In [1]:

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
```

In [2]:

```
df=pd.read_csv(r"C:\Users\USER\Downloads\ionosphere.csv")
df
```

Out[2]:

```
       1    0  0.99539  -0.05889   0.85243   0.02306   0.83398  -0.37708      1.1   0.03760  ...  -0.51171   0.41078  -0.46168   0.21266  -0.340
  0    1    0  1.00000  -0.18829   0.93035  -0.36156  -0.10868  -0.93597  1.00000  -0.04549  ...  -0.26569  -0.20468  -0.18401  -0.19040  -0.115
  1    1    0  1.00000  -0.03365   1.00000   0.00485   1.00000  -0.12062  0.88965   0.01198  ...  -0.40220   0.58984  -0.22145   0.43100  -0.173
  2    1    0  1.00000  -0.45161   1.00000   1.00000   0.71216  -1.00000  0.00000   0.00000  ...   0.90695   0.51613   1.00000   1.00000  -0.200
  3    1    0  1.00000  -0.02401   0.94140   0.06531   0.92106  -0.23255  0.77152  -0.16399  ...  -0.65158   0.13290  -0.53206   0.02431  -0.621
  4    1    0  0.02337  -0.00592  -0.09924  -0.11949  -0.00763  -0.11824  0.14706   0.06637  ...  -0.01535  -0.03240   0.09223  -0.07859   0.007
...  ...  ...      ...       ...       ...       ...       ...       ...      ...       ...  ...       ...       ...       ...       ...
345    1    0  0.83508   0.08298   0.73739  -0.14706   0.84349  -0.05567  0.90441  -0.04622  ...  -0.04202   0.83479   0.00123   1.00000   0.128
346    1    0  0.95113   0.00419   0.95183  -0.02723   0.93438  -0.01920  0.94590   0.01606  ...   0.01361   0.93522   0.04925   0.93159   0.081
347    1    0  0.94701  -0.00034   0.93207  -0.03227   0.95177  -0.03431  0.95584   0.02446  ...   0.03193   0.92489   0.02542   0.92120   0.022
348    1    0  0.90608  -0.01657   0.98122  -0.01989   0.95691  -0.03646  0.85746   0.00110  ...  -0.02099   0.89147  -0.07760   0.82983  -0.172
349    1    0  0.84710   0.13533   0.73638  -0.06151   0.87873   0.08260  0.88928  -0.09139  ...  -0.15114   0.81147  -0.04822   0.78207  -0.007

350 rows × 35 columns
```

In [ ]:

In [3]:
```
pd.set_option('display.max_rows',10000000000)
pd.set_option('display.max_columns',10000000000)
pd.set_option('display.width',95)
print('This DataFrame has %d rows and %d columns'%(df.shape))
```

```
This DataFrame has 350 rows and 35 columns
```

In [4]:
```
df.head(10)
```

Out[4]:

	1	0	0.99539	-0.05889	0.85243	0.02306	0.83398	-0.37708	1.1	0.03760	0.85243.1	-0.17755	0.59755	-0.44945	0.60536	-0
0	1	0	1.00000	-0.18829	0.93035	-0.36156	-0.10868	-0.93597	1.00000	-0.04549	0.50874	-0.67743	0.34432	-0.69707	-0.51685	-0
1	1	0	1.00000	-0.03365	1.00000	0.00485	1.00000	-0.12062	0.88965	0.01198	0.73082	0.05346	0.85443	0.00827	0.54591	0
2	1	0	1.00000	-0.45161	1.00000	1.00000	0.71216	-1.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	-1.00000	0
3	1	0	1.00000	-0.02401	0.94140	0.06531	0.92106	-0.23255	0.77152	-0.16399	0.52798	-0.20275	0.56409	-0.00712	0.34395	-0
4	1	0	0.02337	-0.00592	-0.09924	-0.11949	-0.00763	-0.11824	0.14706	0.06637	0.03786	-0.06302	0.00000	0.00000	-0.04572	-0
5	1	0	0.97588	-0.10602	0.94601	-0.20800	0.92806	-0.28350	0.85996	-0.27342	0.79766	-0.47929	0.78225	-0.50764	0.74628	-0
6	0	0	0.00000	0.00000	0.00000	0.00000	1.00000	-1.00000	0.00000	0.00000	-1.00000	-1.00000	0.00000	0.00000	0.00000	0
7	1	0	0.96355	-0.07198	1.00000	-0.14333	1.00000	-0.21313	1.00000	-0.36174	0.92570	-0.43569	0.94510	-0.40668	0.90392	-0
8	1	0	-0.01864	-0.08459	0.00000	0.00000	0.00000	0.00000	0.11470	-0.26810	-0.45663	-0.38172	0.00000	0.00000	-0.33656	0
9	1	0	1.00000	0.06655	1.00000	-0.18388	1.00000	-0.27320	1.00000	-0.43107	1.00000	-0.41349	0.96232	-0.51874	0.90711	-0

In [5]:
```
features_matrix=df.iloc[:,0:34]
```

In [6]:
```
target_vector=df.iloc[:,-1]
```

In [7]:
```
print('The features matrix has %d rows and %d column(s)'%(features_matrix.shape))
```

```
The features matrix has 350 rows and 34 column(s)
```

In [8]:
```
print('The target matrix has %d rows and %d column(s)'%(np.array(target_vector).reshape(-1,1).shape))
```

```
The target matrix has 350 rows and 1 column(s)
```

In [9]:
```
features_matrix_standardized=StandardScaler().fit_transform(features_matrix)
```

In [10]:
```
algorithm=LogisticRegression(penalty=None,dual=False,tol=1e-4,C=1.0,fit_intercept=True,intercept_scaling=1,cla
```

In [11]:
```
Logistic_Regression_Model=algorithm.fit(features_matrix_standardized,target_vector)
```

```
C:\Users\USER\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\linear_model\_logistic.py:45
8: 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 (https://scikit-learn.org/stable/modules/prep
rocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/
stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

observation[[1,0,0.99539,-0.5889,0.8524299999999999,0.02306,0.8339799999999999,-0.37708,1.0,0.0376,0.8524299999999999,-0.17

In [12]:
```
observation=[[1,0,0.99539,-0.5889,0.8524299999999999,0.02306,0.8339799999999999,-0.37708,1.0,0.0376,0.852429999
```

In [14]:
```
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 One Of The Two Classes:%s'%(algorithm.classes_))
```

```
The Model Predicted The Observation To Belong to Class ['g']
The Algorithm Was Trained To Predict One Of The Two Classes:['b' 'g']
```

In [16]:
```
print("""The Model Says The Probability Of The Observation We Passed Belonging To Class['b'] is %s"""%(algorit
print()
print("""The Model Says The Probability Of The Observation We Passed Belonging To Class['g'] is %s"""%(algorit
```

```
The Model Says The Probability Of The Observation We Passed Belonging To Class['b'] is 4.591503101458727e-05

The Model Says The Probability Of The Observation We Passed Belonging To Class['g'] is 0.9999540849689854
```

In [2]:
```
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()
```

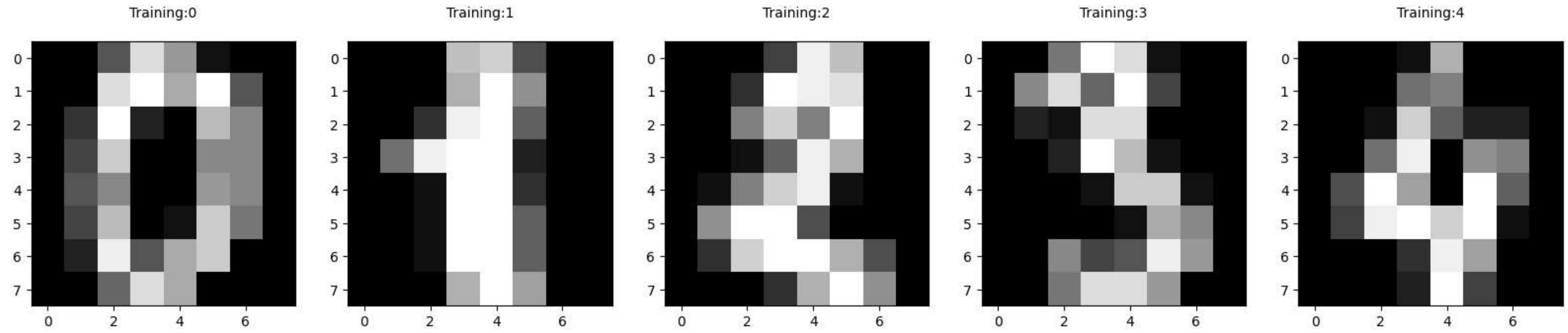
Type *Markdown* and LaTeX: α^2

In [3]:
```
print("Image Data Shape",digits.data.shape)
print("Label Data Shape",digits.target.shape)
```

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

In [4]:

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

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(digits.data,digits.target,test_size=0.30,random_state=2)
print(x_train.shape)
```

```
(1257, 64)
```

In [6]:

```
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(1257,)
(540, 64)
(540,)
```

In [7]:

```
from sklearn.linear_model import LogisticRegression
LogisticRegr=LogisticRegression(max_iter=10000)
LogisticRegr.fit(x_train,y_train)
print(LogisticRegr.predict(x_test))
```

```
[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]
```

In [8]:

```
score=LogisticRegr.score(x_test,y_test)
print(score)
```

```
0.9537037037037037
```

In [ ]: