Name : Syed Riaz Ali

Email : syedriazali1997@gmail.com

Whatsapp : 923002502513

Day_25 : Logistic Regression Analysis

In [7]:
```
# Importing Libraries

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

In [8]:
```
# Importing online data

from sklearn.datasets import load_digits

digits = load_digits()
```

In [14]:
```
# Input variable / Features (X)

digits.data.shape
X = digits.data

#It means 1797 pictures with the size 64 = 8x8
```

In [15]:
```
# Output variable / Lables (y)

digits.target.shape
y = digits.target
```

In [13]:
```
sns.set_style("darkgrid")

plt.figure(figsize=(20,4))
for index, (image, label) in enumerate(zip(digits.data[0:10], digits.target[0:10])):
    plt.subplot(1, 10, index + 1)
    plt.imshow(np.reshape(image, (8,8)), cmap = plt.cm.gray)
    plt.title('Training: %i\n' % label, fontsize = 20)
```

In [16]:
```
# Splitting the data

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= 0.2, random_state= 0)
```

In [17]:
```
print("Train input data: ", X_train.shape)
print("Test input data: ", X_test.shape)
print("Train output data: ", y_train.shape)
print("Test output data: ", y_test.shape)
```

```
Train input data:  (1437, 64)
Test input data:  (360, 64)
Train output data:  (1437,)
Test output data:  (360,)
```

In [19]:
```
# Model train

from sklearn.linear_model import LogisticRegression

log_reg = LogisticRegression().fit(X_train, y_train)
log_reg
```

```
C:\Users\syedriaz\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[19]:
```
LogisticRegression()
```

In [20]:
```
log_reg.predict(X_test[0:5])
```

Out[20]:
```
array([2, 8, 2, 6, 6])
```

In [24]:
```
predictions = log_reg.predict(X_test)
predictions
```

Out[24]:
```
array([2, 8, 2, 6, 6, 7, 1, 9, 8, 5, 2, 8, 6, 6, 6, 6, 1, 0, 5, 8, 8, 7,
       8, 4, 7, 5, 4, 9, 2, 9, 4, 7, 6, 8, 9, 4, 3, 1, 9, 1, 8, 6, 7, 7,
       1, 0, 7, 6, 2, 1, 9, 6, 7, 9, 0, 0, 9, 1, 6, 3, 0, 2, 3, 4, 1, 9,
       2, 6, 9, 1, 8, 3, 5, 1, 2, 8, 2, 2, 9, 7, 2, 3, 5, 0, 9, 3, 7, 5,
       1, 2, 9, 9, 3, 1, 4, 7, 4, 8, 5, 8, 5, 5, 2, 5, 9, 0, 7, 1, 4, 7,
       3, 4, 8, 9, 7, 9, 8, 2, 1, 5, 2, 5, 8, 4, 1, 7, 0, 6, 1, 5, 5, 9,
       9, 5, 9, 9, 5, 7, 5, 6, 2, 8, 6, 9, 6, 1, 5, 1, 5, 9, 9, 1, 5, 3,
       6, 1, 8, 9, 8, 7, 8, 7, 6, 5, 6, 0, 8, 8, 9, 8, 6, 1, 0, 4, 1, 6,
       3, 8, 6, 7, 4, 9, 6, 3, 0, 3, 3, 3, 0, 7, 7, 5, 7, 8, 0, 7, 1, 9,
       6, 4, 5, 0, 1, 4, 6, 4, 3, 3, 0, 9, 5, 9, 2, 1, 4, 2, 1, 6, 8, 9,
       2, 4, 9, 3, 7, 6, 2, 3, 3, 1, 6, 9, 3, 6, 3, 3, 2, 0, 7, 6, 1, 1,
       9, 7, 2, 7, 8, 5, 5, 7, 5, 2, 3, 7, 2, 7, 5, 5, 7, 0, 9, 1, 6, 5,
       9, 7, 4, 3, 8, 0, 3, 6, 4, 6, 3, 2, 6, 8, 8, 8, 4, 6, 7, 5, 2, 4,
       5, 3, 2, 4, 6, 9, 4, 5, 4, 3, 4, 6, 2, 9, 0, 1, 7, 2, 0, 9, 6, 0,
       4, 2, 0, 7, 9, 8, 5, 7, 8, 2, 8, 4, 3, 7, 2, 6, 9, 1, 5, 1, 0, 8,
       2, 8, 9, 5, 6, 2, 2, 7, 2, 1, 5, 1, 6, 4, 5, 0, 9, 4, 1, 1, 7, 0,
       8, 9, 0, 5, 4, 3, 8, 8])
```

In [21]:
```
# Accuracy test

score = log_reg.score(X_test, y_test)
print("The accuracy score is : ", score)
```

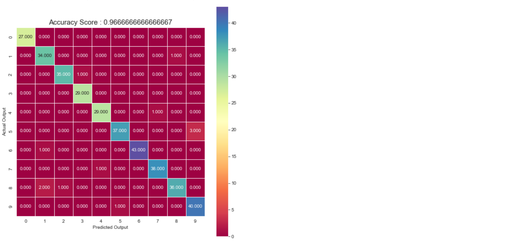
```
The accuracy score is :  0.9666666666666667
```

In [25]:
```
# Confusion Matrix

from sklearn import metrics

cm = metrics.confusion_matrix(y_test, predictions)
cm
```

Out[25]:
```
array([[27,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0, 34,  0,  0,  0,  0,  0,  0,  1,  0],
       [ 0,  0, 35,  1,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0, 29,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0, 29,  0,  0,  1,  0,  0],
       [ 0,  0,  0,  0,  0, 37,  0,  0,  0,  1],
       [ 0,  1,  0,  0,  0,  0, 43,  0,  0,  0],
       [ 0,  0,  0,  0,  1,  0,  0, 38,  0,  0],
       [ 0,  2,  1,  0,  0,  0,  0,  0, 36,  0],
       [ 0,  0,  0,  0,  1,  0,  0,  0, 40]], dtype=int64)
```

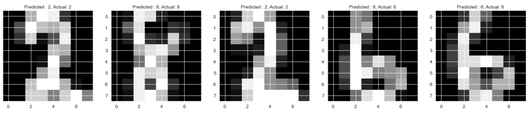
In [26]:
```
plt.figure(figsize = (9,9))
sns.heatmap(cm, annot=True, fmt ="".3f", linewidth= .5, square = True, cmap = "Spectral")
plt.ylabel("Actual Output")
plt.xlabel("Predicted Output")
all_sample_title= "Accuracy Score : {0}" .format(score)
plt.title(all_sample_title, size = 15)
```

Out[26]:
```
Text(0.5, 1.0, 'Accuracy Score : 0.9666666666666667')
```



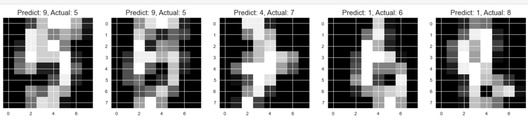
In [27]:
```
# Getting miss classified labels

index = 0
misclassifiedIndexes = []
for label, predict in zip(y_test, predictions):
    if label != predict:
        misclassifiedIndexes.append(index)
        index +=1
```

In [35]:
```
# Plotting misscalssified labels with known labels

plt.figure(figsize = (20,4))
for plotIndex, badIndex in enumerate(misclassifiedIndexes[0:5]):
    plt.subplot(1, 5, plotIndex + 1)
    plt.imshow(np.reshape(X_test[badIndex], (8,8)), cmap = plt.cm.gray)
    plt.title("Predicted : {}, Actual: {}".format(predictions[badIndex], y_test[badIndex]), fontsize = 10 )
```



Assignment

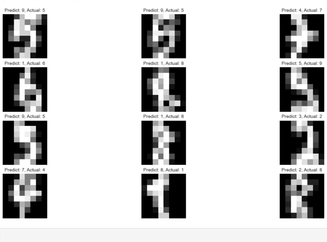
In [47]:
```
index = 0
misclassifiedIndexes = []
for label, predict in zip(y_test, predictions):
    if label != predict:
        misclassifiedIndexes.append(index)
    index +=1

plt.figure(figsize=(20,4))
for plotIndex, badIndex in enumerate(misclassifiedIndexes[0:5]):
    plt.subplot(1, 5, plotIndex + 1)
    plt.imshow(np.reshape(X_test[badIndex], (8,8)), cmap=plt.cm.gray)
    plt.title('Predict: {}, Actual: {}'.format(predictions[badIndex], y_test[badIndex]), fontsize = 15)
```



Another way of writing that code

In [36]:
```
import matplotlib.pyplot as plt
misclassifiedIndexes = np.where(y_test!=predictions)[0]


fig, ax = plt.subplots(4, 3,figsize=(15,8))
ax = ax.ravel()
for i, badIndex in enumerate(misclassifiedIndexes):
    ax[i].imshow(np.reshape(X_test[badIndex], (8, 8)), cmap=plt.cm.gray)
    ax[i].set_title(f'Predict: {predictions[badIndex]}, '
                    f'Actual: {y_test[badIndex]}', fontsize = 10)
    ax[i].set(frame_on=False)
    ax[i].axis('off')
plt.box(False)
plt.axis('off')
```

Out[36]:
```
(-0.5, 7.5, 7.5, -0.5)
```



In [ ]:
```
```