In [1]:
```
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
df=pd.read_csv(r"C:\Users\P. VIJAY KUMAR\Downloads\archive (2)\ionosphere_data.csv")
df
```

Out[1]:

	column_a	column_b	column_c	column_d	column_e	column_f	column_g	column_h	column_i	column_j	...	column_z	colu
0	True	False	0.99539	-0.05889	0.85243	0.02306	0.83398	-0.37708	1.00000	0.03760	...	-0.51171	
1	True	False	1.00000	-0.18829	0.93035	-0.36156	-0.10868	-0.93597	1.00000	-0.04549	...	-0.26569	
2	True	False	1.00000	-0.03365	1.00000	0.00485	1.00000	-0.12062	0.88965	0.01198	...	-0.40220	
3	True	False	1.00000	-0.45161	1.00000	1.00000	0.71216	-1.00000	0.00000	0.00000	...	0.90695	
4	True	False	1.00000	-0.02401	0.94140	0.06531	0.92106	-0.23255	0.77152	-0.16399	...	-0.65158	
...	...	...	...	...	...	...	...	...	...	...	...	...	
346	True	False	0.83508	0.08298	0.73739	-0.14706	0.84349	-0.05567	0.90441	-0.04622	...	-0.04202	
347	True	False	0.95113	0.00419	0.95183	-0.02723	0.93438	-0.01920	0.94590	0.01606	...	0.01361	
348	True	False	0.94701	-0.00034	0.93207	-0.03227	0.95177	-0.03431	0.95584	0.02446	...	0.03193	
349	True	False	0.90608	-0.01657	0.98122	-0.01989	0.95691	-0.03646	0.85746	0.00110	...	-0.02099	
350	True	False	0.84710	0.13533	0.73638	-0.06151	0.87873	0.08260	0.88928	-0.09139	...	-0.15114	

351 rows × 35 columns

In [2]:
```
pd.set_option("Display.max_rows",100000000)
pd.set_option("Display.max_column",10000000)
pd.set_option("Display.width",95)
print("This DataFrame has %d Rows & %d columns"%(df.shape))

```

```
This DataFrame has 351 Rows & 35 columns
```

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

Out[3]:

	column_a	column_b	column_c	column_d	column_e	column_f	column_g	column_h	column_i	column_j	column_k	column_l
0	True	False	0.99539	-0.05889	0.85243	0.02306	0.83398	-0.37708	1.00000	0.03760	0.85243	-0.17755
1	True	False	1.00000	-0.18829	0.93035	-0.36156	-0.10868	-0.93597	1.00000	-0.04549	0.50874	-0.67743
2	True	False	1.00000	-0.03365	1.00000	0.00485	1.00000	-0.12062	0.88965	0.01198	0.73082	0.05346
3	True	False	1.00000	-0.45161	1.00000	1.00000	0.71216	-1.00000	0.00000	0.00000	0.00000	0.00000
4	True	False	1.00000	-0.02401	0.94140	0.06531	0.92106	-0.23255	0.77152	-0.16399	0.52798	-0.20275

In [4]:
```
features_matrix=df.iloc[:,0:34]
target_vector=df.iloc[:,-1]
```

In [5]:
```
print("The feature matrix has %d Rows and %d Columns"%(features_matrix.shape))
print("The Target Vector Matrix has %d Rows and %d Columns"%(np.array(target_vector).reshape(-1,1).sha
```

```
The feature matrix has 351 Rows and 34 Columns
The Target Vector Matrix has 351 Rows and 1 Columns
```

In [6]:
```
features_matrix_standardized=StandardScaler().fit_transform(features_matrix)
```

In [7]:
```
algorithm=LogisticRegression(penalty='l2',dual=False,tol=1e-4,C=1.0,fit_intercept=True,intercept_scali
                             solver='lbfgs',max_iter=100,multi_class='auto',verbose=0,warm_start=False,
```

In [8]:
```
Logistic_Regression_Model=algorithm.fit(features_matrix_standardized,target_vector)
```

In [9]:
```
observation=[[1,0,0.99539,-0.05889,0.8524299999999999,0.02306,0.8339799999999999,-0.37708,1.0,0.0376,0
```

In [10]:
```
predictions=Logistic_Regression_Model.predict(observation)
print('The Model Predicted the observation to belong to class %s'%(predictions))
print('The Algorithm was trained to predict n=one of the two classes:%s'%(algorithm.classes_))
print("""The model says the probability of the observation we passed Belonging to class['b'] Is %s"""%
```

```
The Model Predicted the observation to belong to class ['g']
The Algorithm was trained to predict n=one of the two classes:['b' 'g']
The model says the probability of the observation we passed Belonging to class['b'] Is 0.0077739316001378
4
```

In [11]:
```
print()
```

In [12]:
```
print("""The model says the probability of the observation we passed Belonging to class['g'] Is %s"""%
```

```
The model says the probability of the observation we passed Belonging to class['g'] Is 0.9922260683998622
```

In [1]:
```
import re
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import metrics
%matplotlib inline
digits=load_digits()
```

In [2]:
```
print("Image Data Shape",digits.data.shape)
print("Label Data Shape",digits.target.shape)
```

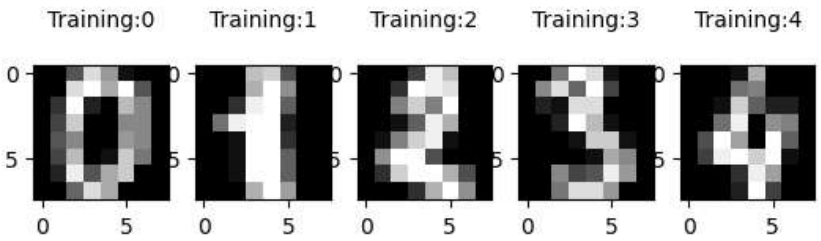
```
Image Data Shape (1797, 64)
Label Data Shape (1797,)
```

In [3]:
```
plt.figure(figsize=(20,4))
```

Out[3]:
```
<Figure size 2000x400 with 0 Axes>
```

```
<Figure size 2000x400 with 0 Axes>
```

In [4]:
```
for index,(image,label) in enumerate(zip(digits.data[0:5],digits.target[0:5])):
    plt.subplot(1,5,index+1)
    plt.imshow(np.reshape(image,(8,8)),cmap=plt.cm.gray)
    plt.title('Training:%i\n'%label,fontsize=10)
```



In [5]:
```
X_train,X_test,Y_train,Y_test=train_test_split(digits.data,digits.target,test_size=0.30,random_state=2
```

In [6]:
```
print(X_train.shape)
```

```
(1257, 64)
```

In [8]:
```
print(Y_train.shape)
```

```
(1257,)
```

In [9]:
```
print(X_test.shape)
```

```
(540, 64)
```

In [10]:
```
print(Y_test.shape)
```

```
(540,)
```

In [11]:
```
from sklearn.linear_model import LogisticRegression
lRegr=LogisticRegression(max_iter=10000)
lRegr.fit(X_train,Y_train)
print(lRegr.predict(X_test))
score=lRegr.score(X_test,Y_test)
print("Score:",score)
```

```
[4 0 9 1 8 7 1 5 1 6 6 7 6 1 5 5 8 6 2 7 4 6 4 1 5 2 9 5 4 6 5 6 3 4 0 9 9
 8 4 6 8 8 5 7 9 8 9 6 1 7 0 1 9 7 3 3 1 8 8 8 9 8 5 8 4 9 3 5 8 4 3 1 3 8
 7 3 3 0 8 7 2 8 5 3 8 7 6 4 6 2 2 0 1 1 5 3 5 7 1 8 2 2 6 4 6 7 3 7 3 9 4
 7 0 3 5 1 5 0 3 9 2 7 3 2 0 8 1 9 2 1 5 1 0 3 4 3 0 8 3 2 2 7 3 1 6 7 2 8
 3 1 1 6 4 8 2 1 8 4 1 3 1 1 9 5 4 8 7 4 8 9 5 7 6 9 4 0 4 0 0 9 0 6 5 8 8
 3 7 9 2 0 8 2 7 3 0 2 1 9 2 7 0 6 9 3 1 1 3 5 2 5 5 2 1 2 9 4 6 5 5 5 9 7
 1 5 9 6 3 7 1 7 5 1 7 2 7 5 5 4 8 6 6 2 8 7 3 7 8 0 9 5 7 4 3 4 1 0 3 3 5
 4 1 3 1 2 5 1 4 0 3 1 5 5 7 4 0 1 0 9 5 5 5 4 0 1 8 6 2 1 1 1 7 9 6 7 9 7
 0 4 9 6 9 2 7 2 1 0 8 2 8 6 5 7 8 4 5 7 8 6 4 2 6 9 3 0 0 8 0 6 6 7 1 4 5
 6 9 7 2 8 5 1 2 4 1 8 8 7 6 0 8 0 6 1 5 7 8 0 4 1 4 5 9 2 2 3 9 1 3 9 3 2
 8 0 6 5 6 2 5 2 3 2 6 1 0 7 6 0 6 2 7 0 3 2 4 2 3 6 9 7 7 0 3 5 4 1 2 2 1
 2 7 7 0 4 9 8 5 6 1 6 5 2 0 8 2 4 3 3 2 9 3 8 9 9 5 9 0 3 4 7 9 8 5 7 5 0
 5 3 5 0 2 7 3 0 4 3 6 6 1 9 6 3 4 6 4 6 7 2 7 6 3 0 3 0 1 3 6 1 0 4 3 8 4
 3 3 4 8 6 9 6 3 3 0 5 7 8 9 1 5 3 2 5 1 7 6 0 6 9 5 2 4 4 7 2 0 5 6 2 0 8
 4 4 4 7 1 0 4 1 9 2 1 3 0 5 3 9 8 2 6 0 0 4]
Score: 0.9537037037037037
```

In [ ]:
```

```