LOGISTIC REGRESSION WITH TITANIC DATASET

In [3]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score
from sklearn.model_selection import train_test_split
```

In [4]:
```
titanic = pd.read_csv('titanic.csv')
```

In [4]:
```
titanic.head()
```

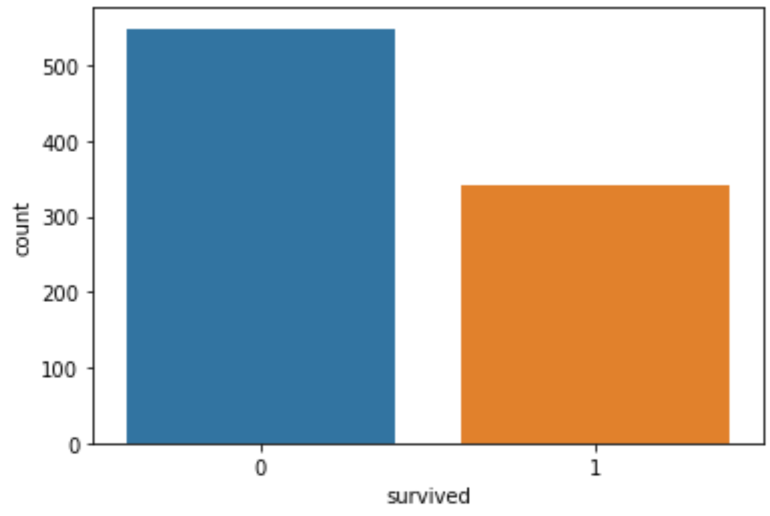
Out[4]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

EDA

In [5]:
```
sns.countplot(x='survived', data = titanic)
```

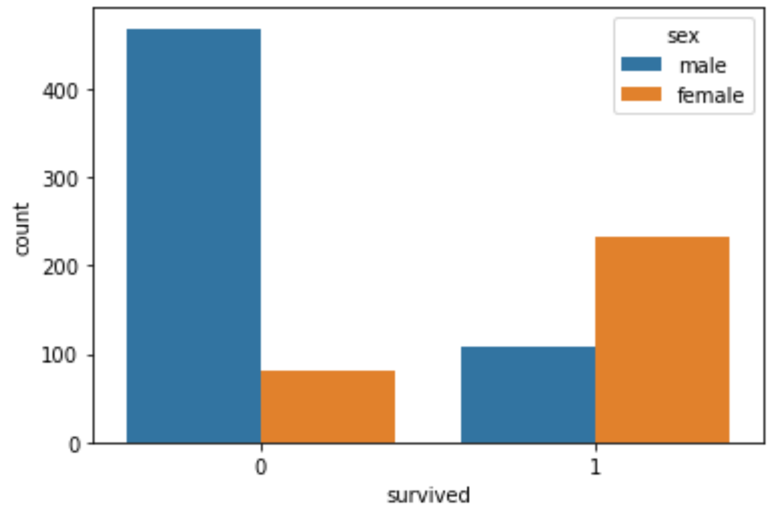
Out[5]: <AxesSubplot:xlabel='survived', ylabel='count'>



In [2]:
```
import pandas as pd
import seaborn as sns
from pandas_profiling import ProfileReport
#df = pd.read_csv('https://www.kaggle.com/competitions/titanic/data?select=train.csv', )
tips = sns.load_dataset('tips')
#EDA using pandas-profiling
profile = ProfileReport(tips, explorative=True)
#Saving results to a HTML file
profile.to_file("tips-eda.html")
```

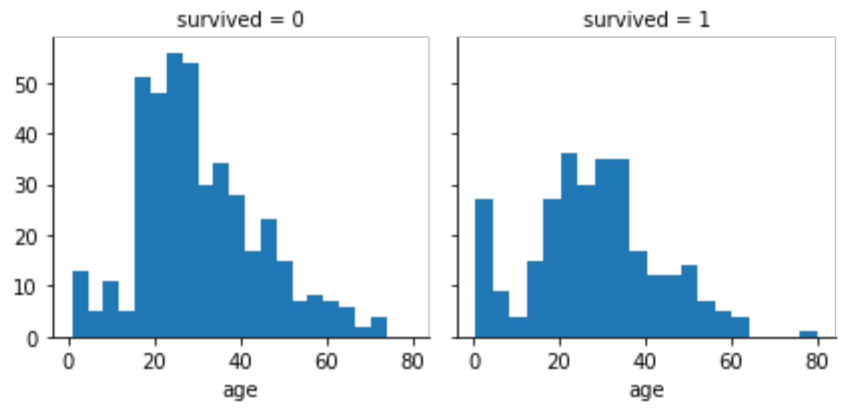
In [8]:
```
sns.countplot(x='survived', hue = 'sex', data = titanic)
```

Out[8]: <AxesSubplot:xlabel='survived', ylabel='count'>



In [11]:
```
h = sns.FacetGrid(titanic, col = 'survived')
h.map(plt.hist, "age", bins = 20)
```

Out[11]: <seaborn.axisgrid.FacetGrid at 0x7f96381228e0>



Data Wrangling

In [8]:
```
titanic.isnull().any().sum()
```

Out[8]: 4

In [18]:
```
titanic.dropna(inplace=True)
```

In [19]:
```
titanic.shape[0]
```

Out[19]: 182

Pre processings

In [22]:
```
sex = pd.get_dummies(titanic['sex'],drop_first=True)
sex[:5]
```

Out[22]:

	male
1	0
3	0
6	1
10	0
11	0

In [23]:
```
embark = pd.get_dummies(titanic['embarked'], drop_first=True)
embark[:5]
```

Out[23]:

	Q	S
1	0	0
3	0	1
6	0	1
10	0	1
11	0	1

In [24]:
```
cl = pd.get_dummies(titanic['pclass'], drop_first=True)
cl[:5]
```

Out[24]:

	2	3
1	0	0
3	0	0
6	0	0
10	0	1
11	0	0

In [25]:
```
titanic = pd.concat([titanic, sex, cl, embark], axis = 1)
```

In [26]:
```
titanic.head()
```

Out[26]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
6	0	1	male	54.0	0	0	51.8625	S	First	man	True	E	Southampton	no	True

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone

In [28]:
```
titanic.columns.values
```

Out[28]:
```
array(['survived', 'pclass', 'sex', 'age', 'sibsp', 'parch', 'fare',
       'embarked', 'class', 'who', 'adult_male', 'deck', 'embark_town',
       'alive', 'alone', 'male', 2, 3, 'Q', 'S'], dtype=object)
```

In [29]:
```
titanic.drop(['pclass', 'sex','embarked', 'class', 'who', 'adult_male', 'deck', 'embark_town',
       'alive', 'alone'], axis = 1, inplace=True)
```

In [30]:
```
titanic.head()
```

Out[30]:

	survived	age	sibsp	parch	fare	male	2	3	Q	S
1	1	38.0	1	0	71.2833	0	0	0	0	0
3	1	35.0	1	0	53.1000	0	0	0	0	1
6	0	54.0	0	0	51.8625	1	0	0	0	1
10	1	4.0	1	1	16.7000	0	0	1	0	1
11	1	58.0	0	0	26.5500	0	0	0	0	1

In [31]:
```
X = titanic.drop('survived', axis = 1)
y = titanic['survived']
```

In [45]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.73,random_state=101)
```

