20211014_Supplier_Management_MBW7

October 14, 2021

[3]:
```
# Principal Component Analysis: the directionsin space that best explain the␣
↪variability of data.
```

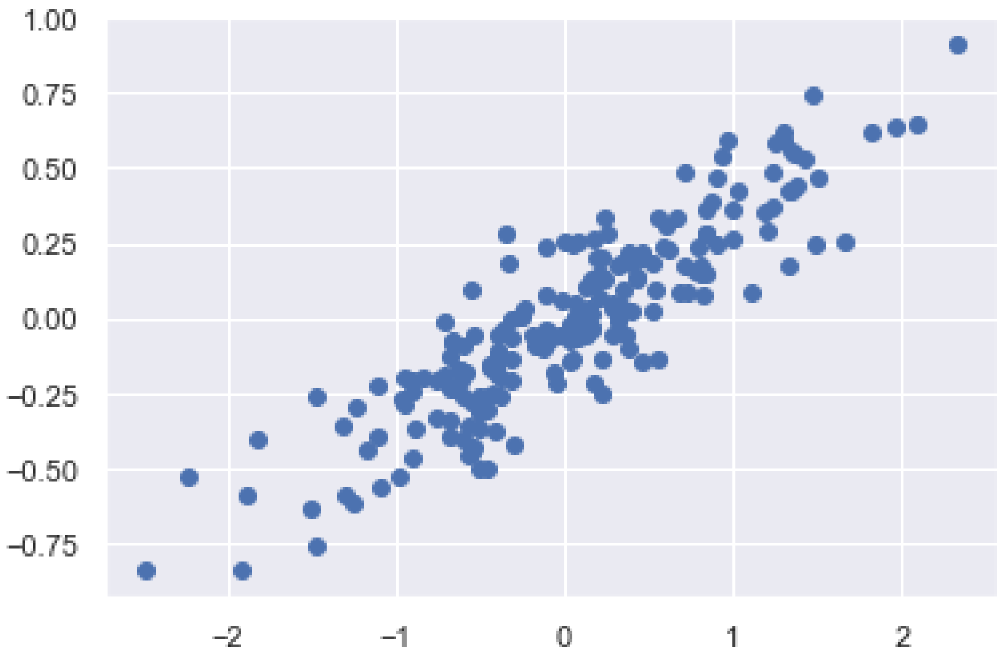
[4]:
```
# Unsupervised Method : no information or knowledge about the dataset is␣
↪necessary previous to analysis
```

[5]:
```
# Upload a 2 dimensional dataset
```

[6]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
```

[9]:
```
rng = np.random.RandomState(1)

X = np.dot(rng.rand(2, 2), rng.randn(2, 200)).T

plt.scatter(X[:,0], X[:,1])
```

[9]:
```
<matplotlib.collections.PathCollection at 0x7fb5176a2dc0>
```



[10]:
```
import sklearn

from sklearn.decomposition import PCA

pca = PCA(n_components=2)

