In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [6]:
```
df=pd.read_csv(r"C:\Users\user\Downloads\C1_ionosphere - C1_ionosphere.csv")
df
```

Out[6]:

	1	0	0.99539	-0.05889	0.85243	0.02306	0.83398	-0.37708	1.1	0.0376	...	-0.51
0	1	0	1.00000	-0.18829	0.93035	-0.36156	-0.10868	-0.93597	1.00000	-0.04549	...	-0.26
1	1	0	1.00000	-0.03365	1.00000	0.00485	1.00000	-0.12062	0.88965	0.01198	...	-0.40
2	1	0	1.00000	-0.45161	1.00000	1.00000	0.71216	-1.00000	0.00000	0.00000	...	0.90
3	1	0	1.00000	-0.02401	0.94140	0.06531	0.92106	-0.23255	0.77152	-0.16399	...	-0.65
4	1	0	0.02337	-0.00592	-0.09924	-0.11949	-0.00763	-0.11824	0.14706	0.06637	...	-0.01
...	...	...	...	...	...	...	...	...	...	...	...	
345	1	0	0.83508	0.08298	0.73739	-0.14706	0.84349	-0.05567	0.90441	-0.04622	...	-0.04
346	1	0	0.95113	0.00419	0.95183	-0.02723	0.93438	-0.01920	0.94590	0.01606	...	0.01
347	1	0	0.94701	-0.00034	0.93207	-0.03227	0.95177	-0.03431	0.95584	0.02446	...	0.03
348	1	0	0.90608	-0.01657	0.98122	-0.01989	0.95691	-0.03646	0.85746	0.00110	...	-0.02
349	1	0	0.84710	0.13533	0.73638	-0.06151	0.87873	0.08260	0.88928	-0.09139	...	-0.15

350 rows × 35 columns

In [7]:
```
from sklearn.linear_model import LogisticRegression
```

In [8]:
```
feature_matrix=df.iloc[:,0:30]
target_vector=df.iloc[:,-1]
```

In [9]:
```
feature_matrix.shape
```

Out[9]:
```
(350, 30)
```

In [10]:
```
target_vector.shape
```

Out[10]:
```
(350,)
```

In [15]:
```
from sklearn.preprocessing import StandardScaler
```

In [16]:
```
fs=StandardScaler().fit_transform(feature_matrix)
```

In [17]:
```
logr=LogisticRegression()
logr.fit(fs,target_vector)
```

Out[17]:
```
LogisticRegression()
```

In [18]:
```
observation=[[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,21,23,24,2
```

In [19]:
```
prediction=logr.predict(observation)
print(prediction)
```

```
['g']
```

In [21]:
```
logr.classes_
```

Out[21]:
```
array(['b', 'g'], dtype=object)
```

In [22]:
```
logr.predict_proba(observation)[0][0]
```

Out[22]:
```
5.773159728050814e-15
```

In [23]:
```
logr.predict_proba(observation)[0][1]
```

Out[23]:
```
0.9999999999999942
```

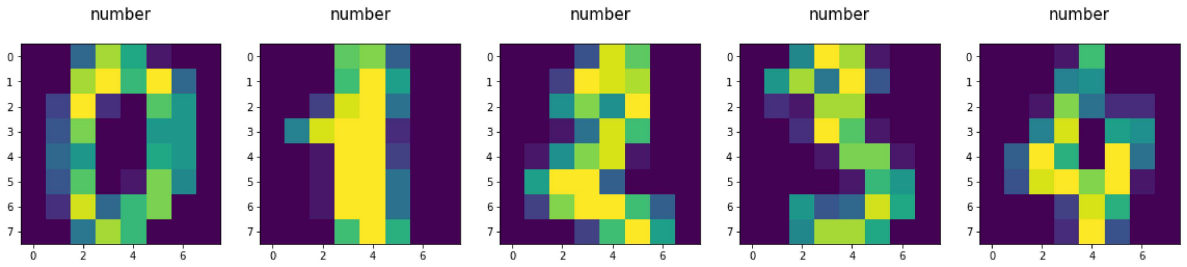
linear regression 2

In [24]:
```
import re
from sklearn.datasets import load_digits
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
```

In [25]:
```
digits=load_digits()
digits
```

Out[25]:
```
{'data': array([[ 0.,  0.,  5., ...,  0.,  0.,  0.],
        [ 0.,  0.,  0., ..., 10.,  0.,  0.],
        [ 0.,  0.,  0., ..., 16.,  9.,  0.],
        ...,
        [ 0.,  0.,  1., ...,  6.,  0.,  0.],
        [ 0.,  0.,  2., ..., 12.,  0.,  0.],
        [ 0.,  0., 10., ..., 12.,  1.,  0.]]),
 'target': array([0, 1, 2, ..., 8, 9, 8]),
 'frame': None,
 'feature_names': ['pixel_0_0',
  'pixel_0_1',
  'pixel_0_2',
  'pixel_0_3',
  'pixel_0_4',
  'pixel_0_5',
  'pixel_0_6',
  'pixel_0_7',
  'pixel_1_0',
  'pixel_1_1',
```

In [28]:
```
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)))
    plt.title("number\n"%label,fontsize=15)
```



In [29]:
```
x_train,x_test,y_train,y_test=train_test_split(digits.data,digits.target,test_
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

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

In [30]:
```
logre=LogisticRegression()
logre.fit(x_train,y_train)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:
763: 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/preprocessing.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(
```

Out[30]:
```
LogisticRegression()
```

In [31]:
```
print(logre.predict(x_test))
```

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

In [33]:
```
print(logre.score(x_test,y_test))
```

```
0.9666666666666667
```

Random Forest

In [34]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [35]:
```
df=pd.read_csv(r"C:\Users\user\Downloads\C1_ionosphere - C1_ionosphere.csv")
df
```

Out[35]:

	1	0	0.99539	-0.05889	0.85243	0.02306	0.83398	-0.37708	1.1	0.0376	...	-0.51
0	1	0	1.00000	-0.18829	0.93035	-0.36156	-0.10868	-0.93597	1.00000	-0.04549	...	-0.26
1	1	0	1.00000	-0.03365	1.00000	0.00485	1.00000	-0.12062	0.88965	0.01198	...	-0.40
2	1	0	1.00000	-0.45161	1.00000	1.00000	0.71216	-1.00000	0.00000	0.00000	...	0.90
3	1	0	1.00000	-0.02401	0.94140	0.06531	0.92106	-0.23255	0.77152	-0.16399	...	-0.65
4	1	0	0.02337	-0.00592	-0.09924	-0.11949	-0.00763	-0.11824	0.14706	0.06637	...	-0.01
...	...	...	...	...	...	...	...	...	...	...	...	
345	1	0	0.83508	0.08298	0.73739	-0.14706	0.84349	-0.05567	0.90441	-0.04622	...	-0.04
346	1	0	0.95113	0.00419	0.95183	-0.02723	0.93438	-0.01920	0.94590	0.01606	...	0.01
347	1	0	0.94701	-0.00034	0.93207	-0.03227	0.95177	-0.03431	0.95584	0.02446	...	0.03
348	1	0	0.90608	-0.01657	0.98122	-0.01989	0.95691	-0.03646	0.85746	0.00110	...	-0.02
349	1	0	0.84710	0.13533	0.73638	-0.06151	0.87873	0.08260	0.88928	-0.09139	...	-0.15

350 rows × 35 columns

In [36]:
```
df['g'].value_counts()
```

Out[36]:
```
g    224
b    126
Name: g, dtype: int64
```

In [37]:
```
x=df.drop('g',axis=1)
y=df['g']
```

In [38]:
```
g1={"g":{'g':1,"b":2}}
df=df.replace(g1)
print(df)
```

```
     1  0  0.99539  -0.05889  0.85243  0.02306  0.83398  -0.37708      1.1  \
0    1  0  1.00000  -0.18829  0.93035 -0.36156 -0.10868  -0.93597  1.00000
1    1  0  1.00000  -0.03365  1.00000  0.00485  1.00000  -0.12062  0.88965
2    1  0  1.00000  -0.45161  1.00000  1.00000  0.71216  -1.00000  0.00000
3    1  0  1.00000  -0.02401  0.94140  0.06531  0.92106  -0.23255  0.77152
4    1  0  0.02337  -0.00592 -0.09924 -0.11949 -0.00763  -0.11824  0.14706
..  .. ..      ...       ...      ...      ...      ...       ...      ...
345  1  0  0.83508   0.08298  0.73739 -0.14706  0.84349  -0.05567  0.90441
346  1  0  0.95113   0.00419  0.95183 -0.02723  0.93438  -0.01920  0.94590
347  1  0  0.94701  -0.00034  0.93207 -0.03227  0.95177  -0.03431  0.95584
348  1  0  0.90608  -0.01657  0.98122 -0.01989  0.95691  -0.03646  0.85746
349  1  0  0.84710   0.13533  0.73638 -0.06151  0.87873   0.08260  0.88928

      0.0376  ...  -0.51171  0.41078  -0.46168  0.21266  -0.3409  0.42267  \
0   -0.04549  ...  -0.26569 -0.20468  -0.18401 -0.19040 -0.11593 -0.16626
1    0.01198  ...  -0.40220  0.58984  -0.22145  0.43100 -0.17365  0.60436
2    0.00000  ...   0.90695  0.51613   1.00000  1.00000 -0.20099  0.25682
3   -0.16399  ...  -0.65158  0.13290  -0.53206  0.02431 -0.62197 -0.05707
4    0.06637  ...  -0.01535 -0.03240   0.09223 -0.07859  0.00732  0.00000
..       ...  ...       ...      ...       ...      ...      ...      ...
345 -0.04622  ...  -0.04202  0.83479   0.00123  1.00000  0.12815  0.86660
346  0.01606  ...   0.01361  0.93522   0.04925  0.93159  0.08168  0.94066
347  0.02446  ...   0.03193  0.92489   0.02542  0.92120  0.02242  0.92459
348  0.00110  ...  -0.02099  0.89147  -0.07760  0.82983 -0.17238  0.96022
349 -0.09139  ...  -0.15114  0.81147  -0.04822  0.78207 -0.00703  0.75747

     -0.54487  0.18641   -0.453  g
0    -0.06288 -0.13738 -0.02447  2
1    -0.24180  0.56045 -0.38238  1
2     1.00000 -0.32382  1.00000  2
3    -0.59573 -0.04608 -0.65697  1
4     0.00000 -0.00039  0.12011  2
..        ...      ...      ... ..
345  -0.10714  0.90546 -0.04307  1
346  -0.00035  0.91483  0.04712  1
347   0.00442  0.92697 -0.00577  1
348  -0.03757  0.87403 -0.16243  1
349  -0.06678  0.85764 -0.06151  1

[350 rows x 35 columns]
```

In [65]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.30)
```

In [66]:
```
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[66]:
```
RandomForestClassifier()
```

In [67]:
```
parameters={'max_depth':[1,2,3,4,5],
            'min_samples_leaf':[5,10,15,20,25],
            'n_estimators':[10,20,30,40,50]}
```

In [68]:
```
from sklearn.model_selection import GridSearchCV
grid_search=GridSearchCV(estimator=rfc,param_grid=parameters,cv=2,scoring='acc
grid_search.fit(x_train,y_train)
```

Out[68]:
```
GridSearchCV(cv=2, estimator=RandomForestClassifier(),
             param_grid={'max_depth': [1, 2, 3, 4, 5],
                         'min_samples_leaf': [5, 10, 15, 20, 25],
                         'n_estimators': [10, 20, 30, 40, 50]},
             scoring='accuracy')
```

In [51]:
```
grid_search.best_score_
```

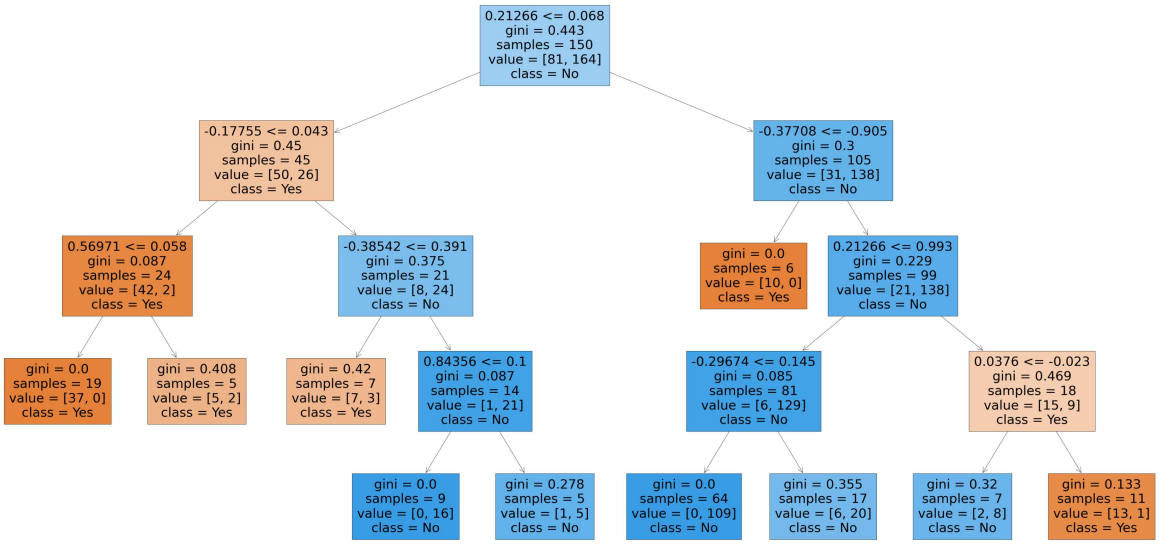
Out[51]:
```
0.9172733992891355
```

In [71]:
```
rfc_best=grid_search.best_estimator_
```

In [72]:
```
from sklearn.tree import plot_tree

plt.figure(figsize=(80,40))
plot_tree(rfc_best.estimators_[5],feature_names=x.columns,class_names=['Yes','
```

Out[72]:
```
[Text(2100.705882352941, 1956.96, '0.21266 <= 0.068\ngini = 0.443\nsamples =
150\nvalue = [81, 164]\nclass = No'),
 Text(1050.3529411764705, 1522.0800000000002, '-0.17755 <= 0.043\ngini = 0.45
\nsamples = 45\nvalue = [50, 26]\nclass = Yes'),
 Text(525.1764705882352, 1087.2, '0.56971 <= 0.058\ngini = 0.087\nsamples = 2
4\nvalue = [42, 2]\nclass = Yes'),
 Text(262.5882352941176, 652.3200000000002, 'gini = 0.0\nsamples = 19\nvalue
= [37, 0]\nclass = Yes'),
 Text(787.7647058823529, 652.3200000000002, 'gini = 0.408\nsamples = 5\nvalue
= [5, 2]\nclass = Yes'),
 Text(1575.5294117647059, 1087.2, '-0.38542 <= 0.391\ngini = 0.375\nsamples =
21\nvalue = [8, 24]\nclass = No'),
 Text(1312.941176470588, 652.3200000000002, 'gini = 0.42\nsamples = 7\nvalue
= [7, 3]\nclass = Yes'),
 Text(1838.1176470588234, 652.3200000000002, '0.84356 <= 0.1\ngini = 0.087\ns
amples = 14\nvalue = [1, 21]\nclass = No'),
 Text(1575.5294117647059, 217.44000000000005, 'gini = 0.0\nsamples = 9\nvalue
= [0, 16]\nclass = No'),
 Text(2100.705882352941, 217.44000000000005, 'gini = 0.278\nsamples = 5\nvalu
e = [1, 5]\nclass = No'),
 Text(3151.0588235294117, 1522.0800000000002, '-0.37708 <= -0.905\ngini = 0.3
\nsamples = 105\nvalue = [31, 138]\nclass = No'),
 Text(2888.4705882352937, 1087.2, 'gini = 0.0\nsamples = 6\nvalue = [10, 0]\n
class = Yes'),
 Text(3413.6470588235293, 1087.2, '0.21266 <= 0.993\ngini = 0.229\nsamples =
99\nvalue = [21, 138]\nclass = No'),
 Text(2888.4705882352937, 652.3200000000002, '-0.29674 <= 0.145\ngini = 0.085
\nsamples = 81\nvalue = [6, 129]\nclass = No'),
 Text(2625.882352941176, 217.44000000000005, 'gini = 0.0\nsamples = 64\nvalue
= [0, 109]\nclass = No'),
 Text(3151.0588235294117, 217.44000000000005, 'gini = 0.355\nsamples = 17\nva
lue = [6, 20]\nclass = No'),
 Text(3938.8235294117644, 652.3200000000002, '0.0376 <= -0.023\ngini = 0.469
\nsamples = 18\nvalue = [15, 9]\nclass = Yes'),
 Text(3676.235294117647, 217.44000000000005, 'gini = 0.32\nsamples = 7\nvalue
= [2, 8]\nclass = No'),
 Text(4201.411764705882, 217.44000000000005, 'gini = 0.133\nsamples = 11\nval
ue = [13, 1]\nclass = Yes')]
```


0.21266 <= 0.068
gini = 0.443
samples = 150
value = [81, 164]
class = No
-0.17755 <= 0.043
gini = 0.45
samples = 45
value = [50, 26]
class = Yes
-0.37708 <= -0.905
gini = 0.3
samples = 105
value = [31, 138]
class = No
0.56971 <= 0.058
gini = 0.087
samples = 24
value = [42, 2]
class = Yes
-0.38542 <= 0.391
gini = 0.375
samples = 21
value = [8, 24]
class = No
gini = 0.0
samples = 6
value = [10, 0]
class = Yes
0.21266 <= 0.993
gini = 0.229
samples = 99
value = [21, 138]
class = No
gini = 0.0
samples = 19
value = [37, 0]
class = Yes
gini = 0.408
samples = 5
value = [5, 2]
class = Yes
gini = 0.42
samples = 7
value = [7, 3]
class = Yes
0.84356 <= 0.1
gini = 0.087
samples = 14
value = [1, 21]
class = No
-0.29674 <= 0.145
gini = 0.085
samples = 81
value = [6, 129]
class = No
0.0376 <= -0.023
gini = 0.469
samples = 18
value = [15, 9]
class = Yes
gini = 0.0
samples = 9
value = [0, 16]
class = No
gini = 0.278
samples = 5
value = [1, 5]
class = No
gini = 0.0
samples = 64
value = [0, 109]
class = No
gini = 0.355
samples = 17
value = [6, 20]
class = No
gini = 0.32
samples = 7
value = [2, 8]
class = No
gini = 0.133
samples = 11
value = [13, 1]
class = Yes

In [ ]: