```python
## set up notebook to display multiple output in one cell
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

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


from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.decomposition import PCA
from sklearn.cluster import MiniBatchKMeans
from sklearn import metrics

from datetime import datetime
import warnings
warnings.filterwarnings("ignore")

%matplotlib inline
sns.set()
pd.set_option('display.max_rows', None)

train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')

train.columns
train.info()
```

```
Index(['label', 'pixel0', 'pixel1', 'pixel2', 'pixel3', 'pixel4',
'pixel5',
       'pixel6', 'pixel7', 'pixel8',
       ...
       'pixel774', 'pixel775', 'pixel776', 'pixel777', 'pixel778',
'pixel779',
       'pixel780', 'pixel781', 'pixel782', 'pixel783'],
      dtype='object', length=785)

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42000 entries, 0 to 41999
Columns: 785 entries, label to pixel783
dtypes: int64(785)
memory usage: 251.5 MB
```

```python
test.columns
test.info()
```

```
Index(['pixel0', 'pixel1', 'pixel2', 'pixel3', 'pixel4', 'pixel5',
'pixel6',
       'pixel7', 'pixel8', 'pixel9',
       ...
       'pixel774', 'pixel775', 'pixel776', 'pixel777', 'pixel778',
'pixel779',
       'pixel780', 'pixel781', 'pixel782', 'pixel783'],
      dtype='object', length=784)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28000 entries, 0 to 27999
Columns: 784 entries, pixel0 to pixel783
dtypes: int64(784)
memory usage: 167.5 MB
```

```
train.isnull().any().describe()
test.isnull().any().describe()
```

```
count       785
unique        1
top       False
freq        785
dtype: object
```

```
count       784
unique        1
top       False
freq        784
dtype: object
```

Random Forest Classifier

```
X_train = train.iloc[:,1:]
Y_train = train.iloc[:,0]
```

```
rfc = RandomForestClassifier(random_state=42, n_jobs=-1)
start = datetime.now()
rfc.fit(X_train, Y_train)
end = datetime.now()
print('Model Fit Timer:', end-start)
```

```
RandomForestClassifier(n_jobs=-1, random_state=42)
```

```
Model Fit Timer: 0:00:07.676975
```

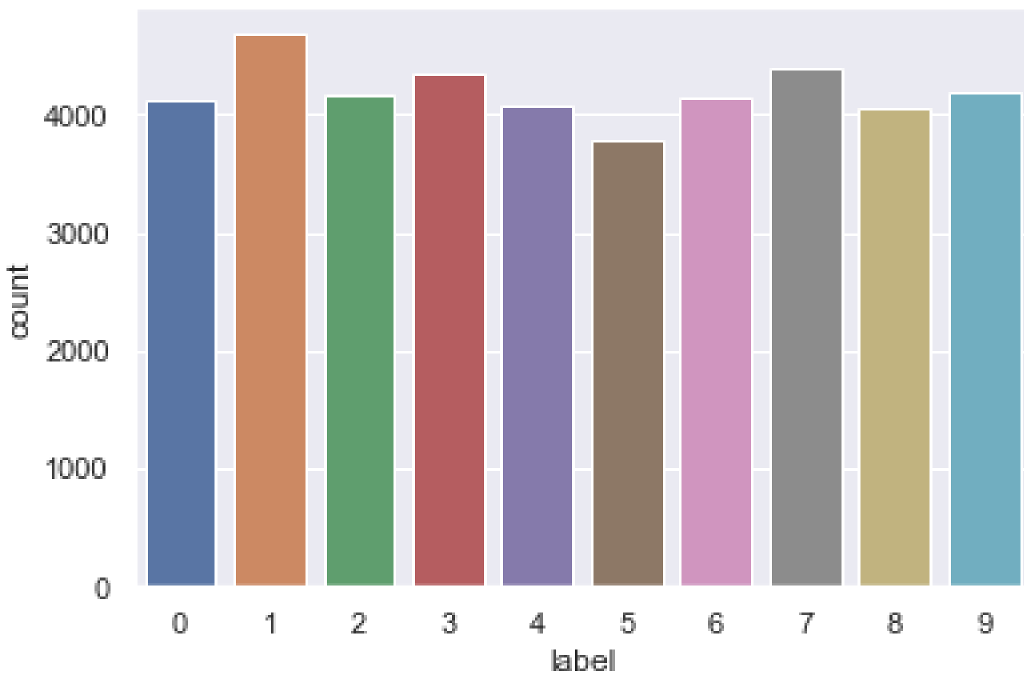
```
yhat = rfc.predict(test)
rfc_df = pd.DataFrame()
rfc_df['ImageID'] = list(range(1,28001))
rfc_df['Label'] = yhat
rfc_df[['ImageID', 'Label']].to_csv('MNIST_RFC.CSV', index=False)
```

PCA

```
pca_train = train.drop('label', axis=1)

label_train = train.label

pca_alldata = pd.concat([pca_train, test], ignore_index=True)
pca_alldata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 70000 entries, 0 to 69999
Columns: 784 entries, pixel0 to pixel783
dtypes: int64(784)
memory usage: 418.7 MB
```

```
L = sns.countplot(label_train)
```



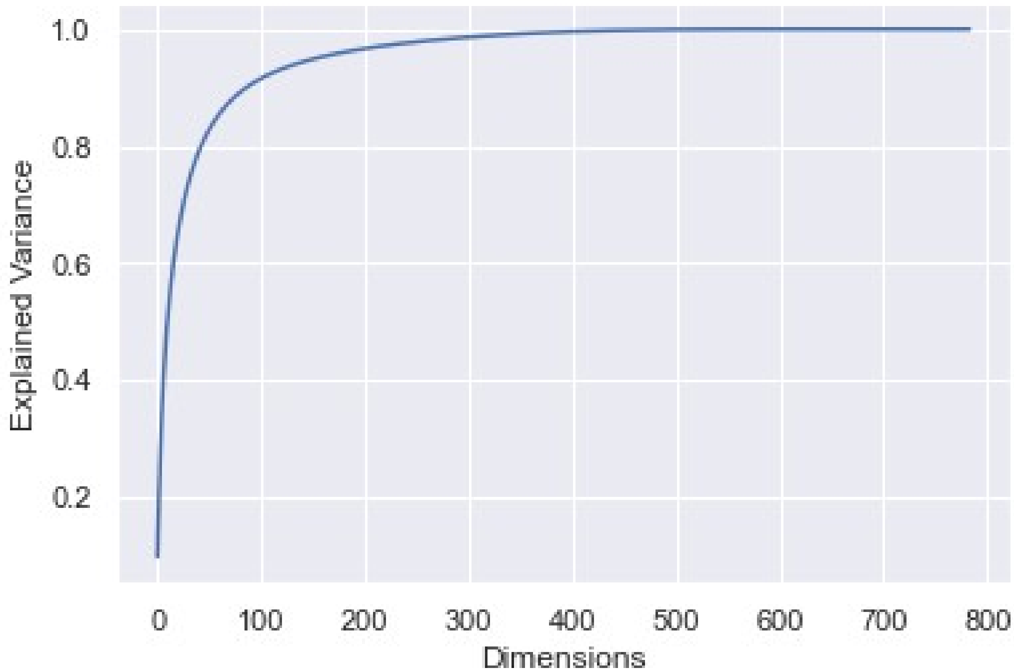

```
pca = PCA()
pca.fit(pca_alldata);
cumsum = np.cumsum(pca.explained_variance_ratio_)

plt.plot(cumsum)
plt.xlabel('Dimensions')
plt.ylabel ('Explained Variance')
```

```
PCA()
```

```
[<matplotlib.lines.Line2D at 0x2491076cdc0>]
```

```
Text(0.5, 0, 'Dimensions')
```

```
Text(0, 0.5, 'Explained Variance')
```




```
pca = PCA(n_components=0.95)
start = datetime.now()
x_reduced = pca.fit_transform(pca_alldata)
end = datetime.now()

print('Principal Component Timer: ', end-start)
print('# of Principal Components: ', pca.n_components)
```

```
Principal Component Timer:  0:00:07.798044
# of Principal Components:  0.95
```

```
x_train_pca = x_reduced[:42000]
x_test_pca = x_reduced[42000:]
```

```
len(x_train_pca)
len(x_test_pca)
```

```
42000
```

```
28000
```

```
rfc_2 = RandomForestClassifier(random_state=40, n_jobs=-1)
start = datetime.now()
rfc_2.fit(x_train_pca, label_train)
end = datetime.now()
print('Model Fit Timer: ', end-start)
```

```
RandomForestClassifier(n_jobs=-1, random_state=40)
```

```
Model Fit Timer:  0:00:18.979521
```

K-Means Clustering

```
from keras.datasets import mnist

(X_train, y_train), (X_test, y_test) = mnist.load_data()

print(type(X_train))
print(type(X_test))
print(type(y_train))
print(type(y_test))
```

```
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
```

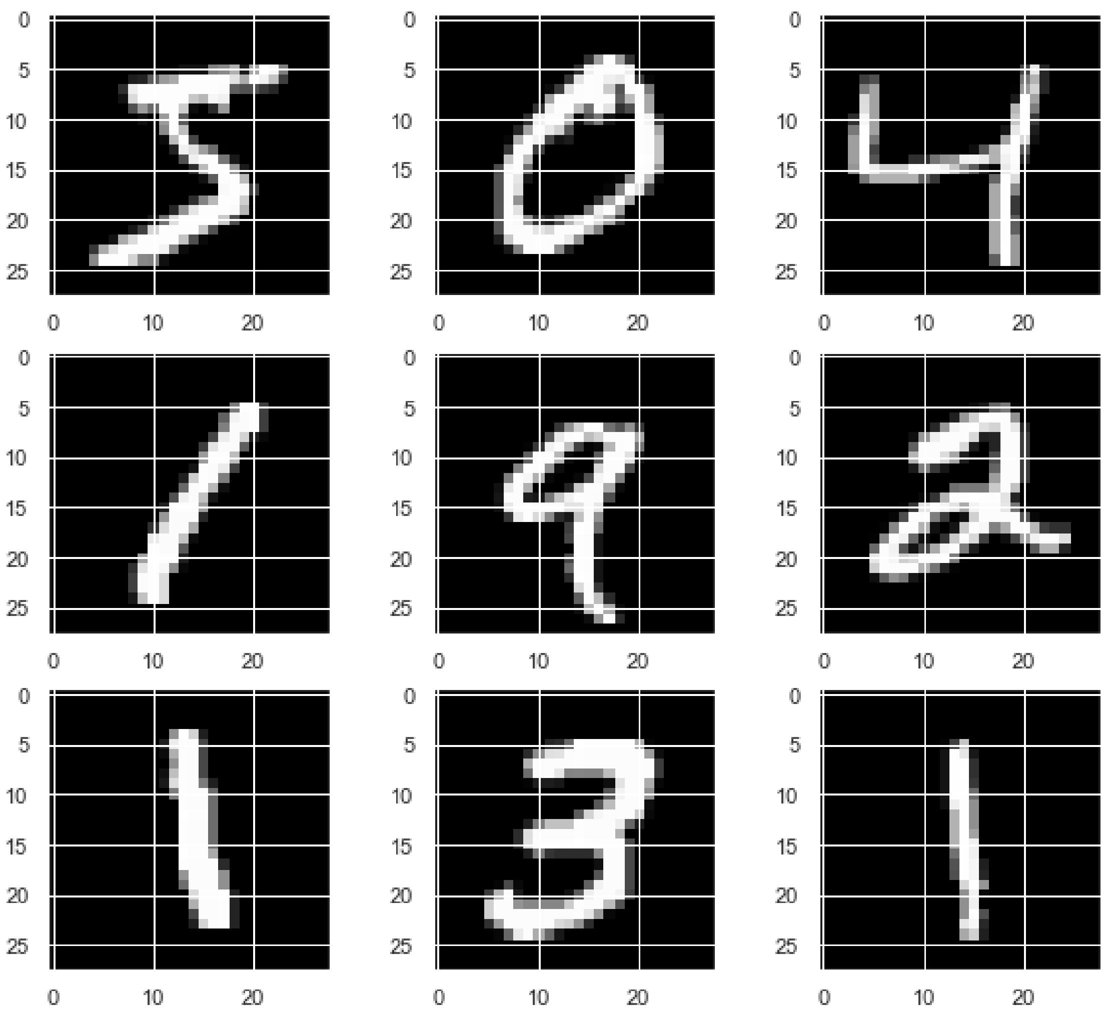
```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(60000, 28, 28)
(10000, 28, 28)
(60000,)
(10000,)
```

```
plt.gray()

plt.figure(figsize = (10,9))

for i in range(9):
    plt.subplot(3,3,i+1)
    plt.imshow(X_train[i])
```

```
<Figure size 720x648 with 0 Axes>

<AxesSubplot:>

<matplotlib.image.AxesImage at 0x24954026190>

<AxesSubplot:>

<matplotlib.image.AxesImage at 0x24953ff9c70>

<AxesSubplot:>

<matplotlib.image.AxesImage at 0x24954086e20>

<AxesSubplot:>

<matplotlib.image.AxesImage at 0x249540c15e0>
```

```
<AxesSubplot:>

<matplotlib.image.AxesImage at 0x249540f3d00>

<AxesSubplot:>

<matplotlib.image.AxesImage at 0x2495412d3d0>

<AxesSubplot:>

<matplotlib.image.AxesImage at 0x24954110e20>

<AxesSubplot:>

<matplotlib.image.AxesImage at 0x2495419b190>

<AxesSubplot:>

<matplotlib.image.AxesImage at 0x249541c6970>

<Figure size 432x288 with 0 Axes>
```



```python
print(X_train.min())
print(X_train.max())
```

```
0
255
```

```python
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

# Normalization
X_train = X_train/255.0
X_test = X_test/255.0

print(X_train.min())
print(X_train.max())
```

```
0.0
1.0
```

```python
X_train = X_train.reshape(len(X_train), -1)
X_test = X_test.reshape(len(X_test), -1)

print(X_train.shape)
print(X_test.shape)
```

```
(60000, 784)
(10000, 784)
```

```python
total_clusters = len(np.unique(y_test))

kmeans = MiniBatchKMeans(n_clusters = total_clusters)

kmeans.fit(X_train)
```

```
MiniBatchKMeans(n_clusters=10)
```

```python
kmeans.labels_
```

```
array([9, 5, 1, ..., 9, 8, 0])
```

```python
def retrieve_info(cluster_labels, y_train):
    reference_labels = {}

    for i in range(len(np.unique(kmeans.labels_))):

        index = np.where(cluster_labels == i, 1, 0)
        num = np.bincount(y_train[index == 1]).argmax()
        reference_labels[i] = num

    return reference_labels

reference_labels = retrieve_info(kmeans.labels_,y_train)
```

```
number_labels = np.random.rand(len(kmeans.labels_))

for i in range(len(kmeans.labels_)):
    number_labels[i] = reference_labels[kmeans.labels_[i]]

print(number_labels[:20].astype('int'))
print(y_train[:20])
```

```
[8 0 4 1 9 6 1 8 1 4 3 1 3 6 1 4 6 4 1 4]
[5 0 4 1 9 2 1 3 1 4 3 5 3 6 1 7 2 8 6 9]
```

```
print(accuracy_score(number_labels,y_train))
```

```
0.50905
```

```
def calculate_metric(model, output):
    print('# of clusters: {}'.format(model.n_clusters))
    print('inertia: {}'.format(model.inertia_))
    print('Homogeneity: {}'.format(metrics.homogeneity_score(output,
model.labels_)))

cluster_number = [10,16,36,64,144,256]
for i in cluster_number:
    total_clusters = len(np.unique(y_test))

    # Initialize the K-Means model
    kmeans = MiniBatchKMeans(n_clusters = i)
    # Fitting the model to training set
    kmeans.fit(X_train)
    # Calculating the metrics
    calculate_metric(kmeans,y_train)
    # Calculating reference_labels
    reference_labels = retrieve_info(kmeans.labels_,y_train)
    # ‘number_labels’ is a list which denotes the number displayed in
image
    number_labels = np.random.rand(len(kmeans.labels_))

    for i in range(len(kmeans.labels_)):
        number_labels[i] = reference_labels[kmeans.labels_[i]]

    print('Accuracy score :
{}'.format(accuracy_score(number_labels,y_train)))
    print('\n')
```

```
MiniBatchKMeans(n_clusters=10)
```

```
# of clusters: 10
inertia: 2382455.0
Homogeneity: 0.43846346576260187
Accuracy score : 0.5468
```

```
MiniBatchKMeans(n_clusters=16)

# of clusters: 16
inertia: 2201185.75
Homogeneity: 0.5622573605986861
Accuracy score : 0.6574
```

```
MiniBatchKMeans(n_clusters=36)

# of clusters: 36
inertia: 1954811.0
Homogeneity: 0.6837853589265859
Accuracy score : 0.7531333333333333
```

```
MiniBatchKMeans(n_clusters=64)

# of clusters: 64
inertia: 1806647.0
Homogeneity: 0.7465652936222945
Accuracy score : 0.8184666666666667
```

```
MiniBatchKMeans(n_clusters=144)

# of clusters: 144
inertia: 1625480.375
Homogeneity: 0.8044457908857507
Accuracy score : 0.8692333333333333
```

```
MiniBatchKMeans(n_clusters=256)

# of clusters: 256
inertia: 1504128.25
Homogeneity: 0.8418798293600633
Accuracy score : 0.8967
```

```
kmeans = MiniBatchKMeans(n_clusters = 256)

kmeans.fit(X_test)
```

```
calculate_metric(kmeans, y_test)

reference_labels = retrieve_info(kmeans.labels_,y_test)

number_labels = np.random.rand(len(kmeans.labels_))

for i in range(len(kmeans.labels_)):
    number_labels[i] = reference_labels[kmeans.labels_[i]]

print('Accuracy Score: {}'.format(accuracy_score(number_labels,
y_test)))
print('\n')
```

```
MiniBatchKMeans(n_clusters=256)

# of clusters: 256
inertia: 241507.578125
Homogeneity: 0.859114220967313
Accuracy Score: 0.8969
```

```
centroids = kmeans.cluster_centers_

centroids.shape
```

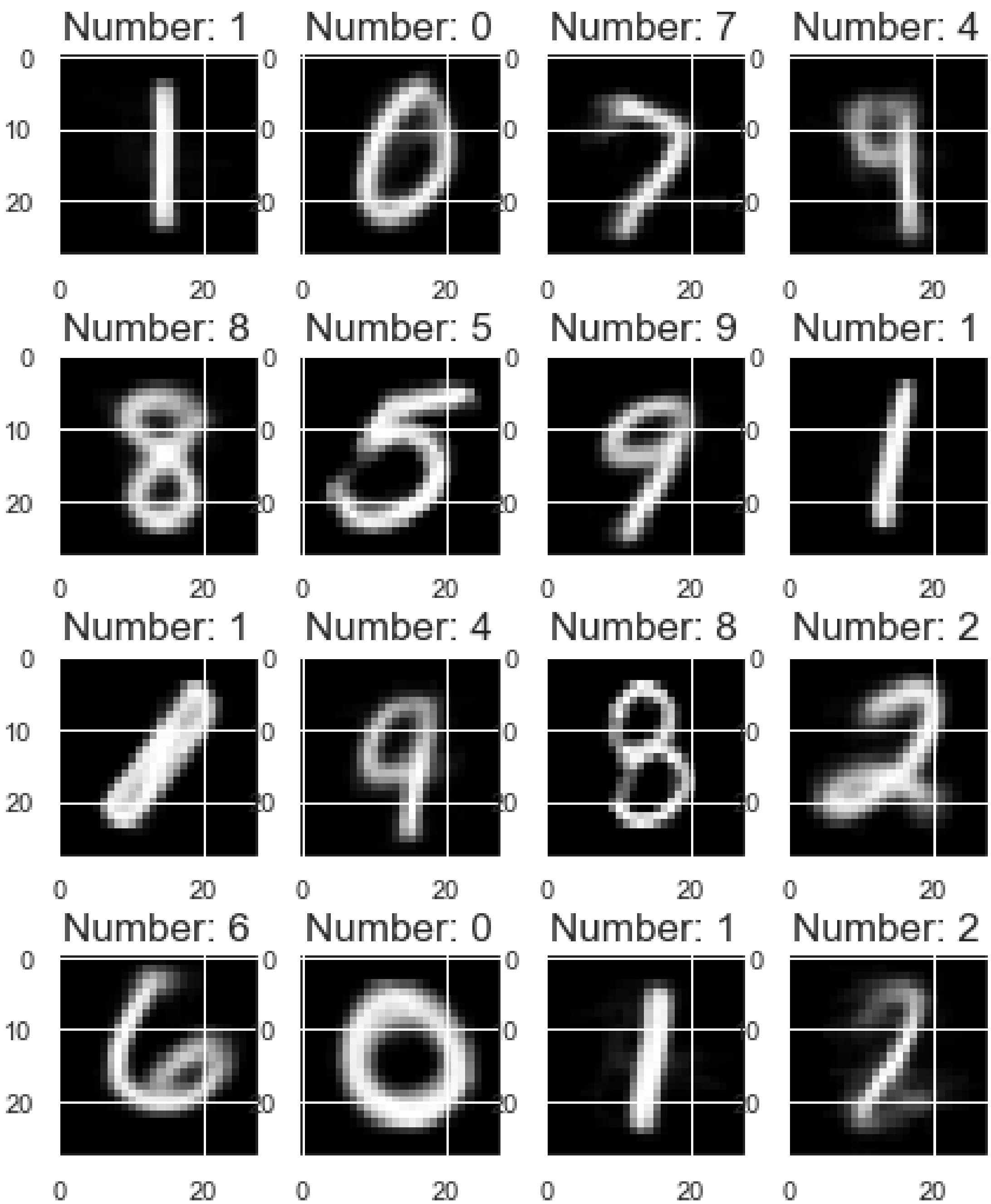
```
(256, 784)
```

```
centroids = centroids.reshape(256,28,28)

centroids = centroids*255

plt.figure(figsize = (10, 9))

bottom = 0.35

for i in range(16):
    plt.subplots_adjust(bottom)
    plt.subplot(4,4,i+1)
    plt.title('Number: {}'.format(reference_labels[i]), fontsize = 17)
    plt.imshow(centroids[i])
```

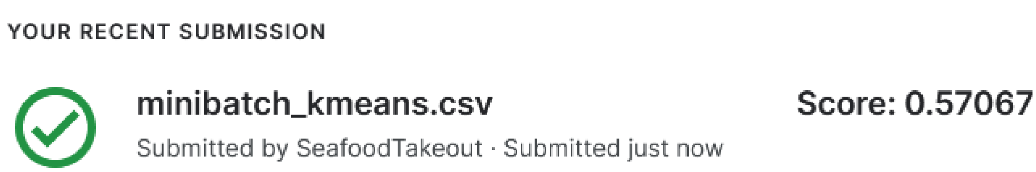
```
<Figure size 720x648 with 0 Axes>

<AxesSubplot:>

Text(0.5, 1.0, 'Number: 1')

<matplotlib.image.AxesImage at 0x2490fc31250>

<AxesSubplot:>
```

```
Text(0.5, 1.0, 'Number: 0')

<matplotlib.image.AxesImage at 0x24950bdb610>

<AxesSubplot:>

Text(0.5, 1.0, 'Number: 7')

<matplotlib.image.AxesImage at 0x24950c203a0>

<AxesSubplot:>

Text(0.5, 1.0, 'Number: 4')

<matplotlib.image.AxesImage at 0x24950c5d5b0>

<AxesSubplot:>

Text(0.5, 1.0, 'Number: 8')

<matplotlib.image.AxesImage at 0x24950c89ee0>

<AxesSubplot:>

Text(0.5, 1.0, 'Number: 5')

<matplotlib.image.AxesImage at 0x24950cc5640>

<AxesSubplot:>

Text(0.5, 1.0, 'Number: 9')

<matplotlib.image.AxesImage at 0x24950cf3dc0>

<AxesSubplot:>

Text(0.5, 1.0, 'Number: 1')

<matplotlib.image.AxesImage at 0x24950d30430>

<AxesSubplot:>

Text(0.5, 1.0, 'Number: 1')

<matplotlib.image.AxesImage at 0x24950d5d9d0>

<AxesSubplot:>

Text(0.5, 1.0, 'Number: 4')

<matplotlib.image.AxesImage at 0x24950d9c130>

<AxesSubplot:>

Text(0.5, 1.0, 'Number: 8')
```

```
<matplotlib.image.AxesImage at 0x24950dc9a00>

<AxesSubplot:>

Text(0.5, 1.0, 'Number: 2')

<matplotlib.image.AxesImage at 0x24950e0b1c0>

<AxesSubplot:>

Text(0.5, 1.0, 'Number: 6')

<matplotlib.image.AxesImage at 0x24950e329d0>

<AxesSubplot:>

Text(0.5, 1.0, 'Number: 0')

<matplotlib.image.AxesImage at 0x24950e53040>

<AxesSubplot:>

Text(0.5, 1.0, 'Number: 1')

<matplotlib.image.AxesImage at 0x24950e9e670>

<AxesSubplot:>

Text(0.5, 1.0, 'Number: 2')

<matplotlib.image.AxesImage at 0x24950eccdf0>
```




```
from IPython.display import Image
Image(filename='Kaggle_Submission.png')
```



Conclusion

The models run above were fairly successful at predicting labels. Using K-Means clustering we could see that the 256-means clustering classifier yielded the highest accuracy score on the training set. It yielded a similar score on the test set of about 0.89. This indicates that the model was not overfitting to the training data. Visualizing the predictive capabilities was difficult to perform, however, we can see that the 256-mean clustering classifer did a relatively good job at predicting the number. However, we can see that a few numbers were improperly labelled. This can be do to a variety of factors such as shape or orientation of the image. Perhaps this can be remedied by increasing the number of clusters to account for this variation.