In [1]:
```
import pandas as pd
import numpy as np
```

In [2]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:
```
%matplotlib inline
```

In [4]:
```
# reading useful datasets into dataframes
investments=pd.read_csv(r'C:\Users\m_joekid\Desktop\Startup Success\investme
funds=pd.read_csv(r'C:\Users\m_joekid\Desktop\Startup Success\funds.csv')
funding_round=pd.read_csv(r'C:\Users\m_joekid\Desktop\Startup Success\fundin
```

BASIC EDA

In [5]:
```
pd.set_option('display.max_columns',None) # collapses all hidden columns
```

In [6]:
```
investments.sample(3)
```

Out[6]:

	id	funding_round_id	funded_object_id	investor_object_id	created_at	updated_at
66559	66560	47262	c:260052	c:141719	2013-08-30 13:35:17	2013-08-30 13:35:17
9364	9365	6564	c:24880	f:372	2009-06-17 01:33:15	2013-11-27 23:06:42
7448	7449	5416	c:20506	f:2178	2009-03-23 18:30:25	2010-02-12 22:48:02

In [7]:
```
funds.sample(3)
```

Out[7]:

	id	fund_id	object_id	name	funded_at	raised_amount	raised_currency_code	
948	1032	1032	f:10144	SB Pan-Asia Fund	2011-01-01	0.0	USD	
1388	1516	1516	f:5832	Lime Rock Resources III, L.P.	2013-10-08	750000000.0	USD	http:
195	211	211	f:542	OpenView Fund I	2006-09-29	100000000.0	USD	http://w

In [8]:
```
1 funding_round.sample(3)
```

Out[8]:

	id	funding_round_id	object_id	funded_at	funding_round_type	funding_round_code	ra
48444	53310	53310	c:275842	2012-10-01	venture	unattributed	
13419	14542	14542	c:6595	2006-08-11	series-a	a	
50783	55739	55739	c:244234	2013-08-20	angel	convertible	

Merging investement and funding round as main dataset

In [9]:
```
1 inv_funrnd=pd.merge(investments,funding_round,on='funding_round_id')
```

INV_FUNRD EDA

In [10]:
```
1 inv_funrnd.sample(3) # get a glimpse into the new dataset
```

Out[10]:

	id_x	funding_round_id	funded_object_id	investor_object_id	created_at_x	updated_at_x
47148	41136	29056	c:32060	f:3806	2012-04-03 02:26:44	2012-04-03 02:26:44
11594	10128	6928	c:25430	f:556	2009-06-26 02:01:39	2011-03-18 21:34:05
64938	55117	44735	c:245365	f:3576	2013-08-06 11:26:53	2013-08-06 11:26:53

In [11]:
```
1 inv_funrnd.describe().T
```

Out[11]:

	count	mean	std	min	25%	50%	
id_x	80902.0	4.045150e+04	2.335454e+04	1.0	20226.25	40451.5	606
funding_round_id	80902.0	2.402017e+04	1.516303e+04	1.0	11747.25	22594.5	347
id_y	80902.0	2.402017e+04	1.516303e+04	1.0	11747.25	22594.5	347
raised_amount_usd	80902.0	1.161108e+07	4.128060e+07	0.0	1000000.00	4480000.0	125000
raised_amount	80902.0	1.188521e+07	4.949011e+07	0.0	1000000.00	4300000.0	122000
pre_money_valuation_usd	80902.0	2.670972e+05	5.299019e+07	0.0	0.00	0.0	
pre_money_valuation	80902.0	2.670972e+05	5.299019e+07	0.0	0.00	0.0	
post_money_valuation_usd	80902.0	1.905719e+06	5.690148e+07	0.0	0.00	0.0	
post_money_valuation	80902.0	1.940594e+06	5.708151e+07	0.0	0.00	0.0	
participants	80902.0	4.305295e+00	3.664336e+00	1.0	2.00	3.0	
is_first_round	80902.0	5.515933e-01	4.973341e-01	0.0	0.00	1.0	
is_last_round	80902.0	5.329658e-01	4.989152e-01	0.0	0.00	1.0	

Feature Engineering -1FE

In [12]:
```
1 # drop1 contains columns to be deleted from the merged dataset
2 drop1=['id_x','funding_round_id','id_y','object_id','raised_amount','raised_
```

In [13]:
```
1 # std_mrg contains a cleaner version of the initially merged datasets
2 std_mrg=inv_funrnd.drop(drop1,axis=1)
```

In [14]:
```
1 std_mrg.sample(3)
```

Out[14]:

	funded_object_id	investor_object_id	created_at_x	updated_at_x	funded_at	funding_round_
51783	c:169322	p:2902	2012-08-25 07:35:12	2013-05-01 02:01:51	2012-08-24	ver
78834	c:83212	p:14	2012-03-20 17:07:41	2013-06-06 11:45:16	2011-05-11	a
58578	c:189208	f:6126	2013-04-04 02:13:40	2013-04-04 02:13:40	2013-04-03	ser

In [15]:
```
1 std_mrg['funding_round_type'].unique() # prints the unique type of funding o
```

Out[15]:
```
array(['series-b', 'series-a', 'angel', 'series-c+', 'venture', 'other',
       'private-equity', 'post-ipo', 'crowdfunding'], dtype=object)
```

In [16]:
```
std_mrg['funding_round_code'].unique() # prints the unique code used for the
```

Out[16]:
```
array(['b', 'a', 'seed', 'angel', 'c', 'd', 'unattributed', 'debt_round',
       'e', 'f', 'private_equity', 'grant', 'post_ipo_debt',
       'post_ipo_equity', 'partial', 'convertible', 'g', 'crowd',
       'secondary_market'], dtype=object)
```

In [17]:
```
# c_std_mrg contains the data usable by clustering algorithms||note the pref
c_std_mrg=std_mrg[['raised_amount_usd','pre_money_valuation_usd','post_money
```

In [18]:
```
c_std_mrg.sample(5)
```

Out[18]:

	raised_amount_usd	pre_money_valuation_usd	post_money_valuation_usd	participants	is_fi
41396	7600000.0	0.0	0.0	3	
78692	1000000.0	0.0	0.0	14	
20184	3015089.0	0.0	0.0	1	
64191	3000000.0	0.0	0.0	4	
74859	0.0	0.0	0.0	2	

In [19]:
```
from sklearn.preprocessing import StandardScaler
```

In [20]:
```
scalar=StandardScaler()
```

In [21]:
```
# the prefix s represents "standardised"|| note the prefix 's' and 'c'
s_c_std_mrg=scalar.fit_transform(c_std_mrg)
```

In [22]:
```
# turns the array to a dataframe
s_c_std_mrg=pd.DataFrame(s_c_std_mrg)
```

In [23]:
```
from scipy.cluster.hierarchy import dendrogram,linkage
```

In [24]:
```
# import k-means clustering algorithm || explained in documentation
from sklearn.cluster import KMeans
```

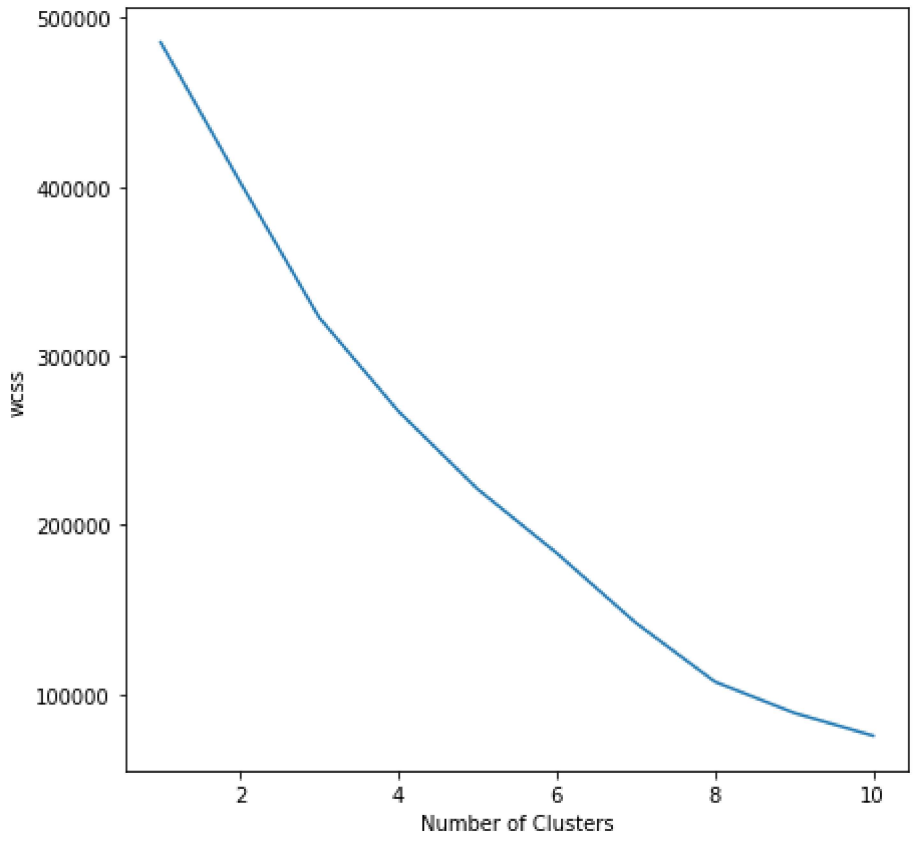
In [25]:
```
# dendrogram=dendrogram(linkage(s_c_std_mrg,method='ward')) #not enough memo
```

Elbow Method for Optimal K || Kmeans Clustering

In [26]:
```
wcss =[]
for i in range(1,11):
        kmeans=KMeans(n_clusters=i,init='k-means++',random_state=101)
        kmeans.fit(s_c_std_mrg)
        wcss.append(kmeans.inertia_)
```

In [27]:
```
plt.figure(figsize=(7,7))
plt.plot(range(1,11),wcss)
plt.xlabel('Number of Clusters')
plt.ylabel('wcss')
```

Out[27]: Text(0, 0.5, 'wcss')



Kmeans Clustering (-1FE||NO-PCA||K=8 after elbow optimisation)

In [28]:
```
kmeans=KMeans(n_clusters=8,init='k-means++',random_state=101)
```

In [29]:
```
kmeans.fit(s_c_std_mrg)
```

Out[29]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=8, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=101, tol=0.0001, verbose=0)
```

In [30]:
```
c_std_mrg_kmeans=c_std_mrg.copy() # made a copy of the clustering dataset
c_std_mrg_kmeans['Segment Kmeans']=kmeans.labels_ #added a new column to rep
```

POST Kmeans Clustering (-1FE||NO-PCA||K=8) EDA

In [31]:
```
1 c_std_mrg_kmeans.sample(5)
```

Out[31]:

	raised_amount_usd	pre_money_valuation_usd	post_money_valuation_usd	participants	is_fi
13601	1507400.0	0.0	0.0	1	
9284	10000000.0	0.0	0.0	4	
53443	18000.0	0.0	300000.0	1	
25875	14000000.0	0.0	0.0	5	
60732	1120000.0	0.0	0.0	4	

In [32]:
```
1 std_mrg_kmeans=std_mrg.copy() # made a copy of the clean merged dataset
2 std_mrg_kmeans['Segment Kmeans']=kmeans.labels_ # added a new column to iden
```

In [33]:
```
1 std_mrg_kmeans.sample(3)
```

Out[33]:

	funded_object_id	investor_object_id	created_at_x	updated_at_x	funded_at	funding_round_
60770	c:82754	p:193872	2013-05-28 12:49:45	2013-05-28 12:49:45	2013-01-15	a
13701	c:30221	f:2257	2009-08-24 17:10:16	2009-08-24 17:10:16	2009-08-24	ver
18233	c:12423	f:127	2010-01-15 02:25:27	2010-01-15 19:10:44	2006-08-17	ser

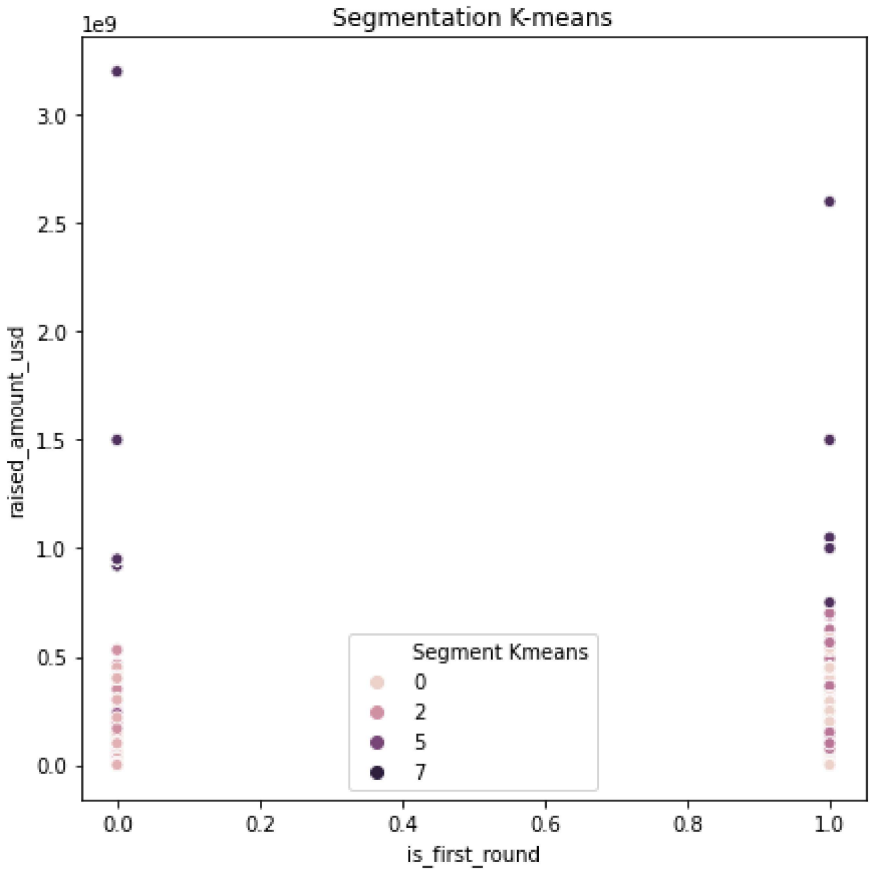
In [34]:
```
1 item_anal_1=c_std_mrg_kmeans.groupby(['Segment Kmeans']).mean()
```

In [35]:
```
1 item_anal_2=c_std_mrg_kmeans.groupby(['Segment Kmeans']).count()
```

```
1 ## POST Kmeans Clustering (-1FE||NO-PCA||K=8) Visualisations
```

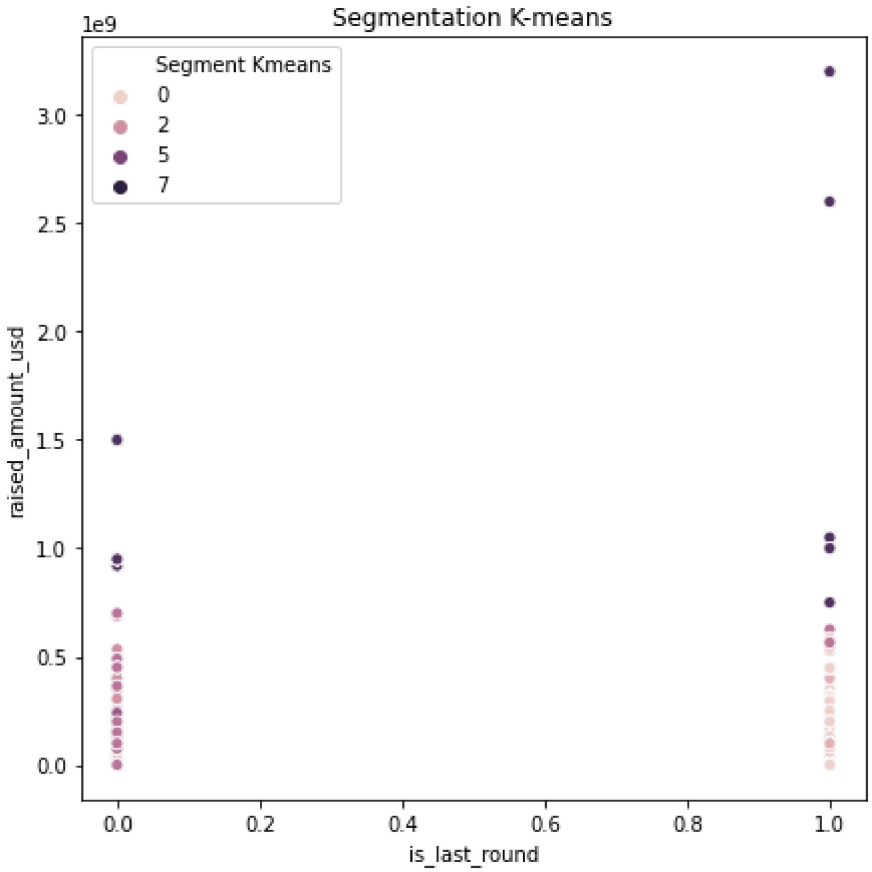
In [36]:
```
plt.figure(figsize=(7,7))
sns.scatterplot(std_mrg_kmeans['is_first_round'],std_mrg_kmeans['raised_amou
plt.title('Segmentation K-means')
```

Out[36]: Text(0.5, 1.0, 'Segmentation K-means')



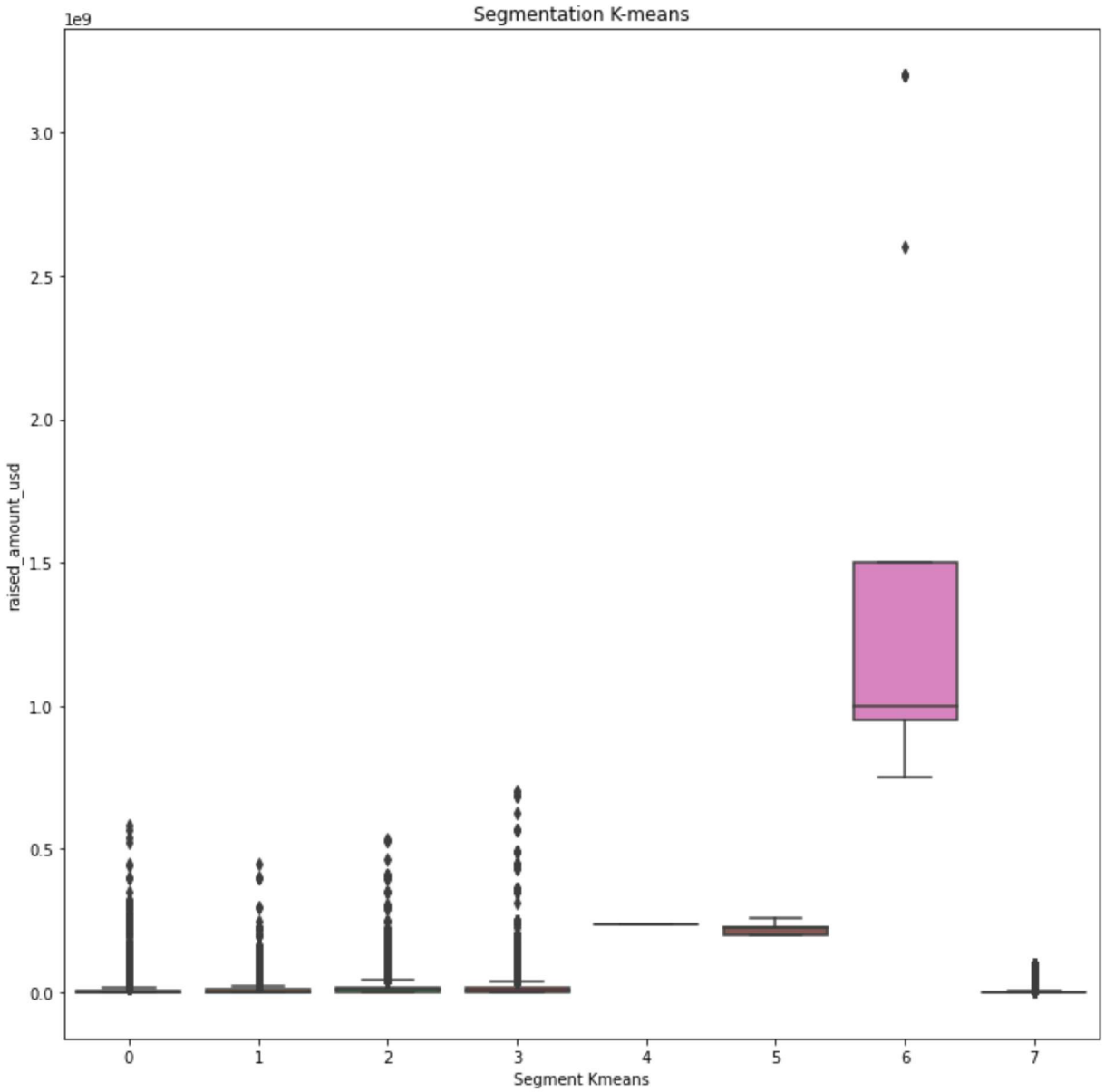
In [37]:
```
plt.figure(figsize=(7,7))
sns.scatterplot(std_mrg_kmeans['is_last_round'],std_mrg_kmeans['raised_amoun
plt.title('Segmentation K-means')
```

Out[37]: Text(0.5, 1.0, 'Segmentation K-means')



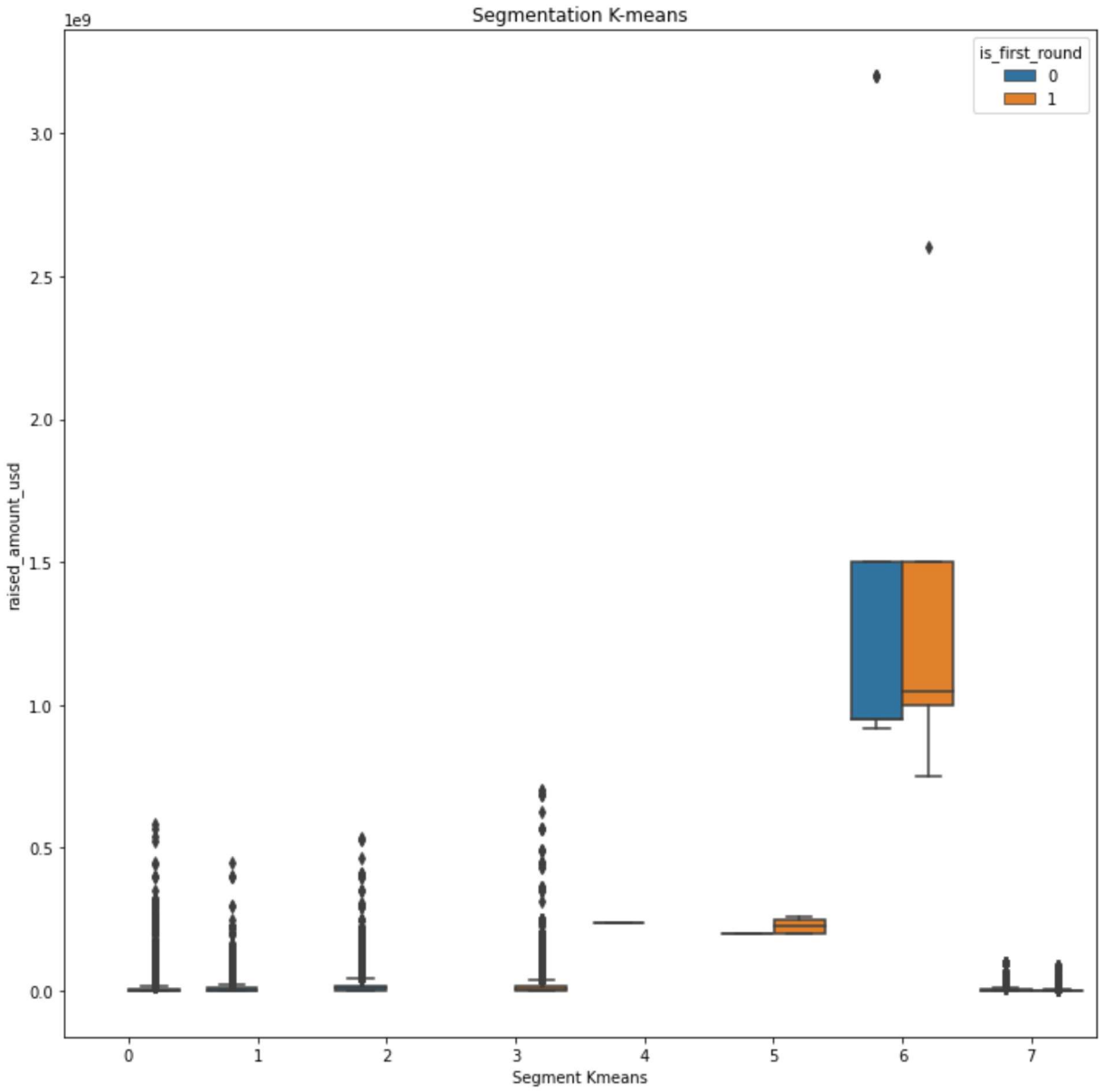
In [38]:
```
plt.figure(figsize=(12,12))
sns.boxplot(x='Segment Kmeans',y='raised_amount_usd',data=std_mrg_kmeans)
plt.title('Segmentation K-means')
```

Out[38]: Text(0.5, 1.0, 'Segmentation K-means')



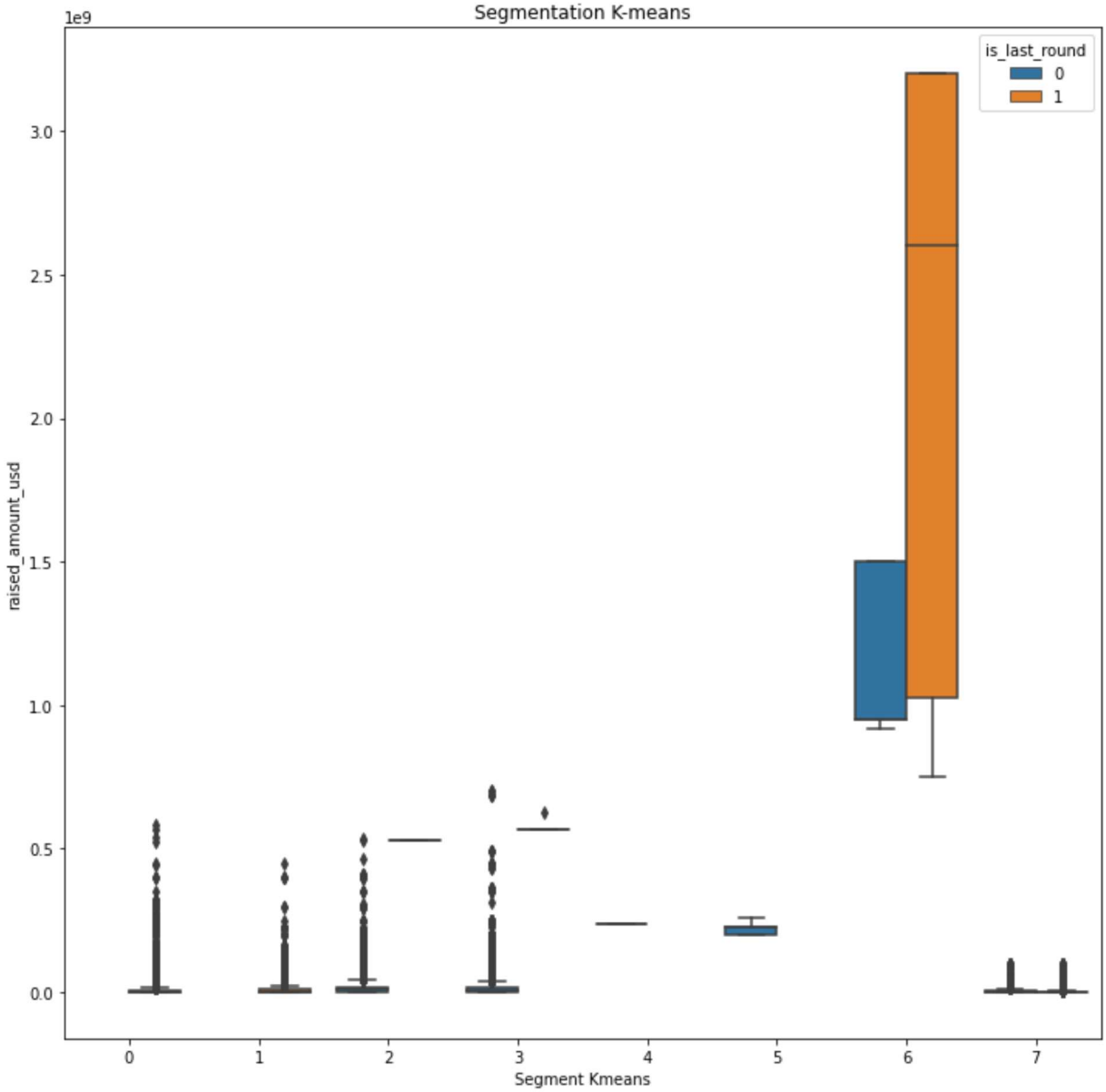
In [39]:
```
plt.figure(figsize=(12,12))
sns.boxplot(x='Segment Kmeans',y='raised_amount_usd',hue='is_first_round',da
plt.title('Segmentation K-means')
```

Out[39]: Text(0.5, 1.0, 'Segmentation K-means')



In [40]:
```
plt.figure(figsize=(12,12))
sns.boxplot(x='Segment Kmeans',y='raised_amount_usd',hue='is_last_round',dat
plt.title('Segmentation K-means')
```

Out[40]: Text(0.5, 1.0, 'Segmentation K-means')



In [41]:
```
# THIS PLOT NEEDS TO BE MAXIMIZED
# plt.figure(figsize=(60,60))
# sns.boxplot(hue='Segment Kmeans',y='raised_amount_usd',x='funding_round_ty
# plt.title('Segmentation K-means')
```

```
## Kmeans Clustering (-1FE||PCA||K=?)
```

In [42]:
```
from sklearn.decomposition import PCA
```

In [43]:
```
pca=PCA()
```

In [44]:
```
pca.fit(s_c_std_mrg)
```

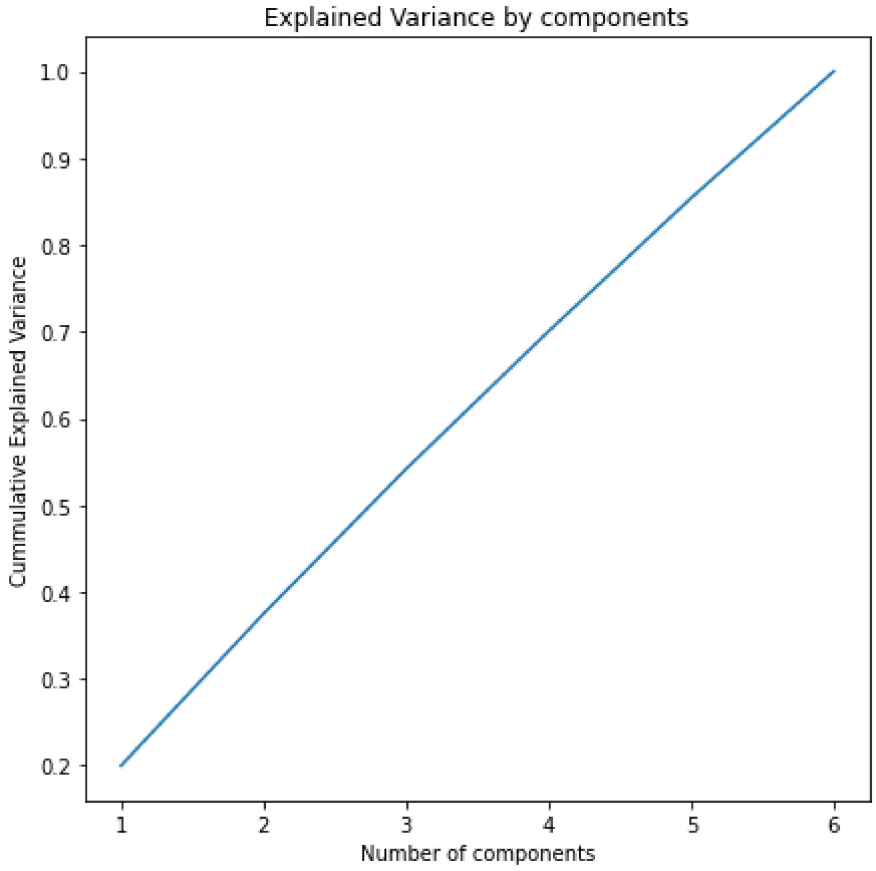
Out[44]:
```
PCA(copy=True, iterated_power='auto', n_components=None, random_state=None,
    svd_solver='auto', tol=0.0, whiten=False)
```

In [45]:
```
pca.explained_variance_ratio_
```

Out[45]:
```
array([0.19984419, 0.17508847, 0.16682985, 0.15873865, 0.15360567,
       0.14589317])
```

In [46]:
```
plt.figure(figsize=(7,7))
plt.plot(range(1,7),pca.explained_variance_ratio_.cumsum())
plt.xlabel('Number of components')
plt.ylabel('Cummulative Explained Variance')
plt.title('Explained Variance by components')
```

Out[46]:
```
Text(0.5, 1.0, 'Explained Variance by components')
```



In [47]:
```
pca=PCA(n_components=4)
```

In [48]:
```
pca.fit(s_c_std_mrg)
```

Out[48]:
```
PCA(copy=True, iterated_power='auto', n_components=4, random_state=None,
    svd_solver='auto', tol=0.0, whiten=False)
```

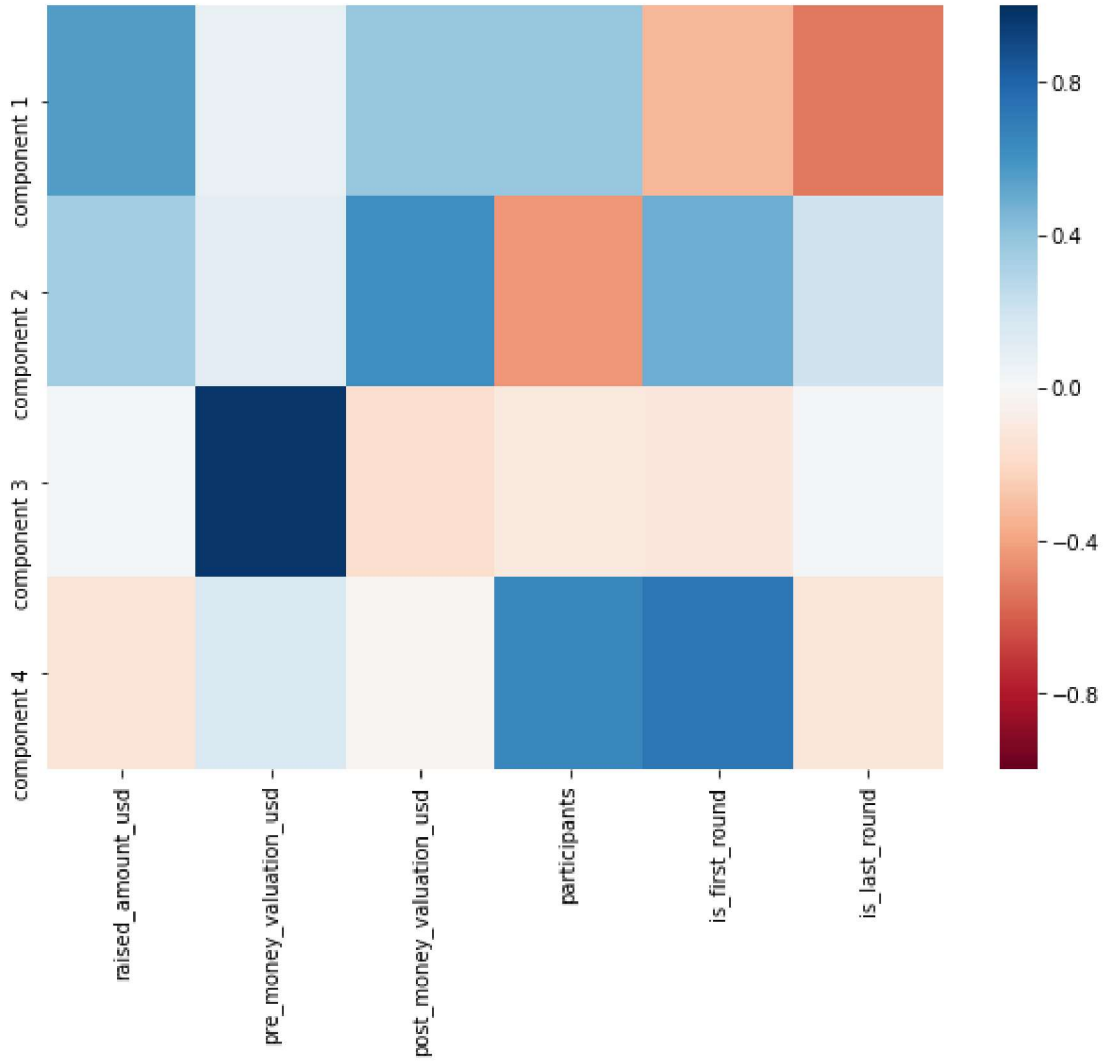
In [49]:
```
pca.components_
```

Out[49]:
```
array([[ 0.5602702 ,  0.07114797,  0.38253944,  0.37883229, -0.33409964,
        -0.52873659],
       [ 0.35105715,  0.10492922,  0.62180842, -0.44024852,  0.49096589,
         0.21032538],
       [ 0.01776182,  0.9744178 , -0.17027974, -0.09996959, -0.10380917,
         0.0207125 ],
       [-0.12629228,  0.14478688, -0.0229149 ,  0.65053163,  0.72414697,
        -0.12239972]])
```

In [50]:
```
p_s_c_std_mrg=pd.DataFrame(pca.components_,columns=c_std_mrg.columns,index=[
```

In [51]:
```
plt.figure(figsize=(10,7))
sns.heatmap(p_s_c_std_mrg,vmin=-1,vmax=1,cmap='RdBu')
```

Out[51]: <matplotlib.axes._subplots.AxesSubplot at 0x19ba9916148>



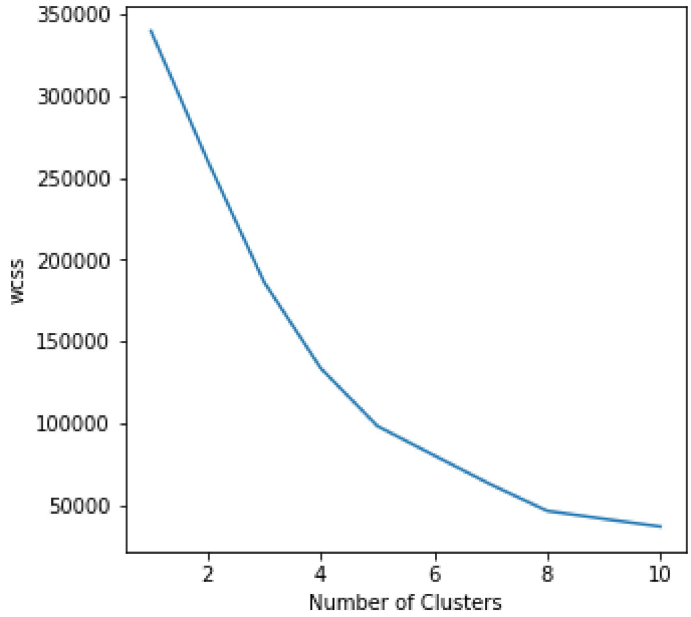
In [52]:
```
# keep note of the prefix|| 'pca'=prinicipal components
pca_s_c_std_mrg=pca.transform(s_c_std_mrg)
```

In [53]:
```
pca_s_c_std_mrg=pd.DataFrame(pca_s_c_std_mrg)
```

In [54]:
```
wcss =[]
for i in range(1,11):
        kmeans=KMeans(n_clusters=i,init='k-means++',random_state=101)
        kmeans.fit(pca_s_c_std_mrg)
        wcss.append(kmeans.inertia_)
```

In [55]:
```
plt.figure(figsize=(5,5))
plt.plot(range(1,11),wcss)
plt.xlabel('Number of Clusters')
plt.ylabel('wcss')
```

Out[55]: Text(0, 0.5, 'wcss')



In [56]:
```
kmeans=KMeans(n_clusters=8,init='k-means++',random_state=101)
```

In [57]:
```
kmeans.fit(pca_s_c_std_mrg)
```

Out[57]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=8, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=101, tol=0.0001, verbose=0)
```

In [58]:
```
pca_s_c_std_mrg_kmeans=pca_s_c_std_mrg.copy()
pca_s_c_std_mrg_kmeans['Segment Kmeans']=kmeans.labels_
```

In [59]:
```
pca_s_c_std_mrg_kmeans.sample(3)
```

Out[59]:

	0	1	2	3	Segment Kmeans
23348	-0.280577	-0.402150	0.140112	-0.946777	0
35864	-0.083065	0.413978	-0.055056	0.395888	1
6129	-0.430108	-0.310920	0.165931	-1.113906	0

In [60]:
```
1 pca_s_c_std_mrg_kmeans.describe().T
```

Out[60]:

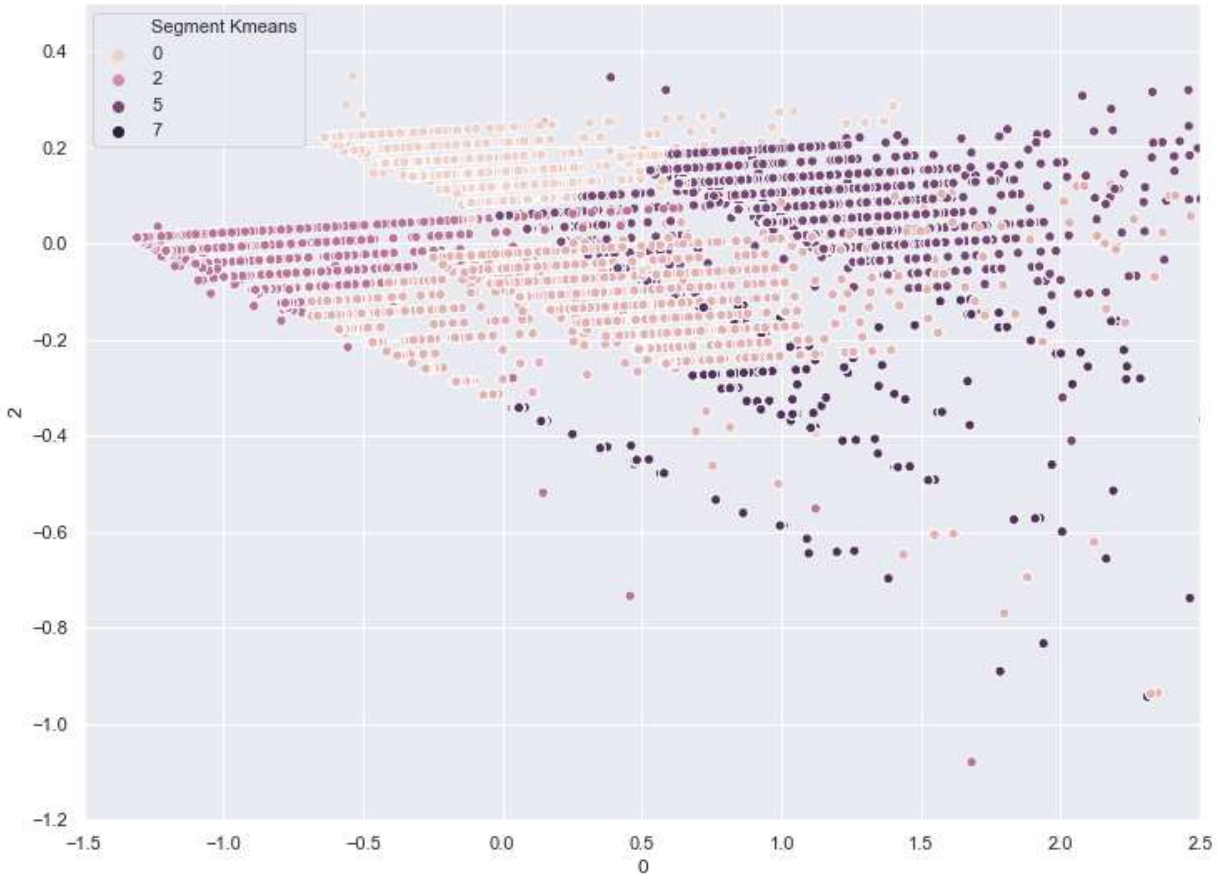
	count	mean	std	min	25%	50%	75%	max
0	80902.0	-5.048906e-15	1.095025	-1.308661	-0.667997	-0.050492	0.621293	43.208095
1	80902.0	2.054257e-15	1.024960	-3.293282	-0.551210	-0.033958	0.558263	46.332559
2	80902.0	-9.684204e-16	1.000496	-11.892404	-0.082983	-0.003819	0.119058	276.113641
3	80902.0	4.001044e-15	0.975932	-10.548860	-0.822507	-0.012903	0.573039	39.027373
Segment Kmeans	80902.0	2.460619e+00	1.971422	0.000000	1.000000	3.000000	5.000000	7.000000

Graphs || Visualizations

In [61]:
```
sns.set(style='darkgrid')
# sns.set_palette('rocket')
sns.set_color_codes(palette='deep')

plt.figure(figsize=(12,9))
g=sns.scatterplot(pca_s_c_std_mrg_kmeans[0],pca_s_c_std_mrg_kmeans[2],hue=pc
g.set_xlim(-1.5,2.5)
g.set_ylim(-1.2,0.5)
```

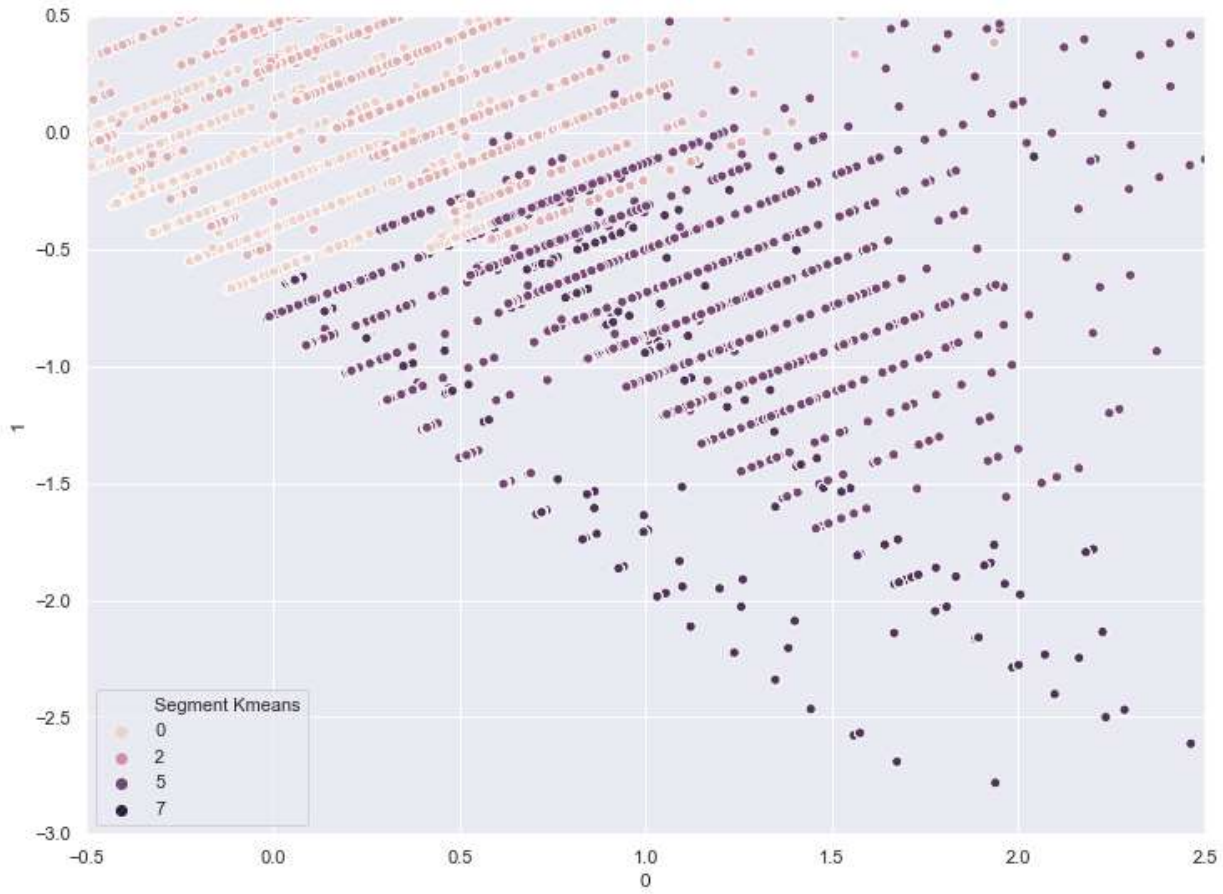
Out[61]: (-1.2, 0.5)



In [62]:
```
sns.set(style='darkgrid')
# sns.set_palette('rocket')
sns.set_color_codes(palette='deep')

plt.figure(figsize=(12,9))
g=sns.scatterplot(pca_s_c_std_mrg_kmeans[0],pca_s_c_std_mrg_kmeans[1],hue=pc
g.set_xlim(-0.5,2.5)
g.set_ylim(-3,0.5)
```

Out[62]: (-3.0, 0.5)



DENSITY BASED SPATIAL CLUSTERING OF APPLICATIONS WITH NOISE

In [76]:
```
from sklearn.cluster import DBSCAN
```

In [77]:
```
dbscan=DBSCAN(eps=3,min_samples=20000)
```

In [83]:
```
dbscan.fit(s_c_std_mrg)
```

In [84]:
```
dbscan.labels_
```

In [85]:
```
n_clustersv=len(set(dbscan.labels_)) - (1 if -1 in dbscan.labels_ else 0)
n_noise=list(dbscan.labels_).count(-1)
```

In [86]:
```
n_clusters
```

In [87]:
```
n_noise
```

Pickle Files

In [78]:
```
import pickle
```

In [83]:
```
pickle.dump(scalar, open('scalar.pickle','wb'))
```

In [84]:
```
pickle.dump(pca, open('pca.pickle','wb'))
```

In [85]:
```
pickle.dump(kmeans, open('kmeans_pca.pickle','wb'))
```

Data Exports

In [87]:
```
inv_funrnd.to_csv(r'C:\Users\m_joekid\Desktop\Startup Success\'inv_fund.csv'
```

In [88]:
```
std_mrg.to_csv(r'C:\Users\m_joekid\Desktop\Startup Success\'std_mrg.csv')
```

In [91]:
```
pca_s_c_std_mrg_kmeans.to_csv(r'C:\Users\m_joekid\Desktop\Startup Success\'c
```

In [90]:
```
item_anal_1.to_csv(r'C:\Users\m_joekid\Desktop\Startup Success\'item_anal_1.
item_anal_2.to_csv(r'C:\Users\m_joekid\Desktop\Startup Success\'item_anal_2.
```

In [ ]:
```

```