pca.fit(X)
```

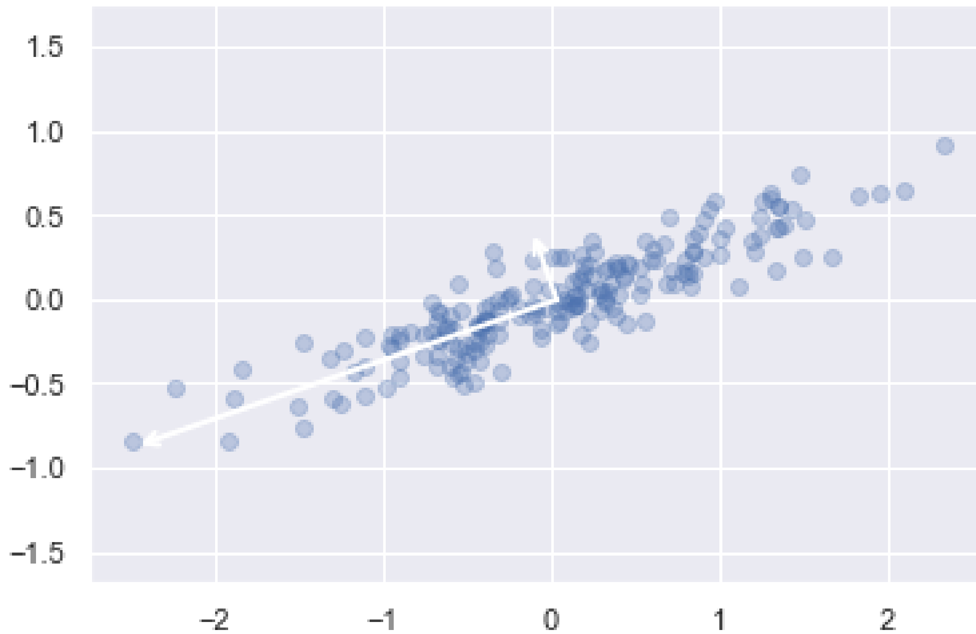
[10]: PCA(n_components=2)

[13]:
```
print(pca.components_)
print(pca.explained_variance_)
```

```
[[-0.94446029 -0.32862557]
 [-0.32862557  0.94446029]]
[0.7625315 0.0184779]
```

[12]:
```
# this line is not relevant for exam

def draw_vector (v0, v1, ax=None):
    ax = ax or plt.gca()
    arrowprops=dict(arrowstyle='->',
                    linewidth=2,
                    shrinkA=0, shrinkB=0)
    ax.annotate('', v1, v0, arrowprops=arrowprops)
```

```
plt.scatter(X[:, 0], X[:, 1], alpha=0.3)
for length, vector in zip(pca.explained_variance_, pca.components_):
    v = vector * 3 * np.sqrt(length)
    draw_vector(pca.mean_, pca.mean_ + v)
plt.axis('equal');
```



[14]:
```
# more than two dimensions
```

[15]:
```
# load packages

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[16]:
```
# load dataset

from sklearn.datasets import load_breast_cancer
```

[17]:
```
cancer = load_breast_cancer()
```

[18]:
```
df = pd.DataFrame(cancer['data'], columns=cancer['feature_names'])
```

[19]:
```
df.head()
```

[19]:

```
   mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
0        17.99         10.38          122.80     1001.0          0.11840
1        20.57         17.77          132.90     1326.0          0.08474
2        19.69         21.25          130.00     1203.0          0.10960
3        11.42         20.38           77.58      386.1          0.14250
4        20.29         14.34          135.10     1297.0          0.10030

   mean compactness  mean concavity  mean concave points  mean symmetry  \
0           0.27760          0.3001              0.14710         0.2419
1           0.07864          0.0869              0.07017         0.1812
2           0.15990          0.1974              0.12790         0.2069
3           0.28390          0.2414              0.10520         0.2597
4           0.13280          0.1980              0.10430         0.1809

   mean fractal dimension  …  worst radius  worst texture  worst perimeter  \
0                 0.07871  …         25.38          17.33           184.60
1                 0.05667  …         24.99          23.41           158.80
2                 0.05999  …         23.57          25.53           152.50
3                 0.09744  …         14.91          26.50            98.87
4                 0.05883  …         22.54          16.67           152.20

   worst area  worst smoothness  worst compactness  worst concavity  \
0      2019.0            0.1622             0.6656           0.7119
1      1956.0            0.1238             0.1866           0.2416
2      1709.0            0.1444             0.4245           0.4504
3       567.7            0.2098             0.8663           0.6869
4      1575.0            0.1374             0.2050           0.4000

   worst concave points  worst symmetry  worst fractal dimension
0                0.2654          0.4601                  0.11890
1                0.1860          0.2750                  0.08902
2                0.2430          0.3613                  0.08758
3                0.2575          0.6638                  0.17300
4                0.1625          0.2364                  0.07678

[5 rows x 30 columns]
```

[20]:

```
# Scale the dataset before processing!
```

[24]:

```
from sklearn.preprocessing import StandardScaler
# we create a dataset with the same statistical properties but with a
# normal distribution with mean = 1 and stddev=0 (Z transform)
scalar = StandardScaler()

scalar.fit(df)
scaled_data = scalar.transform(df)
```

```
scaled_data
```

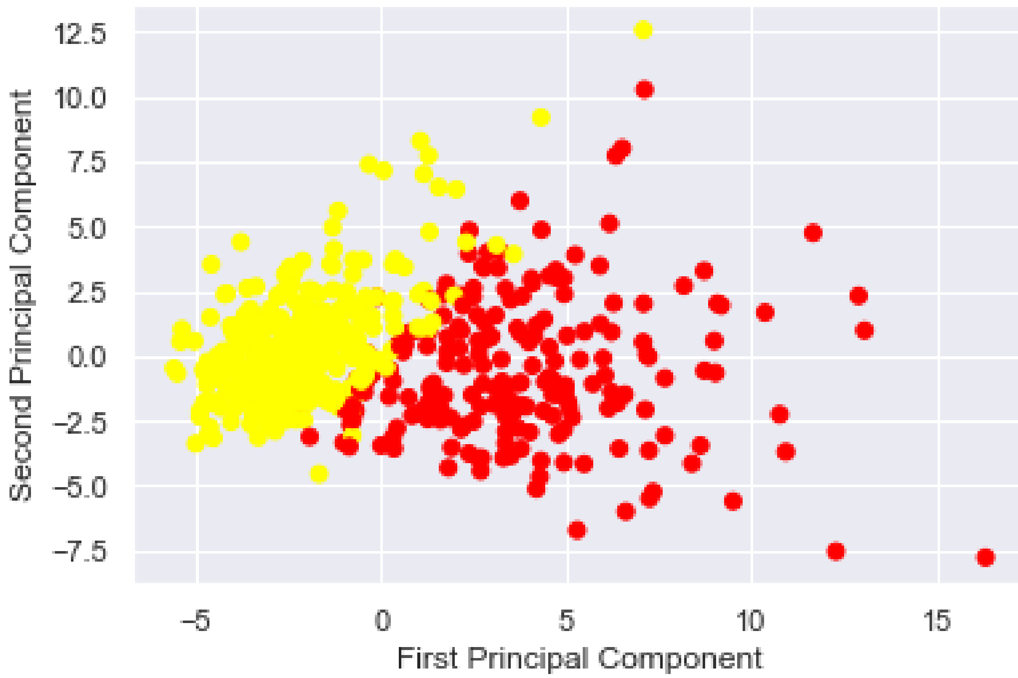
[24]:
```
array([[ 1.09706398, -2.07333501,  1.26993369, …,  2.29607613,
          2.75062224,  1.93701461],
        [ 1.82982061, -0.35363241,  1.68595471, …,  1.0870843 ,
         -0.24388967,  0.28118999],
        [ 1.57988811,  0.45618695,  1.56650313, …,  1.95500035,
          1.152255  ,  0.20139121],
        …,
        [ 0.70228425,  2.0455738 ,  0.67267578, …,  0.41406869,
         -1.10454895, -0.31840916],
        [ 1.83834103,  2.33645719,  1.98252415, …,  2.28998549,
          1.91908301,  2.21963528],
        [-1.80840125,  1.22179204, -1.81438851, …, -1.74506282,
         -0.04813821, -0.75120669]])
```

[26]:
```
pca = PCA(n_components=2)
pca.fit(scaled_data)

x_pca = pca.transform(scaled_data)
x_pca.shape
```

[26]:
```
(569, 2)
```

[27]:
```
# graphic representation

plt.scatter(x_pca[:,0], x_pca[:,1], c=cancer['target'],cmap='autumn')
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
```

