In [1]:
```
import time
start_time = time.time()
```

In [2]:
```
import numpy as np
import pandas as pd
import os, time, pickle, gzip

import matplotlib.pyplot as plt
import seaborn as sns
color = sns.color_palette()
import matplotlib as mpl

from sklearn import preprocessing as pp

%matplotlib inline
```

자료읽기

In [3]:
```
TRX = pd.read_csv('C:\WORK/mnistTRX.csv')
VLX = pd.read_csv('C:\WORK/mnistVLX.csv')
TSX = pd.read_csv('C:\WORK/mnistTSX.csv')

TRy = pd.read_csv('C:\WORK/mnistTRy.csv')['y']
VLy = pd.read_csv('C:\WORK/mnistVLy.csv')['y']
TSy = pd.read_csv('C:\WORK/mnistTSy.csv')['y']

iTR = range(0, len(TRX))
iVL = range(len(TRX), len(TRX)+len(VLX))
iTS = range(len(TRX)+len(VLX), len(TRX)+len(VLX)+len(TSX))

nX = [TRX.shape, VLX.shape, TSX.shape]
ny = [TRy.shape, VLy.shape, TSy.shape]
pd.DataFrame(nX, index=['TRX', 'VLX', 'TSX'], columns=['n', 'p'])
```

Out[3]:

	n	p
TRX	50000	784
VLX	10000	784
TSX	10000	784

In [4]:
```
pd.DataFrame(ny, index=['TRy', 'VLy', 'TSy'], columns=['n'])
```

Out[4]:

	n
TRy	50000
VLy	10000
TSy	10000

In [5]:
```
TRX.describe()
```

Out[5]:

	x01	x02	x03	x04	x05	x06	x07	x08	x09	x10	...	x775	x776	x777
count	50000.0	50000.0	50000.0	50000.0	50000.0	50000.0	50000.0	50000.0	50000.0	50000.0	...	50000.000000	50000.000000	50000.000000
mean	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000739	0.000354	0.000204
std	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.022784	0.015424	0.012080
min	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.000000	0.000000
25%	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.000000	0.000000
50%	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.000000	0.000000
75%	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.000000	0.000000
max	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.992188	0.992188	0.988281

8 rows × 784 columns

In [6]:
```
TRy.head()
```

Out[6]:
```
0    5
1    0
2    4
3    1
4    9
Name: y, dtype: int64
```

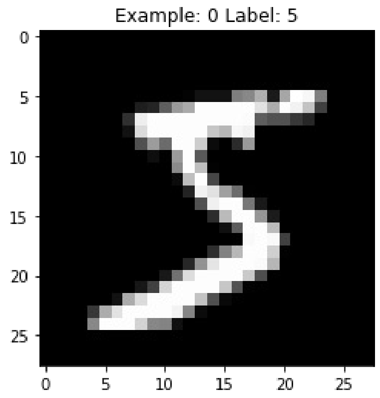
In [7]:
```
TRy.head()
```

Out[7]:
```
0    5
1    0
2    4
3    1
4    9
Name: y, dtype: int64
```

개별 이미지 시각화

In [8]:
```
def view_digit(example):
    label = TRy.loc[example]
    image = TRX.loc[example, :].values.reshape([28, 28])
    plt.title('Example: %d Label: %d' % (example,label))
    plt.imshow(image, cmap=plt.get_cmap('gray'))
    plt.show()
```

In [9]:
```
view_digit(0)
```



레이블 조정하기

In [10]:
```
from sklearn.preprocessing import OneHotEncoder, LabelBinarizer
LB = LabelBinarizer()
TRY = LB.fit_transform(TRy)
TRY[0]
```

Out[10]:
```
array([0, 0, 0, 0, 0, 1, 0, 0, 0, 0])
```

층화추출한 자료

In [11]:
```
TRX0 = TRX.groupby(TRy).apply(lambda x: x.sample(100, random_state=2018))
iTR0 = TRX0.index.to_frame()[1]
TRX0.set_index(iTR0, inplace=True)
TRy0 = TRy.iloc[iTR0]
```

PCA

모델 적합/성분계산

In [12]:
```python
from sklearn.decomposition import PCA

n_components = 784
whiten = False
random_state = 2018

Epca = PCA(n_components=n_components,
           whiten = whiten,
           random_state=random_state)
TRXTpca = Epca.fit_transform(TRX)
TRXTpca = pd.DataFrame(data=TRXTpca, index=iTR)
```

In [13]:
```python
TRXTpca.corr().round(5)
```

Out[13]:

	0	1	2	3	4	5	6	7	8	9	...	774	775	776	777	778	779	780	781	7
0	1.0	-0.0	0.0	-0.0	0.0	0.0	-0.0	0.0	-0.0	0.0	...	-0.00000	-0.00000	0.00000	0.00000	-0.00000	0.00000	0.00000	0.00000	0.000
1	-0.0	1.0	-0.0	-0.0	-0.0	-0.0	0.0	0.0	0.0	0.0	...	0.00000	0.00000	0.00000	0.00000	-0.00000	-0.00000	-0.00000	-0.00000	-0.000
2	0.0	-0.0	1.0	0.0	-0.0	-0.0	-0.0	0.0	-0.0	0.0	...	0.00000	-0.00000	0.00000	-0.00000	-0.00000	-0.00000	-0.00000	-0.00000	-0.000
3	-0.0	-0.0	0.0	1.0	-0.0	0.0	-0.0	0.0	0.0	0.0	...	-0.00000	-0.00000	-0.00000	0.00000	0.00000	-0.00000	-0.00000	-0.00000	0.000
4	0.0	-0.0	-0.0	-0.0	1.0	0.0	-0.0	0.0	0.0	0.0	...	0.00000	-0.00000	-0.00000	0.00000	-0.00000	0.00000	0.00000	0.00000	0.000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
779	0.0	-0.0	-0.0	-0.0	0.0	0.0	0.0	0.0	0.0	-0.0	...	0.00004	0.00003	0.00003	0.00003	0.00003	1.00000	-0.00002	0.00004	0.000
780	0.0	-0.0	-0.0	-0.0	0.0	-0.0	0.0	0.0	0.0	-0.0	...	0.00001	0.00001	0.00001	0.00001	0.00001	-0.00002	1.00000	0.00001	0.000
781	0.0	-0.0	-0.0	-0.0	0.0	-0.0	0.0	0.0	0.0	-0.0	...	-0.00002	-0.00002	-0.00002	-0.00002	-0.00002	0.00004	0.00001	1.00000	-0.000
782	0.0	-0.0	-0.0	0.0	0.0	-0.0	-0.0	-0.0	-0.0	-0.0	...	-0.00001	-0.00001	-0.00001	-0.00001	-0.00001	0.00002	0.00001	-0.00001	1.000
783	-0.0	0.0	0.0	0.0	-0.0	0.0	0.0	0.0	0.0	0.0	...	0.00002	0.00002	0.00002	0.00002	0.00002	-0.00003	-0.00001	0.00002	0.000

784 rows × 784 columns

In [14]:
```python
TRXTpca.var().round(5)
```

Out[14]:
```
0      5.10829
1      3.70098
2      3.25868
3      2.82008
4      2.54673
        ...
779    0.00000
780    0.00000
781    0.00000
782    0.00000
783    0.00000
Length: 784, dtype: float64
```

In [15]:
```python
TRXTpca.cov().round(5)
```

Out[15]:

	0	1	2	3	4	5	6	7	8	9	...	774	775	776	777	778	779	780	781	782	783
0	5.10829	-0.00000	0.00000	-0.00000	-0.00000	0.0	-0.0	0.0	-0.0	0.0	...	-0.0	0.0	-0.0	-0.0	-0.0	0.0	0.0	0.0	0.0	-0.0
1	-0.00000	3.70098	-0.00000	-0.00000	-0.00000	-0.0	0.0	-0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	-0.0	-0.0	-0.0	-0.0	-0.0	0.0
2	0.00000	-0.00000	3.25868	0.00000	-0.00000	-0.0	-0.0	0.0	0.0	0.0	...	0.0	-0.0	0.0	-0.0	-0.0	-0.0	-0.0	-0.0	-0.0	0.0
3	-0.00000	-0.00000	0.00000	2.82008	-0.00000	0.0	-0.0	0.0	0.0	0.0	...	-0.0	-0.0	0.0	-0.0	0.0	-0.0	-0.0	-0.0	0.0	0.0
4	-0.00000	-0.00000	-0.00000	-0.00000	2.54673	0.0	-0.0	0.0	0.0	0.0	...	-0.0	-0.0	0.0	-0.0	0.0	0.0	0.0	0.0	0.0	-0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
779	0.00000	-0.00000	-0.00000	-0.00000	0.00000	0.0	0.0	0.0	0.0	-0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	-0.0	0.0	0.0	-0.0
780	0.00000	-0.00000	-0.00000	-0.00000	0.00000	-0.0	0.0	0.0	0.0	-0.0	...	0.0	0.0	0.0	0.0	0.0	-0.0	0.0	0.0	0.0	-0.0
781	0.00000	-0.00000	-0.00000	-0.00000	0.00000	-0.0	0.0	0.0	0.0	-0.0	...	-0.0	-0.0	-0.0	-0.0	-0.0	0.0	0.0	0.0	-0.0	0.0
782	0.00000	-0.00000	-0.00000	0.00000	0.00000	-0.0	-0.0	-0.0	-0.0	-0.0	...	-0.0	-0.0	-0.0	-0.0	-0.0	0.0	0.0	-0.0	0.0	0.0
783	-0.00000	0.00000	0.00000	0.00000	-0.00000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	-0.0	-0.0	0.0	0.0	0.0

784 rows × 784 columns

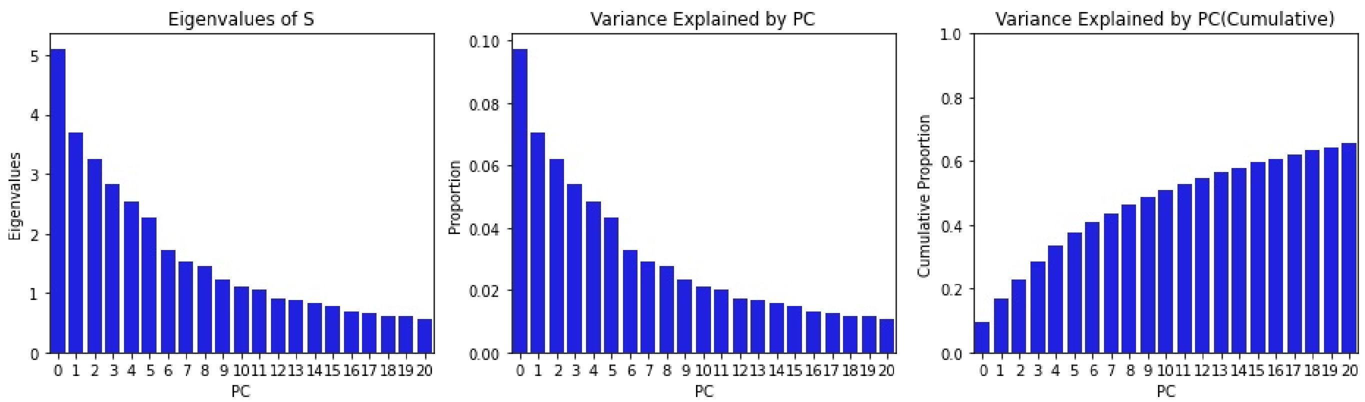
모델 설명력

In [16]:
```python
def summaryPCA(PCAobj):
    eigval = pd.DataFrame(PCAobj.explained_variance_)
    prop = pd.DataFrame(PCAobj.explained_variance_ratio_)
    cumul = pd.DataFrame(np.cumsum(prop)/prop.sum())
    vrexp = pd.concat([eigval, prop, cumul], axis=1).T
    vrexp.index = ['Eigenvalue', 'Prop', 'Cumulative']
    return vrexp
vrexp = summaryPCA(Epca)
vrexp.loc[:, [0, 1, 2, 9, 19, 49, 99, 199, 299, 399, 499, 599, 699]].round(4)
```

Out[16]:

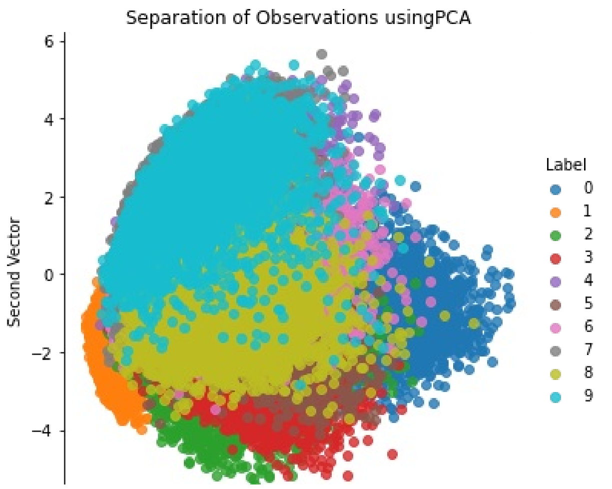
	0	1	2	9	19	49	99	199	299	399	499	599	699
Eigenvalue	5.1083	3.7010	3.2587	1.2403	0.6034	0.1683	0.0528	0.0148	0.0071	0.0031	0.0008	0.0001	0.0
Prop	0.0974	0.0706	0.0622	0.0237	0.0115	0.0032	0.0010	0.0003	0.0001	0.0001	0.0000	0.0000	0.0
Cumulative	0.0974	0.1680	0.2302	0.4888	0.6440	0.8249	0.9147	0.9665	0.9862	0.9958	0.9993	1.0000	1.0

In [17]:
```python
fig, ax = plt.subplots(ncols=3, figsize=(16, 4))

ev = sns.barplot(data=pd.DataFrame(vrexp.loc['Eigenvalue', :20]).T, color='b', ax=ax[0]);
ev.set(title='Eigenvalues of S', xlabel='PC', ylabel='Eigenvalues')
pr = sns.barplot(data=pd.DataFrame(vrexp.loc['Prop', :20]).T, color='b', ax = ax[1]);
pr.set(title='Variance Explained by PC', xlabel='PC', ylabel='Proportion')
cm = sns.barplot(data=pd.DataFrame(vrexp.loc['Cumulative', :20]).T, color='b', ax=ax[2]);
cm.set(title='Variance Explained by PC(Cumulative)', xlabel='PC', ylabel='Cumulative Proportion', ylim=[0, 1]);
```



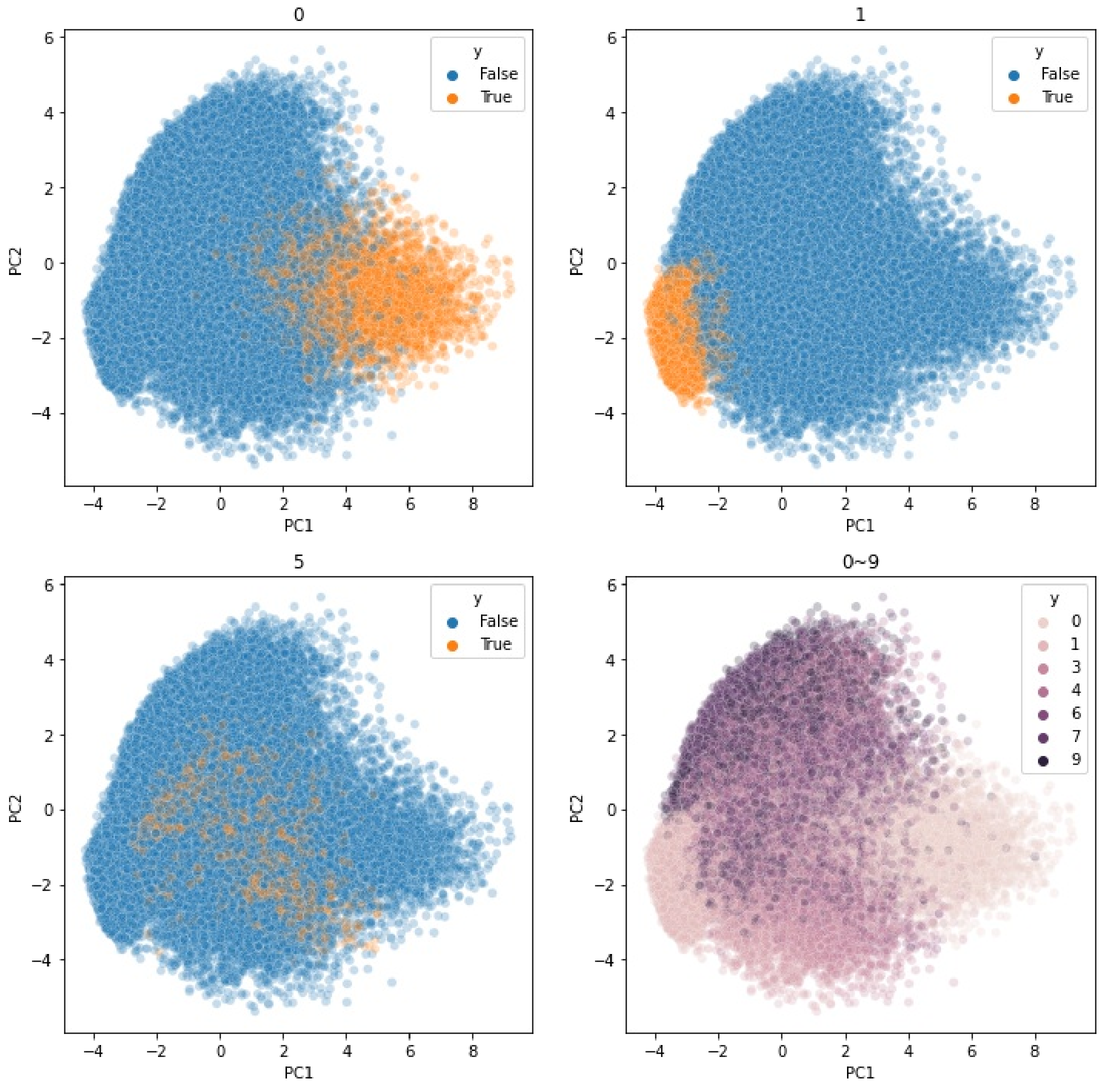
In [18]:
```python
def scatterPlot(xDF, yDF, algoName):
    tempDF = pd.DataFrame(data=xDF.loc[:, 0:1], index=xDF.index)
    tempDF = pd.concat((tempDF, yDF), axis=1, join="inner")
    tempDF.columns = ["First Vector", "Second Vector", "Label"]
    sns.lmplot(x="First Vector", y="Second Vector", hue="Label", data=tempDF, fit_reg=False)
    ax = plt.gca()
    ax.set_title("Separation of Observations using"+algoName)
```

In [19]:
```python
scatterPlot(TRXTpca, TRy, 'PCA')
```



In [20]:
```
fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(12,12))
d0 = sns.scatterplot(x=TRXTpca.loc[:, 0], y=TRXTpca.loc[:, 1], hue=TRy==0, alpha=0.25, ax=axs[0,0]);
d0.set(xlabel='PC1', ylabel='PC2', title='0')
d1 = sns.scatterplot(x=TRXTpca.loc[:, 0], y=TRXTpca.loc[:, 1], hue=TRy==1, alpha=0.25, ax=axs[0,1]);
d1.set(xlabel='PC1', ylabel='PC2', title='1')
d5 = sns.scatterplot(x=TRXTpca.loc[:, 0], y=TRXTpca.loc[:, 1], hue=TRy==5, alpha=0.25, ax=axs[1,0]);
d5.set(xlabel='PC1', ylabel='PC2', title='5')
dd = sns.scatterplot(x=TRXTpca.loc[:, 0], y=TRXTpca.loc[:, 1], hue=TRy, alpha=0.25, ax=axs[1,1]);
dd.set(xlabel='PC1', ylabel='PC2', title='0~9')
```

Out[20]:
```
[Text(0.5, 0, 'PC1'), Text(0, 0.5, 'PC2'), Text(0.5, 1.0, '0~9')]
```



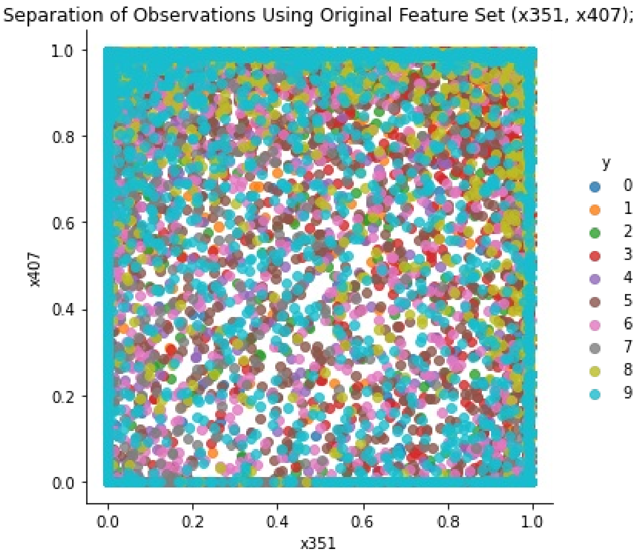
In [21]:
```
TRX.iloc[:, [350,406]].head()
```

Out[21]:

	x351	x407
0	0.273438	0.937500
1	0.000000	0.000000
2	0.000000	0.941406
3	0.800781	0.980469
4	0.031250	0.988281

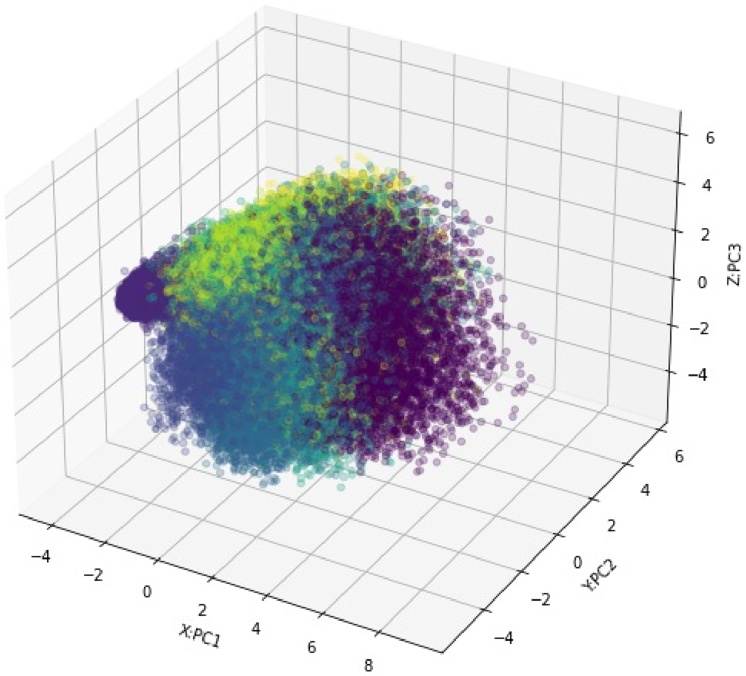
In [22]:
```
XX = pd.DataFrame(data=TRX.iloc[:, [350,406]], index=iTR)
XX = pd.concat([XX, TRy], axis=1, join='inner')
sns.lmplot(x='x351', y='x407', hue='y', data=XX, fit_reg=False)
ax = plt.gca()
ax.set_title('Separation of Observations Using Original Feature Set (x351, x407);')
```

Out[22]: Text(0.5, 1.0, 'Separation of Observations Using Original Feature Set (x351, x407);')



In [23]:
```
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure(figsize=(16,9))
ax = plt.axes(projection='3d')
p = ax.scatter3D(TRXTpca[0], TRXTpca[1], TRXTpca[2], alpha=0.25, c=TRy)
ax.set_xlabel('X:PC1')
ax.set_ylabel('Y:PC2')
ax.set_zlabel('Z:PC3')
```

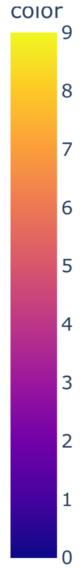
Out[23]: Text(0.5, 0, 'Z:PC3')



In [24]:
```
import plotly.express as px
```

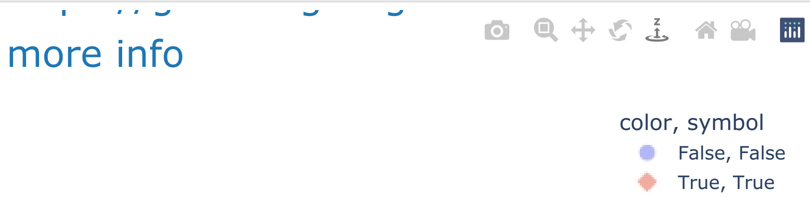
In [25]:
```
fig = px.scatter_3d(TRXTpca, x=0, y=1, z=2, color=TRy, size_max=10)
fig.show()
```

more info



In [26]:

```
TRXTO = TRXTpca.groupby(TRy).apply(lambda x :x.sample(100, random_state=2018))
iTRO = TRXTO.index.to_frame()[1]
TRXTO.set_index(iTRO, inplace=True)
TRy0 = TRy.iloc[iTRO]

fig = px.scatter_3d(TRXTO, x=0, y=1, z=2,
                    color = TRy0==1,
                    symbol=TRy0==1,
                    size=1+(TRy0==1).astype(int),
                    size_max=9,
                    opacity=0.5)
fig.show()
```



차원축소 Tuning

In [27]:

```
from sklearn.model_selection import KFold, StratifiedKFold, GridSearchCV
from sklearn.metrics import mean_squared_error
```

```
KF5 = KFold(n_splits=5, shuffle=True, random_state=2018)

def negmse(estimator, X, y=None):
    XT = estimator.transform(X)
    Xh = estimator.inverse_transform(XT)
    return -1* mean_squared_error(X, Xh)
```

In [28]:

```
parampca = {'n_components': [100, 200, 300, 400, 500]}

Epca2 = PCA()
GSpca = GridSearchCV(Epca2, param_grid=parampca, cv=KF5, scoring=negmse)
%time GSpca.fit(TRX)
GSpca.score(TRX)
```

```
Wall time: 3min 37s
```

Out[28]:

```
-5.0286759245809235e-05
```

In [29]:

```
cvresult = pd.DataFrame(GSpca.cv_results_)
cvresult
```

Out[29]:

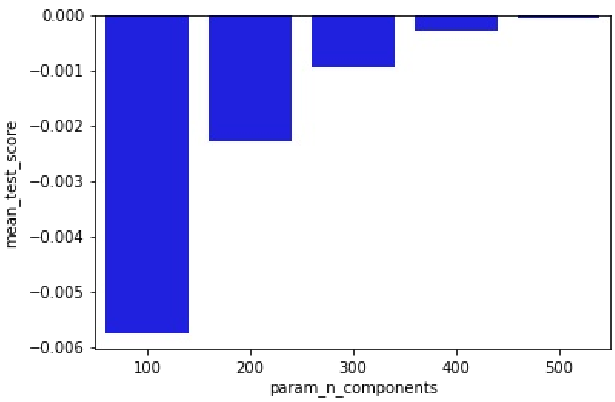
	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_components	params	split0_test_score	split1_test_score	spli
0	4.002801	1.024797	0.395897	0.136476	100	{'n_components': 100}	-0.005773	-0.005762	
1	4.708631	0.249394	0.301810	0.012087	200	{'n_components': 200}	-0.002301	-0.002288	
2	6.910282	0.320145	0.350451	0.016190	300	{'n_components': 300}	-0.000956	-0.000953	
3	9.537575	0.248477	0.423061	0.034302	400	{'n_components': 400}	-0.000292	-0.000290	
4	13.013720	0.612896	0.442413	0.045059	500	{'n_components': 500}	-0.000054	-0.000053	

In [30]:

```
sns.barplot(x='param_n_components', y='mean_test_score', data=cvresult, color='b')
```

Out[30]:

```
<AxesSubplot:xlabel='param_n_components', ylabel='mean_test_score'>
```



In [31]:

```
print(GSpca.best_score_.round(4))
print(GSpca.best_params_)
print(GSpca.best_estimator_)
```

```
-0.0001
{'n_components': 500}
PCA(n_components=500)
```

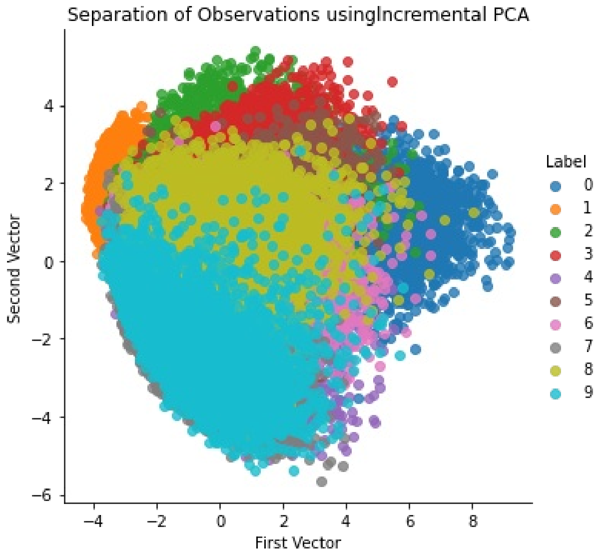
Incremental PCA

In [32]:

```
from sklearn.decomposition import IncrementalPCA
```

```
n_components = 784
batch_size = None

Eipca = IncrementalPCA(n_components=n_components,
                       batch_size=batch_size)

%time TRXTipca = Eipca.fit_transform(TRX)
TRXTipca = pd.DataFrame(data=TRXTipca, index=iTR)

VLXTipca = Eipca.transform(VLX)
VLXTIPCA = pd.DataFrame(data=VLXTipca, index=iVL)

print(TRXTipca.shape)
scatterPlot(TRXTipca, TRy, 'Incremental PCA')
```

```
Wall time: 9.83 s
(50000, 784)
```

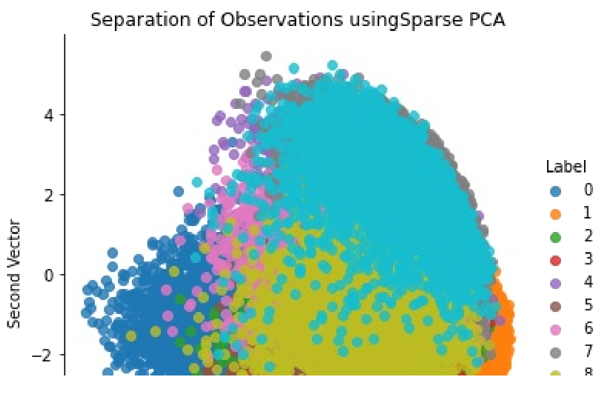


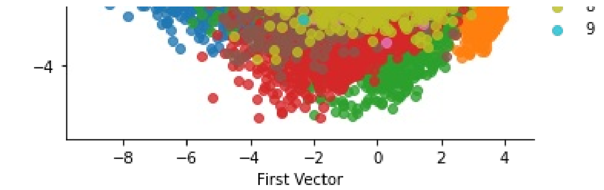
Sparse PCA

In [33]:

```
from sklearn.decomposition import SparsePCA

n_components = 100
alpha = 0.0001
random_state = 2018
n_jobs = -1

Espca = SparsePCA(n_components = n_components,
                  alpha = alpha,
                  random_state = random_state,
                  n_jobs = n_jobs)
%time Espca.fit(TRX.loc[:10000, :])
%time TRXTspca = Espca.transform(TRX)
TRXTspca = pd.DataFrame(data=TRXTspca, index=iTR)

VLXTspca = Espca.transform(VLX)
VLXTspca = pd.DataFrame(data=VLXTspca, index=iVL)

print(TRXTspca.shape)
scatterPlot(TRXTspca, TRy, 'Sparse PCA')
```

```
Wall time: 31.1 s
Wall time: 499 ms
(50000, 100)
```



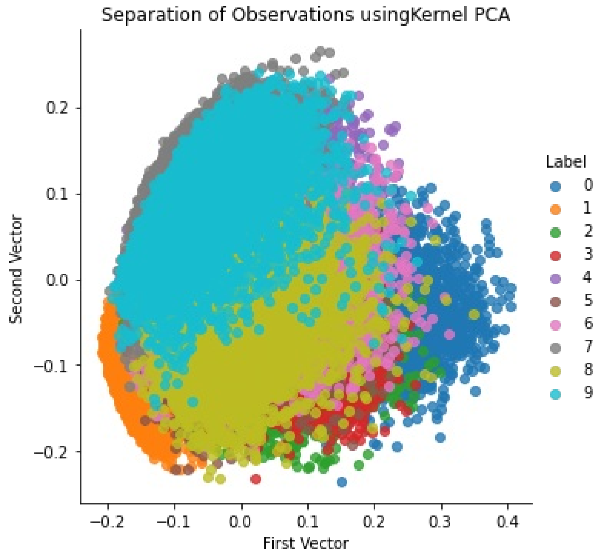


Kernel PCA

In [34]:

```
from sklearn.decomposition import KernelPCA

n_components = 100
kernel = 'rbf'
gamma = None
random_state = 2018
n_jobs = 1

Ekpca = KernelPCA(n_components=n_components,
                  kernel=kernel,
                  gamma=gamma,
                  n_jobs=n_jobs,
                  random_state=random_state)
%time Ekpca.fit(TRX0)
%time TRXTkpca = Ekpca.transform(TRX)
TRXTkpca = pd.DataFrame(data=TRXTkpca, index=iTR)

VLXTkpca = Ekpca.transform(VLX)
VLXTkpca = pd.DataFrame(data=VLXTkpca, index=iVL)

print(TRXTkpca.shape)
scatterPlot(TRXTkpca, TRy, 'Kernel PCA')
```

```
Wall time: 218 ms
Wall time: 2.87 s
(50000, 100)
```



In [35]:

```
paramkpca = {
    'gamma':np.linspace(0.01, 0.05, 5),
    'kernel': ['rbf', 'linear']
}
Ekpca2 = KernelPCA(n_components=100, fit_inverse_transform=True, n_jobs=n_jobs, random_state=random_state)
GSkpca = GridSearchCV(Ekpca2, param_grid=paramkpca, cv=3, scoring=negmse)
%time GSkpca.fit(TRX0)
GSkpca.score(TRX0)
```

```
Wall time: 4.68 s
```

Out[35]:

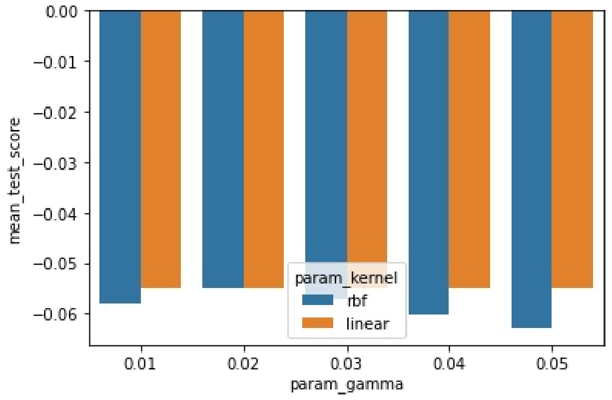
```
-0.05083305496936513
```

```
cvresult = pd.DataFrame(GSkpca.cv_results_)
cvresult
```

Out[36]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_gamma	param_kernel	params	split0_test_score	split1_test_score	sp
0	0.115216	0.004526	0.045042	0.004480	0.01	rbf	{'gamma': 0.01, 'kernel': 'rbf'}	-0.062472	-0.056276	
1	0.099569	0.002432	0.035241	0.000939	0.01	linear	{'gamma': 0.01, 'kernel': 'linear'}	-0.055510	-0.055971	
2	0.115672	0.005090	0.042455	0.001034	0.02	rbf	{'gamma': 0.02, 'kernel': 'rbf'}	-0.059291	-0.053214	
3	0.095320	0.000929	0.033658	0.001276	0.02	linear	{'gamma': 0.02, 'kernel': 'linear'}	-0.055510	-0.055971	
4	0.113283	0.001937	0.046228	0.007153	0.03	rbf	{'gamma': 0.03, 'kernel': 'rbf'}	-0.061458	-0.055114	
5	0.106218	0.006000	0.035902	0.002726	0.03	linear	{'gamma': 0.03, 'kernel': 'linear'}	-0.055510	-0.055971	
6	0.113543	0.002017	0.042211	0.001231	0.04	rbf	{'gamma': 0.04, 'kernel': 'rbf'}	-0.064699	-0.058224	
7	0.098913	0.002717	0.034059	0.000881	0.04	linear	{'gamma': 0.04, 'kernel': 'linear'}	-0.055510	-0.055971	
8	0.114649	0.008092	0.043688	0.003793	0.05	rbf	{'gamma': 0.05, 'kernel': 'rbf'}	-0.067688	-0.061154	
9	0.102030	0.008127	0.040293	0.003676	0.05	linear	{'gamma': 0.05, 'kernel': 'linear'}	-0.055510	-0.055971	

In [37]:
```
sns.barplot(x='param_gamma', y='mean_test_score', hue='param_kernel', data=cvresult);
```



In [38]:
```
print(GSkpca.best_score_.round(4))
print(GSkpca.best_params_)
print(GSkpca.best_estimator_)
```

```
-0.055
{'gamma': 0.01, 'kernel': 'linear'}
KernelPCA(fit_inverse_transform=True, gamma=0.01, n_components=100, n_jobs=1,
          random_state=2018)
```

In [39]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.pipeline import Pipeline
clf = Pipeline([('kpca', KernelPCA()),
               ('glm', LogisticRegression())])

SKF5 = StratifiedKFold(n_splits=5, shuffle=True, random_state=2018)
param_grid = {
    'kpca__n_components':[100, 200],
    'kpca__gamma': np.linspace(0.01, 0.05, 5),
    'kpca__kernel': ['rbf', 'linear', 'sigmoid']
}

grid_search = GridSearchCV(clf, param_grid, cv=SKF5)
%time grid_search.fit(TRX0, TRy0)
print(grid_search.best_score_)
print(grid_search.best_params_)
```

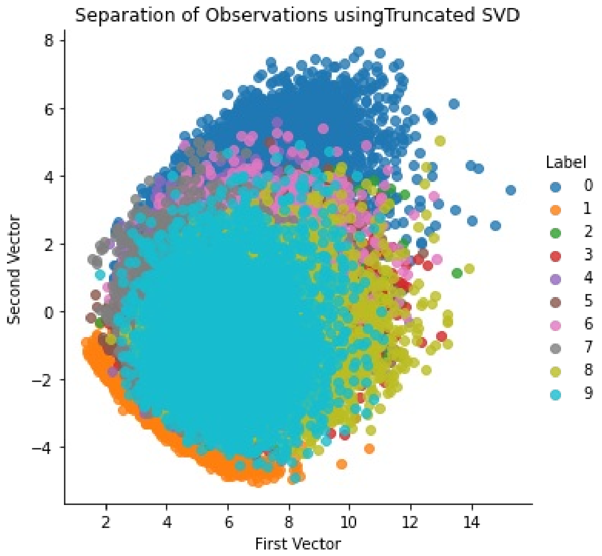
```
Wall time: 33.8 s
0.873
{'kpca__gamma': 0.03, 'kpca__kernel': 'rbf', 'kpca__n_components': 200}
```

tuencatedSVD

In [40]:

```
from sklearn.decomposition import TruncatedSVD

n_components = 200
algorithm = 'randomized'
n_iter = 5
random_state = 2018

Etsvd = TruncatedSVD(n_components = n_components,
                    algorithm = algorithm,
                    n_iter = n_iter,
                    random_state = random_state)

%time TRXTtsvd = Etsvd.fit_transform(TRX)
TRXTtsvd = pd.DataFrame(data=TRXTtsvd, index=iTR)

VLXTtsvd = Etsvd.transform(VLX)
VLXTtsvd = pd.DataFrame(data=VLXTtsvd, index=iVL)

print(TRXTtsvd.shape)
scatterPlot(TRXTtsvd, TRy, 'Truncated SVD')
```

```
Wall time: 6.92 s
(50000, 200)
```



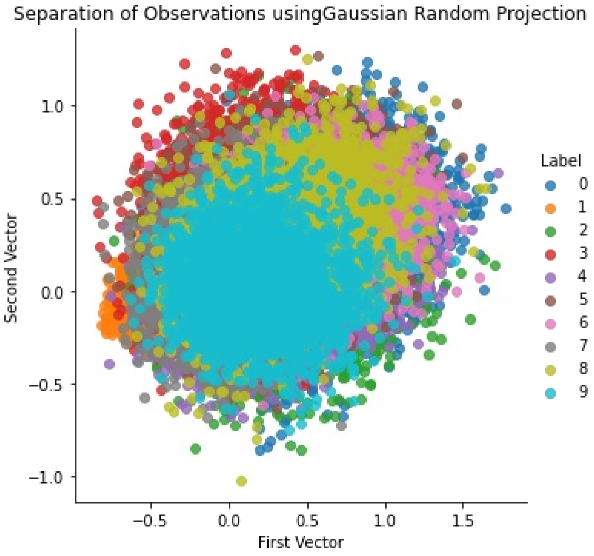
GRP

In [41]:

```
from sklearn.random_projection import GaussianRandomProjection
n_components = 'auto'
eps = 0.5
random_state = 2018

Egrp = GaussianRandomProjection(n_components=n_components,
```

```
                                eps=eps,
                                random_state=random_state)

%time TRXTgrp = Egrp.fit_transform(TRX)
TRXTgrp = pd.DataFrame(data=TRXTgrp, index=iTR)

VLXTgrp = Egrp.transform(VLX)
VLXTgrp = pd.DataFrame(data=VLXTgrp, index=iVL)

print(TRXTgrp.shape)
scatterPlot(TRXTgrp, TRy, 'Gaussian Random Projection')
```

```
Wall time: 670 ms
(50000, 519)
```

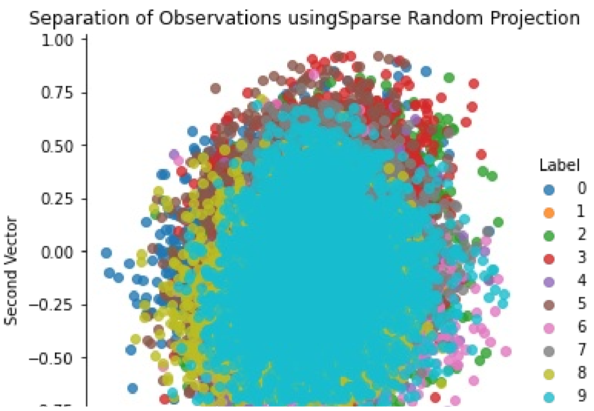


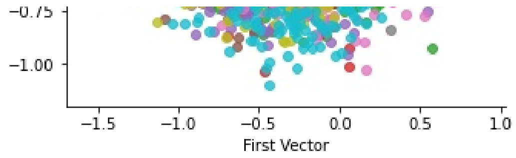
Sparse Random Projection

In [42]:

```
from sklearn.random_projection import SparseRandomProjection

n_components = 'auto'
density = 'auto'
eps = 0.5
dense_output = False
random_state = 2018
Esrp = SparseRandomProjection(n_components = n_components,
                              density = density,
                              eps = eps,
                              dense_output = dense_output,
                              random_state=random_state)

%time TRXTsrp = Esrp.fit_transform(TRX)
TRXTsrp = pd.DataFrame(data=TRXTsrp, index=iTR)

VLXTsrp = Esrp.transform(VLX)
VLXTsrp = pd.DataFrame(data=VLXTsrp, index=iVL)

print(TRXTsrp.shape)
scatterPlot(TRXTsrp, TRy, 'Sparse Random Projection')
```

```
Wall time: 692 ms
(50000, 519)
```

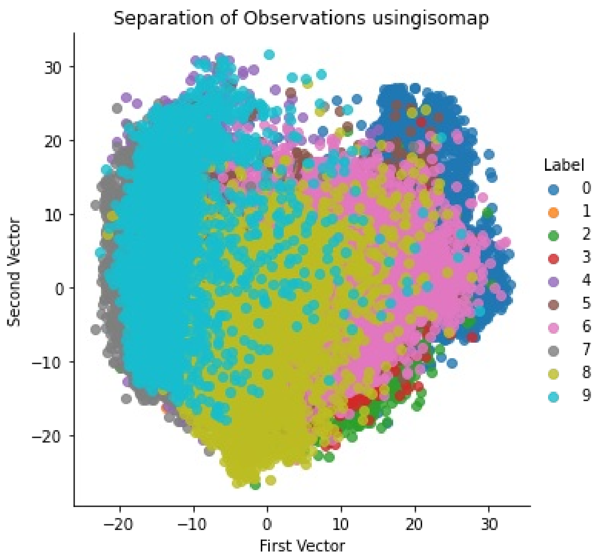




Isomap

In [43]:
```
from sklearn.manifold import Isomap
n_neighbors = 5
n_components = 10
n_jobs = 4

Eimap = Isomap(n_neighbors = n_neighbors,
              n_components = n_components,
              n_jobs=n_jobs)

%time Eimap.fit(TRX0)
%time TRXTimap = Eimap.transform(TRX)
TRXTimap = pd.DataFrame(data=TRXTimap, index=iTR)

VLXTimap = Eimap.transform(VLX)
VLXTimap = pd.DataFrame(data=VLXTimap, index=iVL)

print(TRXTimap.shape)
scatterPlot(TRXTimap, TRy, 'isomap')
```

```
Wall time: 620 ms
Wall time: 8.11 s
(50000, 10)
```

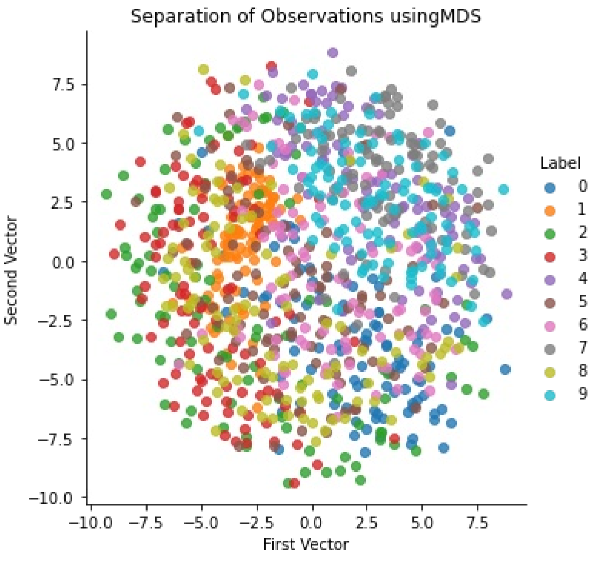


MDS

In [44]:
```
from sklearn.manifold import MDS

n_components = 3
n_init = 2
max_iter = 1200
metric = True
n_jobs = 4
random_state = 2018