In [46]:
```
lr = LogisticRegression()
lr.fit(X_train,y_train)
```

```
/Users/sumitkumarshukla/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_logistic.py:763: Con
vergenceWarning: 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
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[46]:
```
LogisticRegression()
```

In [47]:
```
predictions = lr.predict(X_test)
```

In [48]:
```
predictions[:5]
```

Out[48]:
```
array([1, 0, 0, 1, 0])
```

In [49]:
```
np.array(y[:5])
```

Out[49]:
```
array([1, 1, 0, 1, 1])
```

In [50]:
```
print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.64      0.56      0.60        16
           1       0.81      0.85      0.83        34

    accuracy                           0.76        50
   macro avg       0.72      0.71      0.71        50
weighted avg       0.75      0.76      0.76        50
```

In [51]:
```
accuracy_score(y_test, predictions)*100
```

Out[51]:
```
76.0
```

In [52]:
```
X.columns
```

Out[52]:
```
Index(['age', 'sibsp', 'parch', 'fare', 'male', 2, 3, 'Q', 'S'], dtype='object')
```

prediction

In [56]:
```
lr.predict([[87.5, 0, 1, 87.9, 0, 0, 0, 0, 0]])
```

Out[56]: array([1])

In [59]:
```
titanic.survived.value_counts()/182 * 100
```

Out[59]:
```
1    67.582418
0    32.417582
Name: survived, dtype: float64
```

Image Classification

Predicting the Digits values from images

In [60]:
```
from sklearn.datasets import load_digits
```

In [61]:
```
digits = load_digits()
print(digits.DESCR)
```

