In [1]:

```python
### import python libraries
```

In [2]:

```python
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import RandomForestRegressor
```

In [3]:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets
import plotly.express as px
import matplotlib.pyplot as plt
import plotly.graph_objs as go
from tqdm import tqdm
from sklearn.metrics import mean_squared_error
import tensorflow as tf
from sklearn import model_selection as sk_model_selection
from xgboost.sklearn import XGBRegressor
from sklearn.metrics import mean_squared_error,roc_auc_score,precision_score
from sklearn import metrics
from sklearn.metrics import log_loss
from optuna.samplers import TPESampler
import functools
from functools import partial
import xgboost as xgb
import joblib
from matplotlib_venn import venn2, venn2_circles, venn2_unweighted
from matplotlib_venn import venn3, venn3_circles
import statsmodels.api as sm
import pylab
from xgboost import plot_tree
from xgboost.sklearn import XGBClassifier
from sklearn.metrics import mean_squared_error,roc_auc_score,precision_score
from sklearn import metrics
from sklearn.metrics import log_loss
from sklearn.metrics import confusion_matrix, recall_score, precision_score, precision_recall_curve, auc, f1_score, average_preci
from sklearn.preprocessing import LabelEncoder
import tensorflow as tf
from tensorflow.keras.utils import plot_model
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.layers import Dense, Dropout, Input
from tensorflow.keras.layers import Concatenate, LSTM, GRU
from tensorflow.keras.layers import Bidirectional, Multiply
import seaborn as sns
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.svm import SVC


SEED = 42
```

In [4]:

```
# Suppressing Warnings
import warnings
warnings.filterwarnings('ignore')
```

In [5]:

```
# Importing Pandas and NumPy
import pandas as pd, numpy as np
```

In [6]:

```
# Importing Pandas and NumPy
import pandas as pd, numpy as np
```

In [7]:

```
# Importing all datasets
clustering_data = pd.read_csv("C:/Users/HP/Desktop/Predict_Book_Price/clustering/CCGENERAL.csv")
clustering_data.head(2)
```

Out[7]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	ONEOFF_PU
0	C10001	40.900749	0.818182	95.4	0.0	95.4	0.000000	0.166667	
1	C10002	3202.467416	0.909091	0.0	0.0	0.0	6442.945483	0.000000	

In [8]:

```
clustering_data.dtypes
```

Out[8]:

```
CUST_ID                              object
BALANCE                             float64
BALANCE_FREQUENCY                   float64
PURCHASES                           float64
ONEOFF_PURCHASES                    float64
INSTALLMENTS_PURCHASES              float64
CASH_ADVANCE                        float64
PURCHASES_FREQUENCY                 float64
ONEOFF_PURCHASES_FREQUENCY          float64
PURCHASES_INSTALLMENTS_FREQUENCY    float64
CASH_ADVANCE_FREQUENCY              float64
CASH_ADVANCE_TRX                      int64
PURCHASES_TRX                         int64
```

```
CREDIT_LIMIT                        float64
PAYMENTS                            float64
MINIMUM_PAYMENTS                    float64
PRC_FULL_PAYMENT                    float64
TENURE                                int64
```

`Clean` the data

In [9]:

```
# missing values
round(100*(clustering_data.isnull().sum())/len(clustering_data), 2)
```

Out[9]:

```
CUST_ID                             0.00
BALANCE                             0.00
BALANCE_FREQUENCY                   0.00
PURCHASES                           0.00
ONEOFF_PURCHASES                    0.00
INSTALLMENTS_PURCHASES              0.00
CASH_ADVANCE                        0.00
PURCHASES_FREQUENCY                 0.00
ONEOFF_PURCHASES_FREQUENCY          0.00
PURCHASES_INSTALLMENTS_FREQUENCY    0.00
CASH_ADVANCE_FREQUENCY              0.00
CASH_ADVANCE_TRX                    0.00
PURCHASES_TRX                       0.00
CREDIT_LIMIT                        0.01
PAYMENTS                            0.00
MINIMUM_PAYMENTS                    3.50
PRC_FULL_PAYMENT                    0.00
TENURE                              0.00
dtype: float64
```

-- Here , the missing values are present in the columns -

- CREDIT_LIMIT
- MINIMUM_PAYMENTS

In [10]:

```
def impute_nan(clustering_data,variable,median):
    clustering_data[variable+"_median"]=clustering_data[variable].fillna(median)
    clustering_data[variable+"_random"]=clustering_data[variable]
    ##It will have the random sample to fill the na
    random_sample=clustering_data[variable].dropna().sample(clustering_data[variable].isnull().sum(),random_state=0)
    ##pandas need to have same index in order to merge the dataset
    random_sample.index=clustering_data[clustering_data[variable].isnull()].index
    clustering_data.loc[clustering_data[variable].isnull(),variable+'_random']=random_sample
```

In [11]:
```
median=clustering_data.CREDIT_LIMIT.median()
```

In [12]:
```
median
```

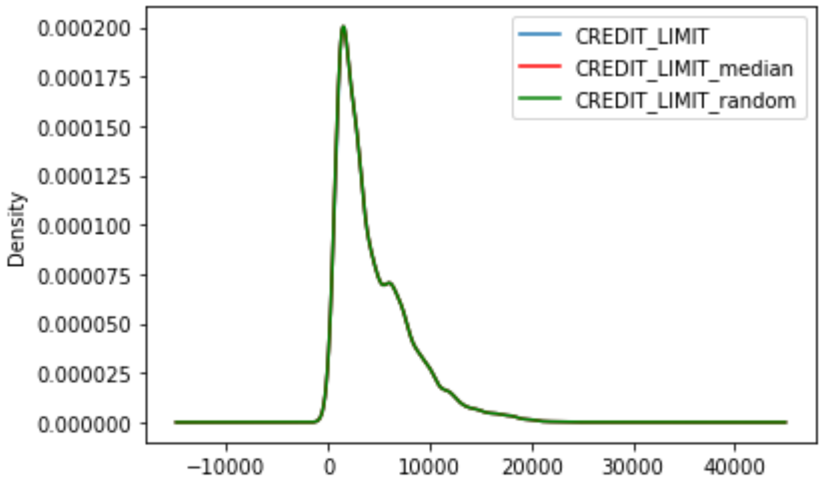
Out[12]: 3000.0

In [13]:
```
impute_nan(clustering_data,"CREDIT_LIMIT",median)
```

In [14]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

In [15]:
```
fig = plt.figure()
ax = fig.add_subplot(111)

clustering_data.CREDIT_LIMIT.plot(kind='kde', ax=ax)
clustering_data.CREDIT_LIMIT_median.plot(kind='kde', ax=ax, color='red')
clustering_data.CREDIT_LIMIT_random.plot(kind='kde', ax=ax, color='green')
lines, labels = ax.get_legend_handles_labels()
ax.legend(lines, labels, loc='best')
```

Out[15]: <matplotlib.legend.Legend at 0x291cc2cfe20>



In [16]:
```
clustering_data = clustering_data.drop(columns='CREDIT_LIMIT')
```

In [17]:
```python
clustering_data = clustering_data.drop(columns='CREDIT_LIMIT_random')
```

In [18]:
```python
clustering_data = clustering_data.rename(columns={"CREDIT_LIMIT_median": "CREDIT_LIMIT"})
```

Similarly,

In [19]:
```python
median=clustering_data.MINIMUM_PAYMENTS.median()
```

In [20]:
```python
median
```

Out[20]:
```
312.343947
```

In [21]:
```python
impute_nan(clustering_data,"MINIMUM_PAYMENTS",median)
```

In [22]:
```python
import matplotlib.pyplot as plt
%matplotlib inline
```

In [23]:
```python
fig = plt.figure()
ax = fig.add_subplot(111)