[27]:
```
Text(0, 0.5, 'Second Principal Component')
```

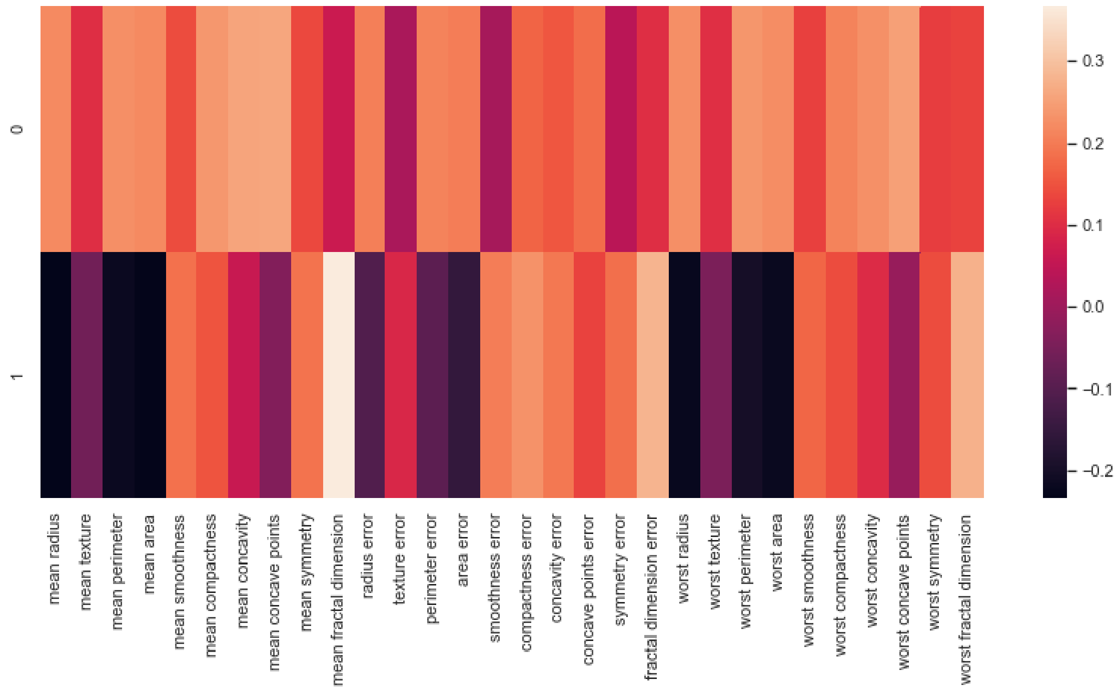


[29]:
```
print(pca.components_)
print(pca.explained_variance_)
```

```
[[ 0.21890244  0.10372458  0.22753729  0.22099499  0.14258969  0.23928535
   0.25840048  0.26085376  0.13816696  0.06436335  0.20597878  0.01742803
   0.21132592  0.20286964  0.01453145  0.17039345  0.15358979  0.1834174
   0.04249842  0.10256832  0.22799663  0.10446933  0.23663968  0.22487053
   0.12795256  0.21009588  0.22876753  0.25088597  0.12290456  0.13178394]
 [-0.23385713 -0.05970609 -0.21518136 -0.23107671  0.18611302  0.15189161
   0.06016536 -0.0347675   0.19034877  0.36657547 -0.10555215  0.08997968
  -0.08945723 -0.15229263  0.20443045  0.2327159   0.19720728  0.13032156
   0.183848    0.28009203 -0.21986638 -0.0454673  -0.19987843 -0.21935186
   0.17230435  0.14359317  0.09796411 -0.00825724  0.14188335  0.27533947]]
[13.30499079  5.7013746 ]
```

[31]:
```
# explain the variability in a "heatmap"
# how much variability is explained by EACH KPI?

df_comp = pd.DataFrame(pca.components_, columns=cancer['feature_names'])
plt.figure(figsize=(14,6))
sns.heatmap(df_comp)
```

[31]: <AxesSubplot:>



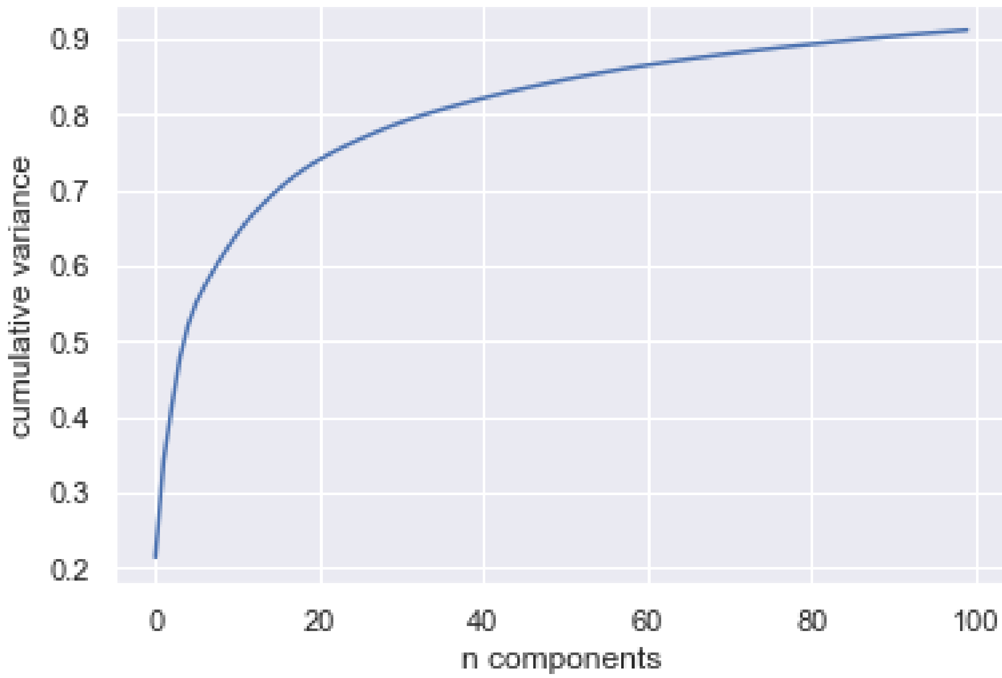
[32]:
```
# face recognition with PCA
```

[33]:
```
from sklearn.datasets import fetch_lfw_people

faces = fetch_lfw_people(min_faces_per_person=30)

faces.data.shape
```

[33]: (2370, 2914)

[34]:
```
# plot some data (not relevant for exam)
fig, ax = plt.subplots(4, 8, subplot_kw=dict(xticks=[], yticks=[]))
for i, axi in enumerate(ax.flat):
    axi.imshow(faces.images[i], cmap='gray')
```

[35]:
```
# we perform a PCA

from sklearn.decomposition import PCA as RandomizedPCA

model = RandomizedPCA(100).fit(faces.data)

plt.plot(np.cumsum(model.explained_variance_ratio_))
plt.xlabel('n components')
plt.ylabel('cumulative variance');
```



[36]:
```
# t-SNE. t-Distributed Stochastic Neighbor Embedding

# Non supervised method. Non-linear technique that explores
# datasets and enables a visualization of multi--dimensional data

# The difference with PCA is that PCA is a Linear dimensional reduction method
# PCA tries to maximize the variance by keeping the distance between the data␣
↪constant.

# t-SNE is a NON--linear method that only keeps the distance between the points␣
↪or local similarities.

# The algorithm calculates the distance between data in the high--dimensional␣
↪space,
# and projects this into the lower dimensional state (2Dimensional in order to␣
↪visualize it)
# trying to keep the relative distances by optimizing a cost function.
```

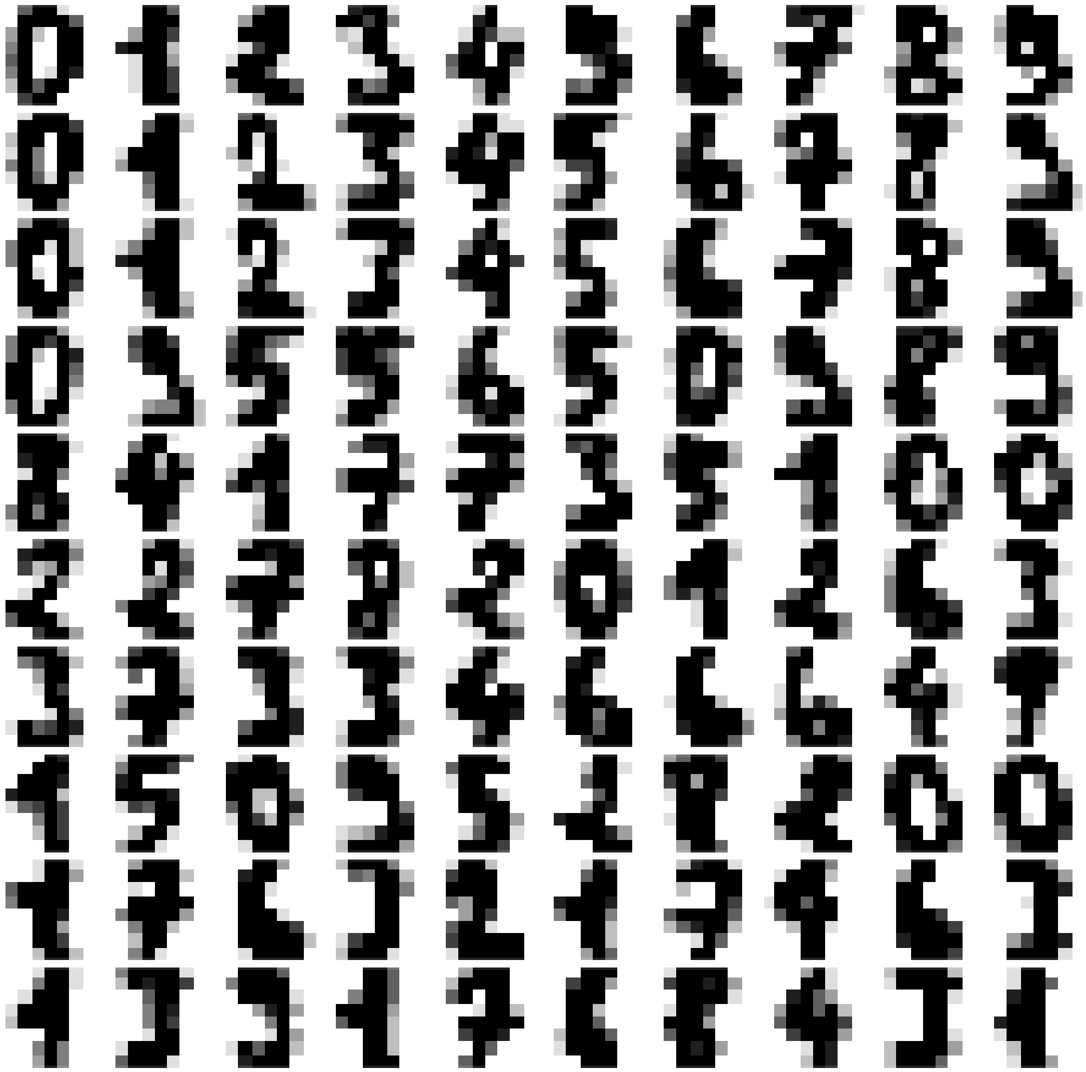
[37]:
```
# 1. Find the similarities between the points bby implementing a t-Student␣
↪distribution.
# 2. Calculate the distance between the points of the high dimensional space␣
↪and project them into the lower dimensional state

import matplotlib.pyplot as plt
```

```python
import seaborn as sns; sns.set()
import numpy as np
```

[38]:
```python
from sklearn.datasets import load_digits
digits = load_digits()
digits.data.shape
```

[38]: (1797, 64)

[40]:
```python
# plot some data (not relevant for exam)

def plot_digits(data):
    fig, ax = plt.subplots(10, 10, figsize=(8, 8),
                           subplot_kw=dict(xticks=[], yticks=[]))
    fig.subplots_adjust(hspace=0.05, wspace=0.05)
    for i, axi in enumerate(ax.flat):
        im = axi.imshow(data[i].reshape(8, 8), cmap='binary')
        im.set_clim(0, 8)
plot_digits(digits.data)
```



[42]:
```
digits.target
```

[42]:
```
array([0, 1, 2, ..., 8, 9, 8])
```

[43]:
```
data_X = digits.data[:600]
y = digits.target[:600]
```

[44]:
```
from sklearn.manifold import TSNE
```

[45]:
```
tsne = TSNE(n_components=2, random_state=0)
```

[46]:
```
tsne_obj=tsne.fit_transform(data_X)
tsne_obj
```

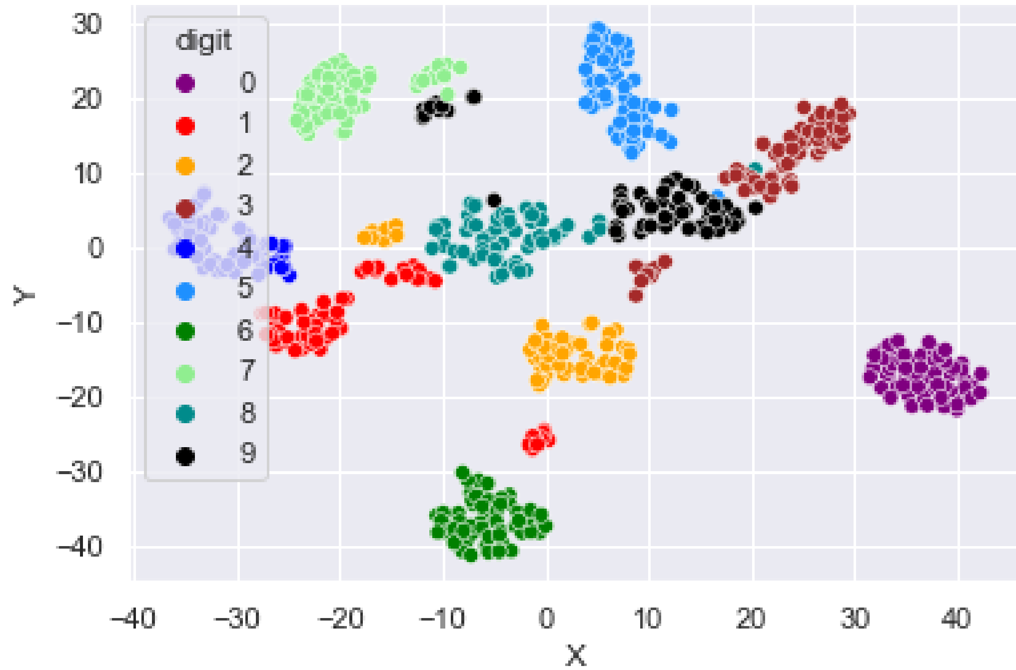
[46]:
```
array([[ 35.729576  , -15.248073  ],
       [-19.279284  ,  -6.8981485 ],
       [-15.420641  ,   0.94437665],
       …,
       [-19.960772  ,  17.956438  ],
       [ -4.50595   , -40.9033    ],
       [  9.309275  ,  -4.539865  ]], dtype=float32)
```

[49]:
```python
import pandas as pd
tsne_df = pd.DataFrame({'X':tsne_obj[:,0],
                        'Y':tsne_obj[:,1],
                        'digit':y})
tsne_df.head()
```

[49]:

	X	Y	digit
0	35.729576	-15.248073	0
1	-19.279284	-6.898149	1
2	-15.420641	0.944377	2
3	22.055639	9.480317	3
4	-33.239338	-2.813015	4

[50]:
```python
sns.scatterplot(x="X", y="Y",
                hue="digit",
                palette=['purple','red','orange','brown','blue',
                         'dodgerblue','green','lightgreen','darkcyan','black'],
                legend='full',
                data=tsne_df);
```



[51]:
```
# www.profh4.com
```

[ ]:
```

```