In [56]:
```python
import os
import  pickle
import pandas as pd  # panda's nickname is pd
import numpy as np  # numpy as np
from pandas import DataFrame, Series     # for convenience
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
```

In [57]:
```python
#load training and testing datasets
train=pd.read_csv('C:/Users/Jimmy/Documents/train.csv')
test=pd.read_csv('C:/Users/Jimmy/Documents/test.csv')
```

In [58]:
```python
#let's see wht this data looks like
train.head()
```

Out[58]:

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775	pixel776	p
0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
2	1	0	0	0	0	0	0	0	0	0	...	0	0	0	
3	4	0	0	0	0	0	0	0	0	0	...	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	

5 rows × 785 columns

In [59]:
```python
test.head()
```

Out[59]:

	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel774	pixel775	pixel776
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0

5 rows × 784 columns

In [60]:
```
train.describe()
```

Out[60]:

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	
count	42000.000000	42000.0	42000.0	42000.0	42000.0	42000.0	42000.0	42000.0	42000.0	42000.0	...	4200
mean	4.456643	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
std	2.887730	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
min	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
25%	2.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
50%	4.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
75%	7.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
max	9.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	25

8 rows × 785 columns

In [63]:
```
print('Training data shape')
print(train.shape)
print('Test data shape')
print(test.shape)
```

```
Training data shape
(42000, 785)
Test data shape
(28000, 784)
```

In [80]:
```
#Let's plot mnist data in order to visualize it
#pulling mnist data - different from kaggle dataset
from six.moves import urllib

from scipy.io import loadmat
mnist_path = "./mnist-original.mat"
response = urllib.request.urlopen(mnist_alternative_url)
with open(mnist_path, "wb") as f:
    content = response.read()
    f.write(content)
mnist_raw = loadmat(mnist_path)
mnist = {
    "data": mnist_raw["data"].T,
    "target": mnist_raw["label"][0],
    "COL_NAMES": ["label", "data"],
    "DESCR": "mldata.org dataset: mnist-original",
}
```

In [81]:
```
X, y = mnist['data'], mnist['target']
```

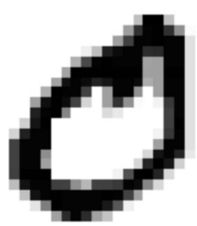
In [82]:
```
print('X shape:')
print(X.shape)
print('y shape:')
print(y.shape)
```

```
X shape:
(70000, 784)
y shape:
(70000,)
```

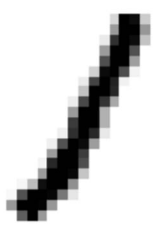
In [87]:
```
def plot_digit(data):
    image = data.reshape(28, 28)
    plt.imshow(image, cmap = matplotlib.cm.binary,
               interpolation="nearest")
    plt.axis("off")
```

In [88]:
```
def plot_digits(instances, images_per_row=10, **options):
    size = 28
    images_per_row = min(len(instances), images_per_row)
    images = [instance.reshape(size,size) for instance in instances]
    n_rows = (len(instances) - 1) // images_per_row + 1
    row_images = []
    n_empty = n_rows * images_per_row - len(instances)
    images.append(np.zeros((size, size * n_empty)))
    for row in range(n_rows):
        rimages = images[row * images_per_row : (row + 1) * images_per_row]
        row_images.append(np.concatenate(rimages, axis=1))
    image = np.concatenate(row_images, axis=0)
    plt.imshow(image, cmap = matplotlib.cm.binary, **options)
    plt.axis("off")
```

In [102]:
```
random_digit1 = X[1]
plt.figure(figsize=(3,3))
plot_digit(random_digit1)
plt.show()
```



In [100]:
```
random_digit2 = X[9200]
plt.figure(figsize=(3,3))
plot_digit(random_digit2)
plt.show()
```



In [101]:
```
random_digit3 = X[25000]
plt.figure(figsize=(3,3))
plot_digit(random_digit3)
plt.show()
```

In [96]:
```
plt.figure(figsize=(8,8))
example_images = np.r_[X[:12000:600], X[13000:30600:600], X[30600:60000:590]]
plot_digits(example_images, images_per_row=10)
plt.show()
```

In [9]:
```
#(1) Begin by fitting a random forest classifier using the full set of 784 explanat
ory
#variables and the model training set (train.csv).
#Record the time it takes to fit the model and then evaluate the model on the test.
csv
#data by submitting to Kaggle.com. Provide your Kaggle.com score and user ID.
from sklearn.ensemble import RandomForestClassifier
```

```
C:\Users\Jimmy\Anaconda3\lib\site-packages\sklearn\ensemble\weight_boosting.py:2
9: DeprecationWarning: numpy.core.umath_tests is an internal NumPy module and sh
ould not be imported. It will be removed in a future NumPy release.
  from numpy.core.umath_tests import inner1d
```

In [10]:
```
x_train = train.iloc[:,1:]
y_train = train['label']

print('Training data x shape')
print(x_train.shape)
print('Training data y shape')
print(y_train.shape)
```

```
Training data x shape
(42000, 784)
Training data y shape
(42000,)
```

In [16]:
```
start=datetime.datetime.now()

random_forest = RandomForestClassifier(n_estimators=10, max_features="sqrt",
                                       bootstrap=True)
random_forest.fit(x_train, y_train)

end=datetime.datetime.now()

print(end-start)
```

```
0:00:12.710691
```

In [17]:
```
rfp = random_forest.predict(test)
rfp.shape
```

Out[17]: (28000,)

In [22]:
```
data = {'ImageId': np.arange(1,28001), 'Label': rfp}
df = pd.DataFrame(data=data)
df.head()
```

Out[22]:

	ImageId	Label
0	1	2
1	2	0
2	3	9
3	4	9
4	5	3

In [24]:
```
df.to_csv('random_forest_1.csv', index=False)
```

In [25]:
```
#Kagg;e Results
print ('Kaggle Results:')
print('Name: James Casey')
print('Score: 0.94342')
print('Rank: 2105')
```

```
Kaggle Results:
Name: James Casey
Score: 0.94342
Rank: 2105
```

In [26]:
```
#(2) Execute principal components analysis (PCA) on the combined training
#and test set data together, generating principal components that represent
#95 percent of the variability in the explanatory variables. The number of
#principal components in the solution should be substantially fewer than the
#784 explanatory variables. Record the time it takes to identify the principal comp
onents
```

In [40]:
```
from sklearn.decomposition import PCA
pca_data = pd.concat([x_train,test])
pca_data.head()
```

Out[40]:

	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel774	pixel775	pixel776
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0

5 rows × 784 columns

In [46]:
```
start=datetime.datetime.now()

pca = PCA(n_components=0.95)
pca_model = pca.fit_transform(pca_data)

end=datetime.datetime.now()
print(end-start)
```

```
0:00:32.338129
```

In [47]:
```
print('PCA number of principal components:')
print(pca.n_components_)
```

```
PCA number of principal components:
154
```

In [48]:
```
#(3) Using the identified principal components from step (2),
#use the train.csv to build another random forest classifier.
#Record the time it takes to fit the model and to evaluate the
#model on the test.csv data by submitting to Kaggle.com.
#Provide your Kaggle.com score and user ID.
```

In [49]:
```
pca_X_train = pca_model[:42000]
print(pca_X_train.shape)
```

```
(42000, 154)
```

In [50]:
```
pca_X_test = pca_model[42000:]
print(pca_X_test.shape)
```

```
(28000, 154)
```

In [51]:
```
start=datetime.datetime.now()

random_forest_2 = RandomForestClassifier(n_estimators=10, max_features="sqrt",
                                         bootstrap=True, )
random_forest_2.fit(pca_X_train, y_train)

end=datetime.datetime.now()
print(end-start)
```

```
0:00:31.192344
```

In [52]:
```
rfp2 = random_forest_2.predict(pca_X_test)
print(rfp2.shape)
print(np.arange(1,28001).shape)
```

```
(28000,)
(28000,)
```

In [53]:
```
data2 = {'ImageId': np.arange(1,28001), 'Label': rfp2}
df2 = pd.DataFrame(data=data2)
df2.to_csv('random_forest_pca.csv', index=False)
```

In [54]:
```
#Kagg;e Results
print ('Kaggle Results:')
print('Name: James Casey')
print('Score: 0.88271')
```

```
Kaggle Results:
Name: James Casey
Score: 0.88271
```

In [ ]: