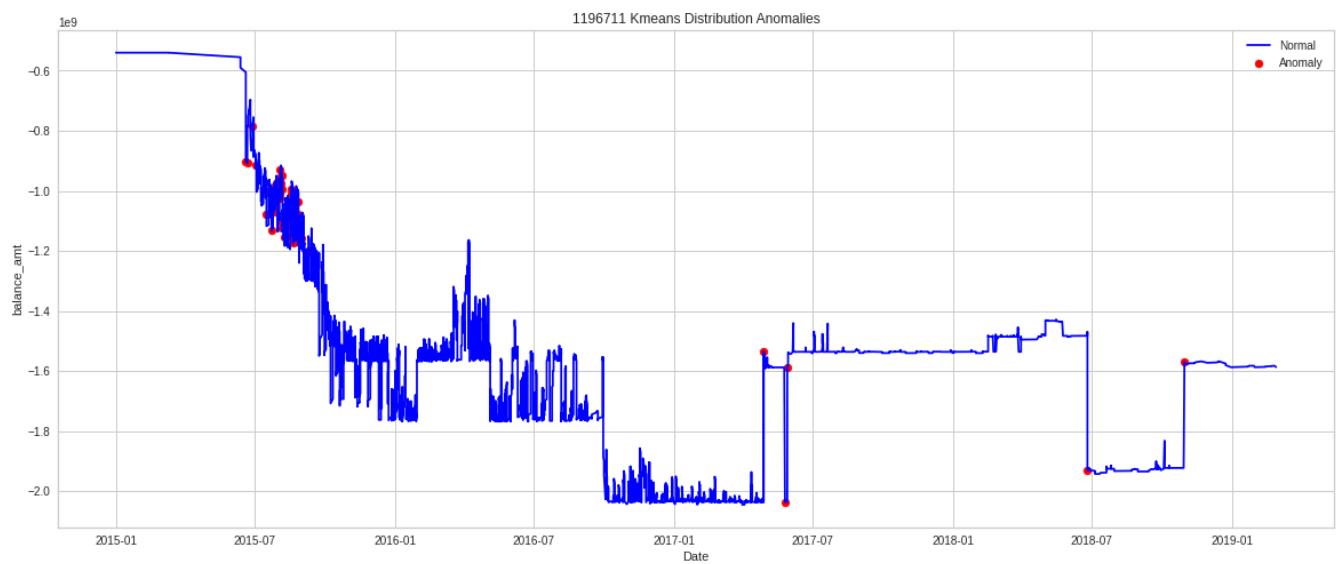
```
labels = acc4_kmeans_scaled.anomaly_kmeans
colors = {0:'blue', 1:'red'}
```

```
fig = plt.figure(1, figsize=(8,8))
ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)
ax.scatter(acc4_kmeans_scaled.iloc[:,3], acc4_kmeans_scaled.iloc[:,4], acc4_kmeans_scaled.iloc[:,5], c=colors[labels])
ax.set_title(f'KMEANS Anomalies account_no = {accounts[3]} (outliers_fraction = 0.5)%')
ax.set_xlabel('withdrawal_amt')
ax.set_ylabel('deposit_amt')
ax.set_zlabel('balance_amt')

Text(0.5, 0, 'balance_amt')
```



```
acc4_kmeans['anomaly_kmeans'] = acc4_kmeans_scaled['anomaly_kmeans']
df = acc4_kmeans.copy()
a= df.loc[df['anomaly_kmeans']==1, ['date', 'balance_amt']]
plt.figure(figsize=(20,8))
plt.plot(df['date'], df['balance_amt'], color='blue', label='Normal')
plt.scatter(a['date'], a['balance_amt'], color='red', label='Anomaly')
plt.title(f'{accounts[3]} Kmeans Distribution Anomalies')
plt.xlabel('Date')
plt.ylabel('balance_amt')
plt.legend()
plt.show()
```



accounts[4]

'409000493210'

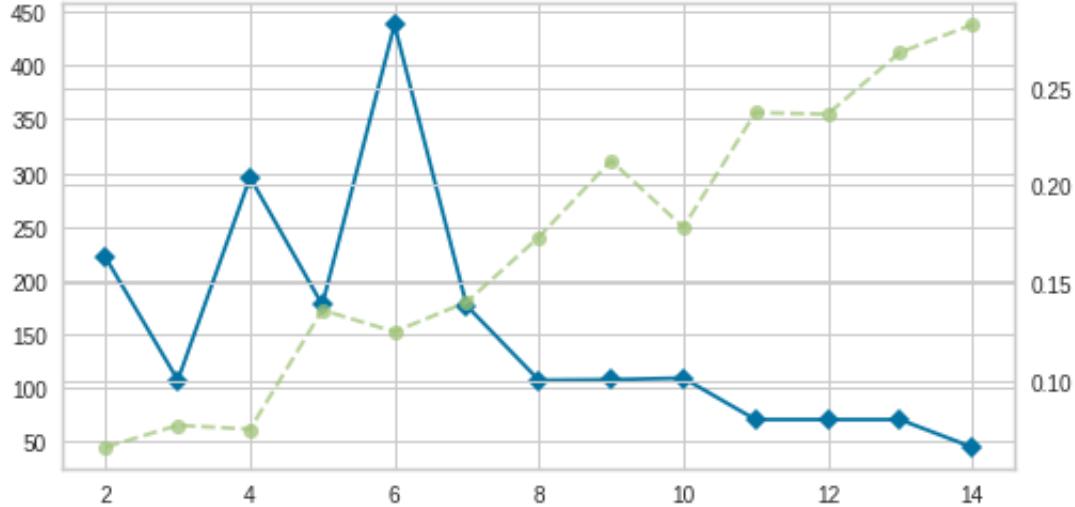
▼ (V) Customer account_no_409000493210

```
acc5_kmeans = acc5.copy()
scaler = StandardScaler()
acc5_kmeans_scaled = pd.DataFrame(scaler.fit_transform(acc5_kmeans[acc5.columns])
acc5_kmeans_scaled.set_index(acc5_kmeans.index, drop=True, inplace=True)
acc5_kmeans_scaled.head(3)
```

	day_diff	weekend_date	weekend_value_date	withdrawal_amt	deposit_am
7578	0.0	-0.249713		-0.249713	-0.042609
7579	0.0	-0.249713		-0.249713	-0.042227
7580	0.0	-0.249713		-0.249713	-0.042555

```
# Instantiate the clustering model and visualizer
visualizer = KElbowVisualizer(KMeans(), k=(2,15))
plt.figure(figsize=(8,4)).patch.set_facecolor('xkcd:white')
visualizer.fit(acc5_kmeans_scaled)          # Fit the data to the visualizer
# visualizer.show()                      # Finalize and render the figure
```

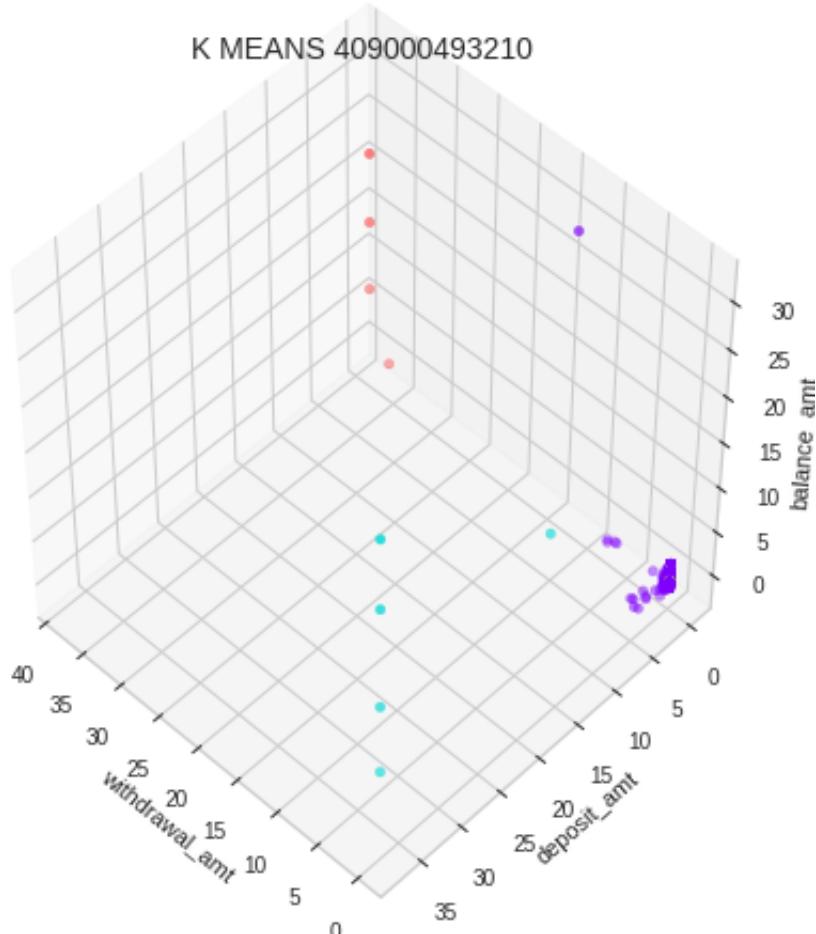
KElbowVisualizer(ax=<matplotlib.axes._subplots.AxesSubplot object at 0x7f25
k=None, metric=None, model=None, timings=True)



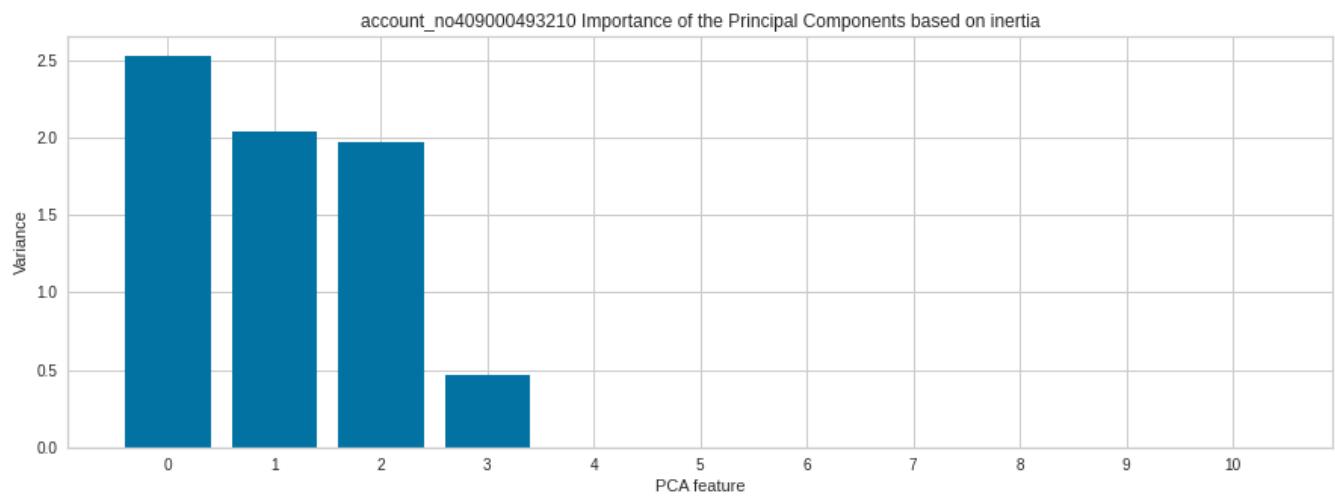
```
kmeans = KMeans(n_clusters=4)
kmeans.fit(acc5_kmeans_scaled)
labels = kmeans.labels_
```

```
fig = plt.figure(1, figsize=(6,6))
ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)
ax.scatter(acc5_kmeans_scaled.iloc[:,3], acc5_kmeans_scaled.iloc[:,4], acc5_kmeans_scaled.iloc[:,5])
ax.set_title(f'K MEANS {accounts[4]}', fontsize=14)
ax.set_xlabel('withdrawal_amt')
ax.set_ylabel('deposit_amt')
ax.set_zlabel('balance_amt')

Text(0.5, 0, 'balance_amt')
```



```
x = acc5_kmeans_scaled
pca = PCA()
pipeline = make_pipeline(pca)
pipeline.fit(x)
# Plot the principal components against their inertia
features = range(pca.n_components_)
_ = plt.figure(figsize=(15, 5))
_ = plt.bar(features, pca.explained_variance_)
_ = plt.xlabel('PCA feature')
_ = plt.ylabel('Variance')
_ = plt.xticks(features)
_ = plt.title(f"account_no{accounts[4]} Importance of the Principal Components")
plt.show()
```



```
acc5_pca = pd.DataFrame(PCA(n_components=3).fit_transform(acc5_kmeans_scaled),
```

```
kmeans = KMeans(n_clusters=4)
kmeans.fit(acc5_pca)
labels = kmeans.predict(acc5_pca)

unique_elements, counts_elements = np.unique(labels, return_counts=True)
clusters = np.asarray((unique_elements, counts_elements))

outliers_fraction = 0.005
distance = getDistanceByPoint(acc5_pca, kmeans)

number_of_outliers = int(outliers_fraction*len(distance))
threshold = distance.nlargest(number_of_outliers).min()
acc5_pca['anomaly_kmeans'] = (distance >= threshold).astype(int)

acc5_pca.index = acc5.index
acc5_pca.anomaly_kmeans.value_counts()

0    5984
1      30
Name: anomaly_kmeans, dtype: int64

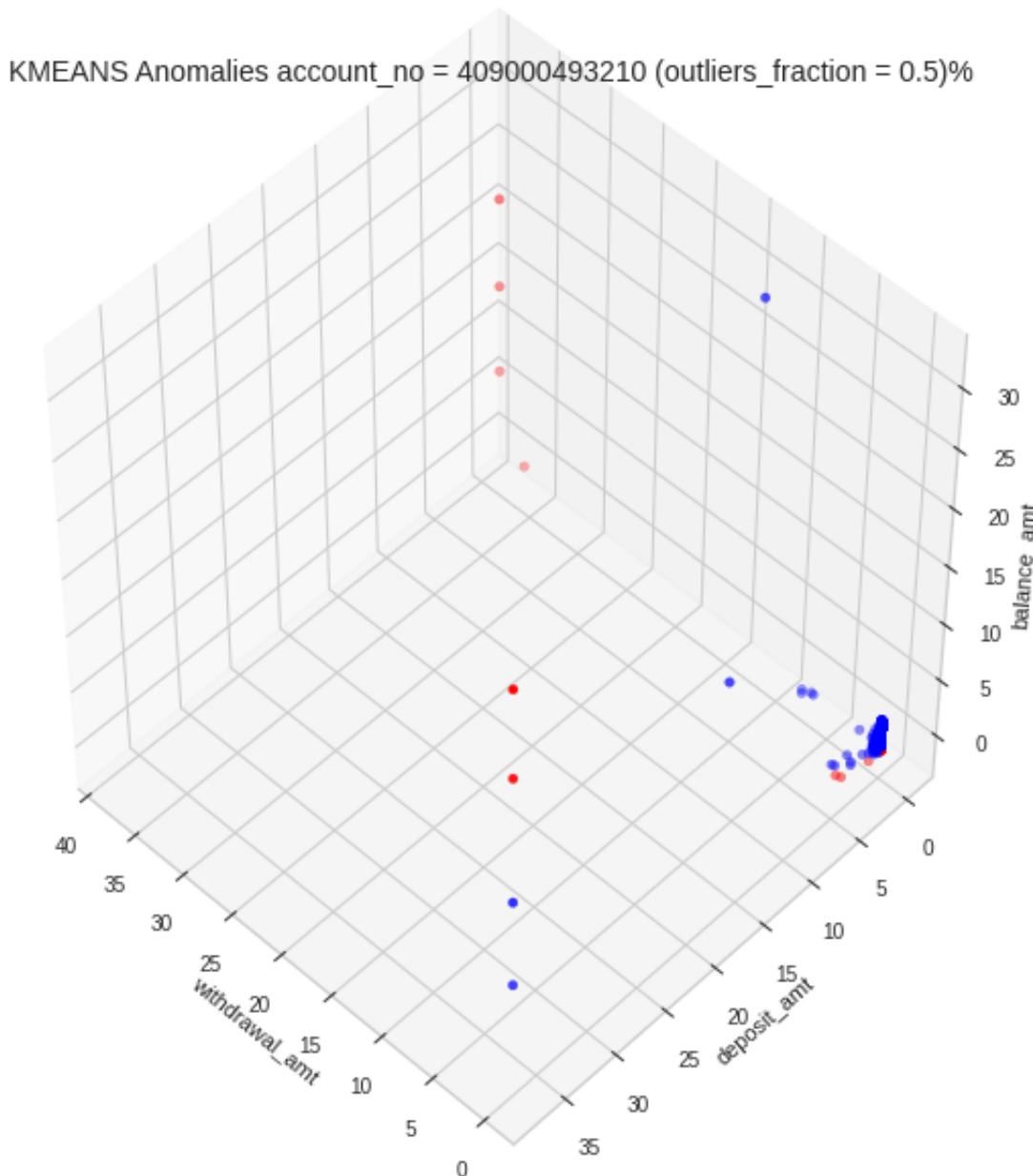
acc5_kmeans_scaled['anomaly_kmeans'] = acc5_pca.anomaly_kmeans
bank_main['km_5'] = 0
bank_main.loc[acc5_kmeans_scaled.index, 'km_5'] = acc5_kmeans_scaled['anomaly_kmeans']
bank_main['km_5'].value_counts()

0    116171
1      30
Name: km_5, dtype: int64
```

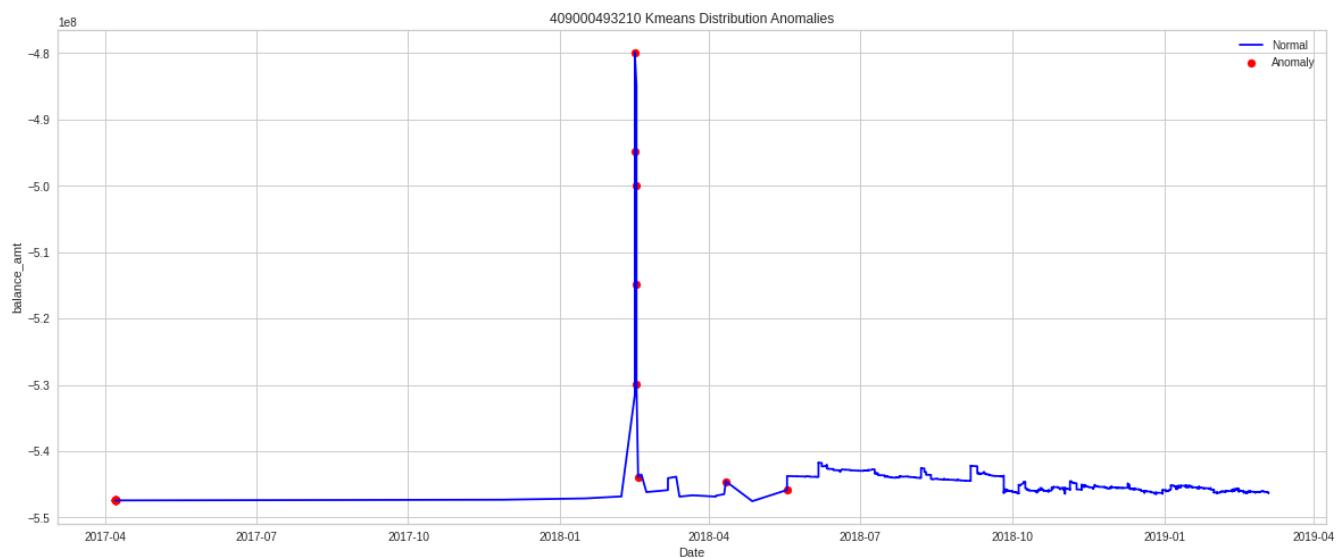
```
labels = acc5_kmeans_scaled.anomaly_kmeans
colors = {0:'blue', 1:'red'}
```

```
fig = plt.figure(1, figsize=(8,8))
ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)
ax.scatter(acc5_kmeans_scaled.iloc[:,3], acc5_kmeans_scaled.iloc[:,4], acc5_kmeans_scaled.iloc[:,5], c=colors[labels])
ax.set_title(f'KMEANS Anomalies account_no = {accounts[4]} (outliers_fraction = 0.5)%')
ax.set_xlabel('withdrawal_amt')
ax.set_ylabel('deposit_amt')
ax.set_zlabel('balance_amt')

Text(0.5, 0, 'balance_amt')
```



```
acc5_kmeans['anomaly_kmeans'] = acc5_kmeans_scaled['anomaly_kmeans']
df = acc5_kmeans.copy()
a= df.loc[df['anomaly_kmeans']==1, ['date', 'balance_amt']]
plt.figure(figsize=(20,8))
plt.plot(df['date'], df['balance_amt'], color='blue', label='Normal')
plt.scatter(a['date'], a['balance_amt'], color='red', label='Anomaly')
plt.title(f'{accounts[4]} Kmeans Distribution Anomalies')
plt.xlabel('Date')
plt.ylabel('balance_amt')
plt.legend()
plt.show()
```



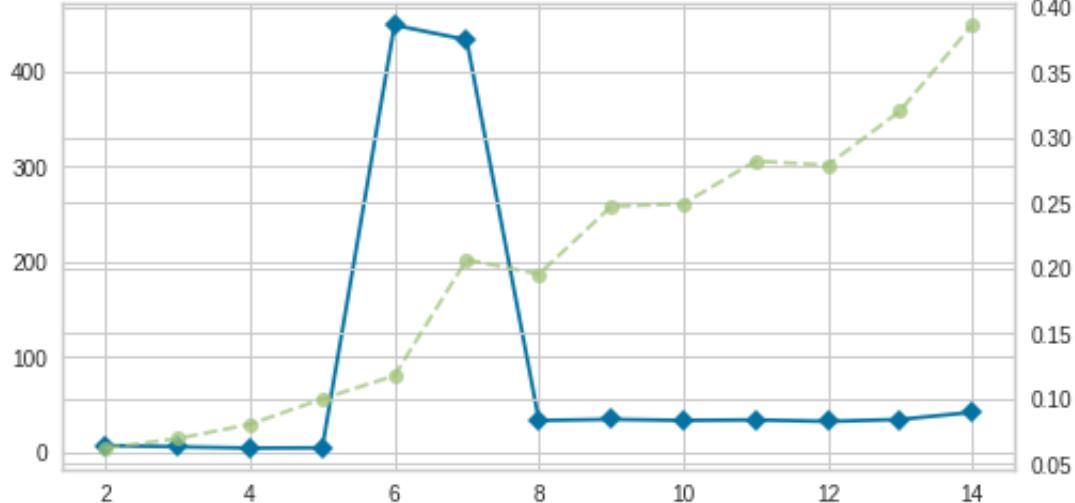
▼ (V) Customer account_no_409000438611

```
acc6_kmeans = acc6.copy()
scaler = StandardScaler()
acc6_kmeans_scaled = pd.DataFrame(scaler.fit_transform(acc6_kmeans[acc6.columns])
acc6_kmeans_scaled.set_index(acc6_kmeans.index, drop=True, inplace=True)
acc6_kmeans_scaled.head(3)
```

	day_diff	weekend_date	weekend_value_date	withdrawal_amt	deposit_am
2990	0.0	-0.280204		-0.280204	-0.217132
2991	0.0	-0.280204		-0.280204	27.304861
2992	0.0	-0.280204		-0.280204	8.251173

```
# Instantiate the clustering model and visualizer
visualizer = KElbowVisualizer(KMeans(), k=(2,15))
plt.figure(figsize=(8,4)).patch.set_facecolor('xkcd:white')
visualizer.fit(acc6_kmeans_scaled)          # Fit the data to the visualizer
# visualizer.show()           # Finalize and render the figure
```

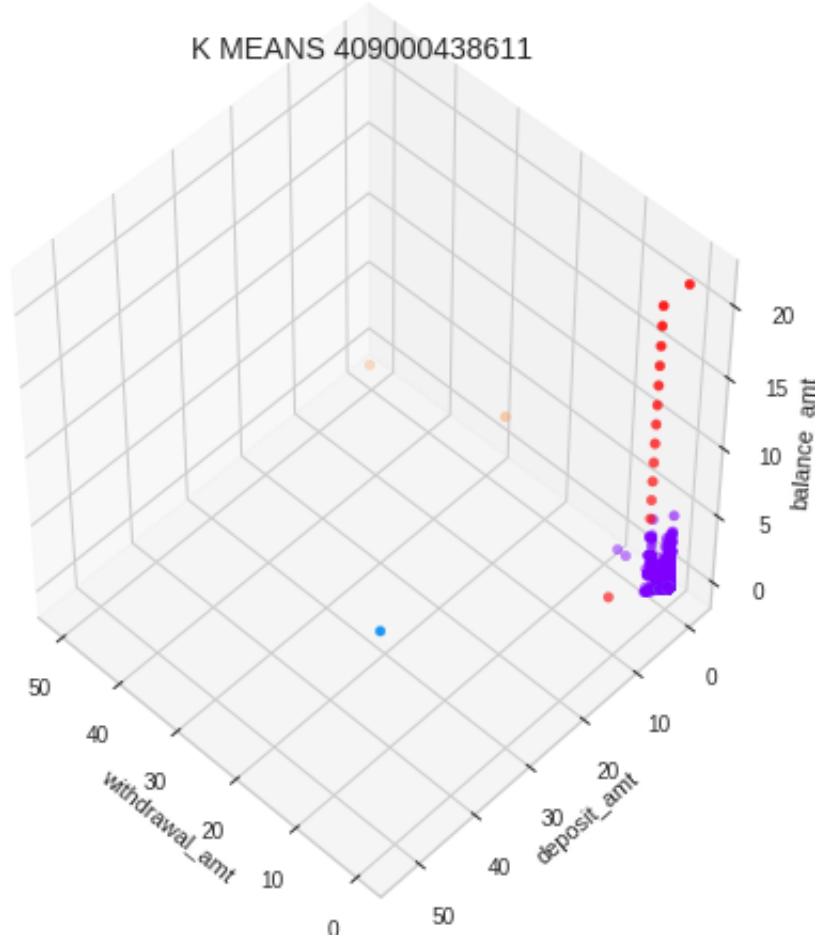
KElbowVisualizer(ax=<matplotlib.axes._subplots.AxesSubplot object at 0x7f26
k=None, metric=None, model=None, timings=True)



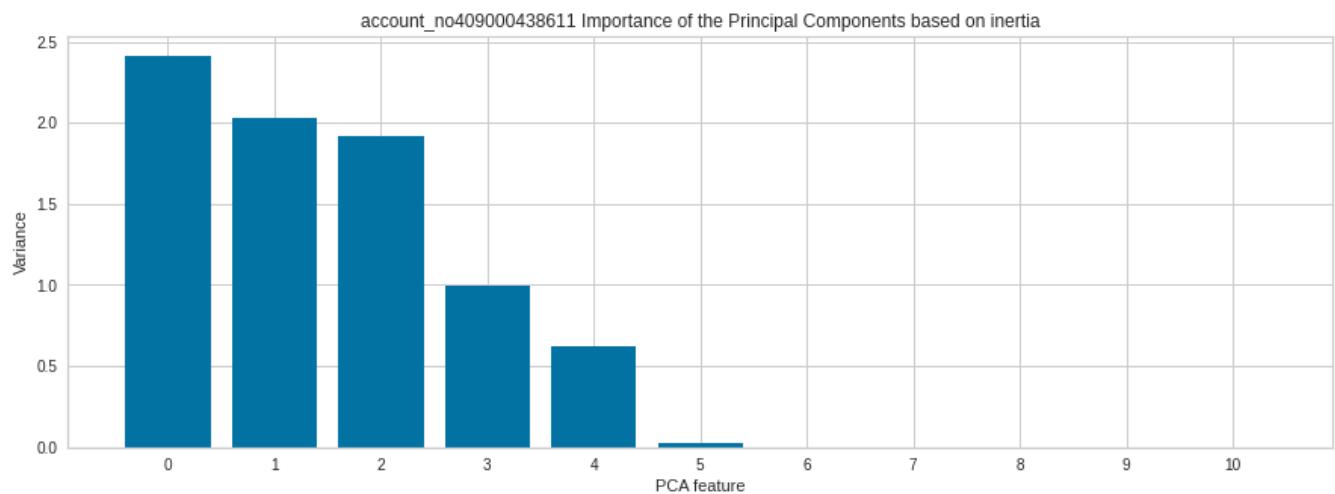
```
kmeans = KMeans(n_clusters=6)
kmeans.fit(acc6_kmeans_scaled)
labels = kmeans.labels_
```

```
fig = plt.figure(1, figsize=(6,6))
ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)
ax.scatter(acc6_kmeans_scaled.iloc[:,3], acc6_kmeans_scaled.iloc[:,4], acc6_kmeans_scaled.iloc[:,5])
ax.set_title(f'K MEANS {accounts[5]}', fontsize=14)
ax.set_xlabel('withdrawal_amt')
ax.set_ylabel('deposit_amt')
ax.set_zlabel('balance_amt')

Text(0.5, 0, 'balance_amt')
```



```
x = acc6_kmeans_scaled
pca = PCA()
pipeline = make_pipeline(pca)
pipeline.fit(x)
# Plot the principal components against their inertia
features = range(pca.n_components_)
_ = plt.figure(figsize=(15, 5))
_ = plt.bar(features, pca.explained_variance_)
_ = plt.xlabel('PCA feature')
_ = plt.ylabel('Variance')
_ = plt.xticks(features)
_ = plt.title(f"account_no{accounts[5]} Importance of the Principal Components")
plt.show()
```



```
acc6_pca = pd.DataFrame(PCA(n_components=5).fit_transform(acc6_kmeans_scaled),
```

```
kmeans = KMeans(n_clusters=6)
kmeans.fit(acc6_pca)
labels = kmeans.predict(acc6_pca)

unique_elements, counts_elements = np.unique(labels, return_counts=True)
clusters = np.asarray((unique_elements, counts_elements))

outliers_fraction = 0.005
distance = getDistanceByPoint(acc6_pca, kmeans)

number_of_outliers = int(outliers_fraction*len(distance))
threshold = distance.nlargest(number_of_outliers).min()
acc6_pca['anomaly_kmeans'] = (distance >= threshold).astype(int)

acc6_pca.index = acc6.index
acc6_pca.anomaly_kmeans.value_counts()

0    4566
1      22
Name: anomaly_kmeans, dtype: int64

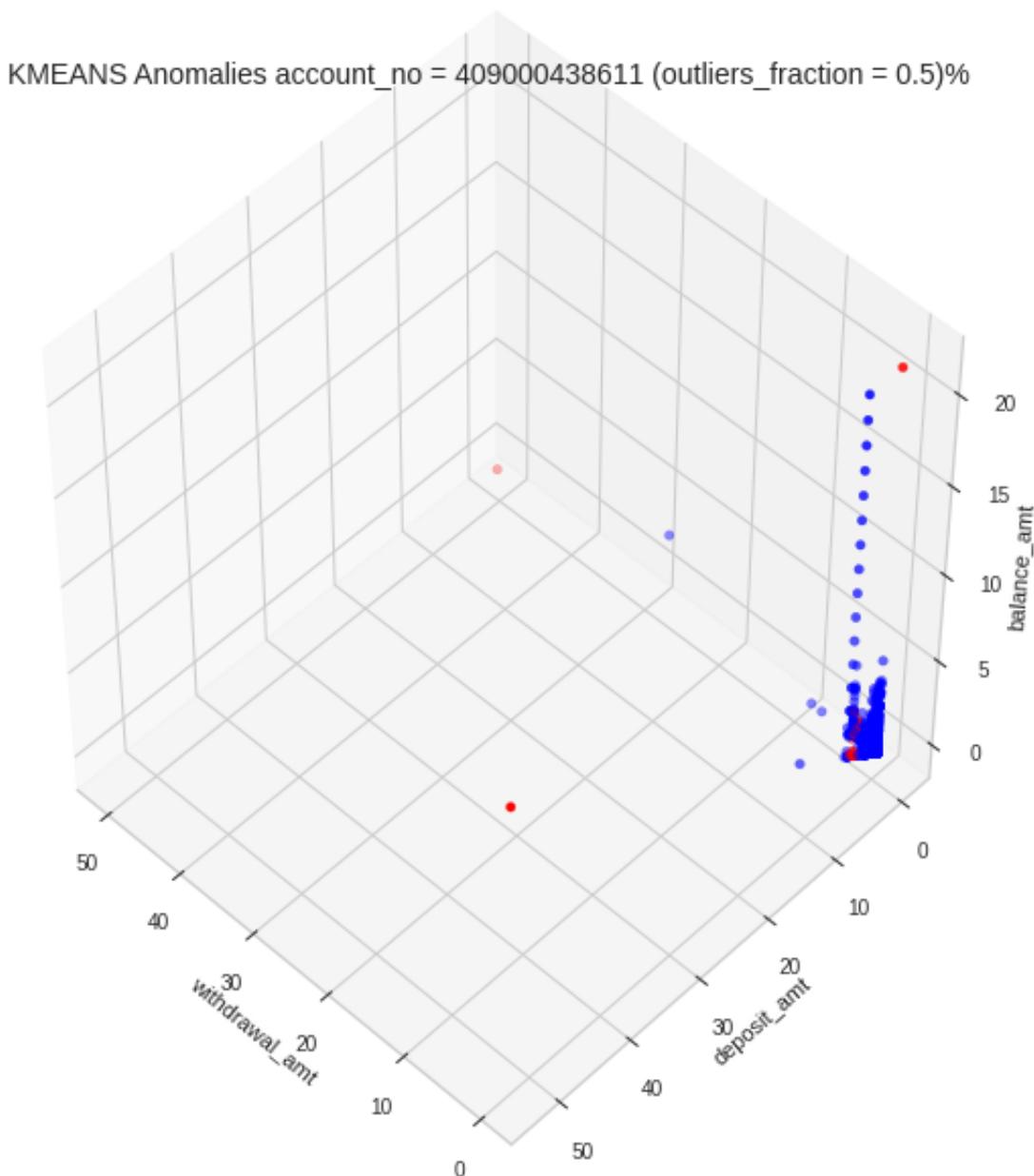
acc6_kmeans_scaled['anomaly_kmeans'] = acc6_pca.anomaly_kmeans
bank_main['km_6'] = 0
bank_main.loc[acc6_kmeans_scaled.index, 'km_6'] = acc6_kmeans_scaled['anomaly_kmeans']
bank_main['km_6'].value_counts()

0    116179
1      22
Name: km_6, dtype: int64
```

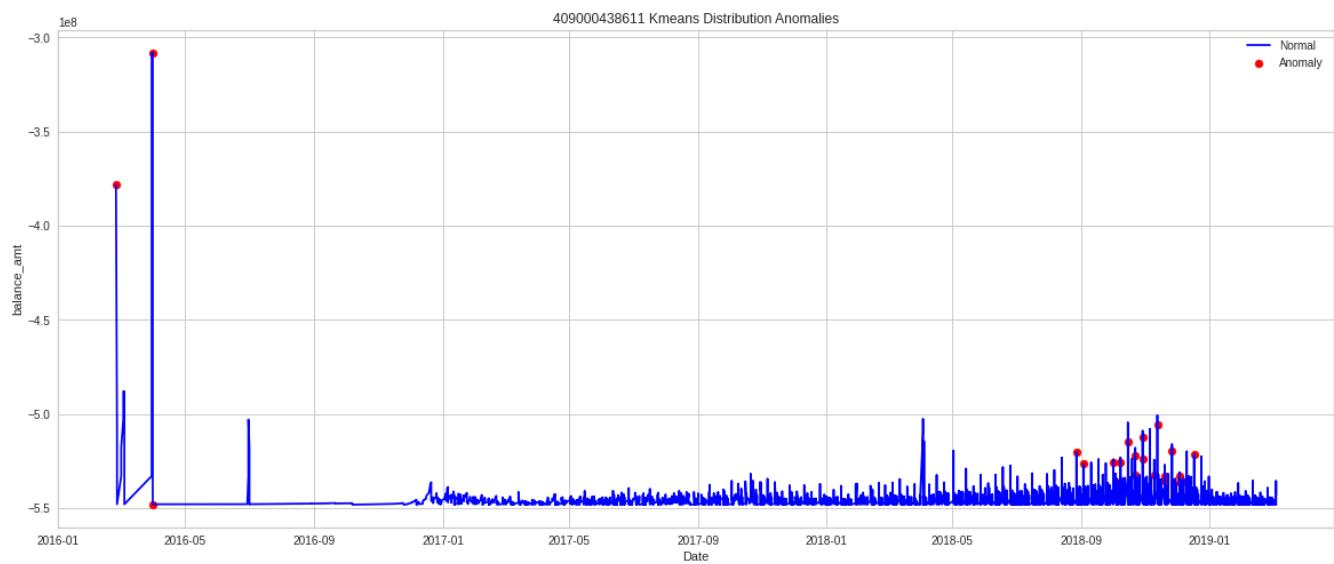
```
labels = acc6_kmeans_scaled.anomaly_kmeans
colors = {0:'blue', 1:'red'}
```

```
fig = plt.figure(1, figsize=(8,8))
ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)
ax.scatter(acc6_kmeans_scaled.iloc[:,3], acc6_kmeans_scaled.iloc[:,4], acc6_kmeans_scaled.iloc[:,5], c=colors[labels])
ax.set_title(f'KMEANS Anomalies account_no = {accounts[5]} (outliers_fraction = 0.5)%')
ax.set_xlabel('withdrawal_amt')
ax.set_ylabel('deposit_amt')
ax.set_zlabel('balance_amt')

Text(0.5, 0, 'balance_amt')
```



```
acc6_kmeans['anomaly_kmeans'] = acc6_kmeans_scaled['anomaly_kmeans']
df = acc6_kmeans.copy()
a= df.loc[df['anomaly_kmeans']==1, ['date', 'balance_amt']]
plt.figure(figsize=(20,8))
plt.plot(df['date'], df['balance_amt'], color='blue', label='Normal')
plt.scatter(a['date'], a['balance_amt'], color='red', label='Anomaly')
plt.title(f'{accounts[5]} Kmeans Distribution Anomalies')
plt.xlabel('Date')
plt.ylabel('balance_amt')
plt.legend()
plt.show()
```



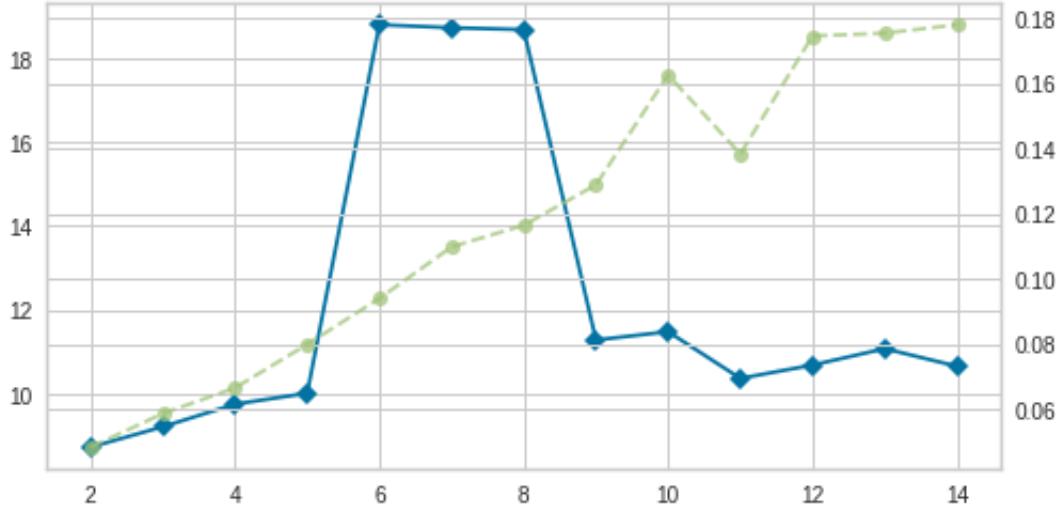
▼ (VII) Customer account_no_409000611074

```
acc7_kmeans = acc7.copy()
scaler = StandardScaler()
acc7_kmeans_scaled = pd.DataFrame(scaler.fit_transform(acc7_kmeans[acc7.columns])
acc7_kmeans_scaled.set_index(acc7_kmeans.index, drop=True, inplace=True)
acc7_kmeans_scaled.head(3)
```

	day_diff	weekend_date	weekend_value_date	withdrawal_amt	deposit_amt
0	0.0	-0.269481		-0.269481	-0.687913
1	0.0	-0.269481		-0.269481	-0.687913
2	0.0	-0.269481		-0.269481	-0.687913

```
# Instantiate the clustering model and visualizer
visualizer = KElbowVisualizer(KMeans(), k=(2,15))
plt.figure(figsize=(8,4)).patch.set_facecolor('xkcd:white')
visualizer.fit(acc7_kmeans_scaled)          # Fit the data to the visualizer
# visualizer.show()           # Finalize and render the figure
```

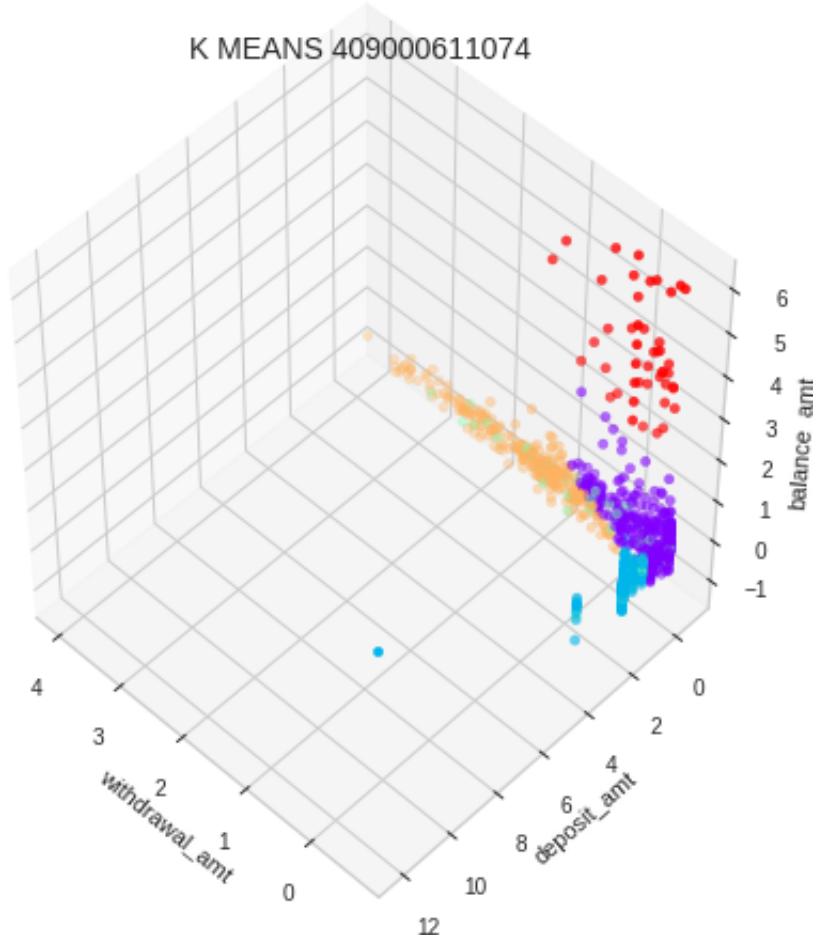
KElbowVisualizer(ax=<matplotlib.axes._subplots.AxesSubplot object at 0x7f25
k=None, metric=None, model=None, timings=True)



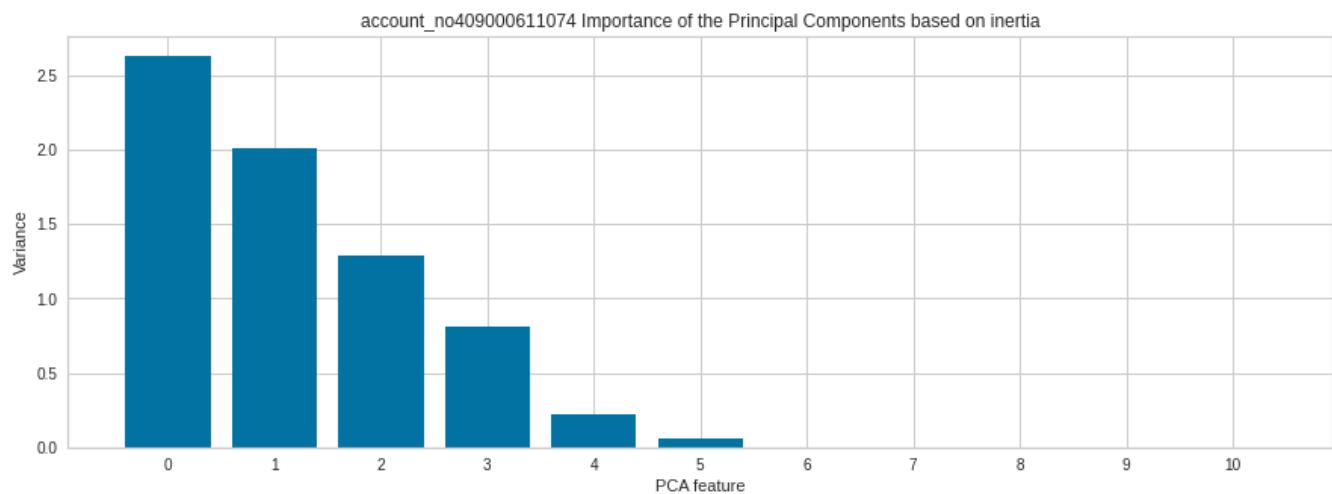
```
kmeans = KMeans(n_clusters=5)
kmeans.fit(acc7_kmeans_scaled)
labels = kmeans.labels_
```

```
fig = plt.figure(1, figsize=(6,6))
ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)
ax.scatter(acc7_kmeans_scaled.iloc[:,3], acc7_kmeans_scaled.iloc[:,4], acc7_kmeans_scaled.iloc[:,6])
ax.set_title(f'K MEANS {accounts[6]}', fontsize=14)
ax.set_xlabel('withdrawal_amt')
ax.set_ylabel('deposit_amt')
ax.set_zlabel('balance_amt')

Text(0.5, 0, 'balance_amt')
```



```
x = acc7_kmeans_scaled
pca = PCA()
pipeline = make_pipeline(pca)
pipeline.fit(x)
# Plot the principal components against their inertia
features = range(pca.n_components_)
_ = plt.figure(figsize=(15, 5))
_ = plt.bar(features, pca.explained_variance_)
_ = plt.xlabel('PCA feature')
_ = plt.ylabel('Variance')
_ = plt.xticks(features)
_ = plt.title(f"account_no{accounts[6]} Importance of the Principal Components")
plt.show()
```



```
acc7_pca = pd.DataFrame(PCA(n_components=5).fit_transform(acc7_kmeans_scaled),
```

```
kmeans = KMeans(n_clusters=5)
kmeans.fit(acc7_pca)
labels = kmeans.predict(acc7_pca)

unique_elements, counts_elements = np.unique(labels, return_counts=True)
clusters = np.asarray((unique_elements, counts_elements))

outliers_fraction = 0.01
distance = getDistanceByPoint(acc7_pca, kmeans)

number_of_outliers = int(outliers_fraction*len(distance))
threshold = distance.nlargest(number_of_outliers).min()
acc7_pca['anomaly_kmeans'] = (distance >= threshold).astype(int)

acc7_pca.index = acc7.index
acc7_pca.anomaly_kmeans.value_counts()

0    1083
1      10
Name: anomaly_kmeans, dtype: int64

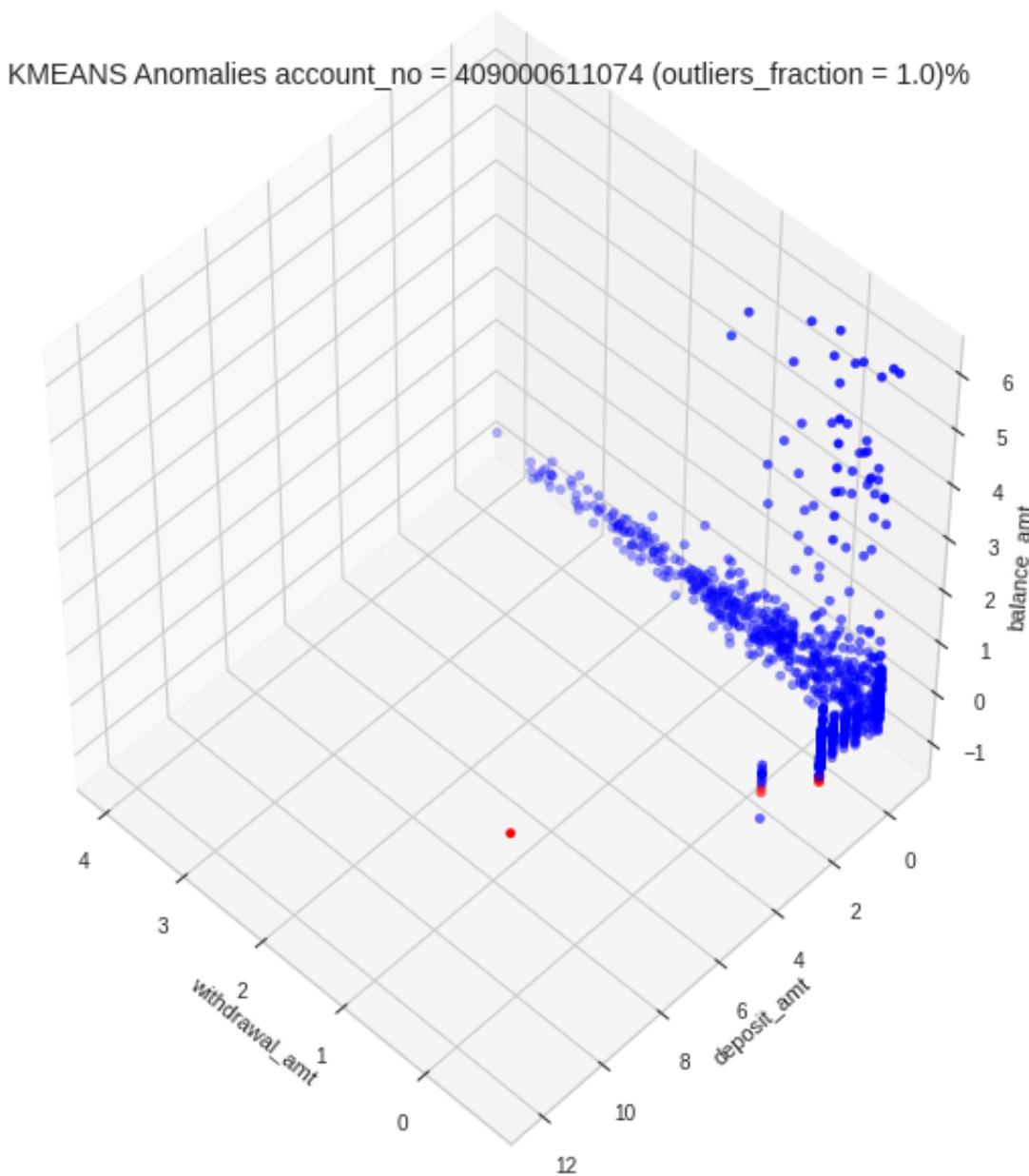
acc7_kmeans_scaled['anomaly_kmeans'] = acc7_pca.anomaly_kmeans
bank_main['km_7'] = 0
bank_main.loc[acc7_kmeans_scaled.index, 'km_7'] = acc7_kmeans_scaled['anomaly_kmeans']
bank_main['km_7'].value_counts()

0    116191
1      10
Name: km_7, dtype: int64
```

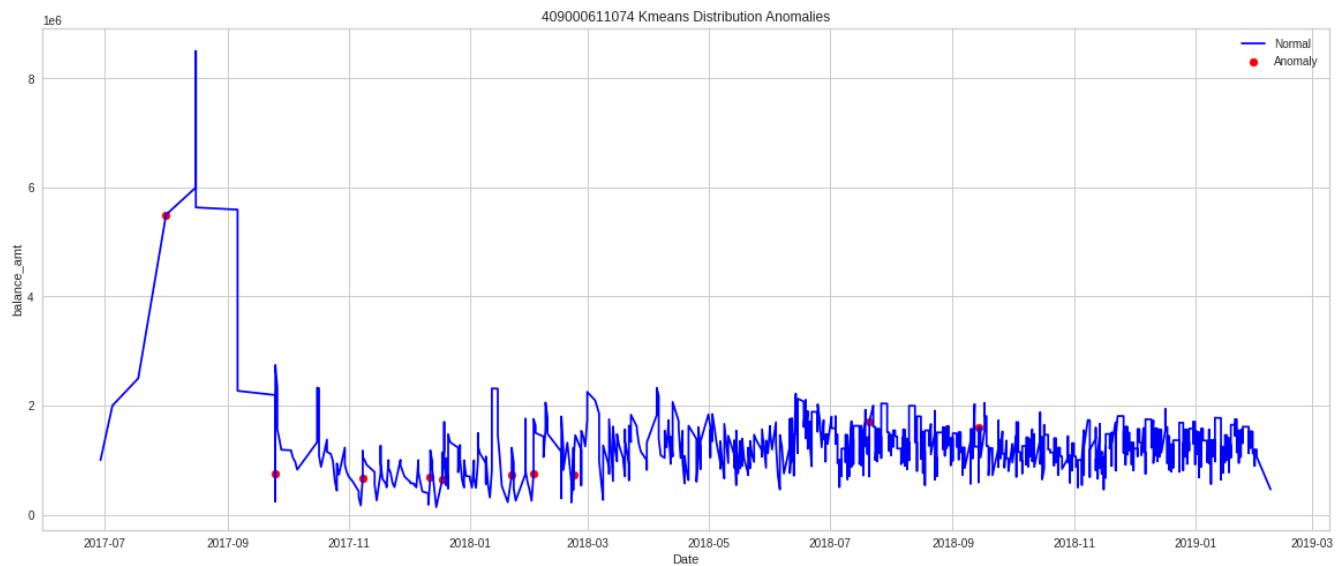
```
labels = acc7_kmeans_scaled.anomaly_kmeans
colors = {0:'blue', 1:'red'}
```

```
fig = plt.figure(1, figsize=(8,8))
ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)
ax.scatter(acc7_kmeans_scaled.iloc[:,3], acc7_kmeans_scaled.iloc[:,4], acc7_kmeans_scaled.iloc[:,5], c=colors[labels])
ax.set_title(f'KMEANS Anomalies account_no = {accounts[6]} (outliers_fraction = 1.0)%')
ax.set_xlabel('withdrawal_amt')
ax.set_ylabel('deposit_amt')
ax.set_zlabel('balance_amt')

Text(0.5, 0, 'balance_amt')
```



```
acc7_kmeans['anomaly_kmeans'] = acc7_kmeans_scaled['anomaly_kmeans']
df = acc7_kmeans.copy()
a= df.loc[df['anomaly_kmeans']==1, ['date', 'balance_amt']]
plt.figure(figsize=(20,8))
plt.plot(df['date'], df['balance_amt'], color='blue', label='Normal')
plt.scatter(a['date'], a['balance_amt'], color='red', label='Anomaly')
plt.title(f'{accounts[6]} Kmeans Distribution Anomalies')
plt.xlabel('Date')
plt.ylabel('balance_amt')
plt.legend()
plt.show()
```



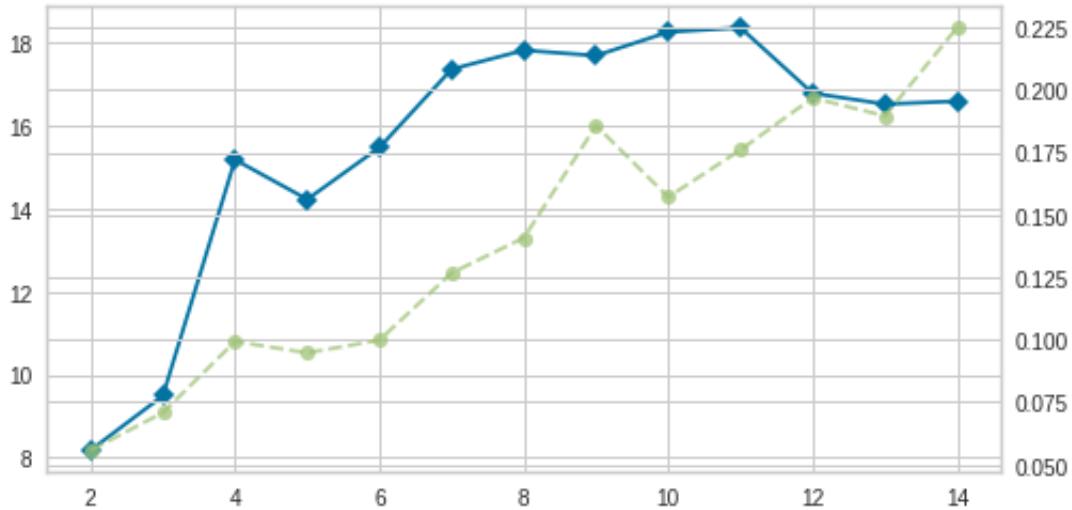
▼ (VIII) Customer account_no_409000493201

```
acc8_kmeans = acc8.copy()
scaler = StandardScaler()
acc8_kmeans_scaled = pd.DataFrame(scaler.fit_transform(acc8_kmeans[acc8.columns])
acc8_kmeans_scaled.set_index(acc8_kmeans.index, drop=True, inplace=True)
acc8_kmeans_scaled.head(3)
```

	day_diff	weekend_date	weekend_value_date	withdrawal_amt	deposit_am
1093	0.0	-0.261873		-0.261873	-0.450731
1094	0.0	-0.261873		-0.261873	-0.446770
1095	0.0	-0.261873		-0.261873	-0.449547

```
# Instantiate the clustering model and visualizer
visualizer = KElbowVisualizer(KMeans(), k=(2,15))
plt.figure(figsize=(8,4)).patch.set_facecolor('xkcd:white')
visualizer.fit(acc8_kmeans_scaled)          # Fit the data to the visualizer
# visualizer.show()                      # Finalize and render the figure
```

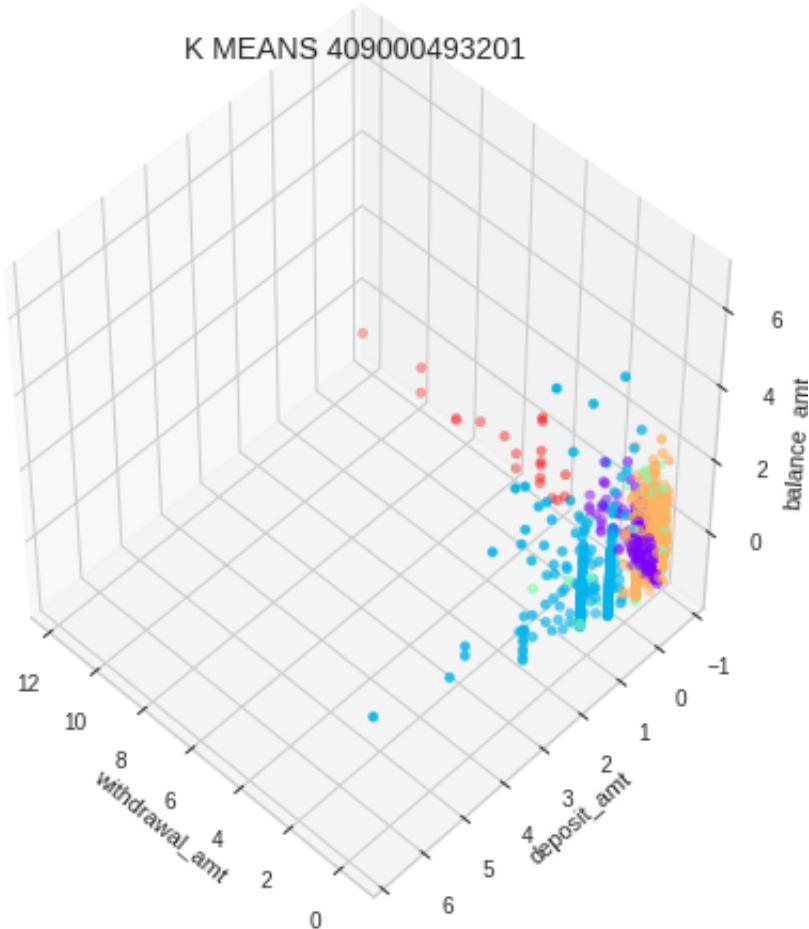
KElbowVisualizer(ax=<matplotlib.axes._subplots.AxesSubplot object at 0x7f25
k=None, metric=None, model=None, timings=True)



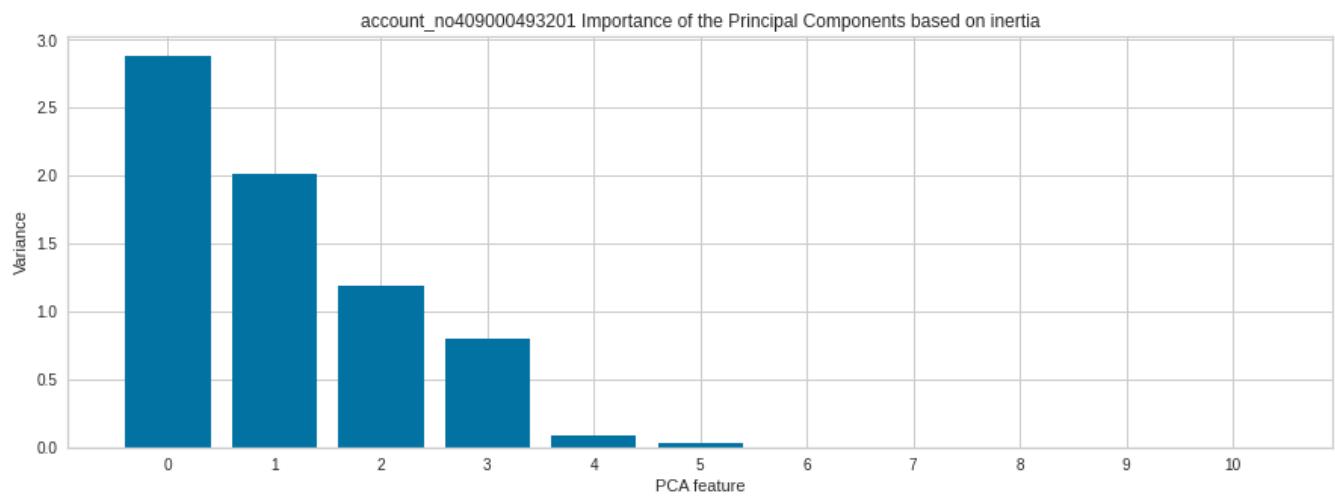
```
kmeans = KMeans(n_clusters=5)
kmeans.fit(acc8_kmeans_scaled)
labels = kmeans.labels_
```

```
fig = plt.figure(1, figsize=(6,6))
ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)
ax.scatter(acc8_kmeans_scaled.iloc[:,3], acc8_kmeans_scaled.iloc[:,4], acc8_kmeans_scaled.iloc[:,5])
ax.set_title(f'K MEANS {accounts[7]}', fontsize=14)
ax.set_xlabel('withdrawal_amt')
ax.set_ylabel('deposit_amt')
ax.set_zlabel('balance_amt')

Text(0.5, 0, 'balance_amt')
```



```
x = acc8_kmeans_scaled
pca = PCA()
pipeline = make_pipeline(pca)
pipeline.fit(x)
# Plot the principal components against their inertia
features = range(pca.n_components_)
_ = plt.figure(figsize=(15, 5))
_ = plt.bar(features, pca.explained_variance_)
_ = plt.xlabel('PCA feature')
_ = plt.ylabel('Variance')
_ = plt.xticks(features)
_ = plt.title(f"account_no{accounts[7]} Importance of the Principal Components")
plt.show()
```



```
acc8_pca = pd.DataFrame(PCA(n_components=5).fit_transform(acc8_kmeans_scaled),
```

```
kmeans = KMeans(n_clusters=5)
kmeans.fit(acc8_pca)
labels = kmeans.predict(acc8_pca)

unique_elements, counts_elements = np.unique(labels, return_counts=True)
clusters = np.asarray((unique_elements, counts_elements))

outliers_fraction = 0.007
distance = getDistanceByPoint(acc8_pca, kmeans)

number_of_outliers = int(outliers_fraction*len(distance))
threshold = distance.nlargest(number_of_outliers).min()
acc8_pca['anomaly_kmeans'] = (distance >= threshold).astype(int)

acc8_pca.index = acc8.index
acc8_pca.anomaly_kmeans.value_counts()

0    1037
1      7
Name: anomaly_kmeans, dtype: int64

acc8_kmeans_scaled['anomaly_kmeans'] = acc8_pca.anomaly_kmeans
bank_main['km_8'] = 0
bank_main.loc[acc8_kmeans_scaled.index, 'km_8'] = acc8_kmeans_scaled['anomaly_kmeans']
bank_main['km_8'].value_counts()

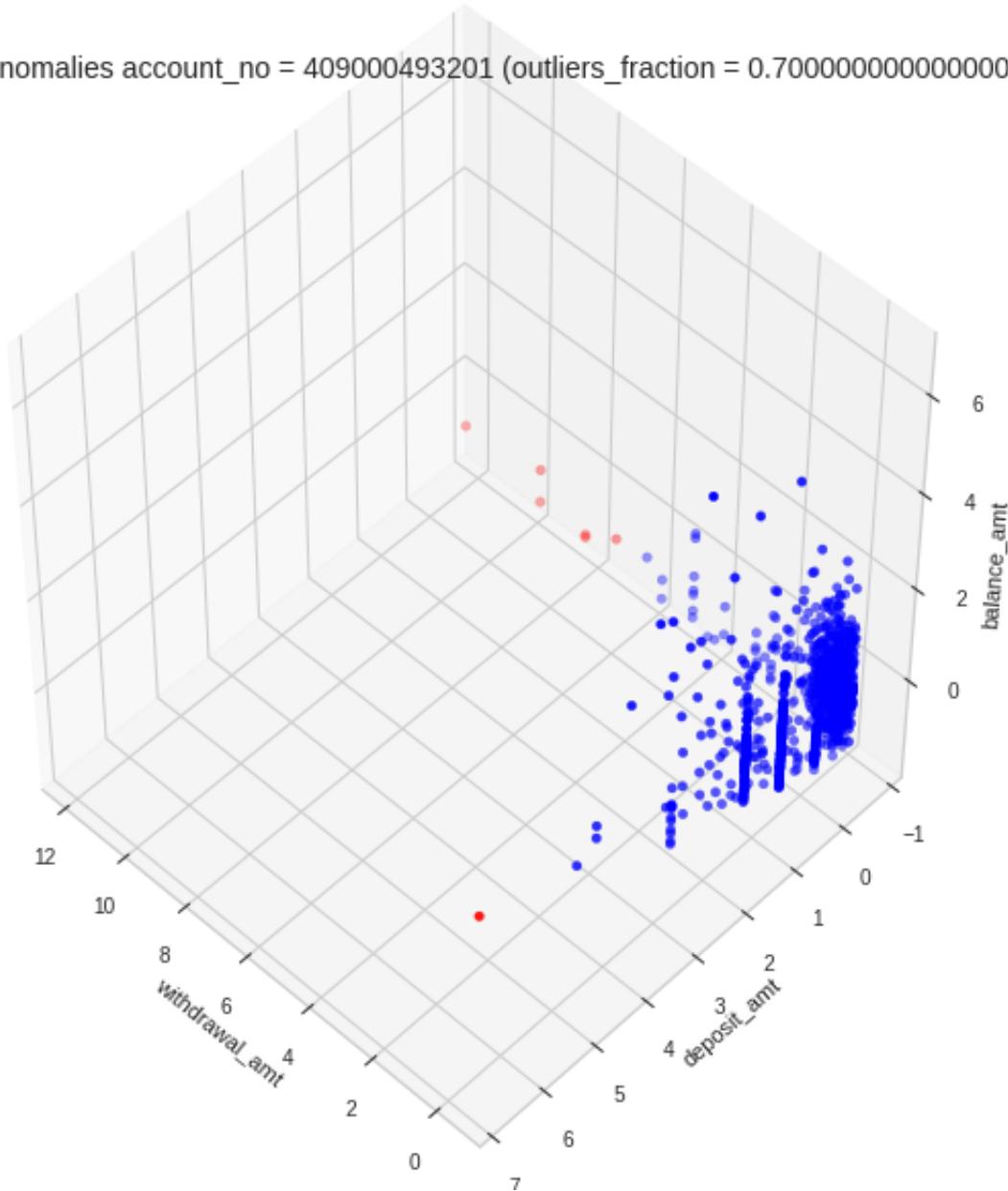
0    116194
1      7
Name: km_8, dtype: int64
```

```
labels = acc8_kmeans_scaled.anomaly_kmeans
colors = {0:'blue', 1:'red'}
```

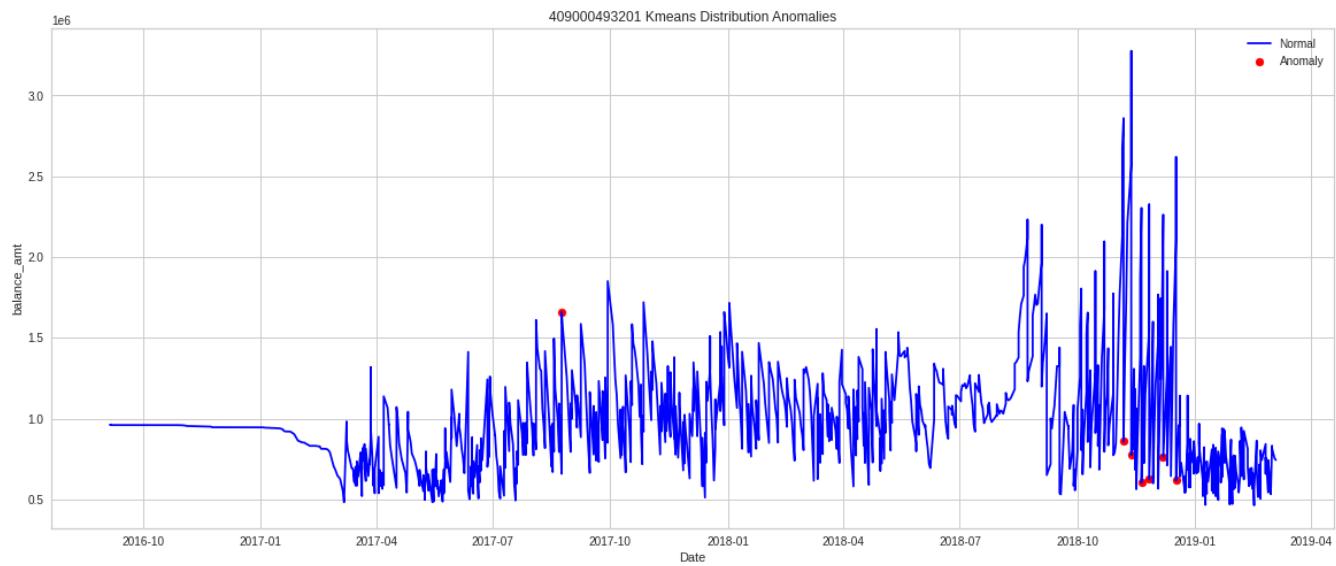
```
fig = plt.figure(1, figsize=(8,8))
ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)
ax.scatter(acc8_kmeans_scaled.iloc[:,3], acc8_kmeans_scaled.iloc[:,4], acc8_kmeans_scaled.iloc[:,5], c=colors[labels])
ax.set_title(f'KMEANS Anomalies account_no = {accounts[7]} (outliers_fraction = {outliers_fraction})')
ax.set_xlabel('withdrawal_amt')
ax.set_ylabel('deposit_amt')
ax.set_zlabel('balance_amt')

Text(0.5, 0, 'balance_amt')
```

KMEANS Anomalies account_no = 409000493201 (outliers_fraction = 0.7000000000000001)%



```
acc8_kmeans['anomaly_kmeans'] = acc8_kmeans_scaled['anomaly_kmeans']
df = acc8_kmeans.copy()
a= df.loc[df['anomaly_kmeans']==1, ['date', 'balance_amt']]
plt.figure(figsize=(20,8))
plt.plot(df['date'], df['balance_amt'], color='blue', label='Normal')
plt.scatter(a['date'], a['balance_amt'], color='red', label='Anomaly')
plt.title(f'{accounts[7]} Kmeans Distribution Anomalies')
plt.xlabel('Date')
plt.ylabel('balance_amt')
plt.legend()
plt.show()
```



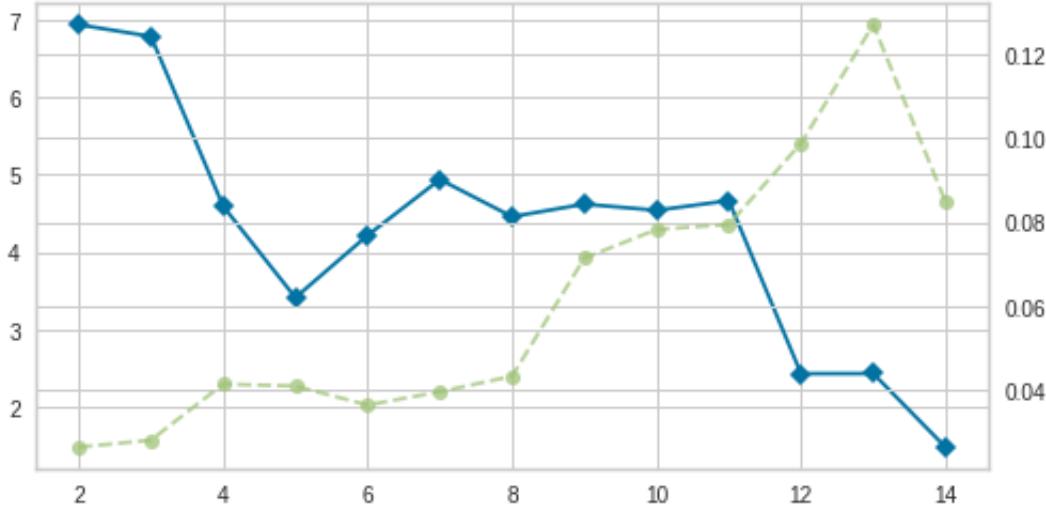
▼ (IX) Customer account_no_409000425051

```
acc9_kmeans = acc9.copy()
scaler = StandardScaler()
acc9_kmeans_scaled = pd.DataFrame(scaler.fit_transform(acc9_kmeans[acc9.columns])
acc9_kmeans_scaled.set_index(acc9_kmeans.index, drop=True, inplace=True)
acc9_kmeans_scaled.head(3)
```

	day_diff	weekend_date	weekend_value_date	withdrawal_amt	deposit_am
2137	0.0	-0.429695		-0.429695	-0.038344
2138	0.0	-0.429695		-0.429695	-0.038344
2139	0.0	-0.429695		-0.429695	-0.038344

```
# Instantiate the clustering model and visualizer
visualizer = KElbowVisualizer(KMeans(), k=(2,15))
plt.figure(figsize=(8,4)).patch.set_facecolor('xkcd:white')
visualizer.fit(acc9_kmeans_scaled)          # Fit the data to the visualizer
# visualizer.show()                      # Finalize and render the figure
```

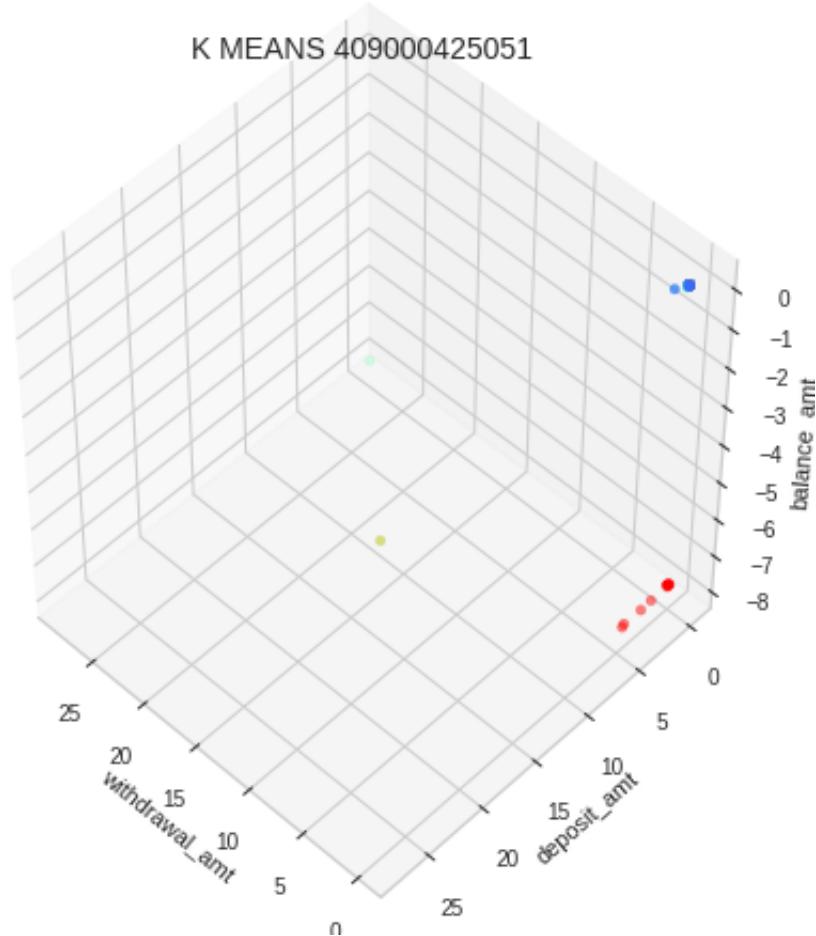
KElbowVisualizer(ax=<matplotlib.axes._subplots.AxesSubplot object at 0x7f25
k=None, metric=None, model=None, timings=True)



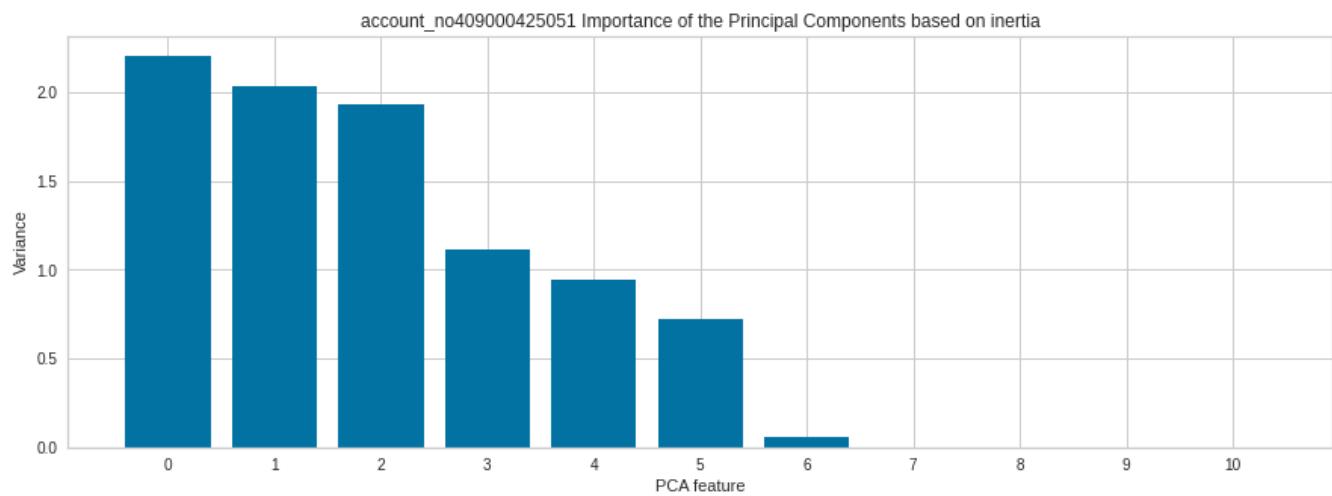
```
kmeans = KMeans(n_clusters=7)
kmeans.fit(acc9_kmeans_scaled)
labels = kmeans.labels_
```

```
fig = plt.figure(1, figsize=(6,6))
ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)
ax.scatter(acc9_kmeans_scaled.iloc[:,3], acc9_kmeans_scaled.iloc[:,4], acc9_kmeans_scaled.iloc[:,8])
ax.set_title(f'K MEANS {accounts[8]}', fontsize=14)
ax.set_xlabel('withdrawal_amt')
ax.set_ylabel('deposit_amt')
ax.set_zlabel('balance_amt')

Text(0.5, 0, 'balance_amt')
```



```
x = acc9_kmeans_scaled
pca = PCA()
pipeline = make_pipeline(pca)
pipeline.fit(x)
# Plot the principal components against their inertia
features = range(pca.n_components_)
_ = plt.figure(figsize=(15, 5))
_ = plt.bar(features, pca.explained_variance_)
_ = plt.xlabel('PCA feature')
_ = plt.ylabel('Variance')
_ = plt.xticks(features)
_ = plt.title(f"account_no{accounts[8]} Importance of the Principal Components")
plt.show()
```



```
acc9_pca = pd.DataFrame(PCA(n_components=6).fit_transform(acc9_kmeans_scaled),
```

```
kmeans = KMeans(n_clusters=7)
kmeans.fit(acc9_pca)
labels = kmeans.predict(acc9_pca)

unique_elements, counts_elements = np.unique(labels, return_counts=True)
clusters = np.asarray((unique_elements, counts_elements))

outliers_fraction = 0.008
distance = getDistanceByPoint(acc9_pca, kmeans)

number_of_outliers = int(outliers_fraction*len(distance))
threshold = distance.nlargest(number_of_outliers).min()
acc9_pca['anomaly_kmeans'] = (distance >= threshold).astype(int)

acc9_pca.index = acc9.index
acc9_pca.anomaly_kmeans.value_counts()

0    796
1      6
Name: anomaly_kmeans, dtype: int64

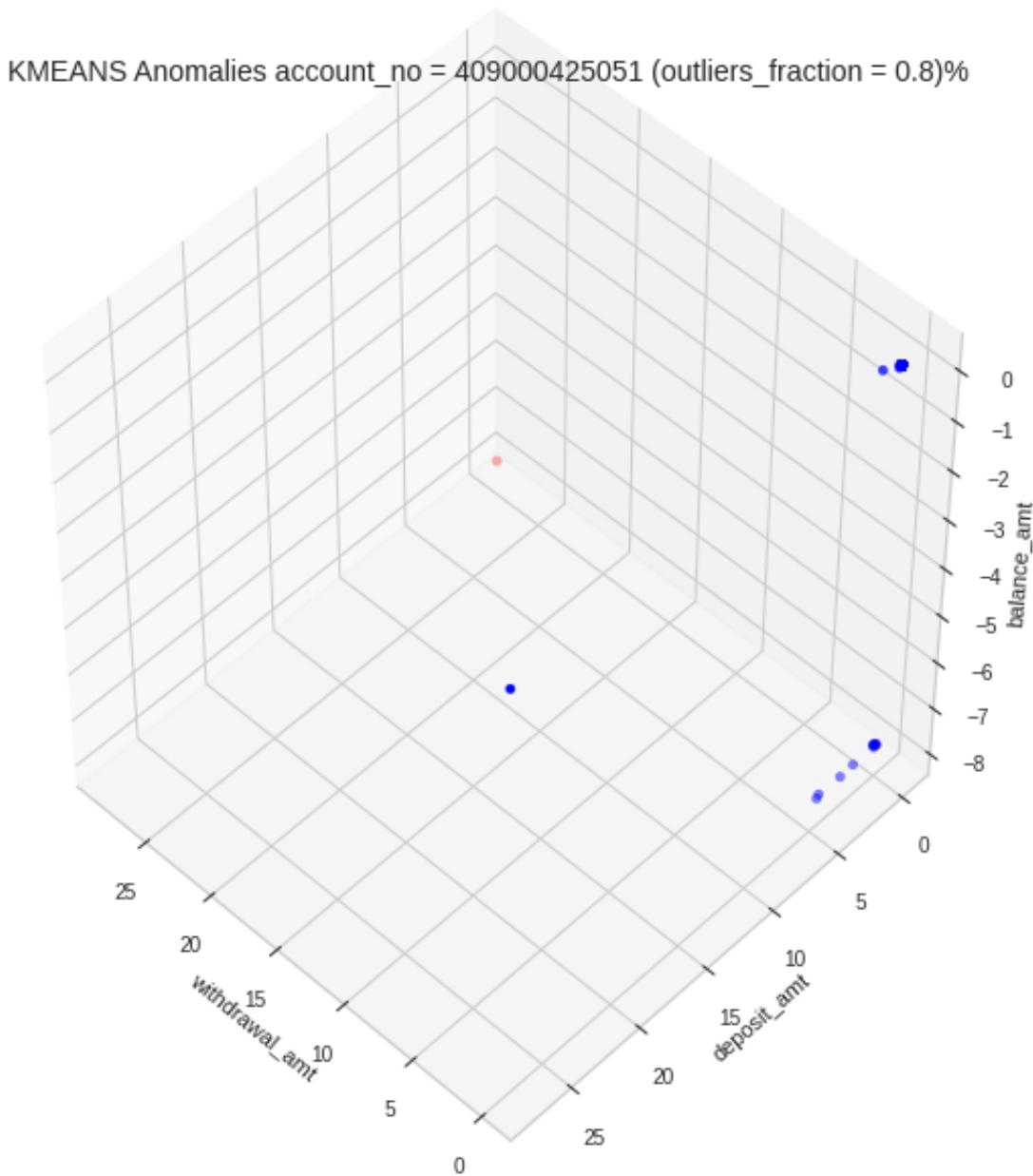
acc9_kmeans_scaled['anomaly_kmeans'] = acc9_pca.anomaly_kmeans
bank_main['km_9'] = 0
bank_main.loc[acc9_kmeans_scaled.index, 'km_9'] = acc9_kmeans_scaled['anomaly_kmeans']
bank_main['km_9'].value_counts()

0    116195
1        6
Name: km_9, dtype: int64
```

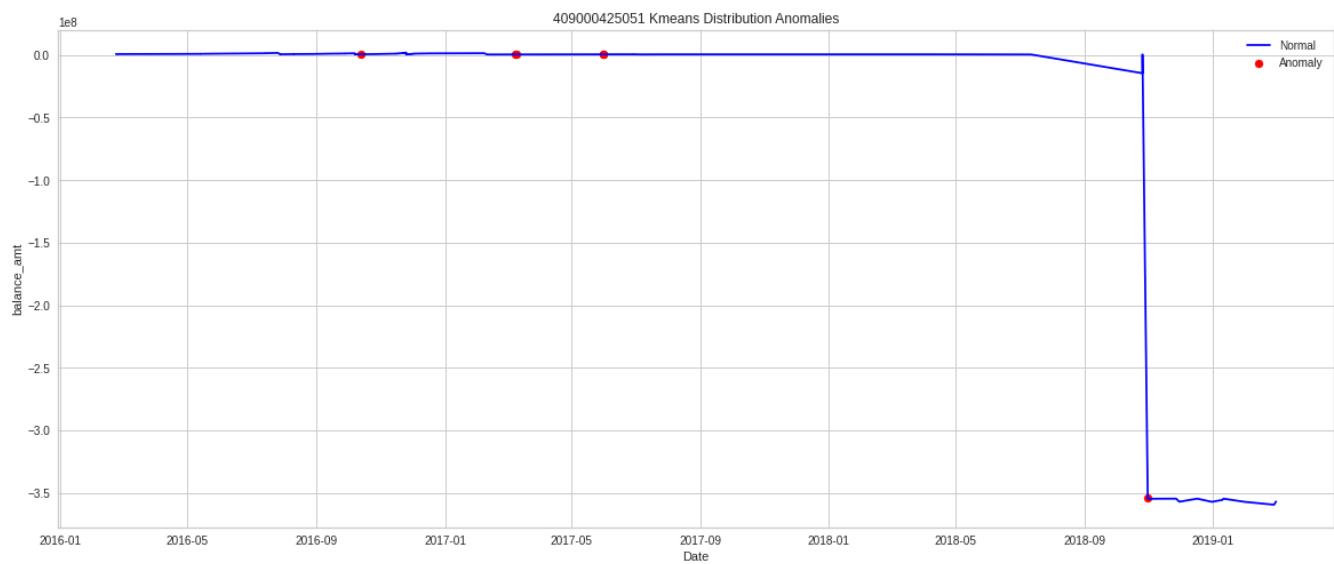
```
labels = acc9_kmeans_scaled.anomaly_kmeans
colors = {0:'blue', 1:'red'}
```

```
fig = plt.figure(1, figsize=(8,8))
ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)
ax.scatter(acc9_kmeans_scaled.iloc[:,3], acc9_kmeans_scaled.iloc[:,4], acc9_kmeans_scaled.iloc[:,5], c=colors[labels])
ax.set_title(f'KMEANS Anomalies account_no = {accounts[8]} (outliers_fraction = 0.8)%')
ax.set_xlabel('withdrawal_amt')
ax.set_ylabel('deposit_amt')
ax.set_zlabel('balance_amt')

Text(0.5, 0, 'balance_amt')
```



```
acc9_kmeans['anomaly_kmeans'] = acc9_kmeans_scaled['anomaly_kmeans']
df = acc9_kmeans.copy()
a= df.loc[df['anomaly_kmeans']==1, ['date', 'balance_amt']]
plt.figure(figsize=(20,8))
plt.plot(df['date'], df['balance_amt'], color='blue', label='Normal')
plt.scatter(a['date'], a['balance_amt'], color='red', label='Anomaly')
plt.title(f'{accounts[8]} Kmeans Distribution Anomalies')
plt.xlabel('Date')
plt.ylabel('balance_amt')
plt.legend()
plt.show()
```



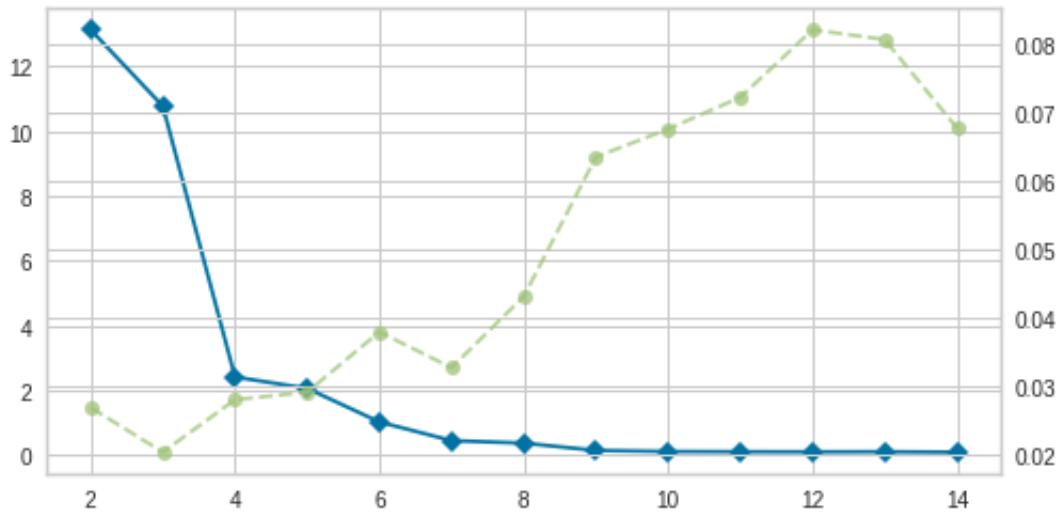
▼ (X) Customer account_no_409000405747

```
acc10_kmeans = acc10.copy()
scaler = StandardScaler()
acc10_kmeans_scaled = pd.DataFrame(scaler.fit_transform(acc10_kmeans[acc10.columns]))
acc10_kmeans_scaled.set_index(acc10_kmeans.index, drop=True, inplace=True)
acc10_kmeans_scaled.head(3)
```

	day_diff	weekend_date	weekend_value_date	withdrawal_amt	deposit_am
2939	0.0	2.318405		2.318405	5.917619
2940	0.0	2.318405		2.318405	0.795991
2941	0.0	2.318405		2.318405	-0.296882

```
# Instantiate the clustering model and visualizer
visualizer = KElbowVisualizer(KMeans(), k=(2,15))
plt.figure(figsize=(8,4)).patch.set_facecolor('xkcd:white')
visualizer.fit(acc10_kmeans_scaled)      # Fit the data to the visualizer
# visualizer.show()          # Finalize and render the figure
```

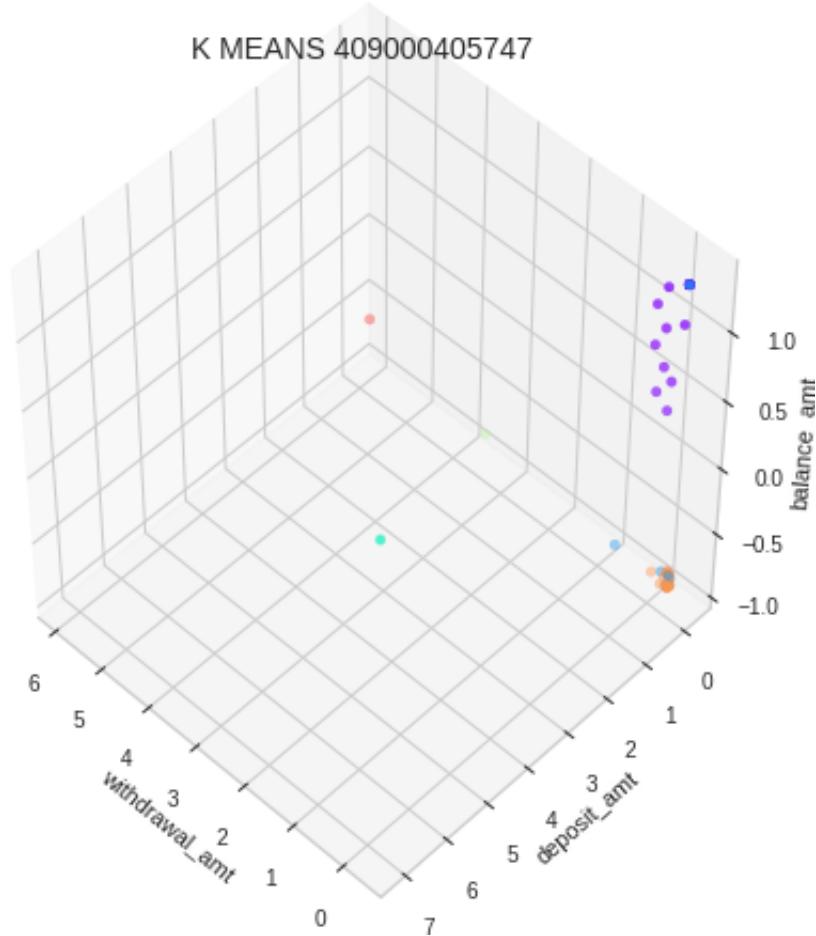
KElbowVisualizer(ax=<matplotlib.axes._subplots.AxesSubplot object at 0x7f25
k=None, metric=None, model=None, timings=True)



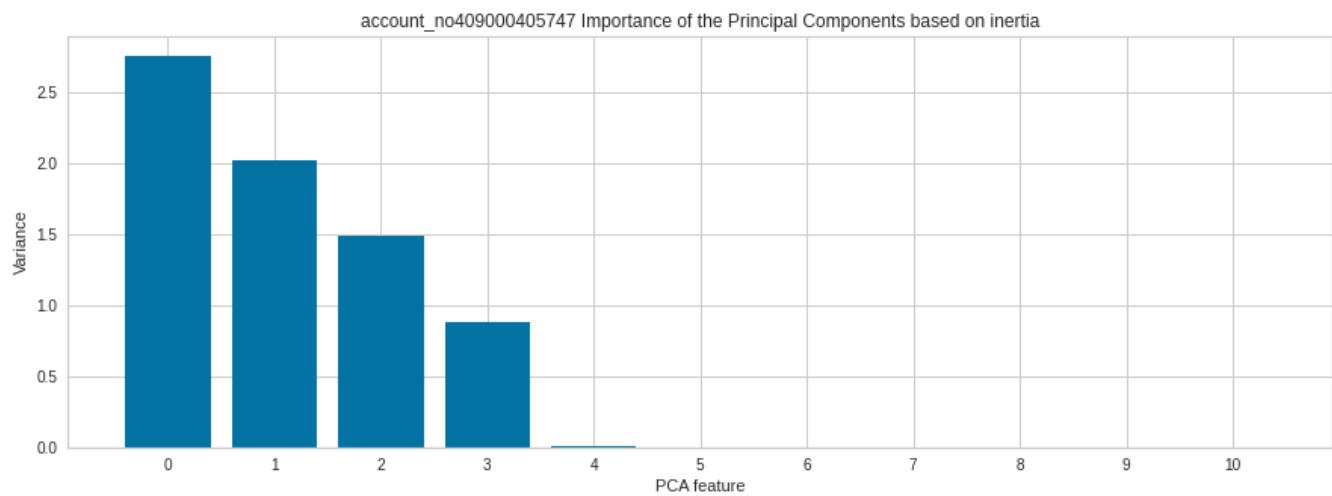
```
kmeans = KMeans(n_clusters=6)
kmeans.fit(acc10_kmeans_scaled)
labels = kmeans.labels_
```

```
fig = plt.figure(1, figsize=(6,6))
ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)
ax.scatter(acc10_kmeans_scaled.iloc[:,3], acc10_kmeans_scaled.iloc[:,4], acc10_kmeans_scaled.iloc[:,5])
ax.set_title(f'K MEANS {accounts[9]}', fontsize=14)
ax.set_xlabel('withdrawal_amt')
ax.set_ylabel('deposit_amt')
ax.set_zlabel('balance_amt')

Text(0.5, 0, 'balance_amt')
```



```
x = acc10_kmeans_scaled
pca = PCA()
pipeline = make_pipeline(pca)
pipeline.fit(x)
# Plot the principal components against their inertia
features = range(pca.n_components_)
_ = plt.figure(figsize=(15, 5))
_ = plt.bar(features, pca.explained_variance_)
_ = plt.xlabel('PCA feature')
_ = plt.ylabel('Variance')
_ = plt.xticks(features)
_ = plt.title(f"account_no{accounts[9]} Importance of the Principal Components")
plt.show()
```



```
acc10_pca = pd.DataFrame(PCA(n_components=3).fit_transform(acc10_kmeans_scaled))
```

```
kmeans = KMeans(n_clusters=4)
kmeans.fit(acc10_pca)
labels = kmeans.predict(acc10_pca)

unique_elements, counts_elements = np.unique(labels, return_counts=True)
clusters = np.asarray((unique_elements, counts_elements))

outliers_fraction = 0.1
distance = getDistanceByPoint(acc10_pca, kmeans)

number_of_outliers = int(outliers_fraction*len(distance))
threshold = distance.nlargest(number_of_outliers).min()
acc10_pca['anomaly_kmeans'] = (distance >= threshold).astype(int)

acc10_pca.index = acc10.index
acc10_pca.anomaly_kmeans.value_counts()

0    46
1     5
Name: anomaly_kmeans, dtype: int64

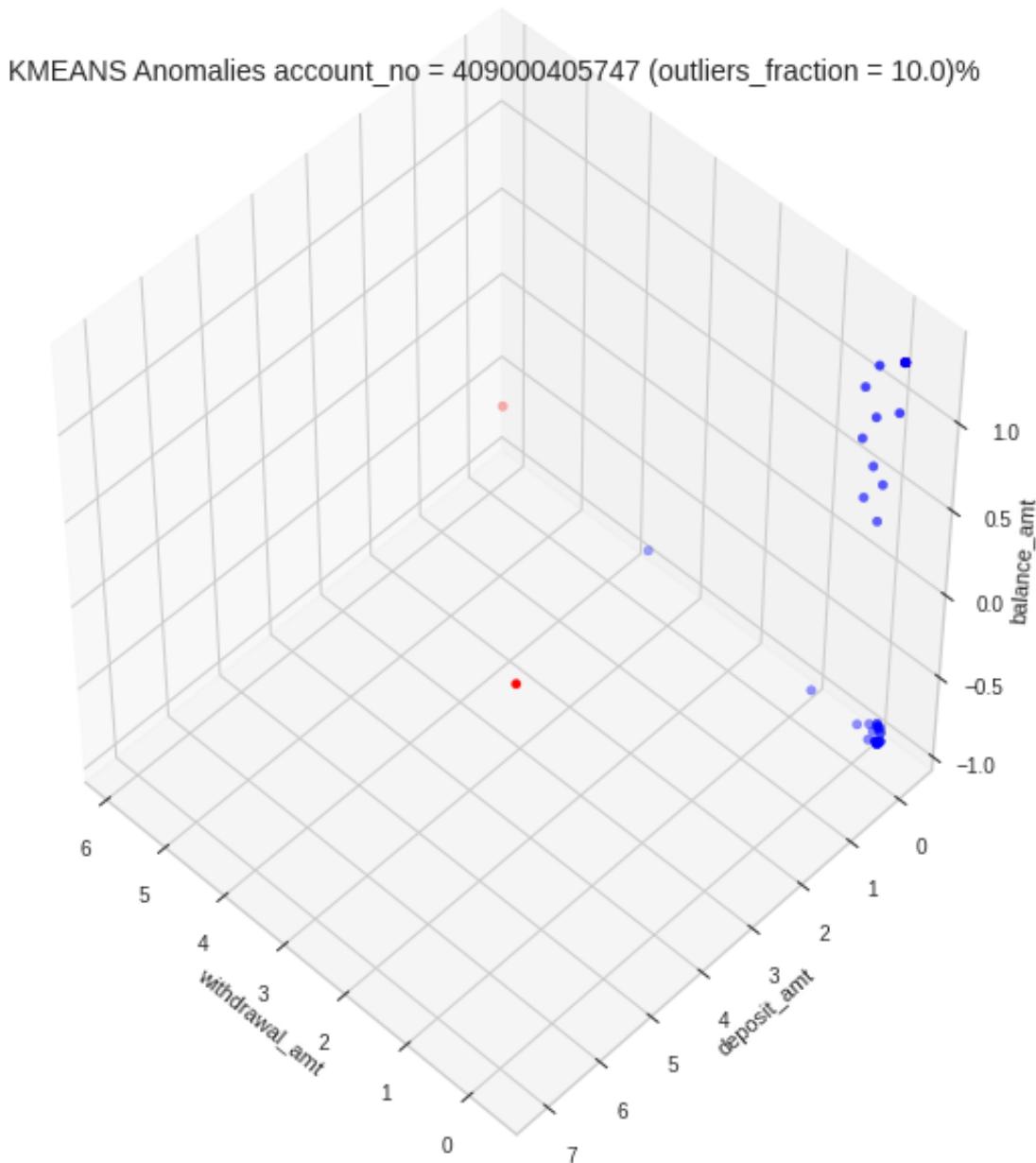
acc10_kmeans_scaled['anomaly_kmeans'] = acc10_pca.anomaly_kmeans
bank_main['km_10'] = 0
bank_main.loc[acc10_kmeans_scaled.index, 'km_10'] = acc10_kmeans_scaled['anomaly_kmeans']
bank_main['km_10'].value_counts()

0    116196
1        5
Name: km_10, dtype: int64
```

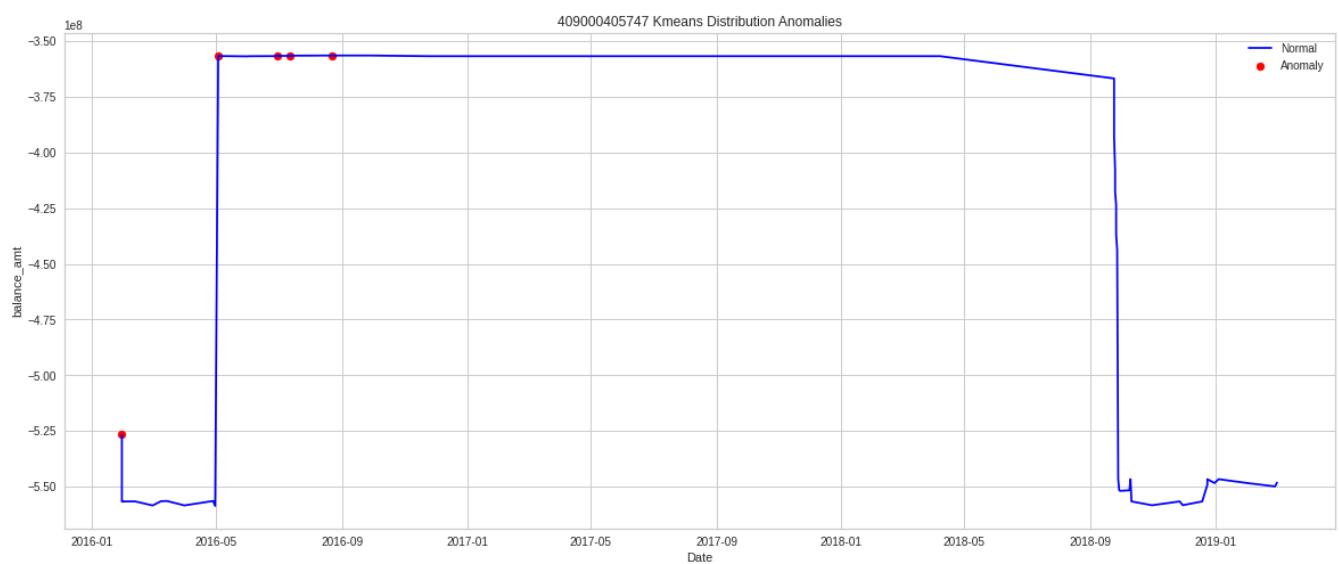
```
labels = acc10_kmeans_scaled.anomaly_kmeans
colors = {0:'blue', 1:'red'}
```

```
fig = plt.figure(1, figsize=(8,8))
ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)
ax.scatter(acc10_kmeans_scaled.iloc[:,3], acc10_kmeans_scaled.iloc[:,4], acc10_kmeans_scaled.iloc[:,5], c=colors[labels])
ax.set_title(f'KMEANS Anomalies account_no = {accounts[9]} (outliers_fraction = 10.0)%')
ax.set_xlabel('withdrawal_amt')
ax.set_ylabel('deposit_amt')
ax.set_zlabel('balance_amt')

Text(0.5, 0, 'balance_amt')
```



```
acc10_kmeans['anomaly_kmeans'] = acc10_kmeans_scaled['anomaly_kmeans']
df = acc10_kmeans.copy()
a= df.loc[df['anomaly_kmeans']==1, ['date', 'balance_amt']]
plt.figure(figsize=(20,8))
plt.plot(df['date'], df['balance_amt'], color='blue', label='Normal')
plt.scatter(a['date'], a['balance_amt'], color='red', label='Anomaly')
plt.title(f'{accounts[9]} Kmeans Distribution Anomalies')
plt.xlabel('Date')
plt.ylabel('balance_amt')
plt.legend()
plt.show()
```



```
bank_main['accounts_kmeans'] = bank_main.km_1 + bank_main.km_2 + bank_main.km_
```

```
bank_main['accounts_kmeans'].value_counts()
```

0	115794
1	407
Name: accounts_kmeans, dtype: int64	

```
bank_main['anomaly_sum_2'] = bank_main.accounts_isolation_forest + bank_main.a
```

▼ 5. Grading Anomalies

```
bank_graded = bank_main.copy()
```

```
drop_lst =['iso_1', 'iso_2', 'iso_3', 'iso_4', 'iso_5', 'iso_6', 'iso_7', 'iso_8', 'iso_9', 'iso_10', 'iso_11', 'iso_12', 'iso_13', 'iso_14', 'iso_15', 'iso_16', 'iso_17', 'iso_18', 'iso_19', 'iso_20', 'svm_1', 'svm_2', 'svm_3', 'svm_4', 'svm_5', 'svm_6', 'svm_7', 'svm_8', 'svm_9', 'svm_10', 'gauss_1', 'gauss_2', 'gauss_3', 'gauss_4', 'gauss_5', 'gauss_6', 'km_1', 'km_2', 'km_3', 'km_4', 'km_5', 'km_6', 'km_7', 'km_8', 'km_9', 'km_10', 'km_11', 'km_12', 'km_13', 'km_14', 'km_15', 'km_16', 'km_17', 'km_18', 'km_19', 'km_20']
```

```
bank_graded.drop(columns=drop_lst, axis=1, inplace=True)
```

```
bank_graded['weighted_anomaly'] = bank_graded.anomaly_sum_1 + 2 * bank_graded.anomaly_sum_2  
bank_graded['weighted_anomaly'].value_counts()
```

```
0      113451  
2       975  
1       822  
3       312  
4       310  
6       103  
8        87  
5        45  
7        37  
10       28  
9        21  
11       9  
12       1  
Name: weighted_anomaly, dtype: int64
```

```
bank_graded['ranks'] = 0
```

```
bank_graded.loc[bank_graded.weighted_anomaly>8, 'ranks'] = 1
```

```
bank_graded.loc[(bank_graded.weighted_anomaly>4) & (bank_graded.weighted_anomaly<=8), 'ranks'] = 2  
bank_graded.loc[(bank_graded.weighted_anomaly>2) & (bank_graded.weighted_anomaly<=4), 'ranks'] = 3  
bank_graded.ranks.value_counts()
```

```
0      115248  
3       622  
2       272  
1        59  
Name: ranks, dtype: int64
```

Since we have essentially determined **weighted_anomaly** using machine learning algorithms, we now want to check some suspicious points that we discovered during the EDA phase. And finally, we synthesize a rank value for each instance.

▼ day_diff vs. weighted_anomaly

```
bank_graded[bank_graded.day_diff<0][['day_diff', 'weighted_anomaly', 'ranks']]
```

	day_diff	weighted_anomaly	ranks
63340	-1	5	2

```
bank_graded.loc[bank_graded.day_diff<0, 'ranks'] = 1
```

```
bank_graded[bank_graded.day_diff>4][['day_diff', 'weighted_anomaly', 'ranks']]
```

	day_diff	weighted_anomaly	ranks
36647	5	6	2
36648	5	4	3
36727	5	6	2
36728	5	4	3
36892	5	6	2
36893	5	4	3
36894	5	6	2
36915	5	4	3
36916	5	6	2
63717	6	6	2
103332	29	6	2
103335	29	6	2
104111	8	6	2
112081	18	6	2

```
bank_graded.loc[(bank_graded.day_diff>4) & (bank_graded.day_diff<6) & (bank_gr  
bank_graded.loc[bank_graded.day_diff>10, 'ranks'] = 1
```

```
bank_graded[bank_graded.day_diff>4].ranks.value_counts()
```

```
2    7  
3    4  
1    3  
Name: ranks, dtype: int64
```

▼ details_null_penalty vs. weighted_anomaly

```
bank_graded[bank_graded.details_null_penalty>0][['details_null_penalty', 'weig
```

	details_null_penalty	weighted_anomaly	ranks
30345	1	1	0
30346	1	3	3
30584	1	6	2
30585	1	6	2
31192	1	1	0
...
97686	1	0	0
97687	1	0	0
97781	1	0	0
97782	1	0	0
97783	1	0	0

2499 rows × 3 columns

```
bank_graded[bank_graded.details_null_penalty>0].ranks.value_counts()
```

```
0    2350  
3    138  
2    11  
Name: ranks, dtype: int64
```

▼ weekend_date vs. weighted_anomaly

```
bank_graded[bank_graded.weekend_date>0][['weekend_date', 'weekend_value_date',
```

	weekend_date	weekend_value_date	weighted_anomaly	ranks
117	1	1	1	4
118	1	1	1	0
119	1	1	1	0
120	1	1	1	0
153	1	1	1	1
...
116167	1	1	1	0
116168	1	1	1	0
116169	1	1	1	0
116170	1	1	1	0
116171	1	1	1	0

10053 rows × 4 columns

```
bank_graded[bank_graded.weekend_date>0].ranks.value_counts()
```

```
0    9690
3    289
2     63
1     11
Name: ranks, dtype: int64
```

```
bank_graded[bank_graded.weekend_date + bank_graded.weekend_value_date==1] [ ['we
```

	weekend_date	weekend_value_date	weighted_anomaly	ranks	
32528	0		1	1	0
34732	0		1	1	0
35192	0		1	9	1
35798	0		1	3	3
37011	0		1	1	0
37221	1		0	4	3
37222	1		0	8	2
41340	0		1	5	2
54062	0		1	4	3
91948	0		1	5	2
104139	1		0	2	0
104159	1		0	0	0
110119	1		0	2	0
113157	1		0	2	0

```
bank_graded.loc[(bank_graded.weekend_date + bank_graded.weekend_value_date==1)
```

```
bank_graded[bank_graded.weekend_date + bank_graded.weekend_value_date==1].rank
```

3	7
2	3
0	3
1	1

Name: ranks, dtype: int64

▼ cheque_penalty vs. weighted_anomaly

```
bank_graded[bank_graded.cheque_penalty>0][['cheque_penalty', 'weighted_anomaly']]
```

	cheque_penalty	weighted_anomaly	ranks
2907	1		2 0
2908	1		2 0
2909	1		2 0
2910	1		2 0
2911	1		2 0
2912	1		2 0
2913	1		2 0
2914	1		2 0
2915	1		2 0
5654	1		4 3
5967	1		4 3
6227	1		4 3
6229	1		4 3
22174	1		4 3
22176	1		4 3
22178	1		4 3
22208	1		4 3
22210	1		4 3
22220	1		4 3
22245	1		4 3
22247	1		4 3

```
bank_graded[bank_graded.cheque_penalty>0].ranks.value_counts()
```

```
3    12
0     9
Name: ranks, dtype: int64
```

```
bank_graded.ranks.value_counts()
```

```
0    115244  
3     626  
2     268  
1     63  
Name: ranks, dtype: int64
```

▼ 6. Conclusion

```
bank_ranked = bank_graded.copy()
```

```
drop_lst = ['account_no_1196428', 'account_no_1196711', 'account_no_4090003624  
           'account_no_409000438611', 'account_no_409000438620', 'account_no_  
           'anomaly_kmeans', 'anomaly_isolation_forest', 'anomaly_one_class_s  
           'accounts_isolation_forest', 'accounts_SVM', 'accounts_gaussian_di
```

```
bank_ranked.drop(columns=drop_lst, axis=1, inplace=True)
```

```
bank_ranked['account_no'] = bank_premodel.account_no
```

```
bank_ranked.head()
```

	date	value_date	day_diff	weekend_date	weekend_value_date	withdrawal_
0	2017-06-29	2017-06-29	0	0	0	0
1	2017-07-05	2017-07-05	0	0	0	0
2	2017-07-18	2017-07-18	0	0	0	0
3	2017-08-01	2017-08-01	0	0	0	0
4	2017-08-16	2017-08-16	0	0	0	0

```
bank_ranked.to_excel('bank_transaction_ranked.xlsx')
```

```
bank_ranked.columns
```

```
Index(['date', 'value_date', 'day_diff', 'weekend_date', 'weekend_value_date',
       'withdrawal_amt', 'deposit_amt', 'balance_amt',
       'withdrawal_over_balance', 'deposit_over_balance',
       'details_null_penalty', 'details_numbers_penalty', 'cheque_penalty',
       'weighted_anomaly', 'ranks', 'account_no'],
      dtype='object')
```

```
bank_ranked[bank_ranked.deposit_over_balance < 0].ranks.value_counts()
```

```
0    60569
3     339
2     161
1      32
Name: ranks, dtype: int64
```

✓ 0s completed at 10:11 AM

✖