Importing Important libraries

In [ ]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import linear_model
from sklearn.model_selection import train_test_split
import gc
from sklearn import svm
from sklearn import metrics
from sklearn.preprocessing import scale
import cv2
from sklearn.preprocessing import scale
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.svm import SVC
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
```

In [ ]:

```
# read the dataset
data = pd.read_csv("train.csv")
data.info()
```

```
<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
```

In [ ]:

```
data.head()
```

Out[ ]:

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	pixel11	pixel12	pixel13	pixel14
0	1	0	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	0	0	0
2	1	0	0	0	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	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

5 rows × 785 columns

In [ ]:

```
four = data.iloc[3, 1:]
four.shape
```

Out[ ]:

```
(784,)
```

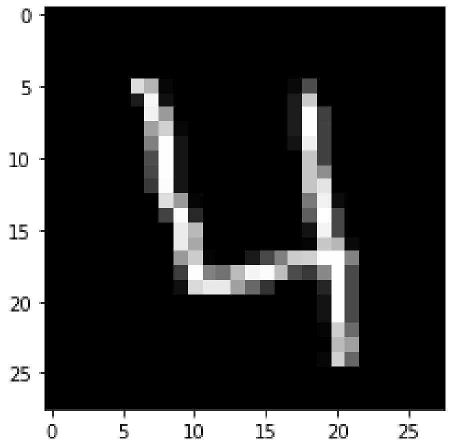
In [ ]:

```
four = four.values.reshape(28, 28)
```

```
plt.imshow(four, cmap='gray')
```

Out[ ]:

```
<matplotlib.image.AxesImage at 0x7f12444ffd68>
```



In [ ]:

```
# Summarise the counts of 'label' to see how many labels of each digit are present
data.label.value_counts()
```

Out[ ]:

```
1    4684
7    4401
3    4351
9    4188
2    4177
6    4137
0    4132
4    4072
8    4063
5    3795
Name: label, dtype: int64
```

In [ ]:

```
100*(round(data.label.astype('category').value_counts()/len(data.index), 4))
```

Out[ ]:

```
1    11.15
7    10.48
3    10.36
9     9.97
2     9.95
6     9.85
0     9.84
4     9.70
8     9.67
5     9.04
Name: label, dtype: float64
```

Check for missing Values

In [ ]:

```
# missing values - there are none
data.isnull().sum()
```

Out[ ]:

```
label       0
pixel0      0
pixel1      0
pixel2      0
pixel3      0
```

```
              ..
pixel779      1
pixel780      1
pixel781      1
pixel782      1
pixel783      1
Length: 785, dtype: int64
```

Visualizing the data

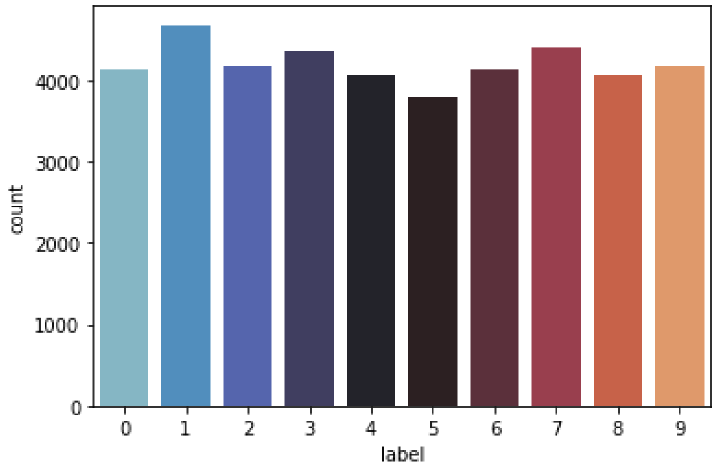
In [ ]:

```python
#visualising the column - label
sns.countplot(data['label'],palette = 'icefire')
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1244335dd8>
```

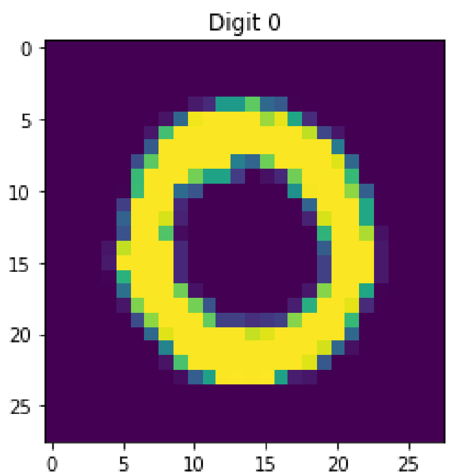


In [ ]:

```python
zero = data.iloc[1, 1:]
zero = zero.values.reshape(28,28)
plt.imshow(zero)
plt.title("Digit 0")
```

Out[ ]:

```
Text(0.5, 1.0, 'Digit 0')
```

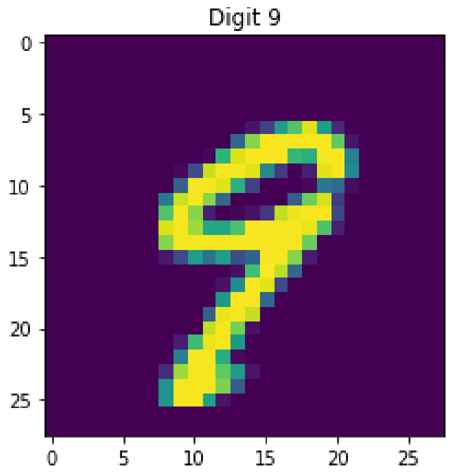


In [ ]:

```python
nine = data.iloc[11, 1:]
nine = nine.values.reshape(28,28)
plt.imshow(nine)
plt.title("Digit 9")
```

Out[ ]:

```
Text(0.5, 1.0, 'Digit 9')
```



In [ ]:

```python
data.describe()
```

Out[ ]:

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	pixel
count	12043.000000	12043.0	12043.0	12043.0	12043.0	12043.0	12043.0	12043.0	12043.0	12043.0	12043.0	12043.0	12042
mean	4.438180	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	(
std	2.877195	0.0	0.0	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	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	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	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	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	0.0	0.0	(

8 rows × 785 columns

Spliting the data

In [ ]:

```python
# Creating training and test sets
# Splitting the data into train and test
X = data.iloc[:, 1:]
Y = data.iloc[:, 0]

# Rescaling the features
X = scale(X)
# train test split with train_size=20% and test size=80%
x_train, x_test, y_train, y_test = train_test_split(X, Y, train_size=0.20, random_state=
101)
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(8400, 784)
(33600, 784)
```

```
(8400,)
(33600,)
```

Model Building

In [ ]:

```
svm_linear = svm.SVC(kernel='linear')
# fit
svm_linear.fit(x_train, y_train)
```

Out[ ]:

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

In [ ]:

```
# predict
predictions = svm_linear.predict(x_test)
predictions[:10]
```

Out[ ]:

```
array([1, 3, 0, 0, 1, 4, 1, 5, 0, 6])
```

In [ ]:

```
y_pred = svm_linear.predict(x_test)
```

In [ ]:

```
# confusion matrix and accuracy, precision, recall

# accuracy
print("accuracy:", metrics.accuracy_score(y_true=y_test, y_pred=y_pred), "\n")

# cm
print(metrics.confusion_matrix(y_true=y_test, y_pred=y_pred))
```

```
accuracy: 0.913125

[[3188    0   10    5   11   20   32    3   15    1]
 [   0 3677   14   11    5    7    4    8   30    4]
 [  36   29 3027   54   55   10   30   42   48   12]
 [  13   12  104 3051    9  181    5   21   54   25]
 [   8   14   33    2 3057    4   25   31    6  110]
 [  30   23   29  136   44 2622   44   12   72   27]
 [  26   11   44    4   28   33 3113    0   18    0]
 [   7   24   36   19   59    9    2 3210    4  134]
 [  13   46   50  120   21  110   30   18 2843   21]
 [  19   17   21   22  172   20    4  161   26 2893]]
```

In [ ]:

```
# class-wise accuracy
score = metrics.classification_report(y_true=y_test, y_pred=predictions)
print(score)
```

```
              precision    recall  f1-score   support

           0       0.95      0.97      0.96      3285
           1       0.95      0.98      0.97      3760
           2       0.90      0.91      0.90      3343
           3       0.89      0.88      0.88      3475
           4       0.88      0.93      0.91      3290
           5       0.87      0.86      0.87      3039
           6       0.95      0.95      0.95      3277
           7       0.92      0.92      0.92      3504
```

```
           8       0.91      0.87      0.89      3272
           9       0.90      0.86      0.88      3355

    accuracy                           0.91     33600
   macro avg       0.91      0.91      0.91     33600
weighted avg       0.91      0.91      0.91     33600
```

Using Grid Search Cross-Validation(K-5)

In [ ]:

```python
# creating a KFold object with 5 splits
folds = KFold(n_splits = 5, shuffle = True, random_state = 101)
# Set the parameters by cross-validation
hyper_params = [ {'gamma': [0.01, 0.001,0.0001],
                     'C': [1, 10, 100]}]
model = SVC(kernel="rbf")
# GridSearchCV()
model_cv = GridSearchCV(estimator = model,
                        param_grid = hyper_params,
                        scoring= 'accuracy',
                        cv = folds,
                        verbose = 1,
                        return_train_score=True,n_jobs = -1)
model_cv.fit(x_train, y_train)
```

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  45 out of  45 | elapsed: 35.8min finished
```

Out[ ]:

```
GridSearchCV(cv=KFold(n_splits=5, random_state=101, shuffle=True),
             error_score=nan,
             estimator=SVC(C=1.0, break_ties=False, cache_size=200,
                           class_weight=None, coef0=0.0,
                           decision_function_shape='ovr', degree=3,
                           gamma='scale', kernel='rbf', max_iter=-1,
                           probability=False, random_state=None, shrinking=True,
                           tol=0.001, verbose=False),
             iid='deprecated', n_jobs=-1,
             param_grid=[{'C': [1, 10, 100], 'gamma': [0.01, 0.001, 0.0001]}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='accuracy', verbose=1)
```

In [ ]:

```python
cv_results = pd.DataFrame(model_cv.cv_results_)
cv_results
```

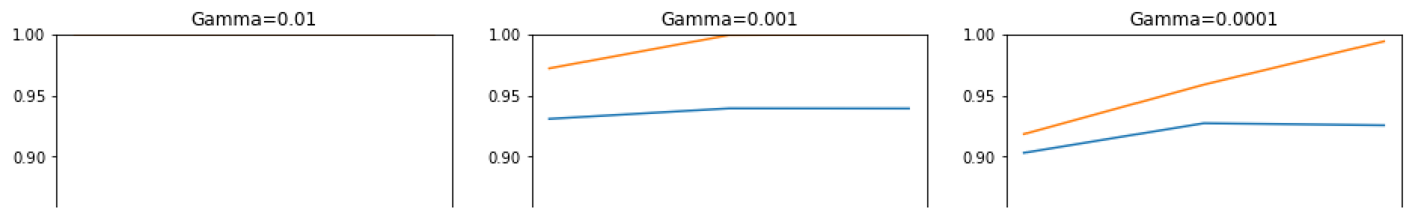
Out[ ]:

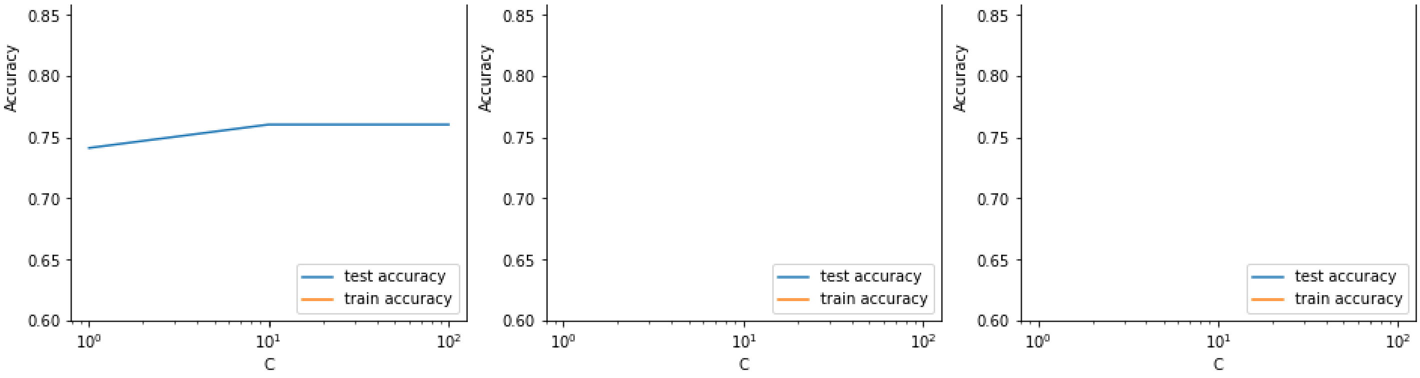
	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	param_gamma	params	split0_test_score	split1_
0	97.649932	2.359833	13.723245	0.060236	1	0.01	{'C': 1, 'gamma': 0.01}	0.752381	
1	23.584376	0.317803	7.792313	0.070050	1	0.001	{'C': 1, 'gamma': 0.001}	0.935119	
2	33.979033	0.310656	10.550277	0.090045	1	0.0001	{'C': 1, 'gamma': 0.0001}	0.910119	
3	98.809526	0.408492	13.668795	0.041593	10	0.01	{'C': 10, 'gamma': 0.01}	0.766071	
							{'C': 10,		

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	param_gamma	params	split0_test_score	split1_
4	20.606161	0.124974	7.027292	0.043067	10	0.001	{'C': 10, 'gamma': 0.001}	0.941071	
5	15.902091	0.154777	6.515077	0.037508	10	0.0001	{'C': 10, 'gamma': 0.0001}	0.933929	
6	102.150486	1.958088	13.824061	0.067490	100	0.01	{'C': 100, 'gamma': 0.01}	0.766071	
7	20.625948	0.282883	7.010906	0.045648	100	0.001	{'C': 100, 'gamma': 0.001}	0.939881	
8	11.971935	0.512912	5.211169	0.197254	100	0.0001	{'C': 100, 'gamma': 0.0001}	0.929762	

In [ ]:

```
cv_results['param_C'] = cv_results['param_C'].astype('int')
plt.figure(figsize=(16,6))

# subplot 1/3
plt.subplot(131)
gamma_01 = cv_results[cv_results['param_gamma']==0.01]

plt.plot(gamma_01["param_C"], gamma_01["mean_test_score"])
plt.plot(gamma_01["param_C"], gamma_01["mean_train_score"])
plt.xlabel('C')
plt.ylabel('Accuracy')
plt.title("Gamma=0.01")
plt.ylim([0.60, 1])
plt.legend(['test accuracy', 'train accuracy'], loc='lower right')
plt.xscale('log')

# subplot 2/3
plt.subplot(132)
gamma_001 = cv_results[cv_results['param_gamma']==0.001]

plt.plot(gamma_001["param_C"], gamma_001["mean_test_score"])
plt.plot(gamma_001["param_C"], gamma_001["mean_train_score"])
plt.xlabel('C')
plt.ylabel('Accuracy')
plt.title("Gamma=0.001")
plt.ylim([0.60, 1])
plt.legend(['test accuracy', 'train accuracy'], loc='lower right')
plt.xscale('log')


# subplot 3/3
plt.subplot(133)
gamma_0001 = cv_results[cv_results['param_gamma']==0.0001]

plt.plot(gamma_0001["param_C"], gamma_0001["mean_test_score"])
plt.plot(gamma_0001["param_C"], gamma_0001["mean_train_score"])
plt.xlabel('C')
plt.ylabel('Accuracy')
plt.title("Gamma=0.0001")
plt.ylim([0.60, 1])
plt.legend(['test accuracy', 'train accuracy'], loc='lower right')
plt.xscale('log')

plt.show()
```





In [ ]:

```
best_score = model_cv.best_score_
best_hyperparams = model_cv.best_params_

print("The best test score is {0} corresponding to hyperparameters {1}".format(best_score
, best_hyperparams))
```

```
The best test score is 0.9394047619047619 corresponding to hyperparameters {'C': 10, 'gam
ma': 0.001}
```

Now Final Model

In [ ]:

```
# optimal hyperparameters
best_C = 10
best_gamma = 0.001

# model
svm_final = svm.SVC(kernel='rbf', C=best_C, gamma=best_gamma)

# fit
svm_final.fit(x_train, y_train)
```

Out[ ]:

```
SVC(C=10, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.001, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

In [ ]:

```
predictions = svm_final.predict(x_test)
confusion = metrics.confusion_matrix(y_true = y_test, y_pred = predictions)

# measure accuracy
test_accuracy = metrics.accuracy_score(y_true=y_test, y_pred=predictions)

print(test_accuracy, "\n")
print(confusion)
```

```
0.9477083333333334

[[3211    0   19    2    2   12   26    3    8    2]
 [   0 3692   26    9    6    3    5    9    7    3]
 [  13   12 3165   29   29    6   20   40   21    8]
 [   4    5   77 3232    4   79    1   23   31   19]
 [   5    8   42    1 3117    5   20   19    9   64]
 [  15    8   33   61   15 2815   35   11   31   15]
 [  19    5   44    1   12   18 3167    1   10    0]
 [   5   17   52   12   29    4    1 3322    4   58]
 [   7   16   42   53   15   51   18   16 3044   10]
 [   9    9   33   20   81   10    0   94   21 3078]]
```

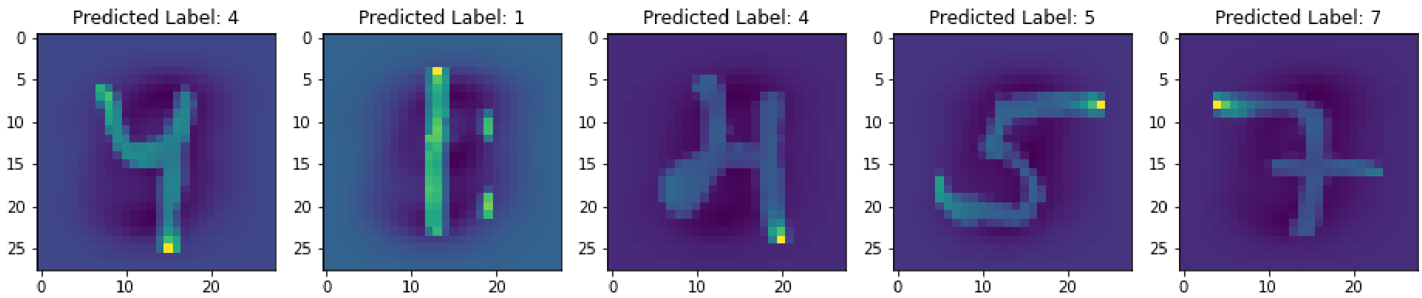
In [ ]:

```
scores=metrics.classification_report(y_test, y_pred, labels=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
print(scores)
```

```
              precision    recall  f1-score   support

           0       0.95      0.97      0.96      3285
           1       0.95      0.98      0.97      3760
           2       0.90      0.91      0.90      3343
           3       0.89      0.88      0.88      3475
           4       0.88      0.93      0.91      3290
           5       0.87      0.86      0.87      3039
           6       0.95      0.95      0.95      3277
           7       0.92      0.92      0.92      3504
           8       0.91      0.87      0.89      3272
           9       0.90      0.86      0.88      3355

    accuracy                           0.91     33600
   macro avg       0.91      0.91      0.91     33600
weighted avg       0.91      0.91      0.91     33600
```

In [ ]:

```
df = np.random.randint(1,y_pred.shape[0]+1,5)
plt.figure(figsize=(16,4))
for i,j in enumerate(df):
    plt.subplot(150+i+1)
    d = x_test[j].reshape(28,28)
    plt.title(f'Predicted Label: {y_pred[j]}')
    plt.imshow(d)
plt.show()
```



Now work with orignal test.csv

In [ ]:

```
#import file and reading few lines
test_df = pd.read_csv('test.csv')
test_df.head(10)
```

Out[ ]:

	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	pixel11	pixel12	pixel13	pixel14	pixel15
0	0	0	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	0	0	0
2	0	0	0	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	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	pixel11	pixel12	pixel13	pixel14	pixel15
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

10 rows × 784 columns

In [ ]:

```
test_df.info()
```

```
<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
```

In [ ]:

```
# scaling the features
test_scaled = scale(test_df)
```

In [ ]:

```
test_predict = svm_final.predict(test_scaled)
```

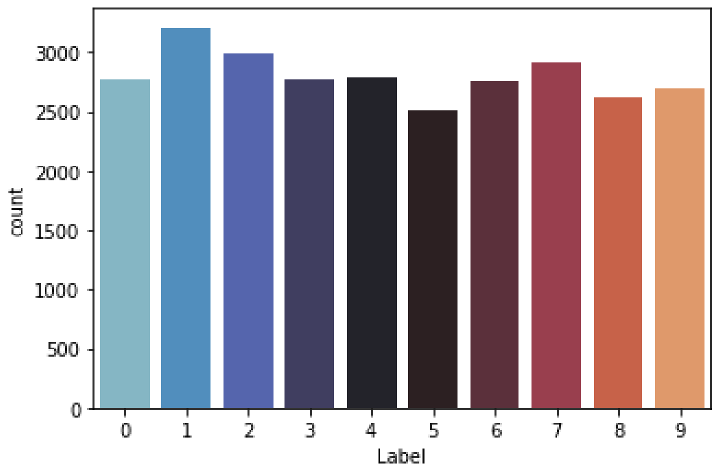
In [ ]:

```
# Plotting the distribution of prediction
a = {'ImageId': np.arange(1,test_predict.shape[0]+1), 'Label': test_predict}
data_to_export = pd.DataFrame(a)
sns.countplot(data_to_export['Label'], palette = 'icefire')
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the
following variable as a keyword arg: x. From version 0.12, the only valid positional argu
ment will be `data`, and passing other arguments without an explicit keyword will result
in an error or misinterpretation.
  FutureWarning
```

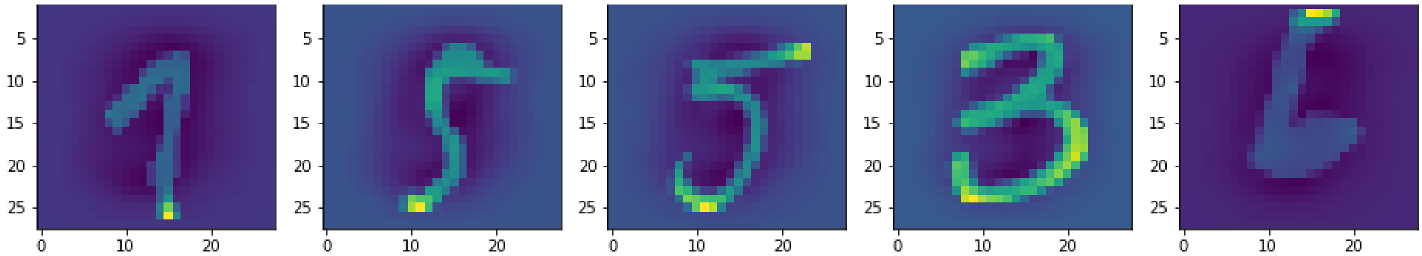
Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1244403470>
```



In [ ]:

```
df = np.random.randint(1,test_predict.shape[0]+1,5)

plt.figure(figsize=(16,4))
for i,j in enumerate(df):
    plt.subplot(150+i+1)
    d = test_scaled[j].reshape(28,28)
    plt.title(f'Predicted Label: {test_predict[j]}')
    plt.imshow(d)
plt.show()
```



Thank You