clustering_data.MINIMUM_PAYMENTS.plot(kind='kde', ax=ax)
clustering_data.MINIMUM_PAYMENTS_median.plot(kind='kde', ax=ax, color='red')
clustering_data.MINIMUM_PAYMENTS_random.plot(kind='kde', ax=ax, color='green')
lines, labels = ax.get_legend_handles_labels()
ax.legend(lines, labels, loc='best')
```

Out[23]:
```
<matplotlib.legend.Legend at 0x291cc529220>
```

In [24]:
```
clustering_data = clustering_data.drop(columns='MINIMUM_PAYMENTS')
```

In [25]:
```
clustering_data = clustering_data.drop(columns='MINIMUM_PAYMENTS_random')
```

In [26]:
```
clustering_data = clustering_data.rename(columns={"MINIMUM_PAYMENTS_median": "MINIMUM_PAYMENTS"})
```

In [27]:
```
clustering_data.head(2)
```

Out[27]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	ONEOFF_PU
0	C10001	40.900749	0.818182	95.4	0.0	95.4	0.000000	0.166667	
1	C10002	3202.467416	0.909091	0.0	0.0	0.0	6442.945483	0.000000	

In [28]:
```
clustering_data.isnull().sum()
```

Out[28]:
```
CUST_ID                             0
BALANCE                             0
BALANCE_FREQUENCY                   0
PURCHASES                           0
ONEOFF_PURCHASES                    0
INSTALLMENTS_PURCHASES              0
CASH_ADVANCE                        0
PURCHASES_FREQUENCY                 0
ONEOFF_PURCHASES_FREQUENCY          0
PURCHASES_INSTALLMENTS_FREQUENCY    0
CASH_ADVANCE_FREQUENCY              0
CASH_ADVANCE_TRX                    0
PURCHASES_TRX                       0
PAYMENTS                            0
PRC_FULL_PAYMENT                    0
TENURE                              0
CREDIT_LIMIT                        0
MINIMUM_PAYMENTS                    0
dtype: int64
```

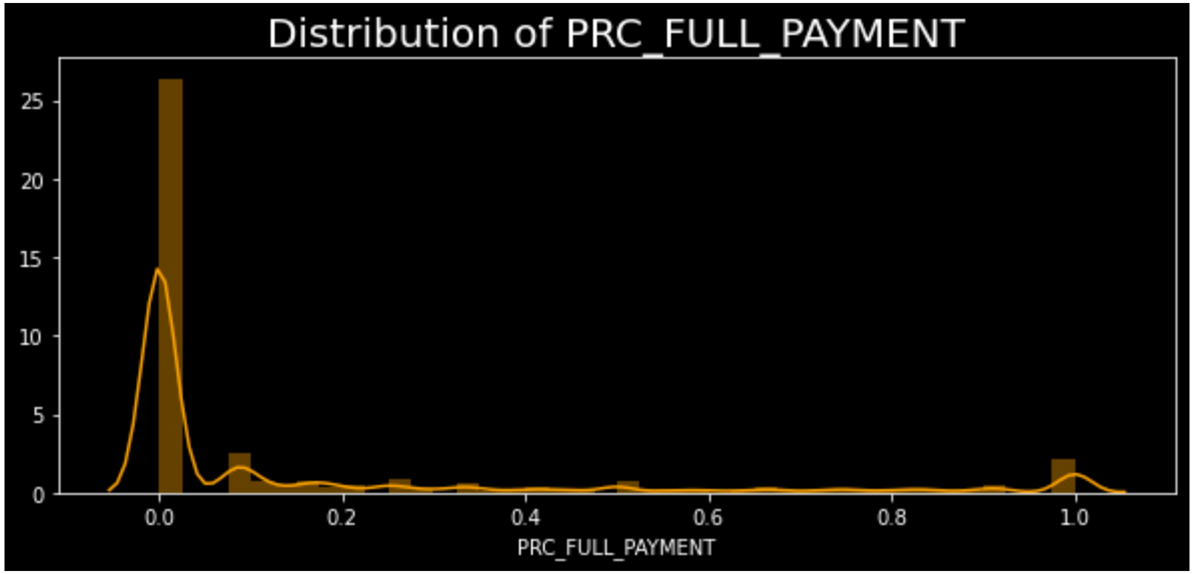
Univariate Analysis

In [29]:

```
import seaborn as sns
import matplotlib.pyplot as plt
plt.style.use("dark_background")
```

In [30]:

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

import seaborn as sns
plt.figure(figsize = [10,4])
sns.distplot(clustering_data.PRC_FULL_PAYMENT,  bins = 40, color = "orange")
plt.title("Distribution of PRC_FULL_PAYMENT", fontsize = 20, fontweight = 10, verticalalignment = 'baseline')

plt.show()
```



In [31]:

```
clustering_data.head(2)
```

Out[31]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	ONEOFF_PU
0	C10001	40.900749	0.818182	95.4	0.0	95.4	0.000000	0.166667	
1	C10002	3202.467416	0.909091	0.0	0.0	0.0	6442.945483	0.000000	

Records In Clustering Data

In [32]:

```
print('# Records in train data:',clustering_data.shape[0])
clustering_data.nunique().sort_values().head(29)
```

```
# Records in train data: 8950
```

Out[32]:

```
TENURE                                 7
BALANCE_FREQUENCY                     43
ONEOFF_PURCHASES_FREQUENCY            47
PURCHASES_FREQUENCY                   47
PURCHASES_INSTALLMENTS_FREQUENCY      47
PRC_FULL_PAYMENT                      47
CASH_ADVANCE_FREQUENCY                54
CASH_ADVANCE_TRX                      65
PURCHASES_TRX                        173
CREDIT_LIMIT                         205
ONEOFF_PURCHASES                    4014
CASH_ADVANCE                        4323
INSTALLMENTS_PURCHASES              4452
PURCHASES                           6203
MINIMUM_PAYMENTS                    8636
PAYMENTS                            8711
BALANCE                             8871
CUST_ID                             8950
dtype: int64
```

In [33]:

```
for col in clustering_data.nunique().sort_values().head(13).reset_index()['index'].tolist():
    print(col,'\n')
    display(clustering_data.groupby(col).size().reset_index())
    print('--'*50,'\n')
```

```
TENURE
```

	TENURE	0
0	6	204
1	7	190
2	8	196
3	9	175
4	10	236
5	11	365
6	12	7584

BALANCE_FREQUENCY

	BALANCE_FREQUENCY	0
0	0.000000	80
1	0.090909	67
2	0.100000	8
3	0.111111	5
4	0.125000	9
5	0.142857	7
6	0.166667	7
7	0.181818	146
8	0.200000	9
9	0.222222	5
10	0.250000	8
11	0.272727	151
12	0.285714	8
13	0.300000	9
14	0.333333	22
15	0.363636	170
16	0.375000	9
17	0.400000	10
18	0.428571	5
19	0.444444	7
20	0.454545	172
21	0.500000	40
22	0.545455	219
23	0.555556	10
24	0.571429	19

	BALANCE_FREQUENCY	0
25	0.600000	6
26	0.625000	11
27	0.636364	209
28	0.666667	37
29	0.700000	13
30	0.714286	15
31	0.727273	223
32	0.750000	17
33	0.777778	22
34	0.800000	20
35	0.818182	278
36	0.833333	60
37	0.857143	51
38	0.875000	57
39	0.888889	53
40	0.900000	55
41	0.909091	410

```
----------------------------------------------------------------------------------------------------
ONEOFF_PURCHASES_FREQUENCY
```

	ONEOFF_PURCHASES_FREQUENCY	0
0	0.000000	4302
1	0.083333	1104
2	0.090909	56
3	0.100000	39
4	0.111111	26
5	0.125000	41
6	0.142857	37

	ONEOFF_PURCHASES_FREQUENCY	0
7	0.166667	592
8	0.181818	34
9	0.200000	27
10	0.222222	12
11	0.250000	418
12	0.272727	12
13	0.285714	9
14	0.300000	10
15	0.333333	355
16	0.363636	13
17	0.375000	11
18	0.400000	5
19	0.416667	244
20	0.428571	8
21	0.444444	4
22	0.454545	13
23	0.500000	235
24	0.545455	8
25	0.555556	2
26	0.571429	11
27	0.583333	197
28	0.600000	7
29	0.625000	3
30	0.636364	7
31	0.666667	167
32	0.700000	4
33	0.714286	7

	ONEOFF_PURCHASES_FREQUENCY	0
34	0.727273	6
35	0.750000	142
36	0.777778	2
37	0.800000	4
38	0.818182	10
39	0.833333	120
40	0.857143	1
41	0.875000	6
42	0.888889	2
43	0.900000	1
44	0.909091	4

```
PURCHASES_FREQUENCY
```

	PURCHASES_FREQUENCY	0
0	0.000000	2043
1	0.083333	677
2	0.090909	43
3	0.100000	27
4	0.111111	18
5	0.125000	32
6	0.142857	26
7	0.166667	392
8	0.181818	16
9	0.200000	19
10	0.222222	12
11	0.250000	345
12	0.272727	19

	PURCHASES_FREQUENCY	0
13	0.285714	8
14	0.300000	13
15	0.333333	367
16	0.363636	10
17	0.375000	10
18	0.400000	9
19	0.416667	289
20	0.428571	9
21	0.444444	5
22	0.454545	19
23	0.500000	395
24	0.545455	20
25	0.555556	7
26	0.571429	16
27	0.583333	316
28	0.600000	11
29	0.625000	8
30	0.636364	17
31	0.666667	310
32	0.700000	11
33	0.714286	13
34	0.727273	15
35	0.750000	299
36	0.777778	6
37	0.800000	9
38	0.818182	21
39	0.833333	373

	PURCHASES_FREQUENCY	0
40	0.857143	25
41	0.875000	26
42	0.888889	18
43	0.900000	24
44	0.909091	28

```
PURCHASES_INSTALLMENTS_FREQUENCY
```

	PURCHASES_INSTALLMENTS_FREQUENCY	0
0	0.000000	3915
1	0.083333	275
2	0.090909	12
3	0.100000	6
4	0.111111	9
5	0.125000	5
6	0.142857	6
7	0.166667	305
8	0.181818	14
9	0.200000	9
10	0.222222	5
11	0.250000	255
12	0.272727	13
13	0.285714	9
14	0.300000	10
15	0.333333	255
16	0.363636	11
17	0.375000	6
18	0.400000	8

	PURCHASES_INSTALLMENTS_FREQUENCY	0
19	0.416667	388
20	0.428571	7
21	0.444444	8
22	0.454545	19
23	0.500000	310
24	0.545455	13
25	0.555556	10
26	0.571429	10
27	0.583333	225
28	0.600000	12
29	0.625000	10
30	0.636364	16
31	0.666667	292
32	0.700000	11
33	0.714286	22
34	0.727273	9
35	0.750000	291
36	0.777778	13
37	0.800000	18
38	0.818182	21
39	0.833333	311
40	0.857143	30
41	0.875000	28
42	0.888889	28
43	0.900000	19
44	0.909091	25
45	0.916667	345

PRC_FULL_PAYMENT

	PRC_FULL_PAYMENT	0
0	0.000000	5903
1	0.083333	426
2	0.090909	153
3	0.100000	94
4	0.111111	61
5	0.125000	52
6	0.142857	54
7	0.166667	166
8	0.181818	75
9	0.200000	83
10	0.222222	20
11	0.250000	156
12	0.272727	35
13	0.285714	24
14	0.300000	40
15	0.333333	134
16	0.363636	32
17	0.375000	13
18	0.400000	42
19	0.416667	44
20	0.428571	14
21	0.444444	17
22	0.454545	36
23	0.500000	156
24	0.545455	27

	PRC_FULL_PAYMENT	0
25	0.555556	12
26	0.571429	14
27	0.583333	31
28	0.600000	28
29	0.625000	9
30	0.636364	26
31	0.666667	78
32	0.700000	12
33	0.714286	19
34	0.727273	22
35	0.750000	68
36	0.777778	19
37	0.800000	33
38	0.818182	17
39	0.833333	63
40	0.857143	12
41	0.875000	18
42	0.888889	12
43	0.900000	16
44	0.909091	19
45	0.916667	77

```
--------------------------------------------------------------------------------

CASH_ADVANCE_FREQUENCY
```

	CASH_ADVANCE_FREQUENCY	0
0	0.000000	4628
1	0.083333	1021
2	0.090909	70

	CASH_ADVANCE_FREQUENCY	0
3	0.100000	39
4	0.111111	29
5	0.125000	47
6	0.142857	49
7	0.166667	759
8	0.181818	42
9	0.200000	21
10	0.222222	18
11	0.250000	578
12	0.272727	38
13	0.285714	30
14	0.300000	23
15	0.333333	439
16	0.363636	20
17	0.375000	11
18	0.400000	15
19	0.416667	273
20	0.428571	21
21	0.444444	15
22	0.454545	14
23	0.500000	215
24	0.545455	10
25	0.555556	12
26	0.571429	12
27	0.583333	142
28	0.600000	9
29	0.625000	5

	CASH_ADVANCE_FREQUENCY	0
30	0.636364	8
31	0.666667	125
32	0.700000	1
33	0.714286	4
34	0.727273	8
35	0.750000	63
36	0.777778	3
37	0.800000	6
38	0.818182	2
39	0.833333	48
40	0.857143	5
41	0.875000	5
42	0.888889	2
43	0.900000	2
44	0.909091	3
45	0.916667	27
46	1.000000	25
47	1.090909	1
48	1.100000	1
49	1.125000	1
50	1.142857	1
51	1.166667	2

```
--------------------------------------------------------------------------------------------
CASH_ADVANCE_TRX
```

	CASH_ADVANCE_TRX	0
0	0	4628
1	1	887

	CASH_ADVANCE_TRX	0
2	2	620
3	3	436
4	4	384
...	...	...
60	80	1
61	93	1
62	107	1
63	110	1
64	123	3

```
----------------------------------------------------------------------------------------------------
PURCHASES_TRX
```

	PURCHASES_TRX	0
0	0	2044
1	1	667
2	2	379
3	3	314
4	4	285
...	...	...
168	308	1
169	309	1
170	344	1
171	347	1
172	358	1

173 rows × 2 columns

```
----------------------------------------------------------------------------------------------------
CREDIT_LIMIT
```

	CREDIT_LIMIT	0
0	50.0	1
1	150.0	5
2	200.0	3
3	300.0	14
4	400.0	3
...	...	...
200	22500.0	1
201	23000.0	2
202	25000.0	1
203	28000.0	1
204	30000.0	2

205 rows × 2 columns

```
--------------------------------------------------------------------------------------------

ONEOFF_PURCHASES
```

	ONEOFF_PURCHASES	0
0	0.00	4302
1	0.01	7
2	0.02	2
3	0.05	1
4	0.24	1
...	...	...
4009	26547.43	1
4010	33803.84	1
4011	34087.73	1
4012	40624.06	1
4013	40761.25	1

4014 rows × 2 columns

```
CASH_ADVANCE
```

	CASH_ADVANCE	0
0	0.000000	4628
1	14.222216	1
2	18.042768	1
3	18.117967	1
4	18.123413	1
...	...	...
4318	26194.049540	1
4319	26268.699890	1
4320	27296.485760	1
4321	29282.109150	1
4322	47137.211760	1

4323 rows × 2 columns

```
INSTALLMENTS_PURCHASES
```

	INSTALLMENTS_PURCHASES	0
0	0.00	3916
1	1.95	1
2	4.44	1
3	4.80	1
4	6.33	1
...	...	...
4447	12738.47	1
4448	13184.43	1

	INSTALLMENTS_PURCHASES	0
4449	14686.10	1
4450	15497.19	1
4451	22500.00	1

```
----------------------------------------------------------------------------------------------------
```

In [34]:

```
cols_with_very_low_distinct_values = ['BALANCE','PAYMENTS']
for col in cols_with_very_low_distinct_values:
    print(col,'\n')
    display(clustering_data.groupby(col).size().reset_index())
    print('--'*50,'\n')
```

```
BALANCE
```

	BALANCE	0
0	0.000000	80
1	0.000199	1
2	0.001146	1
3	0.001214	1
4	0.001289	1
...	...	...
8866	16115.596400	1
8867	16259.448570	1
8868	16304.889250	1
8869	18495.558550	1
8870	19043.138560	1

8871 rows × 2 columns

```
----------------------------------------------------------------------------------------------------

PAYMENTS
```

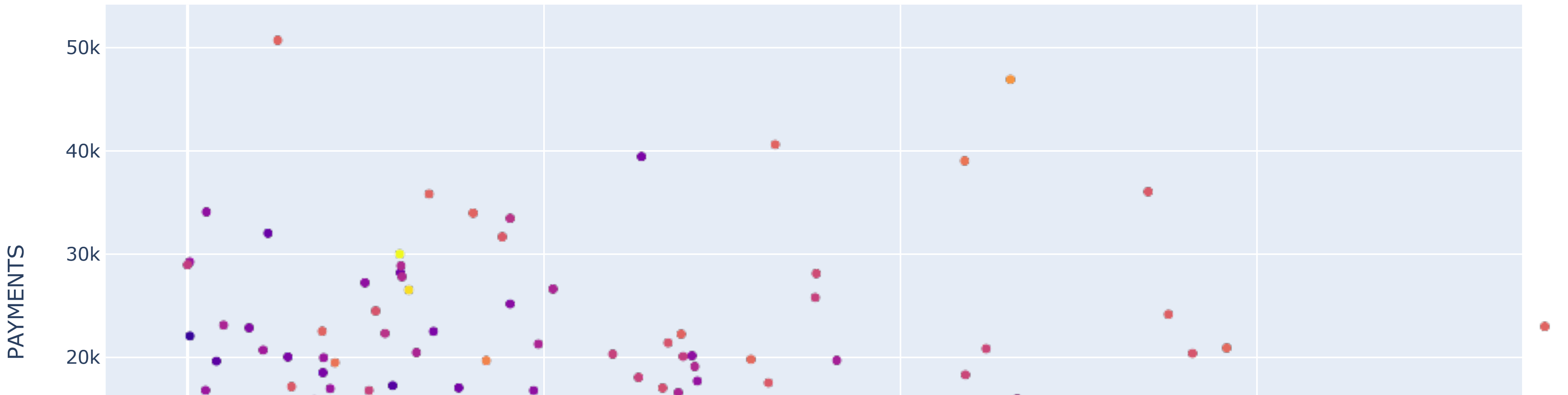
	PAYMENTS	0
0	0.000000	240

	PAYMENTS	0
1	0.049513	1
2	0.056466	1
3	2.389583	1
4	3.500505	1
...	...	...
8706	39048.597620	1
8707	39461.965800	1
8708	40627.595240	1
8709	46930.598240	1
8710	50721.483360	1

SCATTER PLOT WITH BALANCE, PAYMENTS AND CREDIT_LIMITS

In [35]:

```
fig = px.scatter(clustering_data, x="BALANCE", y="PAYMENTS",color="CREDIT_LIMIT", hover_data=['CREDIT_LIMIT'])
fig.show()
```



In [36]:

```python
clustering_data.corr()
```

Out[36]:

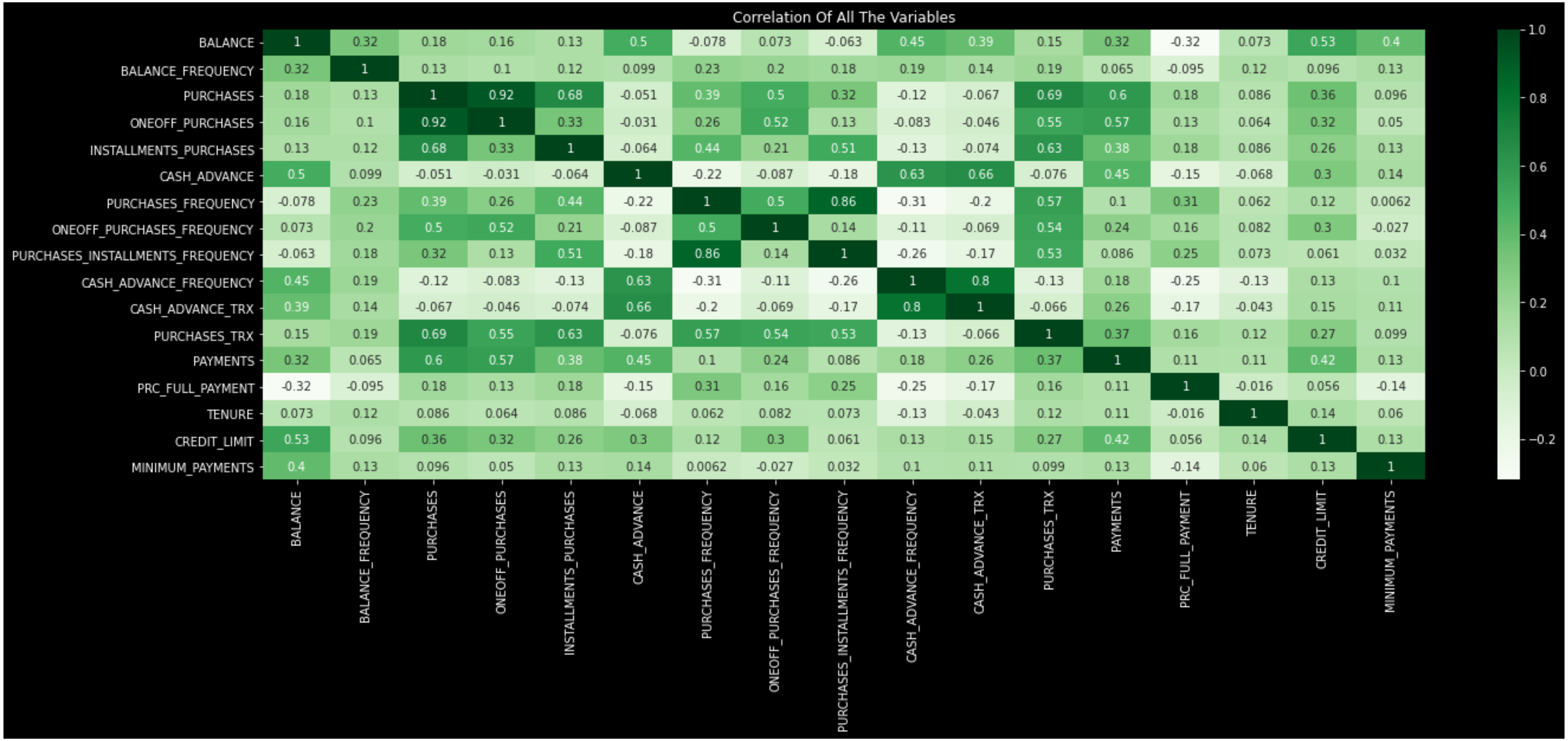
	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES
BALANCE	1.000000	0.322412	0.181261	0.164350	0.126469	0.496692	
BALANCE_FREQUENCY	0.322412	1.000000	0.133674	0.104323	0.124292	0.099388	
PURCHASES	0.181261	0.133674	1.000000	0.916845	0.679896	-0.051474	
ONEOFF_PURCHASES	0.164350	0.104323	0.916845	1.000000	0.330622	-0.031326	
INSTALLMENTS_PURCHASES	0.126469	0.124292	0.679896	0.330622	1.000000	-0.064244	
CASH_ADVANCE	0.496692	0.099388	-0.051474	-0.031326	-0.064244	1.000000	
PURCHASES_FREQUENCY	-0.077944	0.229715	0.393017	0.264937	0.442418	-0.215507	
ONEOFF_PURCHASES_FREQUENCY	0.073166	0.202415	0.498430	0.524891	0.214042	-0.086754	
PURCHASES_INSTALLMENTS_FREQUENCY	-0.063186	0.176079	0.315567	0.127729	0.511351	-0.177070	
CASH_ADVANCE_FREQUENCY	0.449218	0.191873	-0.120143	-0.082628	-0.132318	0.628522	
CASH_ADVANCE_TRX	0.385152	0.141555	-0.067175	-0.046212	-0.073999	0.656498	
PURCHASES_TRX	0.154338	0.189626	0.689561	0.545523	0.628108	-0.075850	
PAYMENTS	0.322802	0.065008	0.603264	0.567292	0.384084	0.453238	
PRC_FULL_PAYMENT	-0.318959	-0.095082	0.180379	0.132763	0.182569	-0.152935	
TENURE	0.072692	0.119776	0.086288	0.064150	0.086143	-0.068312	
CREDIT_LIMIT	0.531296	0.095931	0.356977	0.319735	0.256515	0.303997	
MINIMUM_PAYMENTS	0.397920	0.131181	0.095789	0.050256	0.134019	0.140747	

In [37]:

```python
import seaborn as sns
import matplotlib.pyplot as plt
plt.style.use("dark_background")
plt.figure(figsize = [22,7])
plt.title("Correlation Of All The Variables")
sns.heatmap(clustering_data.corr(),annot=True,cmap="Greens")
```

Out[37]:

```
<AxesSubplot:title={'center':'Correlation Of All The Variables'}>
```



In [38]:

```python
clustering_data.columns
```

Out[38]:

```
Index(['CUST_ID', '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', 'PAYMENTS', 'PRC_FULL_PAYMENT',
```

```
       'TENURE', 'CREDIT_LIMIT', 'MINIMUM_PAYMENTS'],
      dtype='object')
```

Prepare the data for `modelling`

- R (PURCHASES_TRX,CASH_ADVANCE_TRX): PURCHASES_TRX,CASH_ADVANCE_TRX
- F (BALANCE_FREQUENCY,PURCHASES_FREQUENCY,CASH_ADVANCE_FREQUENCY): BALANCE_FREQUENCY,PURCHASES_FREQUENCY,CASH_ADVANCE_FREQUENCY
- M (BALANCE,PURCHASES,CASH_ADVANCE): BALANCE,PURCHASES,CASH_ADVANCE

In [39]:

```
# R : Recency
Recency = clustering_data[['CUST_ID','PURCHASES_TRX','CASH_ADVANCE_TRX']]
Recency.head(4)
```

Out[39]:

	CUST_ID	PURCHASES_TRX	CASH_ADVANCE_TRX
0	C10001	2	0
1	C10002	0	4
2	C10003	12	0
3	C10004	1	1

In [40]:

```
# F : Frequency
frequency = clustering_data[['CUST_ID','BALANCE_FREQUENCY','PURCHASES_FREQUENCY','CASH_ADVANCE_FREQUENCY']]
frequency.head()
```

Out[40]:

	CUST_ID	BALANCE_FREQUENCY	PURCHASES_FREQUENCY	CASH_ADVANCE_FREQUENCY
0	C10001	0.818182	0.166667	0.000000
1	C10002	0.909091	0.000000	0.250000
2	C10003	1.000000	1.000000	0.000000
3	C10004	0.636364	0.083333	0.083333
4	C10005	1.000000	0.083333	0.000000

In [41]:
```python
#M : Monetary

monetary = clustering_data[['CUST_ID','BALANCE','PURCHASES','CASH_ADVANCE']]
monetary.head()
```

Out[41]:

	CUST_ID	BALANCE	PURCHASES	CASH_ADVANCE
0	C10001	40.900749	95.40	0.000000
1	C10002	3202.467416	0.00	6442.945483
2	C10003	2495.148862	773.17	0.000000
3	C10004	1666.670542	1499.00	205.788017
4	C10005	817.714335	16.00	0.000000

In [42]:
```python
# merge the two dfs
grouped_df = pd.merge(Recency, frequency, on='CUST_ID', how='inner')
grouped_df.head()
```

Out[42]:

	CUST_ID	PURCHASES_TRX	CASH_ADVANCE_TRX	BALANCE_FREQUENCY	PURCHASES_FREQUENCY	CASH_ADVANCE_FREQUENCY
0	C10001	2	0	0.818182	0.166667	0.000000
1	C10002	0	4	0.909091	0.000000	0.250000
2	C10003	12	0	1.000000	1.000000	0.000000
3	C10004	1	1	0.636364	0.083333	0.083333
4	C10005	1	0	1.000000	0.083333	0.000000

In [43]:
```python
# merge the two dfs
grouped_df = pd.merge(grouped_df, monetary, on='CUST_ID', how='inner')
grouped_df.head()
```

Out[43]:

	CUST_ID	PURCHASES_TRX	CASH_ADVANCE_TRX	BALANCE_FREQUENCY	PURCHASES_FREQUENCY	CASH_ADVANCE_FREQUENCY	BALANCE	PURCHASES	CASH_ADV
0	C10001	2	0	0.818182	0.166667	0.000000	40.900749	95.40	0.0
1	C10002	0	4	0.909091	0.000000	0.250000	3202.467416	0.00	6442.9
2	C10003	12	0	1.000000	1.000000	0.000000	2495.148862	773.17	0.0
3	C10004	1	1	0.636364	0.083333	0.083333	1666.670542	1499.00	205.7

CUST_ID	PURCHASES_TRX	CASH_ADVANCE_TRX	BALANCE_FREQUENCY	PURCHASES_FREQUENCY	CASH_ADVANCE_FREQUENCY	BALANCE	PURCHASES	CASH_ADV

OUTLIER TREATMENT

In [44]:

```
# removing (statistical) outliers
Q1 = grouped_df.BALANCE.quantile(0.05)
Q3 = grouped_df.BALANCE.quantile(0.95)
IQR = Q3 - Q1
grouped_df = grouped_df[(grouped_df.BALANCE >= Q1 - 1.5*IQR) & (grouped_df.BALANCE <= Q3 + 1.5*IQR)]

# outlier treatment for Purchases
Q1 = grouped_df.PURCHASES.quantile(0.05)
Q3 = grouped_df.PURCHASES.quantile(0.95)
IQR = Q3 - Q1
grouped_df = grouped_df[(grouped_df.PURCHASES >= Q1 - 1.5*IQR) & (grouped_df.PURCHASES <= Q3 + 1.5*IQR)]

# outlier treatment for Cash_advance
Q1 = grouped_df.CASH_ADVANCE.quantile(0.05)
Q3 = grouped_df.CASH_ADVANCE.quantile(0.95)
IQR = Q3 - Q1
grouped_df = grouped_df[(grouped_df.CASH_ADVANCE >= Q1 - 1.5*IQR) & (grouped_df.CASH_ADVANCE <= Q3 + 1.5*IQR)]
```

In [45]:

```
grouped_df.shape
```

Out[45]:

```
(8829, 9)
```

In [46]:

```
grouped_df.columns
```

Out[46]:

```
Index(['CUST_ID', 'PURCHASES_TRX', 'CASH_ADVANCE_TRX', 'BALANCE_FREQUENCY',
       'PURCHASES_FREQUENCY', 'CASH_ADVANCE_FREQUENCY', 'BALANCE', 'PURCHASES',
       'CASH_ADVANCE'],
      dtype='object')
```

In [47]:

```
# 2. rescaling
rfm_df = grouped_df[['PURCHASES_TRX', 'CASH_ADVANCE_TRX', 'BALANCE_FREQUENCY',
       'PURCHASES_FREQUENCY', 'CASH_ADVANCE_FREQUENCY', 'BALANCE', 'PURCHASES',
       'CASH_ADVANCE']]

# instantiate
scaler = StandardScaler()

# fit_transform
rfm_df_scaled = scaler.fit_transform(rfm_df)
rfm_df_scaled.shape
```

Out[47]: (8829, 8)

In [48]:

```
rfm_df_scaled = pd.DataFrame(rfm_df_scaled)
```

In [49]:

```
rfm_df_scaled.columns = ['PURCHASES_TRX', 'CASH_ADVANCE_TRX', 'BALANCE_FREQUENCY',
       'PURCHASES_FREQUENCY', 'CASH_ADVANCE_FREQUENCY', 'BALANCE', 'PURCHASES',
       'CASH_ADVANCE']
```

In [50]:

```
rfm_df_scaled.head()
```

Out[50]:

	PURCHASES_TRX	CASH_ADVANCE_TRX	BALANCE_FREQUENCY	PURCHASES_FREQUENCY	CASH_ADVANCE_FREQUENCY	BALANCE	PURCHASES	CASH_ADVANCE
0	-0.546755	-0.488643	-0.243046	-0.800798	-0.675904	-0.745374	-0.572673	-0.530617
1	-0.639263	0.134981	0.138998	-1.216788	0.591787	0.861317	-0.643088	3.275908
2	-0.084215	-0.488643	0.521043	1.279150	-0.675904	0.501862	-0.072408	-0.530617
3	-0.593009	-0.332737	-1.007136	-1.008794	-0.253342	0.080833	0.463330	-0.409036
4	-0.593009	-0.488643	0.521043	-1.008794	-0.675904	-0.350602	-0.631278	-0.530617

MODELLING

In [51]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import datetime as dt

import sklearn
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

from scipy.cluster.hierarchy import linkage
from scipy.cluster.hierarchy import dendrogram
from scipy.cluster.hierarchy import cut_tree
```

In [52]:
```
# k-means with some arbitrary k
kmeans = KMeans(n_clusters=4, max_iter=50)
kmeans.fit(rfm_df_scaled)
```

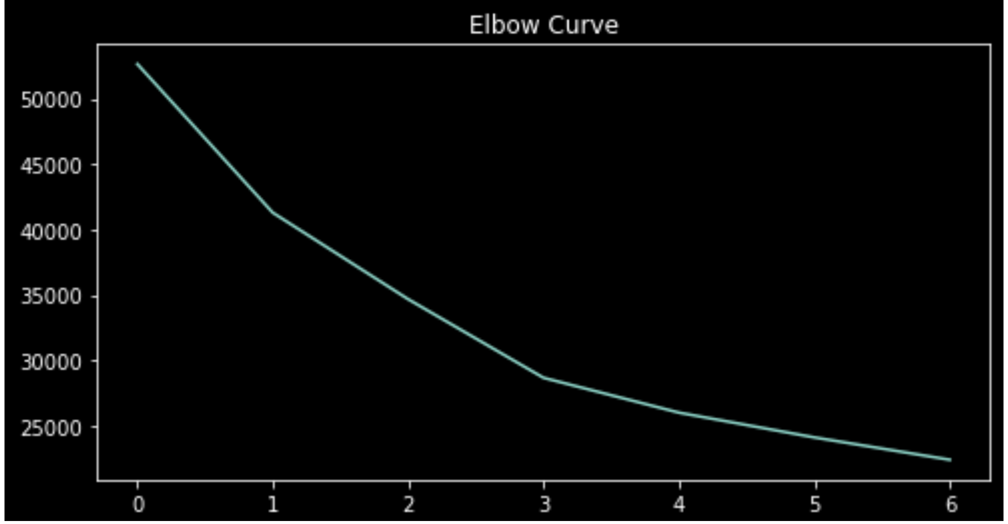
Out[52]:
```
KMeans(max_iter=50, n_clusters=4)
```

In [53]:
```
kmeans.labels_
```

Out[53]:
```
array([0, 1, 0, ..., 0, 0, 0])
```

In [54]:
```
plt.figure(figsize = [8,4])
ssd = []
range_n_clusters = [2, 3, 4, 5, 6, 7, 8]
for num_clusters in range_n_clusters:
    kmeans = KMeans(n_clusters=num_clusters, max_iter=50)
    kmeans.fit(rfm_df_scaled)

    ssd.append(kmeans.inertia_)

# plot the SSDs for each n_clusters
# ssd
plt.title("Elbow Curve")
plt.plot(ssd)
```

Out[54]:
```
[<matplotlib.lines.Line2D at 0x291cc3d9af0>]
```



Silhouette Analysis

$$\text{silhouette score}=\frac{p-q}{max(p,q)}$$

p is the mean distance to the points in the nearest cluster that the data point is not a part of

q is the mean intra-cluster distance to all the points in its own cluster.

- The value of the silhouette score range lies between -1 to 1.
- A score closer to 1 indicates that the data point is very similar to other data points in the cluster,
- A score closer to -1 indicates that the data point is not similar to the data points in its cluster.

In [55]:

```
# silhouette analysis
range_n_clusters = [ 2, 3, 4, 5, 6, 7, 8]

for num_clusters in range_n_clusters:

    # intialise kmeans
    kmeans = KMeans(n_clusters=num_clusters, max_iter=50)
    kmeans.fit(rfm_df_scaled)

    cluster_labels = kmeans.labels_

    # silhouette score
    silhouette_avg = silhouette_score(rfm_df_scaled, cluster_labels)
    print("For n_clusters={0}, the silhouette score is {1}".format(num_clusters, silhouette_avg))
```

```
For n_clusters=2, the silhouette score is 0.3499235353390019
For n_clusters=3, the silhouette score is 0.2745563845223045
For n_clusters=4, the silhouette score is 0.3065103545230291
For n_clusters=5, the silhouette score is 0.3257442070873397
For n_clusters=6, the silhouette score is 0.31010580706625296
For n_clusters=7, the silhouette score is 0.31402082807112125
For n_clusters=8, the silhouette score is 0.2960298382116163
```

In [56]:

```
# final model with k=3
kmeans = KMeans(n_clusters=3, max_iter=50)
kmeans.fit(rfm_df_scaled)
```

Out[56]:

```
KMeans(max_iter=50, n_clusters=3)
```

In [57]:

```
kmeans.labels_
```

Out[57]:

```
array([1, 2, 0, ..., 1, 1, 1])
```

In [58]:

```
# assign the label
grouped_df['cluster_id'] = kmeans.labels_
grouped_df.head()
```

Out[58]:

	CUST_ID	PURCHASES_TRX	CASH_ADVANCE_TRX	BALANCE_FREQUENCY	PURCHASES_FREQUENCY	CASH_ADVANCE_FREQUENCY	BALANCE	PURCHASES	CASH_ADV
0	C10001	2	0	0.818182	0.166667	0.000000	40.900749	95.40	0.0
1	C10002	0	4	0.909091	0.000000	0.250000	3202.467416	0.00	6442.9
2	C10003	12	0	1.000000	1.000000	0.000000	2495.148862	773.17	0.0
3	C10004	1	1	0.636364	0.083333	0.083333	1666.670542	1499.00	205.7
4	C10005	1	0	1.000000	0.083333	0.000000	817.714335	16.00	0.0

In [59]:

```
grouped_df.cluster_id.value_counts()
```

Out[59]:

```
1    4038
0    3332
2    1459
Name: cluster_id, dtype: int64
```

In [63]:
```
from sklearn import metrics
score = metrics.silhouette_score(rfm_df_scaled, grouped_df["cluster_id"])
score
```

Out[63]: 0.2747262163390628

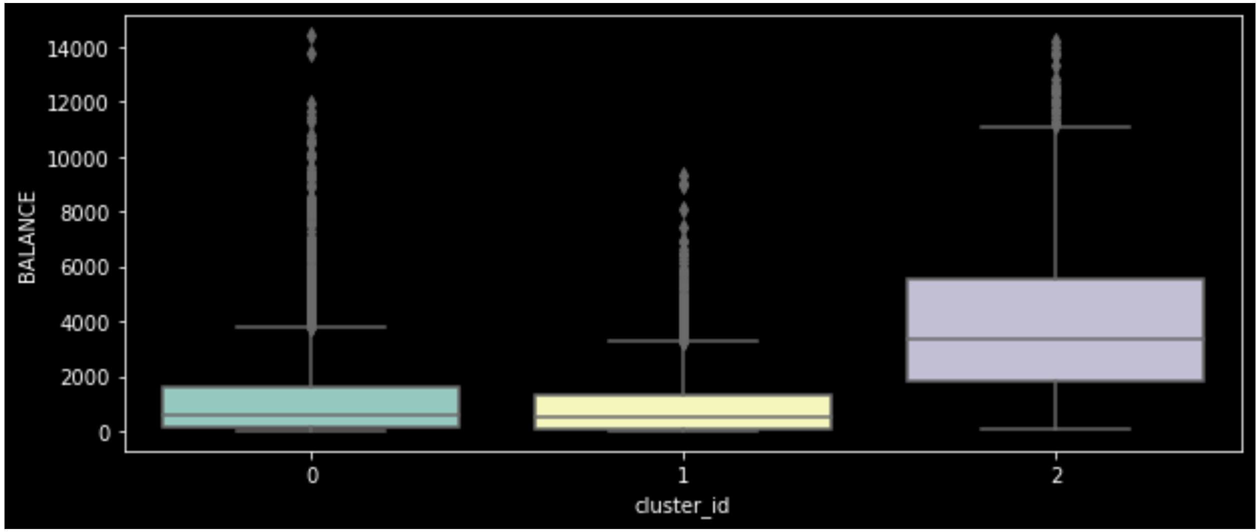
In [84]:
```
print("The Score For the K-means Cluster",score)
```

The Score For the K-means Cluster 0.6216666669733575

Cluster Formation

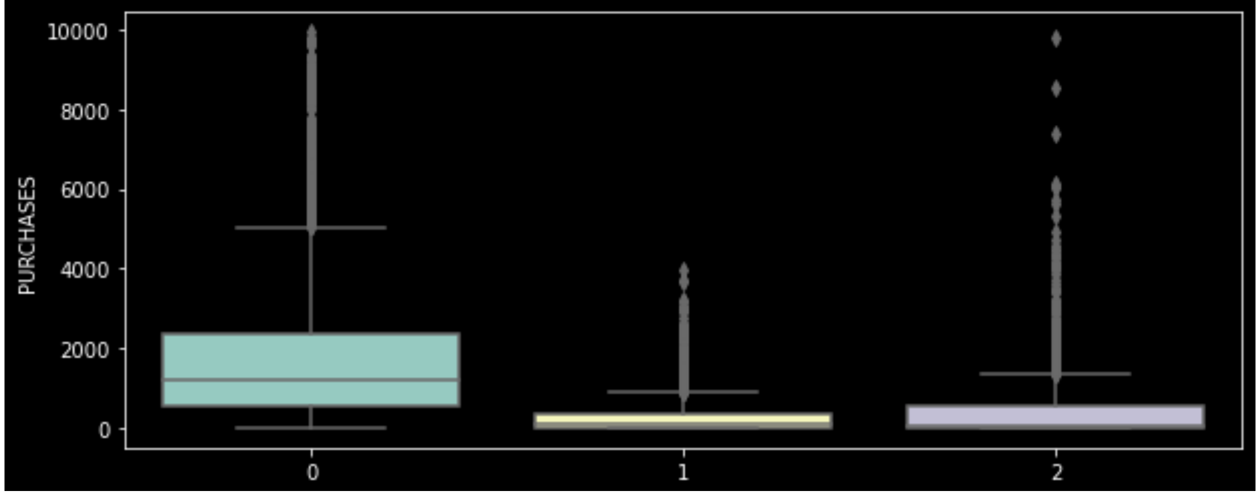
In [65]:
```
# Income
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=[10,4])

sns.boxplot(x='cluster_id', y='BALANCE', data=grouped_df)
```

Out[65]: <AxesSubplot:xlabel='cluster_id', ylabel='BALANCE'>



In [66]:
```
plt.figure(figsize=[10,4])
sns.boxplot(x='cluster_id', y='PURCHASES', data=grouped_df)
```

Out[66]: <AxesSubplot:xlabel='cluster_id', ylabel='PURCHASES'>



In [67]:

```
grouped_df.columns
```

Out[67]:

```
Index(['CUST_ID', 'PURCHASES_TRX', 'CASH_ADVANCE_TRX', 'BALANCE_FREQUENCY',
       'PURCHASES_FREQUENCY', 'CASH_ADVANCE_FREQUENCY', 'BALANCE', 'PURCHASES',
       'CASH_ADVANCE', 'cluster_id'],
      dtype='object')
```

In [68]:

```
cluster_1 = grouped_df.where(grouped_df.cluster_id == 1)
cluster_1 = cluster_1.dropna()
cluster_1 = cluster_1.sort_values(by = ['BALANCE','PURCHASES','CASH_ADVANCE'], ascending= False)
cluster_1.head()
```

Out[68]:

	CUST_ID	PURCHASES_TRX	CASH_ADVANCE_TRX	BALANCE_FREQUENCY	PURCHASES_FREQUENCY	CASH_ADVANCE_FREQUENCY	BALANCE	PURCHASES	CASH_
416	C10431	5.0	0.0	1.0	0.166667	0.000000	9335.314170	226.23	
3921	C14032	0.0	1.0	1.0	0.000000	0.083333	9061.317491	0.00	3
1655	C11709	5.0	0.0	1.0	0.333333	0.000000	8953.743398	254.85	
1109	C11146	7.0	0.0	1.0	0.500000	0.000000	8115.039014	383.42	
2672	C12749	4.0	0.0	1.0	0.333333	0.000000	7418.314012	901.62	

In [69]:

```
cluster_0 = grouped_df.where(grouped_df.cluster_id == 0)
cluster_0 = cluster_0.dropna()
cluster_0 = cluster_0.sort_values(by = ['BALANCE','PURCHASES','CASH_ADVANCE'], ascending= False)
cluster_0.head()
```

Out[69]:

	CUST_ID	PURCHASES_TRX	CASH_ADVANCE_TRX	BALANCE_FREQUENCY	PURCHASES_FREQUENCY	CASH_ADVANCE_FREQUENCY	BALANCE	PURCHASES	CASH_
643	C10669	98.0	0.0	1.0	1.000000	0.000000	14411.95798	5958.17	
708	C10735	88.0	2.0	1.0	1.000000	0.166667	13763.47358	9670.84	18
174	C10180	90.0	3.0	1.0	1.000000	0.166667	11972.01104	5715.00	16
1697	C11753	52.0	0.0	1.0	1.000000	0.000000	11670.17985	4872.60	
2485	C12559	9.0	0.0	1.0	0.583333	0.000000	11416.64736	1347.70	

In [70]:

```
cluster_2 = grouped_df.where(grouped_df.cluster_id == 2)
cluster_2 = cluster_2.dropna()
cluster_2 = cluster_2.sort_values(by = ['BALANCE','PURCHASES','CASH_ADVANCE'], ascending= False)
cluster_2.head()
```

Out[70]:

	CUST_ID	PURCHASES_TRX	CASH_ADVANCE_TRX	BALANCE_FREQUENCY	PURCHASES_FREQUENCY	CASH_ADVANCE_FREQUENCY	BALANCE	PURCHASES	CASH_
124	C10130	0.0	9.0	1.0	0.000000	0.333333	14224.11541	0.00	46
4089	C14205	9.0	12.0	1.0	0.416667	0.666667	13968.47957	281.71	27
5913	C16079	0.0	11.0	1.0	0.000000	0.666667	13777.37772	0.00	16
723	C10750	3.0	7.0	1.0	0.250000	0.500000	13774.74154	404.24	33
153	C10159	216.0	26.0	1.0	1.000000	0.750000	13673.07961	9792.23	24

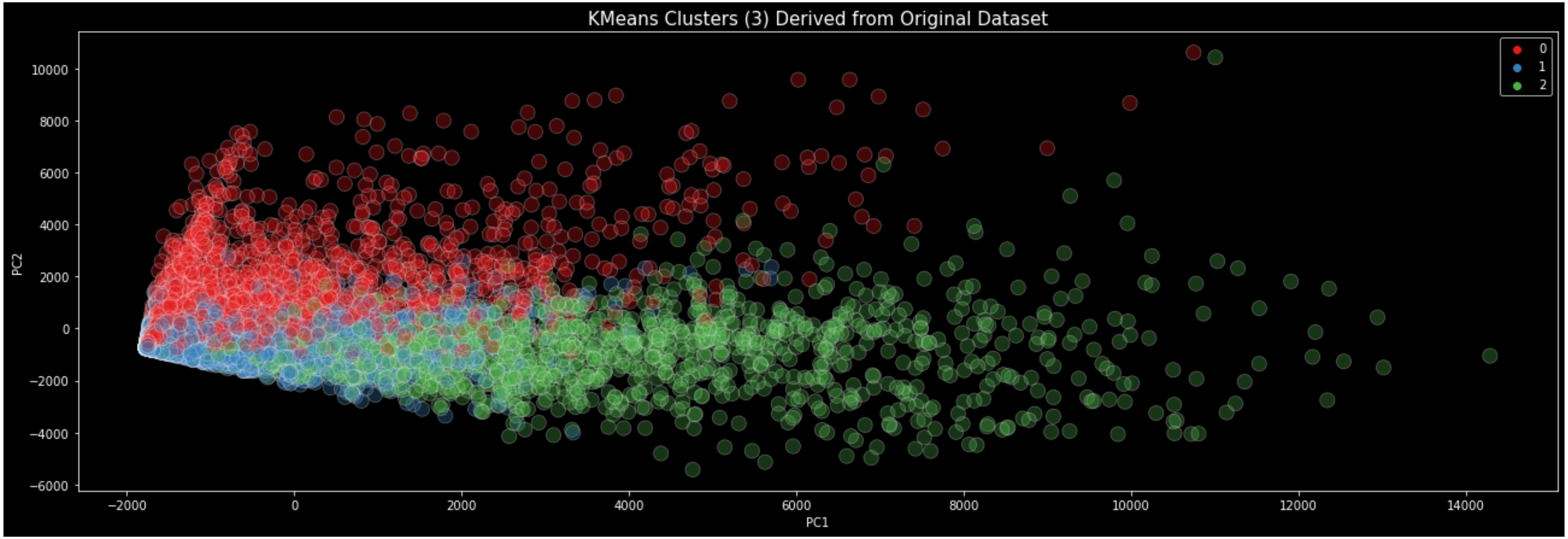
In [71]:

```
grouped_df
cluster_01 = grouped_df.drop(columns="CUST_ID")
labels_scale = kmeans.labels_
```

Principal Component Analysis

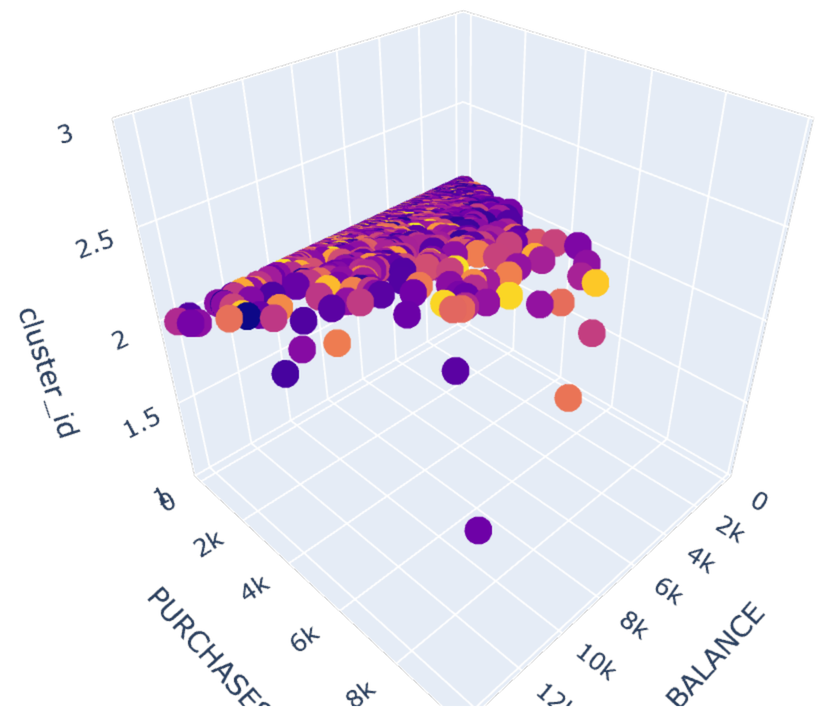
In [72]:

```
from sklearn import svm
import numpy as np
import glob
import os
from PIL import Image
from sklearn.decomposition import PCA
plt.style.use("dark_background")
plt.figure(figsize = [22,7])
pca2 = PCA(n_components=3).fit(cluster_01)
pca2d = pca2.transform(cluster_01)
sns.scatterplot(pca2d[:,0], pca2d[:,1],
                hue=labels_scale,
                palette='Set1',
                s=150, alpha=0.3).set_title('KMeans Clusters (3) Derived from Original Dataset', fontsize=15)

plt.legend()
plt.ylabel('PC2')
plt.xlabel('PC1')
plt.show()
```



In [92]:

```
import plotly.express as px

fig = px.scatter_3d(cluster_2, x='BALANCE', y='PURCHASES', z='cluster_id',
              color='CASH_ADVANCE')
fig.show()
```



HIERARCHICAL CLUSTERING

In [74]:

```
rfm_df_scaled.head(3)
```

Out[74]:

PURCHASES_TRX	CASH_ADVANCE_TRX	BALANCE_FREQUENCY	PURCHASES_FREQUENCY	CASH_ADVANCE_FREQUENCY	BALANCE	PURCHASES	CASH_ADVANCE

	PURCHASES_TRX	CASH_ADVANCE_TRX	BALANCE_FREQUENCY	PURCHASES_FREQUENCY	CASH_ADVANCE_FREQUENCY	BALANCE	PURCHASES	CASH_ADVANCE
0	-0.546755	-0.488643	-0.243046	-0.800798	-0.675904	-0.745374	-0.572673	-0.530617
1	-0.639263	0.134981	0.138998	-1.216788	0.591787	0.861317	-0.643088	3.275908

In [75]:

```
grouped_df.head(3)
```

Out[75]:

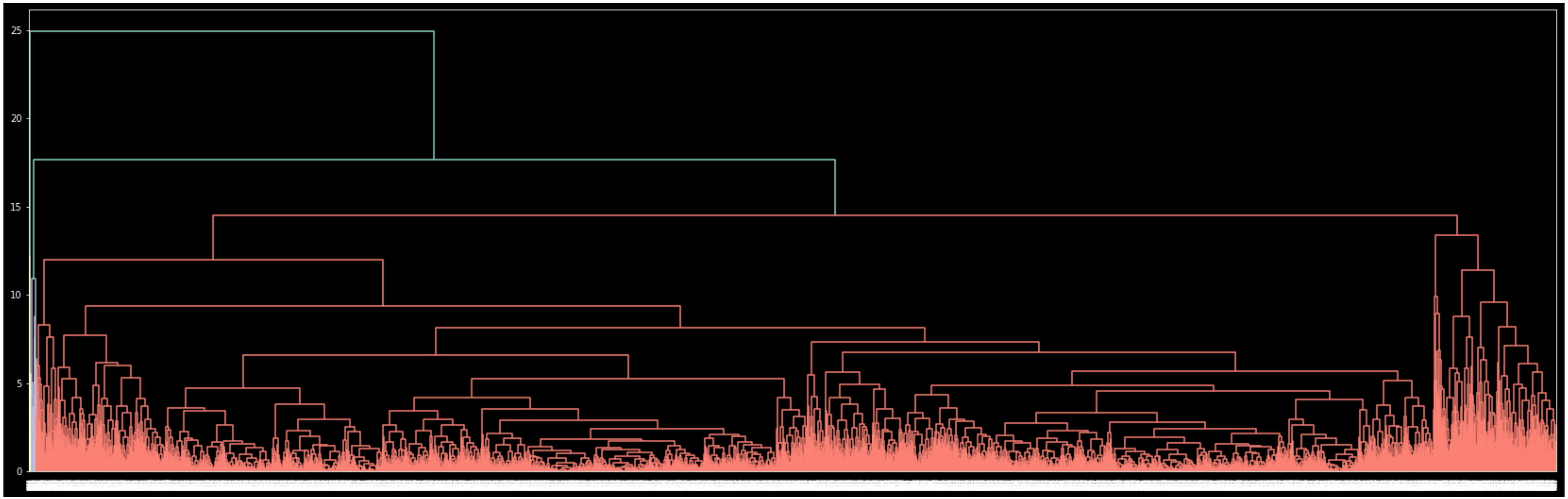
	CUST_ID	PURCHASES_TRX	CASH_ADVANCE_TRX	BALANCE_FREQUENCY	PURCHASES_FREQUENCY	CASH_ADVANCE_FREQUENCY	BALANCE	PURCHASES	CASH_ADV
0	C10001	2	0	0.818182	0.166667	0.00	40.900749	95.40	0.(
1	C10002	0	4	0.909091	0.000000	0.25	3202.467416	0.00	6442.9
2	C10003	12	0	1.000000	1.000000	0.00	2495.148862	773.17	0.(

In [76]:

```
import scipy.cluster.hierarchy as sch
```

In [77]:

```
# complete linkage --  complete linkage give us the better flow of chart
plt.figure(figsize = [29,9])
mergings = linkage(rfm_df_scaled, method="complete", metric='euclidean')
dendrogram(mergings)
plt.show()
```



3 clusters - Here we consider the `Optimum Value` for K == 3 `BLUE LINES`

In [78]:
```
# 3 clusters - Here we consider the Optimum Value for K == 3
cluster_labels = cut_tree(mergings, n_clusters=3).reshape(-1, )
cluster_labels
```

Out[78]:
```
array([0, 0, 0, ..., 0, 0, 0])
```

In [79]:
```
# assign cluster labels
grouped_df['cluster_labels'] = cluster_labels
grouped_df.head()
```

Out[79]:

	CUST_ID	PURCHASES_TRX	CASH_ADVANCE_TRX	BALANCE_FREQUENCY	PURCHASES_FREQUENCY	CASH_ADVANCE_FREQUENCY	BALANCE	PURCHASES	CASH_ADV
0	C10001	2	0	0.818182	0.166667	0.000000	40.900749	95.40	0.0
1	C10002	0	4	0.909091	0.000000	0.250000	3202.467416	0.00	6442.9
2	C10003	12	0	1.000000	1.000000	0.000000	2495.148862	773.17	0.0
3	C10004	1	1	0.636364	0.083333	0.083333	1666.670542	1499.00	205.7
4	C10005	1	0	1.000000	0.083333	0.000000	817.714335	16.00	0.0

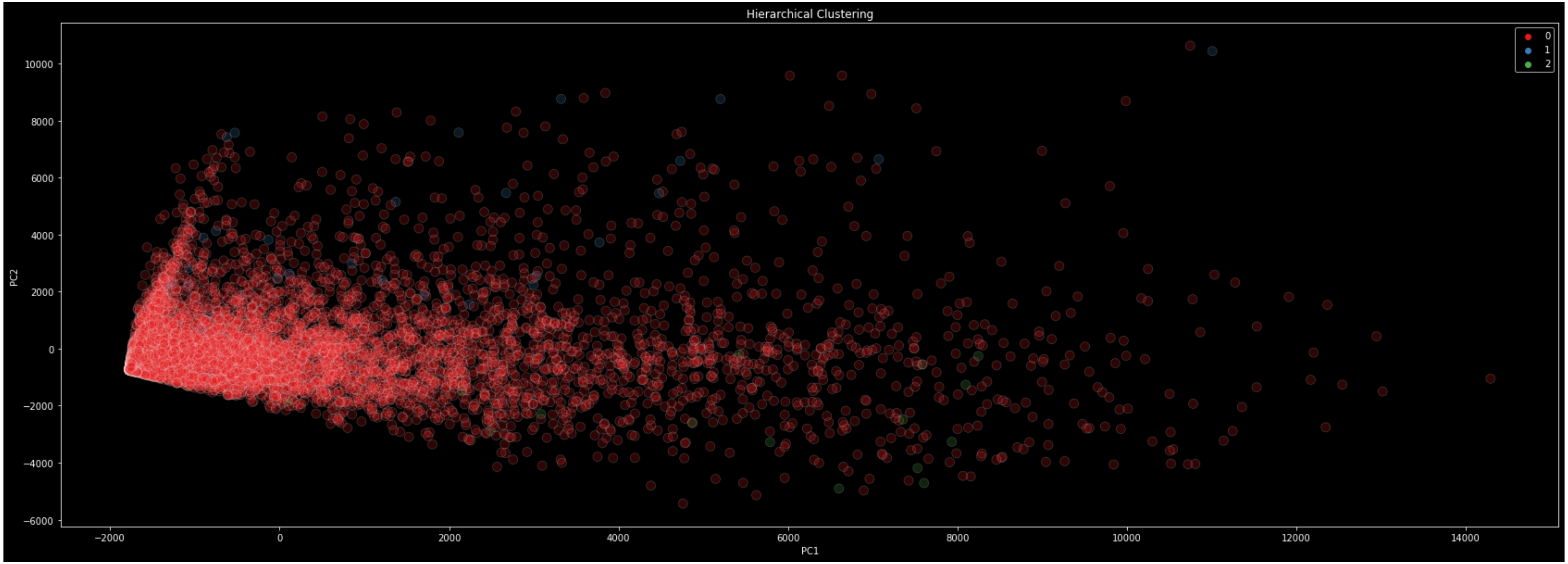
In [80]:
```
grouped_df.cluster_labels.value_counts()
```

Out[80]:
```
0    8783
1      31
2      15
Name: cluster_labels, dtype: int64
```

In [81]:
```
grouped_df
cluster_02 = grouped_df.drop(columns="CUST_ID")
labels_scale = cluster_labels
```

In [82]:

```
from sklearn import svm
import numpy as np
import glob
import os
from PIL import Image
from sklearn.decomposition import PCA
#plt.style.use("seaborn-dark")
plt.figure(figsize = [29,10])
pca2 = PCA(n_components=3).fit(cluster_02)
pca2d = pca2.transform(cluster_02)
sns.scatterplot(pca2d[:,0], pca2d[:,1],
                hue=labels_scale,
                palette='Set1',
                s=100, alpha=0.2).set_title('Hierarchical Clusters  Derived from Original Dataset', fontsize=15)

plt.legend()
plt.ylabel('PC2')
plt.xlabel('PC1')
plt.title('Hierarchical Clustering')
plt.show()
```



In [83]:

```
from sklearn import metrics
score = metrics.silhouette_score(rfm_df_scaled, grouped_df["cluster_labels"])
score
```

Out[83]: 0.6216666669733575

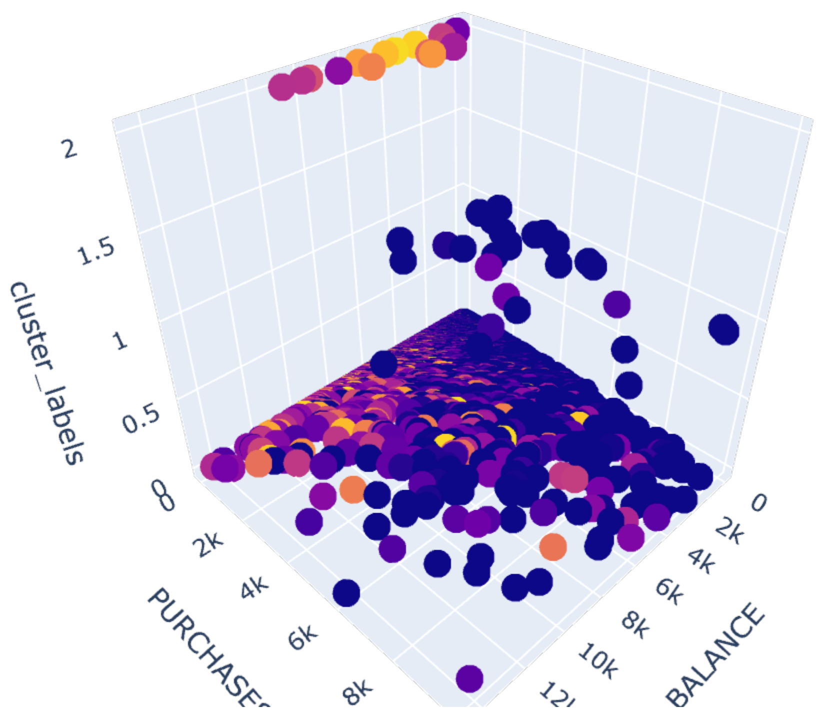
In [90]:

```
print("The cluster performance for the hierarchical cluster", score)
```

The cluster performance for the hierarchical cluster 0.6216666669733575

In [86]:

```
import plotly.express as px

fig = px.scatter_3d(grouped_df, x='BALANCE', y='PURCHASES', z='cluster_labels',
              color='CASH_ADVANCE')
fig.show()
```



In [91]:

```
grouped_df.cluster_labels.value_counts()
```

Out[91]:

```
0    8783
1      31
2      15
Name: cluster_labels, dtype: int64
```

In [ ]: