In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
```

In [2]:

```
df=pd.read_csv(r"C:\Users\user\Downloads\used cars.csv")
df
```

Out[2]:

	Unnamed: 0	model	year	price	transmission	mileage	fuelType	tax	mpg	engineSize	Make
0	0	T-Roc	2019	25000	Automatic	13904	Diesel	145	49.6	2.0	VW
1	1	T-Roc	2019	26883	Automatic	4562	Diesel	145	49.6	2.0	VW
2	2	T-Roc	2019	20000	Manual	7414	Diesel	145	50.4	2.0	VW
3	3	T-Roc	2019	33492	Automatic	4825	Petrol	145	32.5	2.0	VW
4	4	T-Roc	2019	22900	Semi-Auto	6500	Petrol	150	39.8	1.5	VW
...	...	...	...	...	...	...	...	...	...	...	...
99182	10663	A3	2020	16999	Manual	4018	Petrol	145	49.6	1.0	Audi
99183	10664	A3	2020	16999	Manual	1978	Petrol	150	49.6	1.0	Audi
99184	10665	A3	2020	17199	Manual	609	Petrol	150	49.6	1.0	Audi
99185	10666	Q3	2017	19499	Automatic	8646	Petrol	150	47.9	1.4	Audi
99186	10667	Q3	2016	15999	Manual	11855	Petrol	150	47.9	1.4	Audi

99187 rows × 11 columns

In [3]:

```
feature_matrix=df.iloc[:,2:3]
target_vector=df.iloc[:,4]
```

In [4]:

```
feature_matrix.shape
```

Out[4]:

```
(99187, 1)
```

In [5]:

```
target_vector.shape
```

Out[5]:

```
(99187,)
```

In [6]:

```
from sklearn.preprocessing import StandardScaler
```

In [7]:

```
fs=StandardScaler().fit_transform(feature_matrix)
```

In [8]:
```
logr=LogisticRegression()
```

In [9]:
```
logr.fit(fs,target_vector)
```

Out[9]:
```
LogisticRegression()
```

In [10]:
```
observation=[[1]]
```

In [11]:
```
prediction=logr.predict(observation)
```

In [12]:
```
print(prediction)
```

```
['Manual']
```

In [13]:
```
logr.classes_
```

Out[13]:
```
array(['Automatic', 'Manual', 'Other', 'Semi-Auto'], dtype=object)
```

In [14]:
```
logr.predict_proba(observation)[0][0]
```

Out[14]:
```
0.1927580021954148
```

In [15]:
```
logr.predict_proba(observation)[0][1]
```

Out[15]:
```
0.4866261296054626
```

Logistic Regression 2

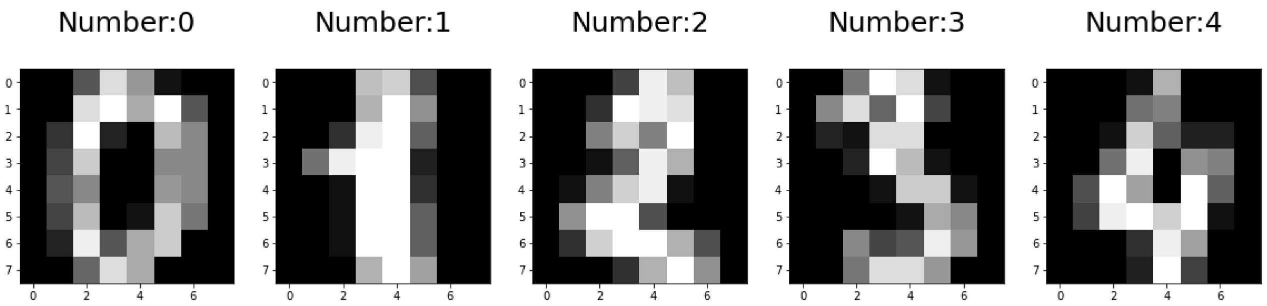
In [16]:
```
import re
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
```

In [17]:
```
digits=load_digits()
digits
```

Out[17]:
```
{'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',
 'pixel_1_2',
 'pixel_1_3',
 'pixel_1_4',
 'pixel_1_5',
 'pixel_1_6',
 'pixel_1_7',
 'pixel_2_0',
 'pixel_2_1',
 'pixel_2_2',
 'pixel_2_3',
 'pixel_2_4',
 'pixel_2_5',
 'pixel_2_6',
 'pixel_2_7',
 'pixel_3_0',
 'pixel_3_1',
 'pixel_3_2',
 'pixel_3_3',
 'pixel_3_4',
 'pixel_3_5',
 'pixel_3_6',
 'pixel_3_7',
 'pixel_4_0',
 'pixel_4_1',
 'pixel_4_2',
 'pixel_4_3',
 'pixel_4_4',
 'pixel_4_5',
 'pixel_4_6',
 'pixel_4_7',
 'pixel_5_0',
 'pixel_5_1',
 'pixel_5_2',
 'pixel_5_3',
 'pixel_5_4',
 'pixel_5_5',
 'pixel_5_6',
 'pixel_5_7',
 'pixel_6_0',
 'pixel_6_1',
 'pixel_6_2',
 'pixel_6_3',
 'pixel_6_4',
 'pixel_6_5',
 'pixel_6_6',
 'pixel_6_7',
 'pixel_7_0',
 'pixel_7_1',
 'pixel_7_2',
 'pixel_7_3',
 'pixel_7_4',
 'pixel_7_5',
 'pixel_7_6',
 'pixel_7_7'],
'target_names': array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
'images': array([[[ 0.,  0.,  5., ...,  1.,  0.,  0.],
        [ 0.,  0., 13., ..., 15.,  5.,  0.],
         [ 0.,  3., 15., ..., 11.,  8.,  0.],
         ...,
         [ 0.,  4., 11., ..., 12.,  7.,  0.],
         [ 0.,  2., 14., ..., 12.,  0.,  0.],
         [ 0.,  0.,  6., ...,  0.,  0.,  0.]],

        [[ 0.,  0.,  0., ...,  5.,  0.,  0.],
         [ 0.,  0.,  0., ...,  9.,  0.,  0.],
         [ 0.,  0.,  3., ...,  6.,  0.,  0.],
         ...,
         [ 0.,  0.,  1., ...,  6.,  0.,  0.],
         [ 0.,  0.,  1., ...,  6.,  0.,  0.],
         [ 0.,  0.,  0., ..., 10.,  0.,  0.]],

        [[ 0.,  0.,  0., ..., 12.,  0.,  0.],
         [ 0.,  0.,  3., ..., 14.,  0.,  0.],
         [ 0.,  0.,  8., ..., 16.,  0.,  0.],
         ...,
         [ 0.,  9., 16., ...,  0.,  0.,  0.],
         [ 0.,  3., 13., ..., 11.,  5.,  0.],
         [ 0.,  0.,  0., ..., 16.,  9.,  0.]],

        ...,

        [[ 0.,  0.,  1., ...,  1.,  0.,  0.],
         [ 0.,  0., 13., ...,  2.,  1.,  0.],
         [ 0.,  0., 16., ..., 16.,  5.,  0.],
         ...,
         [ 0.,  0., 16., ..., 15.,  0.,  0.],
         [ 0.,  0., 15., ..., 16.,  0.,  0.],
         [ 0.,  0.,  2., ...,  6.,  0.,  0.]],

        [[ 0.,  0.,  2., ...,  0.,  0.,  0.],
         [ 0.,  0., 14., ..., 15.,  1.,  0.],
         [ 0.,  4., 16., ..., 16.,  7.,  0.],
         ...,
         [ 0.,  0.,  0., ..., 16.,  2.,  0.],
         [ 0.,  0.,  4., ..., 16.,  2.,  0.],
         [ 0.,  0.,  5., ..., 12.,  0.,  0.]],

        [[ 0.,  0., 10., ...,  1.,  0.,  0.],
         [ 0.,  2., 16., ...,  1.,  0.,  0.],
         [ 0.,  0., 15., ..., 15.,  0.,  0.],
         ...,
         [ 0.,  4., 16., ..., 16.,  6.,  0.],
         [ 0.,  8., 16., ..., 16.,  8.,  0.],
         [ 0.,  1.,  8., ..., 12.,  1.,  0.]]]),
 'DESCR': ".. _digits_dataset:\n\nOptical recognition of handwritten digits dataset\n---
--------------------------------------------\n\n**Data Set Characteristics:**\n\n
:Number of Instances: 1797\n    :Number of Attributes: 64\n    :Attribute Information: 8
x8 image of integer pixels in the range 0..16.\n    :Missing Attribute Values: None\n
:Creator: E. Alpaydin (alpaydin '@' boun.edu.tr)\n    :Date: July; 1998\n\nThis is a cop
y of the test set of the UCI ML hand-written digits datasets\nhttps://archive.ics.uci.ed
u/ml/datasets/Optical+Recognition+of+Handwritten+Digits\n\nThe data set contains images
of hand-written digits: 10 classes where\neach class refers to a digit.\n\nPreprocessing
programs made available by NIST were used to extract\nnormalized bitmaps of handwritten
digits from a preprinted form. From a\ntotal of 43 people, 30 contributed to the trainin
g set and different 13\nto the test set. 32x32 bitmaps are divided into nonoverlapping b
locks of\n4x4 and the number of on pixels are counted in each block. This generates\nan
input matrix of 8x8 where each element is an integer in the range\n0..16. This reduces d
imensionality and gives invariance to small\ndistortions.\n\nFor info on NIST preprocess
ing routines, see M. D. Garris, J. L. Blue, G.\nT. Candela, D. L. Dimmick, J. Geist, P.
J. Grother, S. A. Janet, and C.\nL. Wilson, NIST Form-Based Handprint Recognition Syste
m, NISTIR 5469,\n1994.\n\n.. topic:: References\n\n  - C. Kaynak (1995) Methods of Combi
ning Multiple Classifiers and Their\n    Applications to Handwritten Digit Recognition,
MSc Thesis, Institute of\n    Graduate Studies in Science and Engineering, Bogazici Univ
ersity.\n  - E. Alpaydin, C. Kaynak (1998) Cascading Classifiers, Kybernetika.\n  - Ken
Tang and Ponnuthurai N. Suganthan and Xi Yao and A. Kai Qin.\n    Linear dimensionalityr
eduction using relevance weighted LDA. School of\n    Electrical and Electronic Engineer
ing Nanyang Technological University.\n    2005.\n  - Claudio Gentile. A New Approximate
Maximal Margin Classification\n    Algorithm. NIPS. 2000.\n"}
```

In [18]:

```
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=25)
```



In [19]:

```
x_train,x_test,y_train,y_test=train_test_split(digits.data,digits.target,test_size=0.30
```

In [20]:

```
logr=LogisticRegression(max_iter=10000)
```

In [21]:

```
logr.fit(x_train,y_train)
```

Out[21]:

```
LogisticRegression(max_iter=10000)
```

In [22]:

```
print(logr.predict(x_test))
```

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

In [23]:

```
print(logr.score(x_test,y_test))
```

```
0.9592592592592593
```