```
.. _digits_dataset:

Optical recognition of handwritten digits dataset
--------------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 1797
    :Number of Attributes: 64
    :Attribute Information: 8x8 image of integer pixels in the range 0..16.
    :Missing Attribute Values: None
    :Creator: E. Alpaydin (alpaydin '@' boun.edu.tr)
    :Date: July; 1998

This is a copy of the test set of the UCI ML hand-written digits datasets
https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits

The data set contains images of hand-written digits: 10 classes where
each class refers to a digit.

Preprocessing programs made available by NIST were used to extract
normalized bitmaps of handwritten digits from a preprinted form. From a
total of 43 people, 30 contributed to the training set and different 13
to the test set. 32x32 bitmaps are divided into nonoverlapping blocks of
4x4 and the number of on pixels are counted in each block. This generates
an input matrix of 8x8 where each element is an integer in the range
0..16. This reduces dimensionality and gives invariance to small
distortions.

For info on NIST preprocessing routines, see M. D. Garris, J. L. Blue, G.
T. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Janet, and C.
L. Wilson, NIST Form-Based Handprint Recognition System, NISTIR 5469,
1994.

.. topic:: References

  - C. Kaynak (1995) Methods of Combining Multiple Classifiers and Their
    Applications to Handwritten Digit Recognition, MSc Thesis, Institute of
    Graduate Studies in Science and Engineering, Bogazici University.
  - E. Alpaydin, C. Kaynak (1998) Cascading Classifiers, Kybernetika.
  - Ken Tang and Ponnuthurai N. Suganthan and Xi Yao and A. Kai Qin.
    Linear dimensionalityreduction using relevance weighted LDA. School of
    Electrical and Electronic Engineering Nanyang Technological University.
    2005.
  - Claudio Gentile. A New Approximate Maximal Margin Classification
    Algorithm. NIPS. 2000.
```

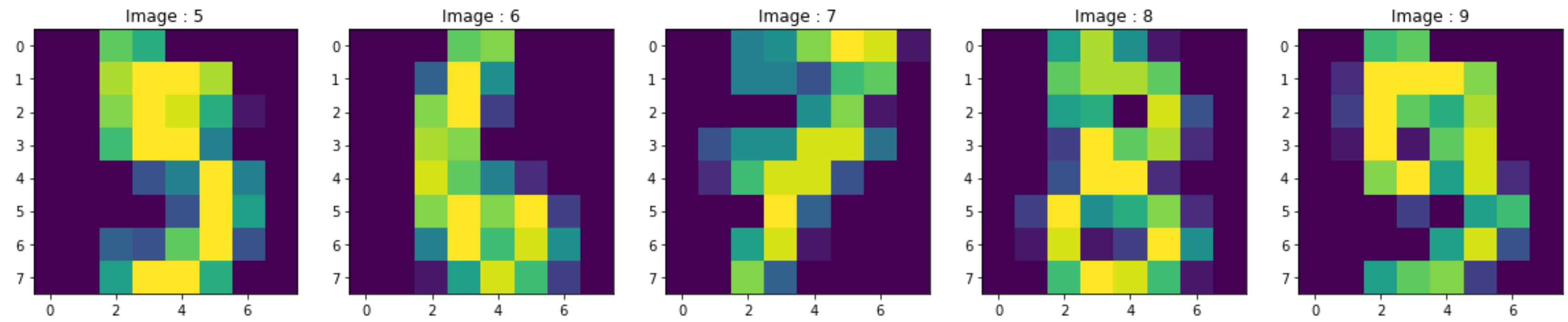
determine the total number of images and labels

In [62]:
```
print('Image Data shape = ',digits.data.shape)
print('Label data shape = ',digits.target.shape)
```

```
Image Data shape =  (1797, 64)
Label data shape =  (1797,)
```

Displaying some of the images with their labels

In [67]:
```
plt.figure(figsize = (20, 4))
for index, (image, label) in enumerate(zip(digits.data[5:10], digits.target[5:10])):
    plt.subplot(1, 5, index+1)
    plt.imshow(np.reshape(image,(8,8)))
    plt.title('Image : {}'.format(label))
```



dataset splitting

In [68]:
```
X_train, X_test, y_train, y_test = train_test_split(digits.data, digits.target, test_size=0.23, random_state
```

In [70]:
```
print(X_train.shape, X_test.shape)
```

```
(1383, 64) (414, 64)
```

In [71]:
```
ld = LogisticRegression()
ld.fit(X_train, y_train)
```

```
/Users/sumitkumarshukla/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_logistic.py:763: Con
vergenceWarning: 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
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[71]:
```
LogisticRegression()
```

In [73]:
```
print(classification_report(y_test, ld.predict(X_test)))
```

```
              precision    recall  f1-score   support

           0       1.00      0.97      0.99        39
           1       0.92      0.95      0.93        37
           2       0.95      1.00      0.98        40
           3       1.00      0.98      0.99        43
           4       0.98      0.93      0.95        44
           5       0.98      0.93      0.95        44
           6       1.00      1.00      1.00        42
           7       0.98      1.00      0.99        42
           8       0.89      0.94      0.91        33
           9       0.94      0.94      0.94        50

    accuracy                           0.96       414
   macro avg       0.96      0.96      0.96       414
weighted avg       0.96      0.96      0.96       414
```

In [75]:
```
predictions = ld.predict(X_test)
```

In [76]:
```
accuracy_score(y_test, predictions)*100
```

Out[76]:
```
96.37681159420289
```

In [80]:
```
print(ld.predict(X_test[0:10]))
```

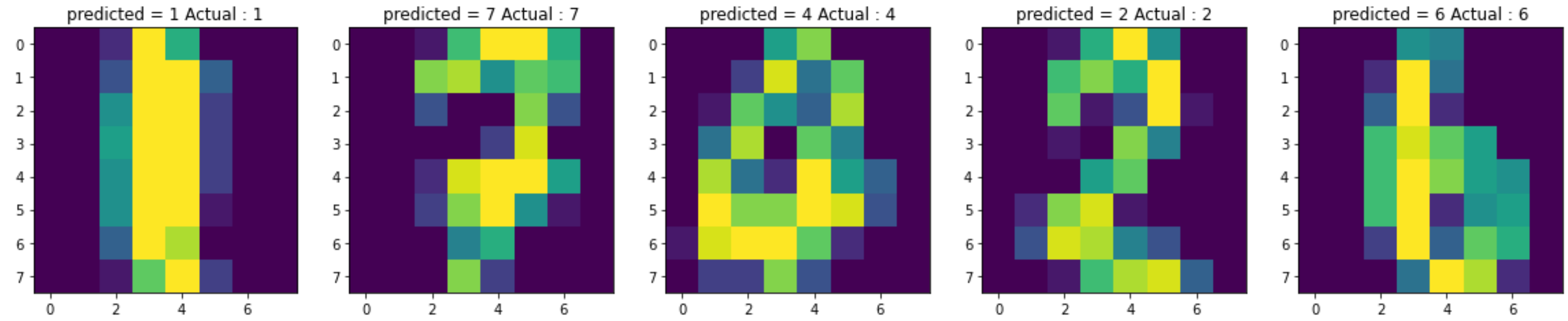
```
[1 8 4 6 3 5 1 7 4 8]
```

In [81]:
```
print(y_test[0:10])
```

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

In [82]:
```
index = 0
logre = []
for Predicted, actual in zip(predictions, y_test):
    if Predicted == actual:
        logre.append(index)
    index += 1
```

In [91]:
```
plt.figure(figsize = (20, 4))
for image, label in enumerate(logre[5:10]):
    plt.subplot(1, 5, image+1)
    plt.imshow(np.reshape(X_test[label], (8,8)))
    plt.title('predicted = {} Actual : {}'.format(predictions[label], y_test[label]))
```


predicted = 1 Actual : 1
predicted = 7 Actual : 7
predicted = 4 Actual : 4
predicted = 2 Actual : 2
predicted = 6 Actual : 6

In [ ]: