In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sn
```

In [2]:
```
from sklearn.linear_model import LogisticRegression
```

In [3]:
```
df=pd.read_csv(r"C5_health care diabetes")
```

In [4]:
```
df
```

Out[4]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

In [5]:
```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

In [6]:
```
1 df.dropna(inplace=True)
```

In [7]:
```
1 df.columns
```

Out[7]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

In [8]:
```
1 df
```

Out[8]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

In [9]:
```
1 from sklearn.linear_model import LogisticRegression
```

In [10]:
```
1 logr =LogisticRegression()
```

In [11]:
```
1 feature_matrix=df[['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
2        'BMI', 'DiabetesPedigreeFunction', 'Age']]
3 target_vector=df['Outcome']
```

In [12]:
```
1 feature_matrix.shape
```

Out[12]: (768, 8)

In [13]:
```
1 target_vector.shape
```

Out[13]: (768,)

In [14]:
```
1 from sklearn.preprocessing import StandardScaler
```

In [15]:
```
1 fs=StandardScaler().fit_transform(feature_matrix)
```

In [16]:
```
1 logr=LogisticRegression()
2 logr.fit(fs,target_vector)
```

Out[16]:
```
LogisticRegression()
```

In [19]:
```
1 observation=[[1,2,3,4,5,6,7,8]]
```

In [20]:
```
1 prediction = logr.predict(observation)
2 print(prediction)
```

```
[1]
```

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

Out[21]:
```
array([0, 1], dtype=int64)
```

In [22]:
```
1 logr.predict_proba(observation)[0][1]
```

Out[22]:
```
0.9997076305131244
```

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

Out[23]:
```
0.00029236948687560993
```

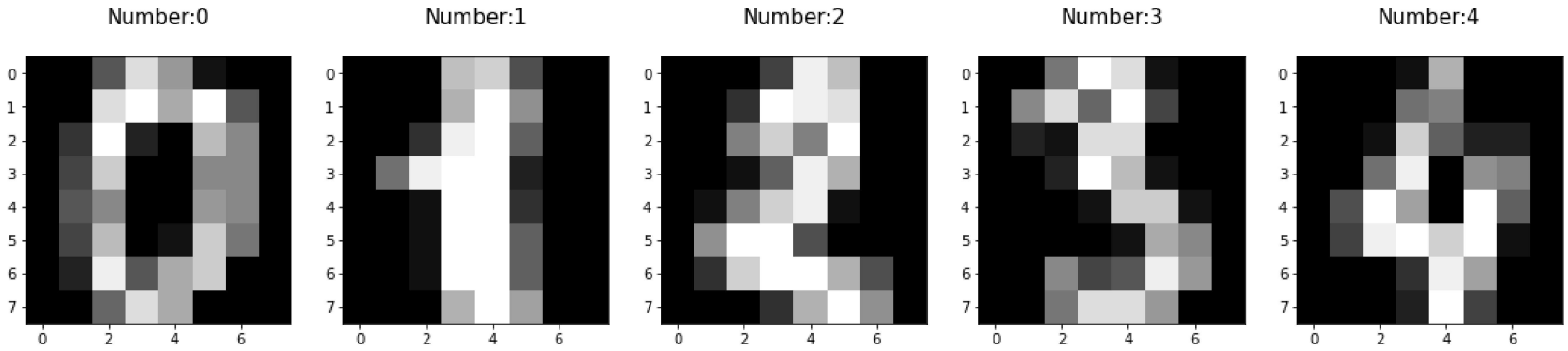
Linear regression 2

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 [26]:
```
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("Number:%i\n"%label,fontsize=15)
```



In [27]:
```
x_train,x_test,y_train,y_test=train_test_split(digits.data,digits.target,test_size=0.30)
```

In [28]:
```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

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

In [29]:
```
logre=LogisticRegression(max_iter=10000)
logre.fit(x_train,y_train)
```

Out[29]:
```
LogisticRegression(max_iter=10000)
```

In [30]:
```
print(logre.predict(x_test))
```

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

In [31]:
```
print(logre.score(x_test,y_test))
```

```
0.95
```