Code Summary

```
import numpy as np
import pandas as pd
from sklearn.decomposition import PCA
from sklearn import random_projection
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix,accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
```

Here we imported all the required libraries.

```
train=pd.read_csv("train.csv")
```

```
train.head()
```

Loading data –

pd.read_csv : imports csv file/dataset

train.head : list first 5 elements of all the features.

```
train.shape
```

train.shape : describes the shape i.e. rows and columns.

```
train.describe()
```

train.describe : describes the list/dataset.

```
train.isnull().sum()
```

It checks whether all the cells are null or not null.

```
train.fillna(0,inplace=True)
```

```
y = train['label']
x = train.drop("label",axis=1)
```

```
x.iloc[0].values.reshape(28,28)
```

train.fillna : It check if null is true then it specifies the given value.

train['label'] : start training the label feature.

train.drop : It removes specified rows and columns.

iloc : It selects particular rows and columns.

And return the data type.

```
def show_images(num_images):
    for n in range(0,num_images):
        plt.subplot(5,5,n+1)
        plt.imshow(x.iloc[n].values.reshape(28,28))#cmap means how the imgae to be presented
        plt.xticks([]) #removes numbered labels on x-axis
        plt.yticks([])
```

```
show_images(25)
```

This function shows, how the image is to be presented.

```
y.unique()
```

This function finds the unique element of the array i.e. neglects the repeated values.

```python
#def show_images by digit(digit):
digit=1
if digit in list(range(10)):
    indices=np.where(y==digit)#extract indecies where y==1
    for d in range(0,50):
        plt.subplot(5,10,d+1)
        data=x.iloc[indices[0][d]].values.reshape(28,28)
        plt.imshow(data)
        plt.xticks([])
        plt.yticks([])
else:
    print("number doesn't exist")
```

It takes the 3 coordinates in a 3-D space and describes the position.

```python
def fit_random_forest_classifier_with_plot(X, y):

    #First let's create training and testing data
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42

    #We could grid search and tune, but let's just fit a simple model to see how it does
    #instantiate
    clf = RandomForestClassifier(n_estimators=100, max_depth=None)

    #fit
    clf.fit(X_train, y_train)

    #predict
    y_preds = clf.predict(X_test)

    #score
    mat = confusion_matrix(y_test, y_preds)
    print(mat)
#    print(sns.heatmap(mat, annot=True, cmap='bwr', linewidths=.5))
    acc = accuracy_score(y_test, y_preds)
    print(acc)
    return acc

fit_random_forest_classifier_with_plot(x, y)
```

It imports & implement random forest classifier and returns the accuracy of the applied random forest algorithm.

```
from sklearn.random_projection import GaussianRandomProjection,SparseRandomProjection
```

```
rp=SparseRandomProjection(eps=0.5)#n_components='auto' or eps=''
```

```
x_rp=rp.fit_transform(x)
```

```
x
```

```
x_rp.shape
```

```
fit_random_forest_classifier_with_plot(x_rp, y)
```

It imports & implies Gaussian sparse random projection to plot the shape & reduce the feature of the dataset.

```
for ep in np.arange(0.5,1,0.2):
    rp=SparseRandomProjection(eps=ep)
    x_rp=rp.fit_transform(x)
    acc=fit_random_forest_classifier_with_plot(x_rp, y)
    print("With epsilon = {:.2f}, the transformed data has {} components, a random forest acheived an accuracy of {}.".format(ep,
```

Here we define the learning rate.

```
eps=np.arange(0.1,1,0.01)
n_comp = johnson_lindenstrauss_min_dim(n_samples=1e6, eps=eps)

plt.plot(eps, n_comp, 'bo');
plt.xlabel('eps');
plt.ylabel('Number of Components');
plt.title('Number of Components by eps');
```

Here we compare the no of features with different learning rates.

```
x_samples,x_comps=x.shape
print("The orignial data has {} samples with dimension {}.".format(x_samples, x_comps))
```

Here we print/plot the original dataset before dimensionality reduction.

```
n_components = 30

rp = SparseRandomProjection(n_components=n_components)
x_rp = rp.fit_transform(x)
```

Here we specify the dimension of the feature we want to make using random projection.