Nearest Neighbor on MNIST Dataset

Note: Whenever you see '...', replace with a line of code

In [1]:
```
import numpy as np
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
import random
```

In [2]:
```
train_data = pd.read_csv('mnist-in-csv/mnist_train_1.csv')
train_data = train_data.append(pd.read_csv('mnist-in-csv/mnis
t_train_2.csv')).reset_index(drop=True)
test_data = pd.read_csv('mnist-in-csv/mnist_test.csv')
```

In [3]:
```
train_data.head()
```

Out[3]:

	label	1x1	1x2	1x3	1x4	1x5	1x6	1x7	1x8	1x9	...	28x19	28x20	28x21	28x
0	5	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	4	0	0	0	0	0	0	0	0	0	...	0	0	0	
3	1	0	0	0	0	0	0	0	0	0	...	0	0	0	
4	9	0	0	0	0	0	0	0	0	0	...	0	0	0	

5 rows × 785 columns

In [4]:
```
X_train = train_data[train_data.columns[1:]]
X_test = test_data[test_data.columns[1:]]
y_train = train_data['label']
y_test = test_data['label']
```

Using the PCA class from sklearn, fit to your training data and transform your train and test set accordingly. Each data set should only have 30 elements per data sample after transforming via PCA. Hint: your train_data should have the shape (60000, 30). Check this with the following command: X_train_pca.shape

In [5]:
```
X_train.head()
```

Out[5]:

	1x1	1x2	1x3	1x4	1x5	1x6	1x7	1x8	1x9	1x10	...	28x19	28x20	28x21	28x
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 [6]:
```
pca = PCA(n_components=30)  # Initialize a PCA object that wi
ll cut our dimensions to 30
```

In [7]:
```
%time
### Fit PCA to our training data, then transform X_train and
X_test
pca.fit(X_train)
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)
```

```
CPU times: user 1 µs, sys: 1 µs, total: 2 µs
Wall time: 2.38 µs
```

Implement a KNN classifier using the KNeighborsClassifier from sklearn. Fit to your train_pca data, then generate predictions on your test_pca data.

Note: Generating predictions will talk approximately 20-25 seconds when you have 30 components via PCA.

In [8]:
```
knn = KNeighborsClassifier()  # Intialize KNeighborsClassifie
r with a value of k <= 7
```

In [9]:
```
knn.fit(X_train_pca, y_train)   # Fit knn classifier to X_tra
in_pca
y_pred = knn.predict(X_test_pca)   # Generate predictions bas
ed on X_test_pca
```

In [10]:
```
sum(y_pred == y_test) / len(y_pred)  # Outputs the accuracy s
core for your model
```

Out[10]:
```
0.9751
```

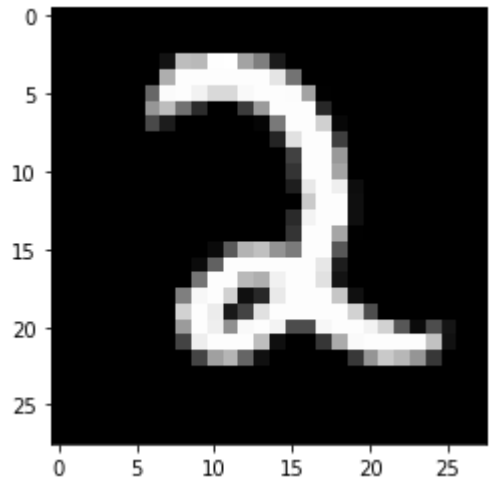
Note that there are two parameters you can change in this process that will affect model accuracy: Number of components for PCA, and the value of k in the KNeighbors Classifier. Try to adjust these parameters to achieve optimal model accuracy.

Keep in mind that the runtime of kNN is approximately linear as you increase the number of components in PCA. Ideally, PCA should have <= 100 components to avoid long runtimes.

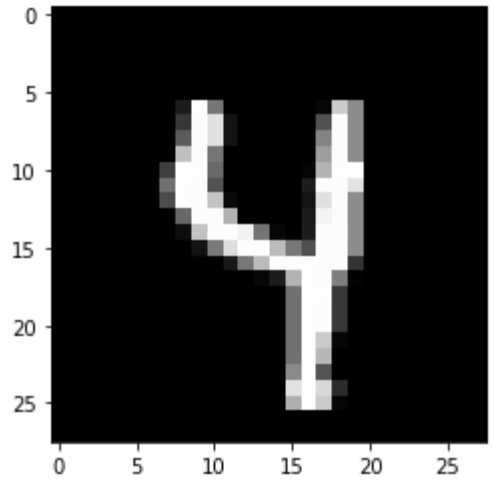
Visualization

The below cell will plot 10 examples from our test set at random, and print the corresponding prediction our model made.

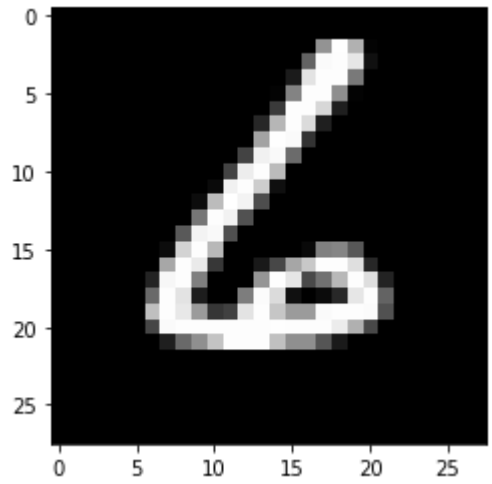
In [12]:
```
indices = random.choices(range(len(X_test)), k=10)
for i in indices:
    plt.imshow(X_test.loc[i].values.reshape((28,28)), cmap='g
ray')
    plt.show()
    print("Prediction:", y_pred[i])
    print("Correct:", y_test.loc[i])
    print()
```



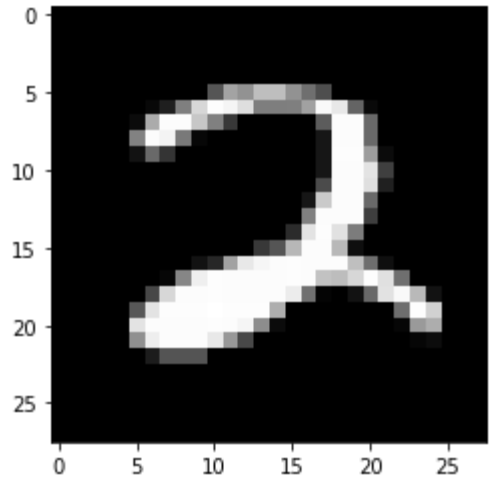
```
Prediction: 2
Correct: 2
```



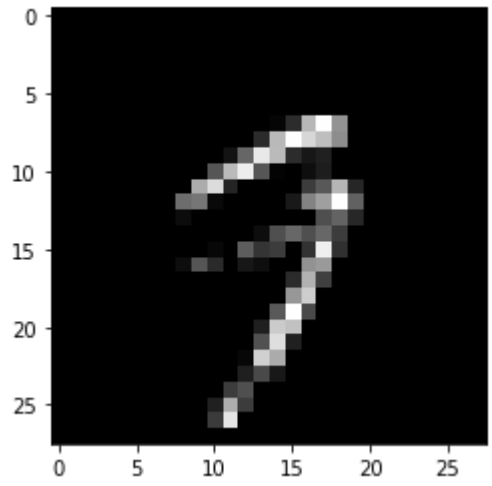
```
Prediction: 4
Correct: 4
```



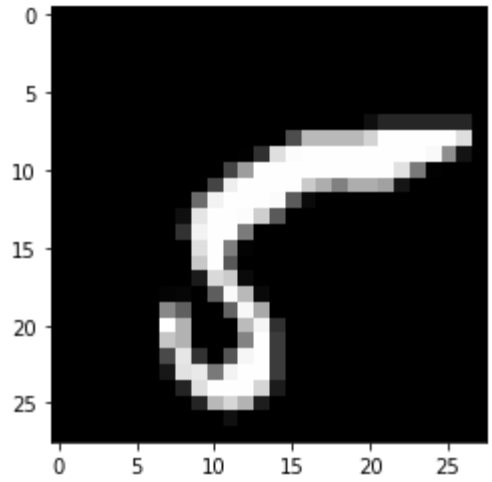
```
Prediction: 6
Correct: 6
```



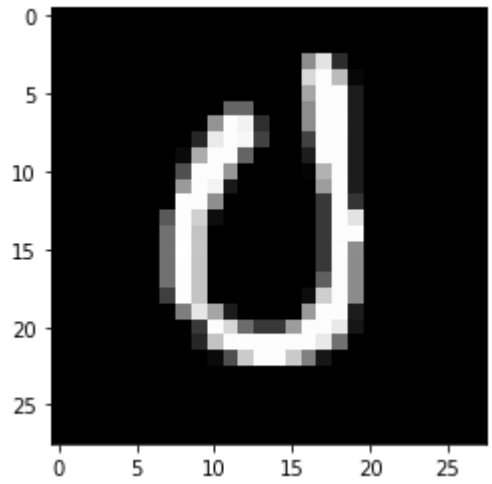
```
Prediction: 2
Correct: 2
```



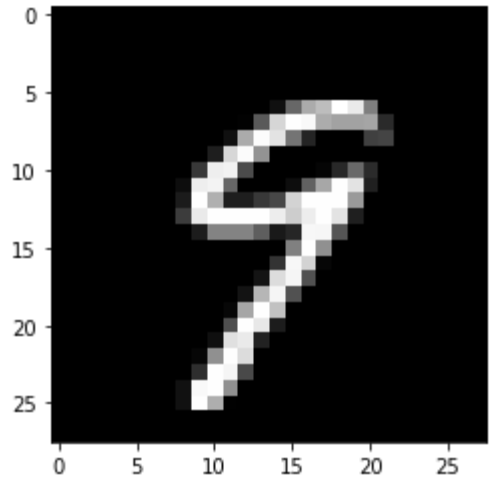
```
Prediction: 7
Correct: 9
```



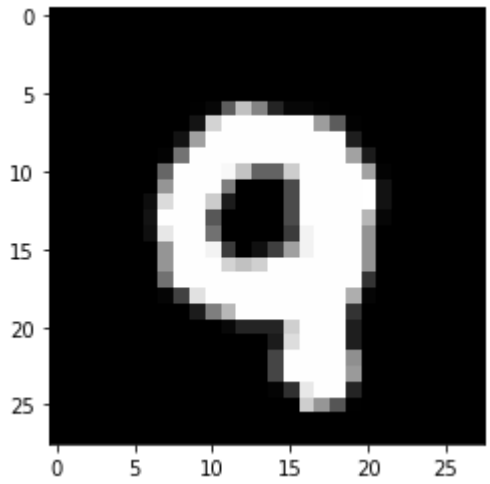
```
Prediction: 5
Correct: 5
```



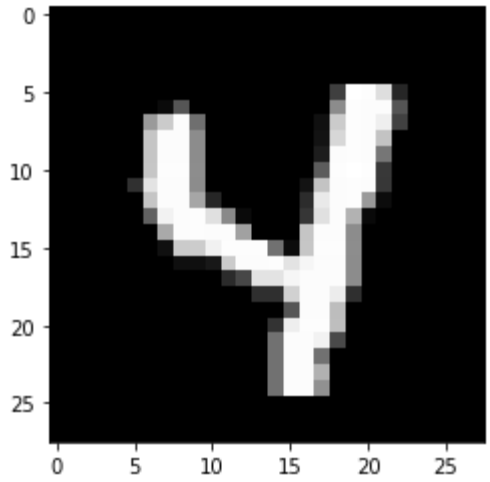
```
Prediction: 0
Correct: 0
```



```
Prediction: 9
Correct: 9
```



```
Prediction: 9
Correct: 9
```



```
Prediction: 4
Correct: 4
```

In [ ]:

In [ ]:

Optional:

Implement Nearest Neighbor Classifier. Using Euclidean distance, find the closest point in the training set to each row in the test set, and return the class of the closest point.

Hint: Use np.argmin() to find the index of the smallest value in an array.

In [13]:
```
%time y_pred = [y_train[np.sum(((X_train_pca - i) ** 2), axis
=1).argmin()] for i in X_test_pca]
```

```
CPU times: user 39.4 s, sys: 7.43 ms, total: 39.4 s
Wall time: 39.4 s
```

In [14]:
```
sum(np.array(y_pred) == y_test.values) / len(y_pred)
```

Out[14]:
```
0.972
```