Emds = MDS(n_components=n_components,
          n_init=n_init,
          max_iter=max_iter,
          metric=metric,
          n_jobs=n_jobs,
          random_state=random_state)

%time TRXTmds = Emds.fit_transform(TRX0)
TRXTmds = pd.DataFrame(data=TRXTmds, index=iTR0)

print(TRXTmds.shape)
```

```
scatterPlot(TRXTmds, TRy0, 'MDS')
```

```
Wall time: 1min 11s
(1000, 3)
```



In [45]:

```
TRXTmds
```

Out[45]:

1	0	1	2
1907	3.742804	-5.522545	5.454732
5809	2.836946	-6.351985	-0.773530
32846	-0.034010	-4.593104	8.550494
8838	-0.778592	-3.789294	8.203202
13044	-1.427745	-1.683926	8.627664
...	...	...	...
5459	5.406290	0.307090	-3.424906
34983	4.955261	1.458435	-2.962013
16622	0.360030	2.772790	-4.439113
32504	-1.408353	6.400787	3.725603
27565	3.446744	-0.049185	-3.483470

1000 rows × 3 columns

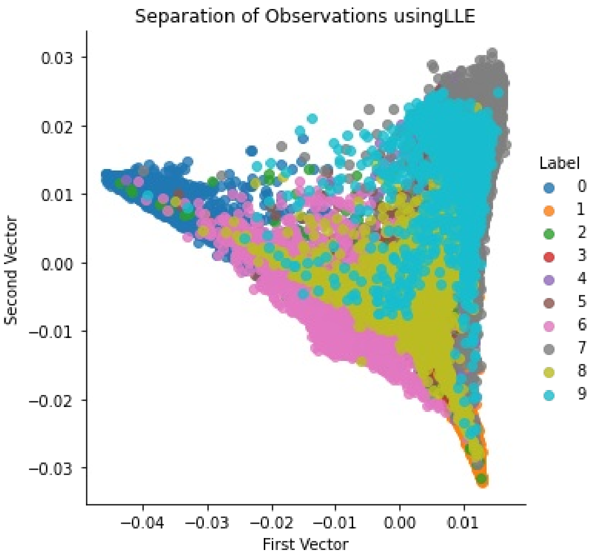
LLE

In [46]:

```
from sklearn.manifold import LocallyLinearEmbedding

n_neighbors = 10
n_components = 3
ethod = 'modified'
n_jobs = 4
random_state = 2018

Elle = LocallyLinearEmbedding(n_neighbors=n_neighbors,
                              n_components = n_components,
                              method='modified',
                              random_state=random_state,
                              n_jobs=n_jobs)

%time Elle.fit(TRX.loc[0:5000, :])
TRXTlle = Elle.transform(TRX)
TRXTlle = pd.DataFrame(data=TRXTlle, index=iTR)

VLXTlle = Elle.transform(VLX)
VLXTlle = pd.DataFrame(data=VLXTlle, index=iVL)

print(TRXTlle.shape)
```

```
scatterPlot(TRXTlle, TRy, 'LLE')
```

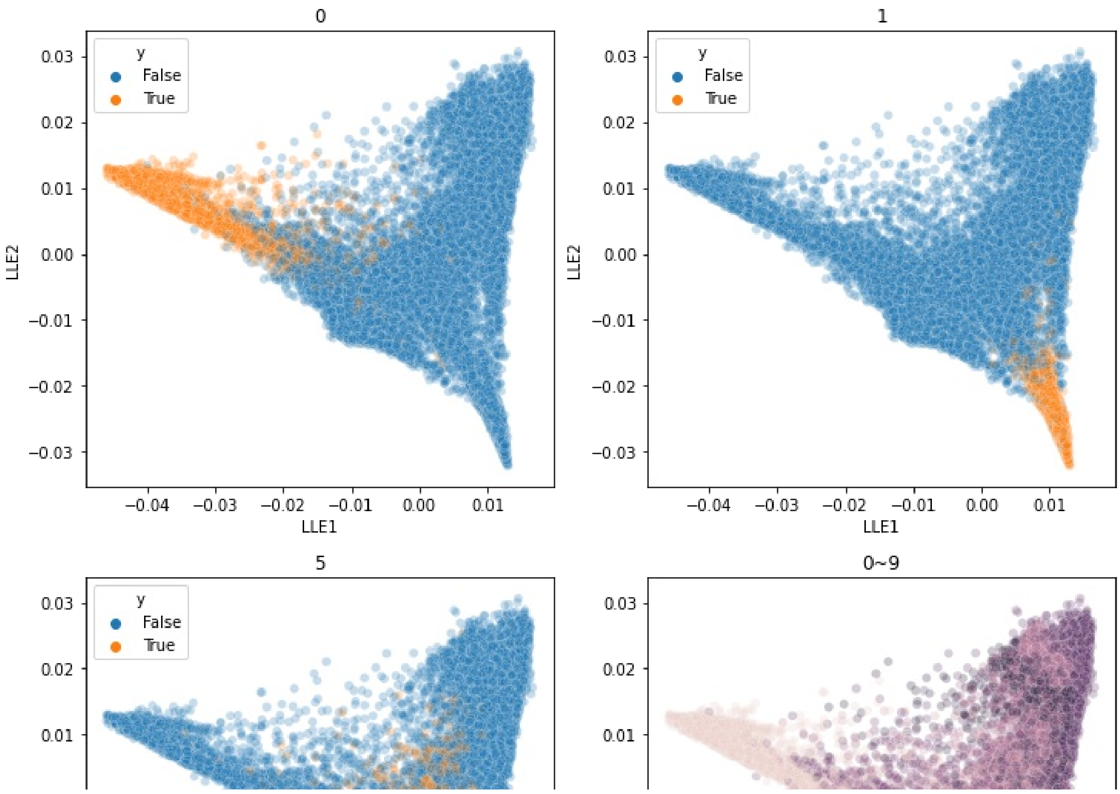
```
Wall time: 9.52 s
(50000, 3)
```

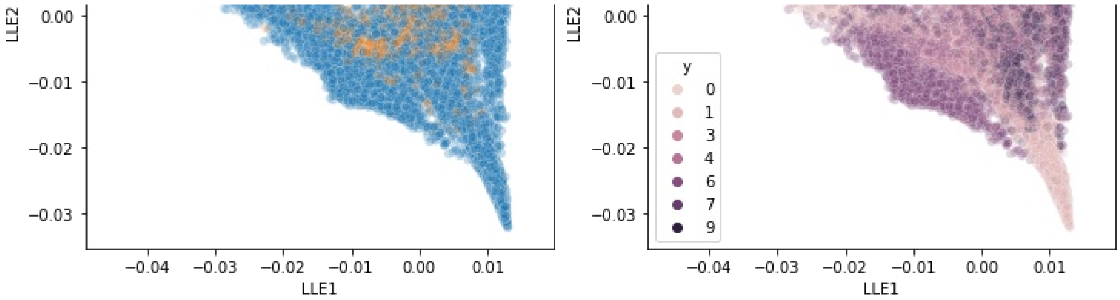


In [47]:
```
Elle.embedding_
```

Out[47]:
```
array([[-0.00111031, -0.00486779, -0.00313505],
       [-0.03931153,  0.0089401 ,  0.0206251 ],
       [ 0.00713626,  0.02153579, -0.00420588],
       ...,
       [ 0.01072742, -0.02356577,  0.01404733],
       [ 0.00227599, -0.00813298,  0.00379371],
       [ 0.00834223,  0.01594336, -0.0012472 ]])
```

In [48]:
```
fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(12, 12))
d0 = sns.scatterplot(x=TRXTlle.loc[:, 0], y=TRXTlle.loc[:, 1], hue=TRy==0, alpha=0.25, ax=axs[0,0]);
d0.set(xlabel='LLE1', ylabel='LLE2', title='0')
d1 = sns.scatterplot(x=TRXTlle.loc[:, 0], y=TRXTlle.loc[:, 1], hue=TRy==1, alpha=0.25, ax=axs[0,1]);
d1.set(xlabel='LLE1', ylabel='LLE2', title='1')
d5 = sns.scatterplot(x=TRXTlle.loc[:, 0], y=TRXTlle.loc[:, 1], hue=TRy==5, alpha=0.25, ax=axs[1,0]);
d5.set(xlabel='LLE1', ylabel='LLE2', title='5')
dd = sns.scatterplot(x=TRXTlle.loc[:, 0], y=TRXTlle.loc[:, 1], hue=TRy, alpha=0.25, ax=axs[1,1]);
dd.set(xlabel='LLE1', ylabel='LLE2', title='0~9');
```





In [49]:

```
TRXT0 = TRXTlle.groupby(TRy).apply(lambda x:x.sample(100, random_state=2018))
iTR0 = TRXT0.index.to_frame()[1]
TRXT0.set_index(iTR0, inplace=True)
TRy0 = TRy.iloc[iTR0]

fig = px.scatter_3d(TRXT0, x=0, y=1, z=2,
                   color=TRy0==0,
                   symbol=TRy0==0,
                   size=1+(TRy0==0).astype(int),
                   size_max=9,
                   opacity=0.5)
fig.show()
```

In [50]:

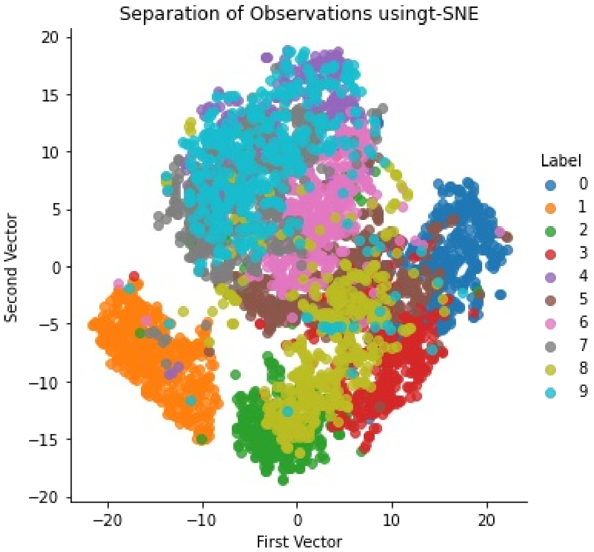
```
fig = px.scatter_3d(TRXTlle, x=0, y=1, z=2, color=TRy, size_max=10)
fig.show()
```

t-SNE

In [51]:

```
from sklearn.manifold import TSNE

n_components = 3
learning_rate = 300
perplexity = 30
early_exaggeration = 12
init = 'random'
random_state = 2018

Etsne = TSNE(n_components=n_components,
            learning_rate=learning_rate,
            perplexity=perplexity,
            early_exaggeration=early_exaggeration,
            init=init,
            random_state=random_state)

%time TRXTtsne = Etsne.fit_transform(TRXTpca.iloc[:5000, :9])
TRXTtsne = pd.DataFrame(data=TRXTtsne, index=iTR[:5000])

print(TRXTtsne.shape)
scatterPlot(TRXTtsne, TRy, 't-SNE')
```

```
Wall time: 1min 20s
(5000, 3)
```



In [52]:

```
TMPy = TRy.loc[TRXTtsne.index]
fig = px.scatter_3d(TRXTtsne, x=0, y=1, z=2, color=TMPy, size=TMPy+1, size_max=8)
fig.show()
```

In [53]:

```
fig = px.scatter_3d(TRXTtsne, x=0, y=1, z=2, color=(TMPy==0), size=(TMPy==0)+1, size_max=8)
fig.show()
```

MiniBatch Dictionary Learning

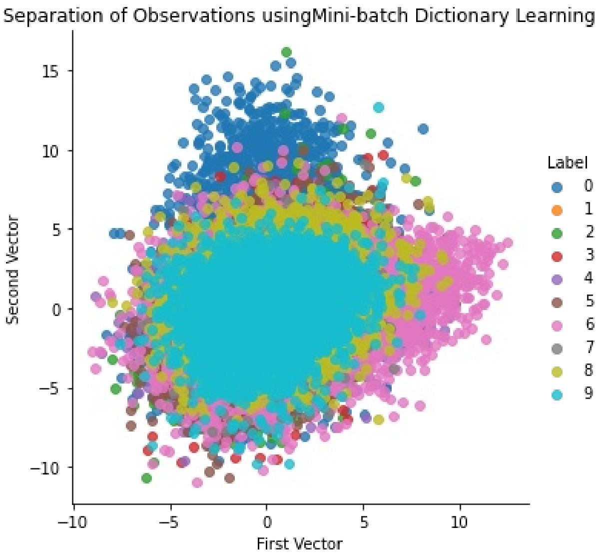
In [54]:

```
from sklearn.decomposition import MiniBatchDictionaryLearning

n_components = 50
alpha = 1
batch_size = 200
n_iter = 1000
random_state = 2018

Edl = MiniBatchDictionaryLearning(n_components = n_components,
                                  alpha=alpha,
                                  batch_size=batch_size,
                                  n_iter=n_iter,
                                  random_state=random_state)

%time TRXTdl =Edl.fit_transform(TRX)
TRXTdl = pd.DataFrame(data=TRXTdl, index=iTR)

VLXTdl = Edl.transform(VLX)
VLXTdl = pd.DataFrame(data=VLXTdl, index=iVL)

print(TRXTdl.shape)
scatterPlot(TRXTdl, TRy, 'Mini-batch Dictionary Learning')
```

```
C:\Users\wngus\anaconda3\envs\tf\lib\site-packages\sklearn\utils\validation.py:63: RuntimeWarning:
```

```
 Orthogonal matching pursuit ended prematurely due to linear
dependence in the dictionary. The requested precision might not have been met.
```

```
Wall time: 5min 7s
(50000, 50)
```

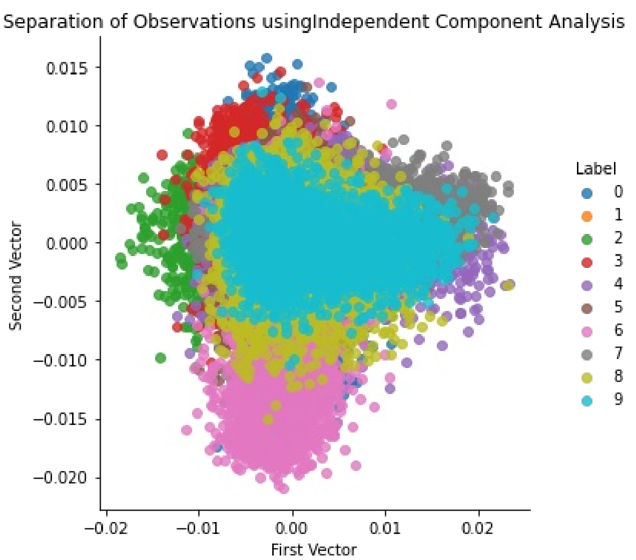


ICA

In [55]:

```python
from sklearn.decomposition import FastICA

n_components = 25
algorithm = 'parallel'
whiten = True
max_iter = 100
random_state = 2018

Eica = FastICA(n_components = n_components,
               algorithm = algorithm,
               whiten = whiten,
               max_iter = max_iter,
               random_state = random_state)

%time Eica.fit(TRX)
TRXTica = Eica.fit_transform(TRX)
TRXTica = pd.DataFrame(data=TRXTica, index=iTR)

VLXTica = Eica.transform(VLX)
VLXTica = pd.DataFrame(data=VLXTica, index=iVL)

scatterPlot(TRXTica, TRy, 'Independent Component Analysis')
```

```
Wall time: 15.9 s
```



In [56]:
```
fig = px.scatter_3d(TRXTica, x=0, y=1, z=2, color=(TRy==2), size=(TRy==2)+1, size_max=8)
fig.show()
```

In [57]:
```
print("--- %s seconds ---" % (time.time() - start_time))
```

```
--- 1034.4201645851135 seconds ---
```