NETAJI SUBHAS UNIVERSITY OF TECHNOLOGY

MACHINE LEARNING

Lab Assignment 6

(CACSC17)

SUBMITTED BY: -
Name: *HARSH KUMAR*
Branch: CSAI
Roll no.:2021UCA1829

Objective: Write a program implement a Artificial neural network (ANN) and predict the accuracy of churn modeling dataset to check the customer who close their account.

Dataset overview and preprocessing

Loading the Churn modeling dataset and performing initial data exploration. The dataset contains 14 features,and the target variable 'Exited' indicates the presence (1) or deletion (0) of the particular bank account.

```
[ ] import numpy as np
    import pandas as pd

[ ] #Import Dataset
    df = pd.read_csv('Churn_Modelling.csv')

    df.shape

    (10000, 14)

    df.head()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
2	3	15619304	Onio	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
3	4	15701354	Boni	699	France	Female	39	1	0.00	2	0	0	93826.63	0
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0

```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   RowNumber        10000 non-null  int64
 1   CustomerId       10000 non-null  int64
 2   Surname          10000 non-null  object
 3   CreditScore      10000 non-null  int64
 4   Geography        10000 non-null  object
 5   Gender           10000 non-null  object
 6   Age              10000 non-null  int64
 7   Tenure           10000 non-null  int64
 8   Balance          10000 non-null  float64
 9   NumOfProducts    10000 non-null  int64
 10  HasCrCard        10000 non-null  int64
 11  IsActiveMember   10000 non-null  int64
 12  EstimatedSalary  10000 non-null  float64
 13  Exited           10000 non-null  int64
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB

[ ] df.drop(columns = ['RowNumber','CustomerId','Surname'],inplace=True)
```

```
df['Geography'].value_counts()
```

```
France     5014
Germany    2509
Spain      2477
Name: Geography, dtype: int64
```

```
df['Gender'].value_counts()
```

```
Male      5457
Female    4543
Name: Gender, dtype: int64
```

```
df.drop(columns = ['RowNumber','CustomerId','Surname'],inplace=True)
```

```
df = pd.get_dummies(df,columns=['Geography','Gender'],drop_first=True)
```

```
df.drop(columns = ['RowNumber','CustomerId','Surname'],inplace=True)
```

```
df = pd.get_dummies(df,columns=['Geography','Gender'],drop_first=True)
```

```
df
```

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited	Geography_Germany	Geography_Spain	Gender_Male
0	619	42	2	0.00	1	1	1	101348.88	1	0	0	0
1	608	41	1	83807.86	1	0	1	112542.58	0	0	1	0
2	502	42	8	159660.80	3	1	0	113931.57	1	0	0	0
3	699	39	1	0.00	2	0	0	93826.63	0	0	0	0
4	850	43	2	125510.82	1	1	1	79084.10	0	0	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...
9995	771	39	5	0.00	2	1	0	96270.64	0	0	0	1
9996	516	35	10	57369.61	1	1	1	101699.77	0	0	0	1
9997	709	36	7	0.00	1	0	1	42085.58	1	0	0	0
9998	772	42	3	75075.31	2	1	0	92888.52	1	1	0	1
9999	792	28	4	130142.79	1	1	0	38190.78	0	0	0	0

Model Training and Evaluation

```
X = df.drop(columns=['Exited'])
y = df['Exited'].values

from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=1)
```

```
[ ] X_train.shape
```

```
(8000, 11)
```

```
[ ] from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaledf = scaler.transform(X_test)
```

```
[ ] X_train_scaled
```

```
array([[-0.23082038, -0.94449979, -0.70174202, ...,  1.71490137,
        -0.57273139,  0.91509065],
       [-0.25150912, -0.94449979, -0.35520275, ..., -0.58312392,
        -0.57273139, -1.09278791],
       [-0.3963303 ,  0.77498705,  0.33787579, ...,  1.71490137,
        -0.57273139, -1.09278791],
       ...,
       [ 0.22433188,  0.58393295,  1.3774936 , ..., -0.58312392,
        -0.57273139, -1.09278791],
       [ 0.13123255,  0.01077067,  1.03095433, ..., -0.58312392,
        -0.57273139, -1.09278791],
       [ 1.1656695 ,  0.29735181,  0.33787579, ...,  1.71490137,
        -0.57273139,  0.91509065]])
```

Theis related to data preprocessing using the StandardScaler from the scikit-learn (sklearn) library in Python. It is commonly used for standardizing (scaling) the features of a dataset, which means it transforms the data in a way that each feature (column) has a mean of 0 and a standard deviation of 1. This can be important for certain machine learning algorithms that are sensitive to the scale of the input features

```
[ ] import tensorflow as tf
    import keras
    from keras.models import Sequential
    from keras.layers import Activation, Dense
```

```
[ ] model = Sequential()

    model.add(Dense(3,activation='sigmoid',input_dim=11))
    model.add(Dense(1,activation='sigmoid'))
    model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_2 (Dense)             (None, 3)                 36

 dense_3 (Dense)             (None, 1)                 4

=================================================================
Total params: 40 (160.00 Byte)
Trainable params: 40 (160.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

This code snippet demonstrates how to create a simple neural network model using the Keras API, which is now integrated with TensorFlow. Here we create an instance of a Sequential model. In Keras, a Sequential model represents a linear stack of layers.

Dense(3, activation='sigmoid', input_dim=11) adds a dense (fully connected) layer to the model with 3 units, a sigmoid activation function, and an input dimension of 11. This layer connects to the input data with 11 features.

The first layer is specified with input_dim because it defines the input shape for the model.

Dense(1, activation='sigmoid') adds another dense layer with 1 unit and a sigmoid activation function. This is the output layer of the neural network.

This code in simple terms creates a simple feed forward neural network with one hidden layer containing 3 units, a sigmoid activation function, and an input dimension of 11 features. The output layer has one unit with a sigmoid activation function. The model.summary() command is used to display a summary of the model's architecture, including the number of parameters in each layer.

```
model.compile(loss='binary_crossentropy', optimizer='Adam',metrics=['accuracy'])
```

```
history = model.fit(X_train_scaled, y_train, batch_size=50,epochs=10,verbose=1,validation_split=0.2)
```

```
Epoch 1/10
128/128 [==============================] - 1s 2ms/step - loss: 1.0269 - accuracy: 0.2028 - val_loss: 0.9514 - val_accuracy: 0.2025
Epoch 2/10
128/128 [==============================] - 0s 2ms/step - loss: 0.8711 - accuracy: 0.2109 - val_loss: 0.8169 - val_accuracy: 0.2338
Epoch 3/10
128/128 [==============================] - 0s 1ms/step - loss: 0.7587 - accuracy: 0.3389 - val_loss: 0.7200 - val_accuracy: 0.4444
Epoch 4/10
128/128 [==============================] - 0s 1ms/step - loss: 0.6777 - accuracy: 0.5886 - val_loss: 0.6502 - val_accuracy: 0.6888
Epoch 5/10
128/128 [==============================] - 0s 2ms/step - loss: 0.6184 - accuracy: 0.7683 - val_loss: 0.5987 - val_accuracy: 0.7894
Epoch 6/10
128/128 [==============================] - 0s 1ms/step - loss: 0.5745 - accuracy: 0.8008 - val_loss: 0.5603 - val_accuracy: 0.7975
Epoch 7/10
128/128 [==============================] - 0s 1ms/step - loss: 0.5417 - accuracy: 0.7970 - val_loss: 0.5317 - val_accuracy: 0.7975
Epoch 8/10
128/128 [==============================] - 0s 2ms/step - loss: 0.5176 - accuracy: 0.7972 - val_loss: 0.5110 - val_accuracy: 0.7975
Epoch 9/10
128/128 [==============================] - 0s 1ms/step - loss: 0.4999 - accuracy: 0.7972 - val_loss: 0.4962 - val_accuracy: 0.7975
Epoch 10/10
128/128 [==============================] - 0s 2ms/step - loss: 0.4872 - accuracy: 0.7972 - val_loss: 0.4851 - val_accuracy: 0.7975
```

```
y_pred = model.predict(X_test)
```

```
63/63 [==============================] - 0s 761us/step
```

After training, the history variable will contain information about the training process, including loss and accuracy values at each epoch. We can use this information to visualize the training progress and evaluate the model's performance

Accuracy

Accuracy represents the overall correctness of a model's predictions. It calculates the ratio of correctly predicted instances to the total instances.

In this case we have a model accuracy of 79.25%.

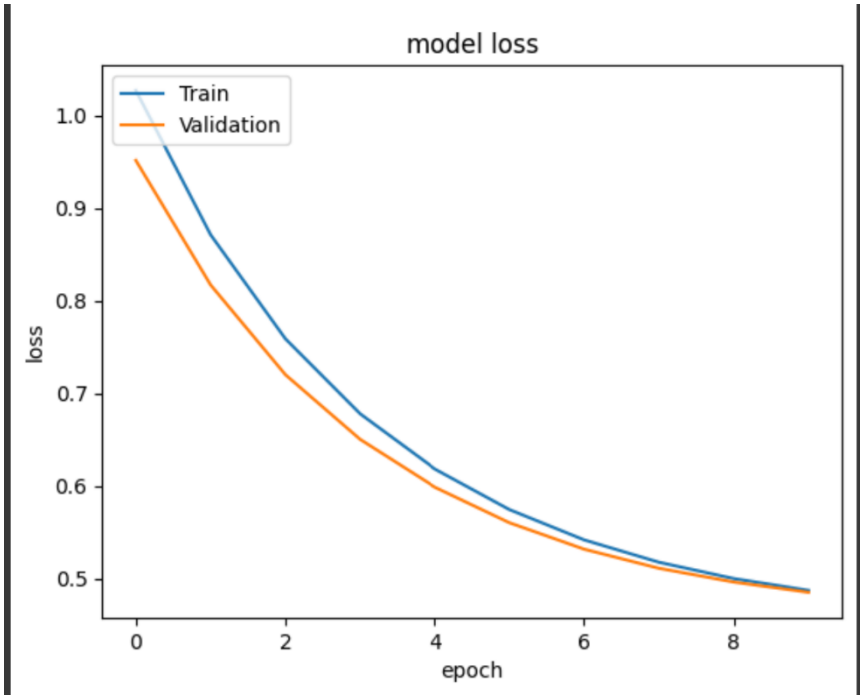
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred)
```

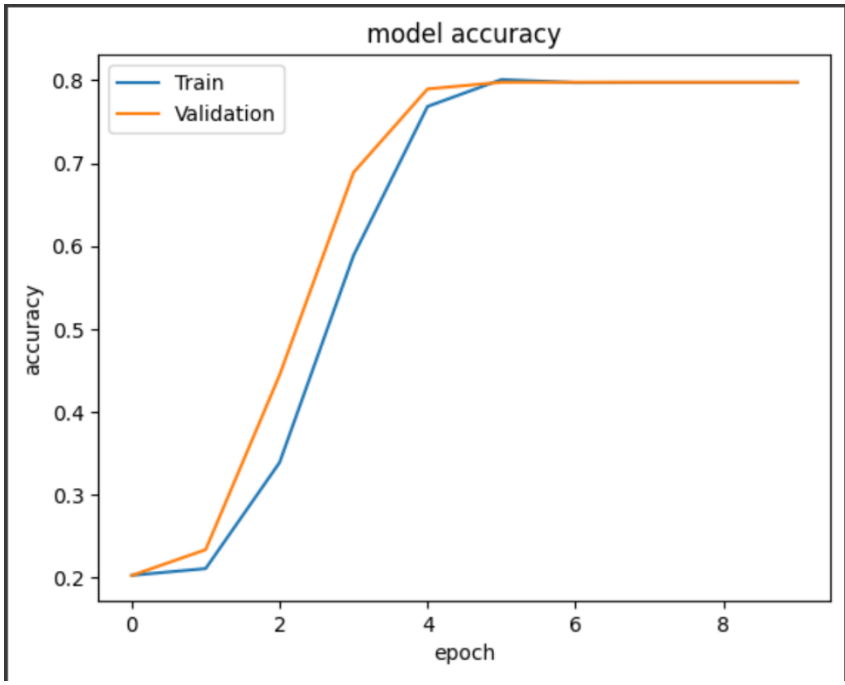
```
0.7925
```

Plotting the Accuracy and Error

```
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```

These plots are useful for visualizing how the model is performing during training. We can analyze them to see if the model is overfitting (when the training loss continues to decrease while the validation loss increases) or if it's converging to a good solution. The accuracy plot shows the training and validation accuracy trends, while the loss plot shows the training and validation loss trends over epochs.





Conclusion

Understood the working of ANN and implemented it in finding the accuracy of the churn modelling dataset .

NETAJI SUBHAS UNIVERSITY OF TECHNOLOGY

MACHINE LEARNING

Lab Assignment 5

(CACSC17)

SUBMITTED BY: -
Name: *HARSH KUMAR*
Branch: CSAI
Roll no.:2021UCA1829

Comapring the performance parameters of logistic regression model for binary classification

In [1]:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

In [2]:

```
heart_data = pd.read_csv("heart.csv")
heart_data
```

Out[2]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	th
0	63	1	3	145	233	1	0	150	0	2.3	0	0	
1	37	1	2	130	250	0	1	187	0	3.5	0	0	
2	41	0	1	130	204	0	0	172	0	1.4	2	0	
3	56	1	1	120	236	0	1	178	0	0.8	2	0	
4	57	0	0	120	354	0	1	163	1	0.6	2	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
298	57	0	0	140	241	0	1	123	1	0.2	1	0	
299	45	1	3	110	264	0	1	132	0	1.2	1	0	
300	68	1	0	144	193	1	1	141	0	3.4	1	2	
301	57	1	0	130	131	0	1	115	1	1.2	1	1	
302	57	0	1	130	236	0	0	174	0	0.0	1	1	

303 rows × 14 columns

Handling Missing Values

In [3]:

```
heart_data.isnull().sum()
```

Out[3]:

```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

No missing data in the dataset

In [4]:

```
#Statiscal measures of the data

heart_data.describe()
```

Out[4]:

	age	sex	cp	trestbps	chol	fbs	rest
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000

target = 1 -> defective heart

target = 0 -> healthy heart

In [5]:

```
# Splitting features and target

X = heart_data.drop(columns = 'target', axis = 1)
Y = heart_data['target']

print(X)
print(Y)
```

```
     age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpe
ak  \
0     63    1   3       145   233    1        0      150      0
2.3
1     37    1   2       130   250    0        1      187      0
3.5
2     41    0   1       130   204    0        0      172      0
1.4
3     56    1   1       120   236    0        1      178      0
0.8
4     57    0   0       120   354    0        1      163      1
0.6
..   ...  ...  ..       ...   ...  ...      ...      ...    ...
...
298   57    0   0       140   241    0        1      123      1
0.2
299   45    1   3       110   264    0        1      132      0
1.2
300   68    1   0       144   193    1        1      141      0
3.4
301   57    1   0       130   131    0        1      115      1
1.2
302   57    0   1       130   236    0        0      174      0
0.0

     slope  ca  thal
0        0   0     1
1        0   0     2
2        2   0     2
3        2   0     2
4        2   0     2
..     ...  ..   ...
298      1   0     3
299      1   0     3
300      1   2     3
301      1   1     3
302      1   1     2

[303 rows x 13 columns]
0      1
1      1
2      1
3      1
4      1
      ..
298    0
299    0
300    0
301    0
302    0
Name: target, Length: 303, dtype: int64
```

Splitting data for training & testing

In [6]:

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.3, stratify
```

Model Training

In [7]:

```
model = LogisticRegression(max_iter = 1000)

model.fit(X_train,  Y_train)
```

Out[7]:

```
LogisticRegression(max_iter=1000)
```

Model Evaluation

Accuracy Score

It may be defined as the number of correct predictions made as a ratio of all predictions made. It can be calculated using the confusion matrix.

Accuracy = (TP + TN)/(TP + FP + TN + FN)

In [8]:

```
from sklearn.metrics import accuracy_score

X_train_prediction  =  model.predict(X_train)
training_data_accuracy  =  accuracy_score(Y_train,  X_train_prediction)

training_data_accuracy
```

Out[8]:

```
0.8490566037735849
```

In [9]:

```
# accuracy on test data
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(Y_test, X_test_prediction)
print(test_data_accuracy)
```

```
0.8461538461538461
```

In [10]:

```
from sklearn.metrics import confusion_matrix

cf_matrix = confusion_matrix(Y_test, X_test_prediction)
print(cf_matrix)
```

```
[[37  4]
 [10 40]]
```

In [11]:

```
tn, fp, fn, tp = cf_matrix.ravel()

print(tn, fp, fn, tp)
```

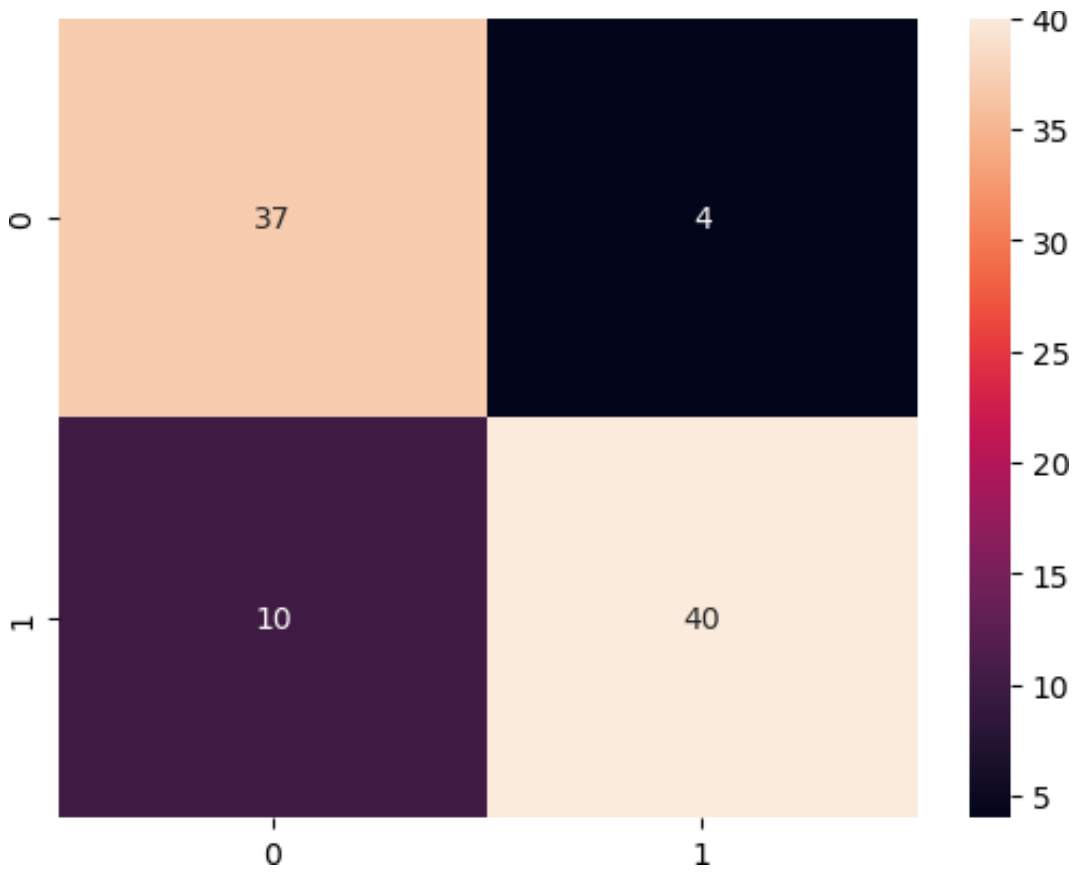
```
37 4 10 40
```

In [12]:

```
import seaborn as sns
sns.heatmap(cf_matrix, annot = True)
```

Out[12]:

<AxesSubplot:>



In [13]:

```
from sklearn.metrics import precision_score

# precision for training data predictions
precision_train = precision_score(Y_train, X_train_prediction)
print('Training data Precision = ', precision_train)

# precision for test data predictions
precision_test = precision_score(Y_test, X_test_prediction)
print('Test data Precision = ', precision_test)
```

```
Training data Precision =  0.8267716535433071
Test data Precision =  0.9090909090909091
```

In [14]:

```
from sklearn.metrics import recall_score

# recall for training data predictions
recall_train = recall_score(Y_train, X_train_prediction)
print('Training data Recall = ', recall_train)

# recall for test data predictions
recall_test = recall_score(Y_test, X_test_prediction)
print('Test data Recall =', recall_test)
```

```
Training data Recall =  0.9130434782608695
Test data Recall = 0.8
```

In [15]:

```
from sklearn.metrics import f1_score

# F1 score for training data predictions
f1_score_train = f1_score(Y_train, X_train_prediction)
print('Training data F1 Score = ', f1_score_train)

# F1 Score for test data predictions
f1_score_test = recall_score(Y_test, X_test_prediction)
print('Test data F1 Score = ', f1_score_test)
```

```
Training data F1 Score =  0.8677685950413223
Test data F1 Score =  0.8
```

In [16]:

```
from sklearn.metrics import log_loss

# loss on training data
X_train_prediction = model.predict(X_train)
training_data_loss = log_loss(Y_train, X_train_prediction, eps = 1e-15)
print('Loss on training data = ', training_data_loss)

# loss on test data
X_test_prediction = model.predict(X_test)
test_data_loss = log_loss(Y_test, X_test_prediction, eps = 1e-15)
print('Loss on testing data = ', test_data_loss)
```

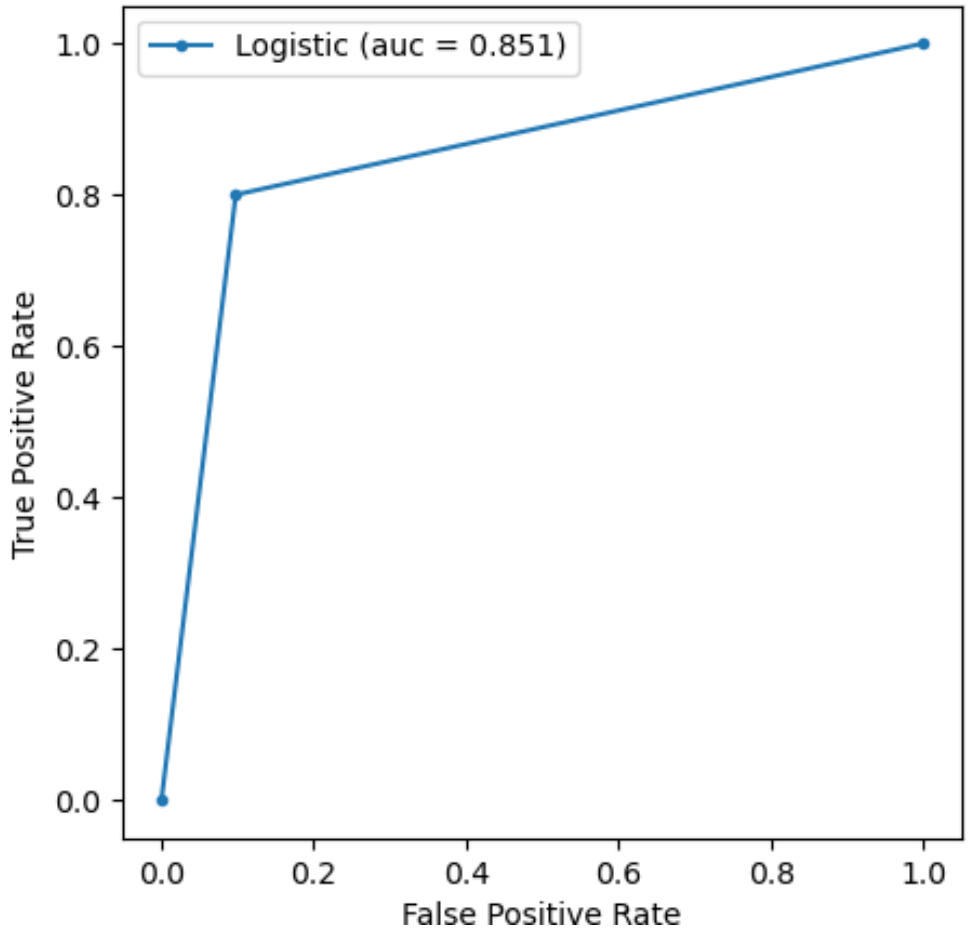
```
Loss on training data =  5.2134831876443695
Loss on testing data =  5.313693054049125
```

In [ ]:

```
```

In [17]:

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

logistic_fpr, logistic_tpr, threshold = roc_curve(Y_test, X_test_prediction)
auc_logistic = auc(logistic_fpr, logistic_tpr)

plt.figure(figsize=(5, 5), dpi=100)
plt.plot(logistic_fpr, logistic_tpr, marker = '.', label = 'Logistic (auc = %0.3f)'

plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')

plt.legend()

plt.show()
```



In [ ]:

NETAJI SUBHAS UNIVERSITY OF TECHNOLOGY

MACHINE LEARNING

Lab Assignment 4

(CACSC17)

SUBMITTED BY: -
Name: *HARSH KUMAR*
Branch: CSAI
Roll no.:2021UCA1829

In [1]:

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from  sklearn.model_selection  import  train_test_split
from sklearn. linear_model import LogisticRegression

from sklearn. linear_model import LinearRegression
from sklearn.metrics import r2_score, accuracy_score, confusion_matrix
```

Linear Regression Model

Reading data

In [2]:

```
dataset  =  pd.read_csv('insurance.csv')
dataset
```

Out[2]:

	age	sex	bmi	children	smoker	region	expenses
0	19	female	27.9	0	yes	southwest	16884.92
1	18	male	33.8	1	no	southeast	1725.55
2	28	male	33.0	3	no	southeast	4449.46
3	33	male	22.7	0	no	northwest	21984.47
4	32	male	28.9	0	no	northwest	3866.86
...	...	...	...	...	...	...	...
1333	50	male	31.0	3	no	northwest	10600.55
1334	18	female	31.9	0	no	northeast	2205.98
1335	18	female	36.9	0	no	southeast	1629.83
1336	21	female	25.8	0	no	southwest	2007.95
1337	61	female	29.1	0	yes	northwest	29141.36

1338 rows × 7 columns

Cleaning Data

i. Handle NULL values

ii. Change string to int using label encoding

In [3]:

```
dataset.isna().sum()
```

Out[3]:

```
age         0
sex         0
bmi         0
children    0
smoker      0
region      0
expenses    0
dtype: int64
```

No NULL values in dataset

In [4]:

```
label = LabelEncoder()

label.fit  (dataset.sex.drop_duplicates())
dataset.sex = label.transform(dataset. sex)

label.fit  (dataset.smoker.drop_duplicates())
dataset.smoker  =  label.transform(dataset.smoker)

label.fit(dataset.region.drop_duplicates())
dataset. region = label.transform(dataset. region)

dataset
```

Out[4]:

	age	sex	bmi	children	smoker	region	expenses
0	19	0	27.9	0	1	3	16884.92
1	18	1	33.8	1	0	2	1725.55
2	28	1	33.0	3	0	2	4449.46
3	33	1	22.7	0	0	1	21984.47
4	32	1	28.9	0	0	1	3866.86

...	...	...	...	...	...	...	...
1333	50	1	31.0	3	0	1	10600.55
1334	18	0	31.9	0	0	0	2205.98
1335	18	0	36.9	0	0	2	1629.83
1336	21	0	25.8	0	0	3	2007.95
1337	61	0	29.1	0	1	1	29141.36

1338 rows × 7 column

Training model of linear regression for predictive analysis

In [5]:

```
x_lin = dataset.drop(['expenses'], axis = 1)
y_lin = dataset[['expenses']]

x_lin_train, x_lin_test, y_lin_train, y_lin_test = train_test_split(x_lin, y_lin, te

linear_model = LinearRegression()
linear_model.fit(x_lin_train,  y_lin_train)
```

Out[5]:

```
LinearRegression()
```

In [6]:

```
y_pred  =  linear_model.predict(x_lin_test)

pred = pd.DataFrame(x_lin_test)
pred['actual cost'] = y_lin_test
pred['predicted cost'] = y_pred

pred
```

Out[6]:

	age	sex	bmi	children	smoker	region	actual cost	predicted cost
764	45	0	25.2	2	0	0	9095.07	8938.656851
887	36	0	30.0	0	0	1	5272.18	7062.787354
890	64	0	26.9	0	1	1	29330.98	36939.995651
1293	46	1	25.7	3	0	1	9301.89	9580.973660
259	19	1	31.9	0	1	1	33750.29	27000.270665
...	...	...	...	...	...	...	...	...

701	50	0	44.7	0	0	0	9541.70	16117.004611
672	36	1	29.7	0	0	2	4399.73	6743.155522
1163	18	0	28.2	0	0	0	2200.83	2059.594911
1103	58	1	36.1	0	0	2	11363.28	14704.031818
1295	20	1	22.0	1	0	3	1964.78	2.541240

402 rows × 8 columns

R2 score of the linear regression model

R2 score is the proportion of the variation in the dependent variable that is predictable from the independent variable.

In [7]:

```
r2_score(y_lin_test,  y_pred)
```

Out[7]:

```
0.7694626233326285
```

Logistic Regression Model

In [8]:

```
df = pd.read_csv('mushrooms.csv')
df
```

Out[8]:

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...

0	p	x	s	n	t	p	f	c	n	k	...
1	e	x	s	y	t	a	f	c	b	k	...
2	e	b	s	w	t	l	f	c	b	n	...
3	p	x	y	w	t	p	f	c	n	n	...
4	e	x	s	g	f	n	f	w	b	k	...
...	...	...	...	...	...	...	...	...	...	...	...
8119	e	k	s	n	f	n	a	c	b	y	...
8120	e	x	s	n	f	n	a	c	b	y	...
8121	e	f	s	n	f	n	a	c	b	n	...
8122	p	k	y	n	f	y	f	c	n	b	...
8123	e	x	s	n	f	n	a	c	b	y	...

8124 rows × 23 columns

Cleaning Data

i. Remove NULL values

ii. Convert categories to integers

In [9]:

```
df.isna().sum()
```

Out[9]:

```
class                       0
cap-shape                   0
cap-surface                 0
cap-color                   0
bruises                     0
odor                        0
gill-attachment             0
gill-spacing                0
gill-size                   0
gill-color                  0
stalk-shape                 0
stalk-root                  0
stalk-surface-above-ring    0
stalk-surface-below-ring    0
stalk-color-above-ring      0
```

```
stalk-color-below-ring      0
veil-type                   0
veil-color                  0
ring-number                 0
ring-type                   0
spore-print-color           0
population                  0
habitat                     0
dtype: int64
```

No NULL values in dataset

In [10]:

```
label_encoder = LabelEncoder()
for i in df.columns:
  df[i]  =  label_encoder.fit_transform(df[i])

df
```

Out[10]:

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...
0	1	5	2	4	1	6	1	0	1	4	...
1	0	5	2	9	1	0	1	0	0	4	...
2	0	0	2	8	1	3	1	0	0	5	...
3	1	5	3	8	1	6	1	0	1	5	...
4	0	5	2	3	0	5	1	1	0	4	...
...	...	...	...	...	...	...	...	...	...	...	...
8119	0	3	2	4	0	5	0	0	0	11	...
8120	0	5	2	4	0	5	0	0	0	11	...
8121	0	2	2	4	0	5	0	0	0	5	...
8122	1	3	3	4	0	8	1	0	1	0	...
8123	0	5	2	4	0	5	0	0	0	11	...

8124 rows × 23 columns

Training logistic regression model

In [11]:

```
x_log = df.drop("class", axis = 1)
y_log = df["class"]

x_log_train, x_log_test, y_log_train, y_log_test = train_test_split(x_log, y_log, te

logistic_model = LogisticRegression(max_iter = 500)
logistic_model.fit(x_log_train,  y_log_train)

pred  =  logistic_model.predict(x_log_test)
```

Calculating accuracy score for logistic model

In [12]:

```
logistic_model.score(x_log_test,  y_log_test)
```

Out[12]:

```
0.9491386382280558
```

In [13]:

```
confusion_matrix(y_log_test,  pred)
```

Out[13]:

```
array([[1201,   56],
       [  68, 1113]], dtype=int64)
```

In [ ]:

```

```

NETAJI SUBHAS UNIVERSITY OF TECHNOLOGY

MACHINE LEARNING

Lab Assignment 3

(CACSC17)

SUBMITTED BY: -
Name: *HARSH KUMAR*
Branch: CSAI
Roll no.:2021UCA1829

1. We first start by importing the csv file through which we perform the logistic regression

In [2]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import sklearn
import matplotlib.pyplot as plt
%matplotlib inline
import math

titanic_data  =  pd.read_csv("titanic.csv")titanic_data.head(10)
```

Out[2]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736

1. Once we have the csv file we start to analyze the data to find the parameters that have a strong relation between them so that we can train the model accordingly-

In [3]:

```
print ("Number Of Passenger in Original data:" +str(len(titanic_data.index)))
```

```
Number Of Passenger in Original data:891
```

Analysing Data

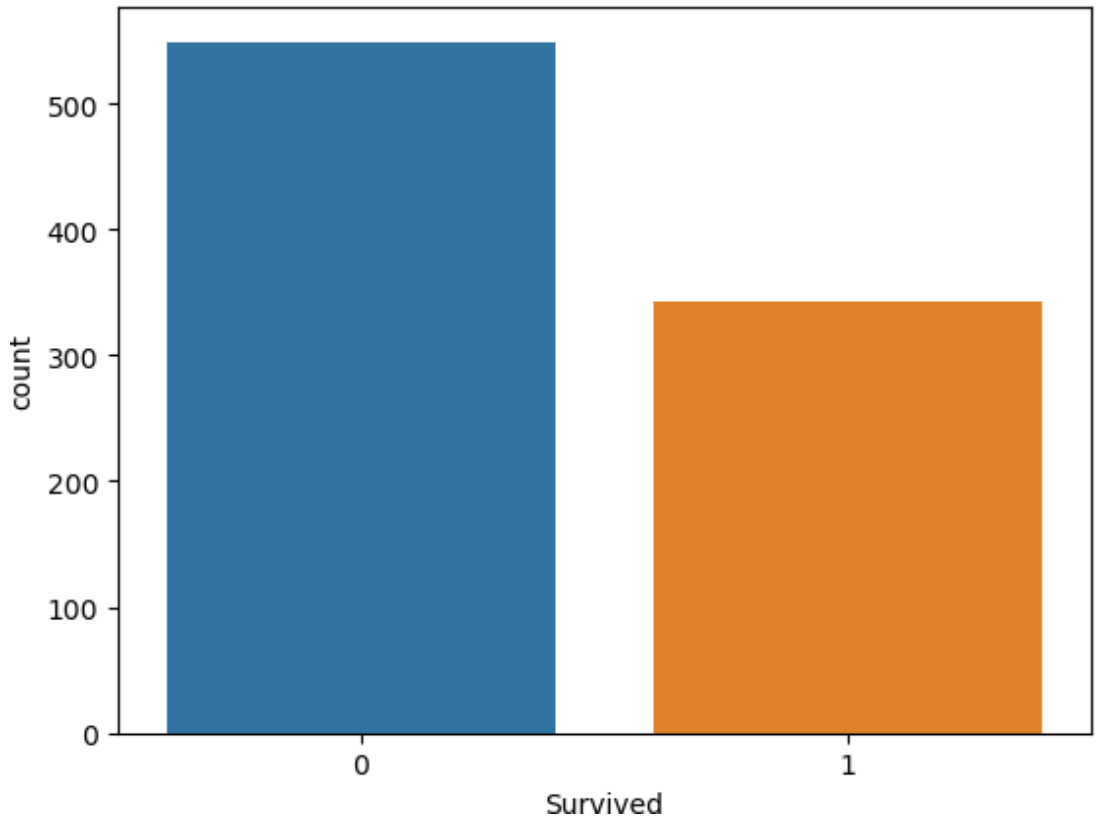
a. A plot that finds the number of people who survived.

In [4]:

```
sns.countplot(x="Survived",  data=titanic_data)
```

Out[4]:

```
<AxesSubplot:xlabel='Survived',  ylabel='count'>
```



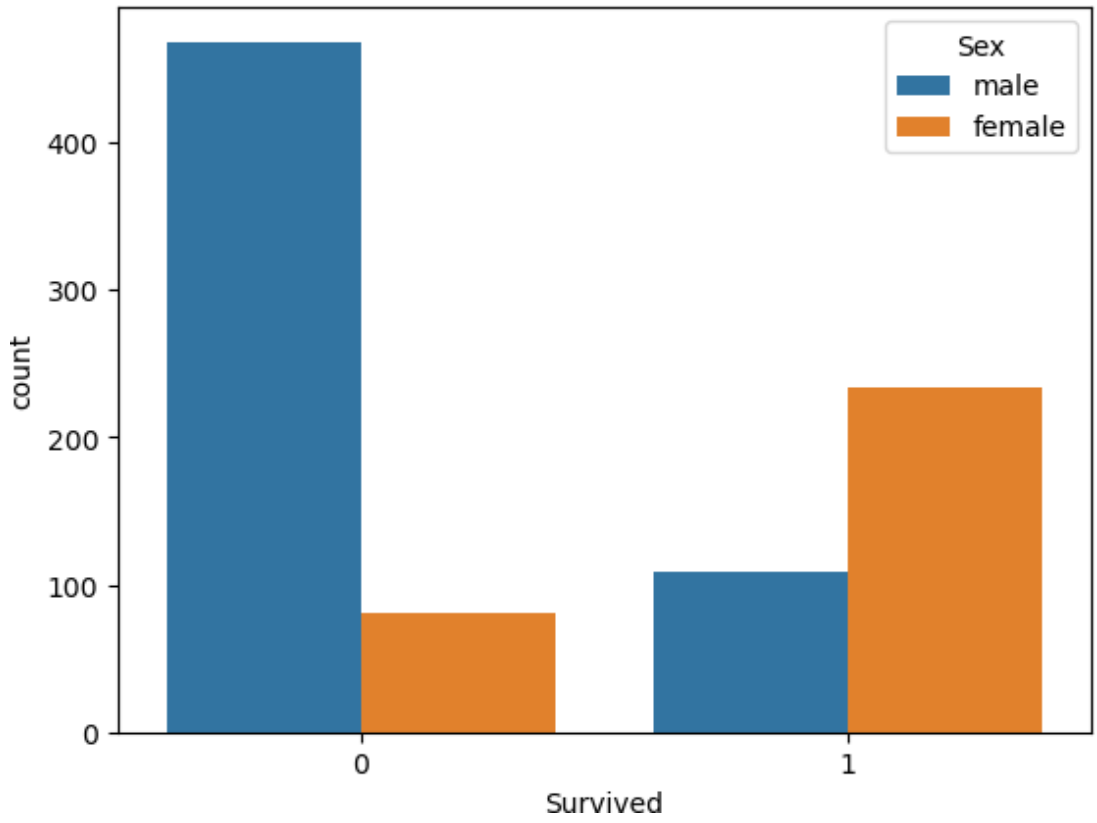
b. A plot that finds the relation between ‘sex’ and the number of people who survived.

In [5]:

```
sns.countplot(x="Survived",  hue="Sex",  data=titanic_data)
```

Out[5]:

```
<AxesSubplot:xlabel='Survived',  ylabel='count'>
```



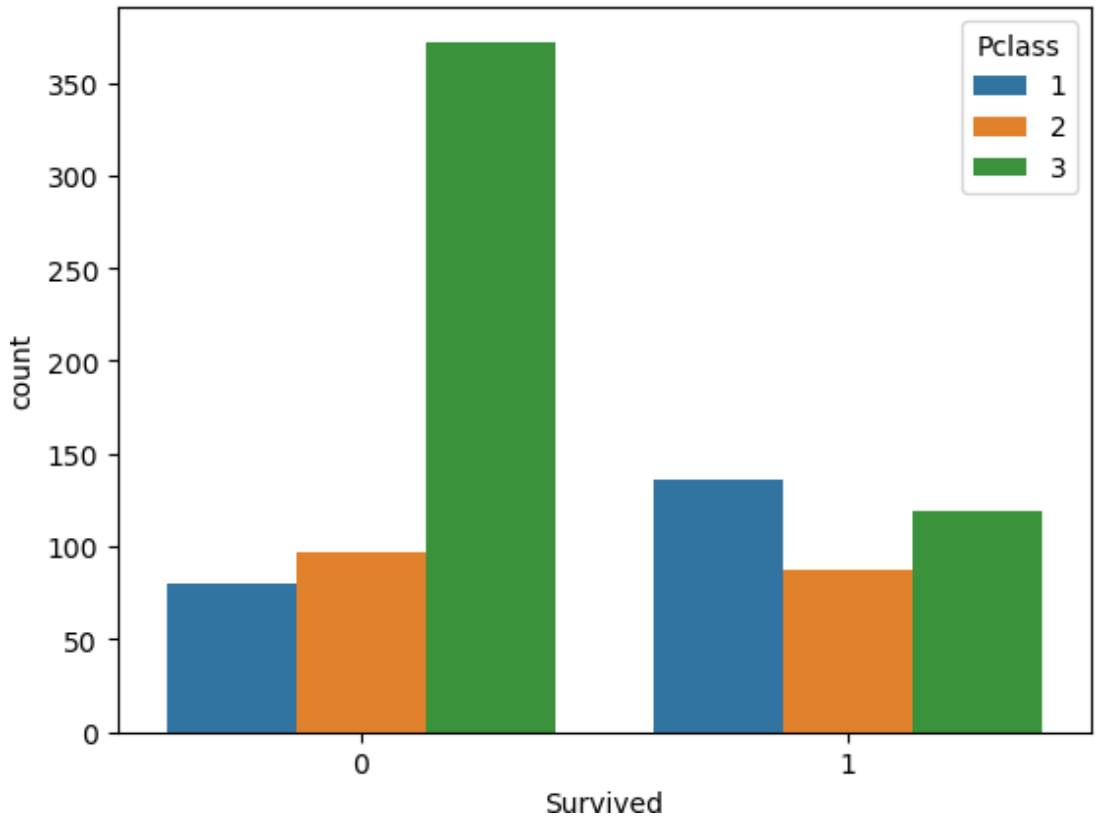
c. Using bar graphs we find the relation between the ‘number of survivors’ and the ‘Pclass’(Passenger Class)

In [6]:

```
sns.countplot(x="Survived",  hue="Pclass",  data=titanic_data)
```

Out[6]:

```
<AxesSubplot:xlabel='Survived',  ylabel='count'>
```



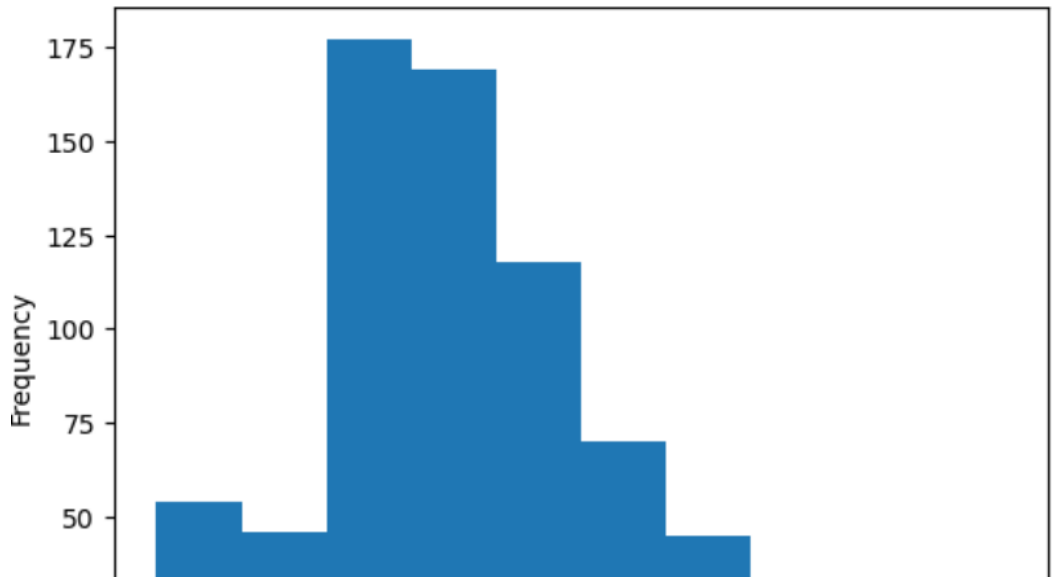
2. Similarly we try to plot various parameters and try to find the relation between the parameters by plotting graphs.

In [7]:

```
titanic_data["Age"].plot.hist()
```

Out[7]:

```
<AxesSubplot:ylabel='Frequency'>
```

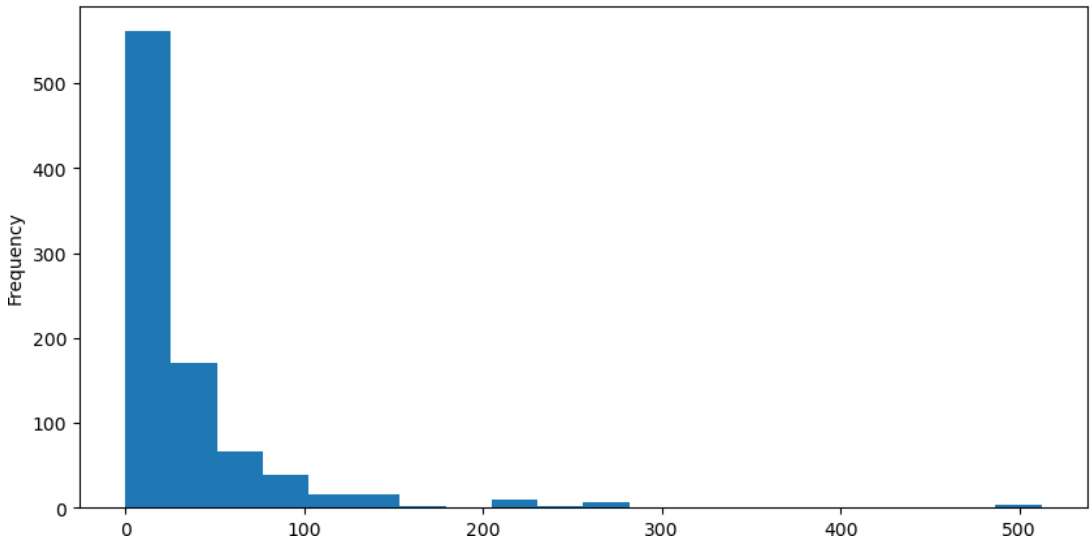


In [8]:

```
titanic_data["Fare"].plot.hist(bins=20,  figsize=(10,5))
```

Out[8]:

```
<AxesSubplot:ylabel='Frequency'>
```



In [9]:

```
titanic_data.info()
```

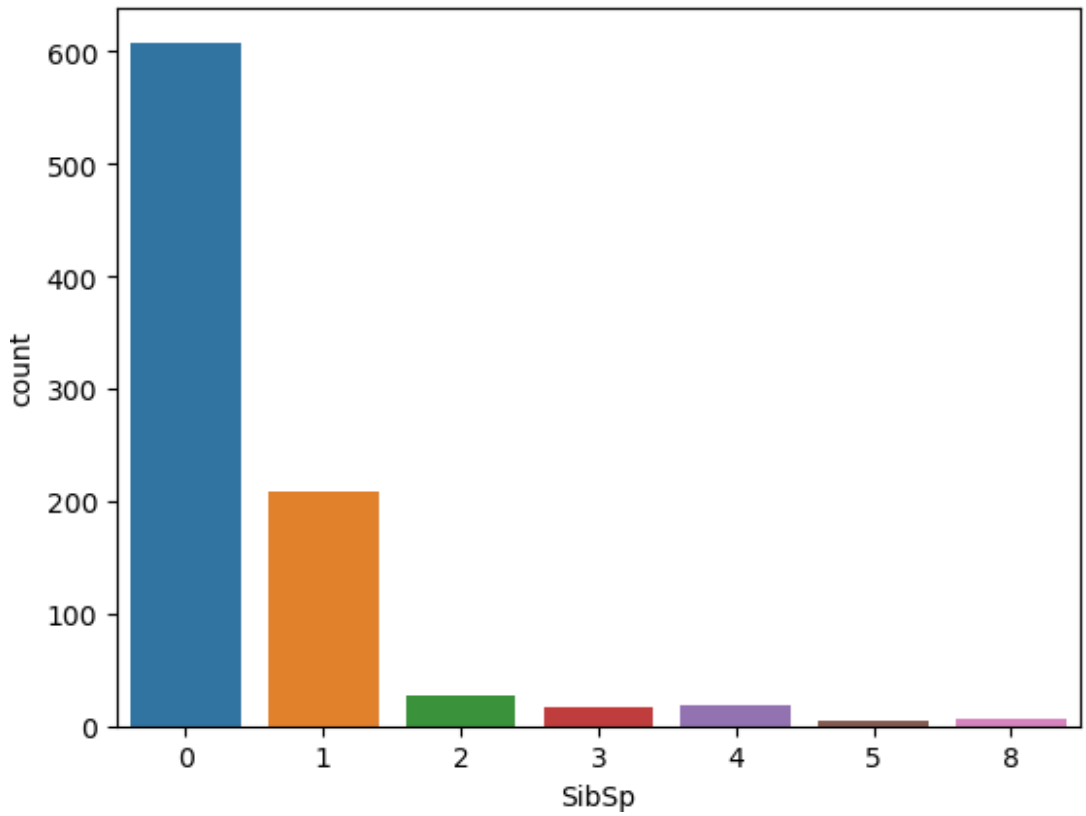
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

In [10]:

```
sns.countplot(x="SibSp",  data=titanic_data)
```

Out[10]:

```
<AxesSubplot:xlabel='SibSp',  ylabel='count'>
```



In [11]:

```
titanic_data.isnull()
```

Out[11]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Far
0	False	False	False	False	False	False	False	False	False	Fals
1	False	False	False	False	False	False	False	False	False	Fals
2	False	False	False	False	False	False	False	False	False	Fals
3	False	False	False	False	False	False	False	False	False	Fals
4	False	False	False	False	False	False	False	False	False	Fals
...	...	...	...	...	...	...	...	...	...	.
886	False	False	False	False	False	False	False	False	False	Fals
887	False	False	False	False	False	False	False	False	False	Fals
888	False	False	False	False	False	True	False	False	False	Fals
889	False	False	False	False	False	False	False	False	False	Fals
890	False	False	False	False	False	False	False	False	False	Fals

891 rows × 12 columns

3. Now we find the number of missing values in the columns by generating the heatmap of all the missing values.

In [12]:

```
titanic_data.isnull().sum()
```

Out[12]:

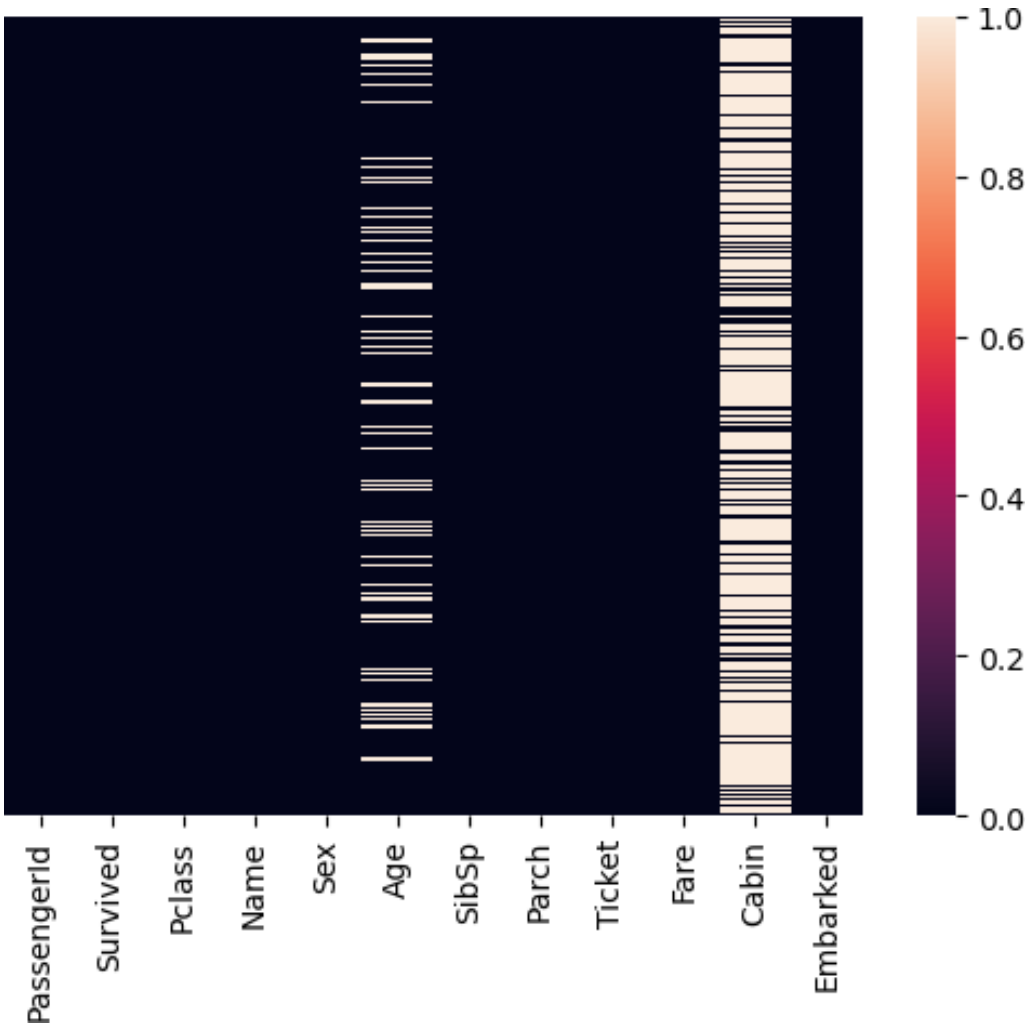
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

In [13]:

```
sns.heatmap(titanic_data.isnull(),   yticklabels=False)
```

Out[13]:

<AxesSubplot:>



4. Now we drop all the null values so that we do not get interference in the test results.

In [15]:

```
titanic_data.dropna(inplace=True)
```

5. We change the data to binary values so as to fit the sigmoid function and logistic regression.

In [16]:

```
pd.get_dummies(titanic_data['Sex'])
```

Out[16]:

	female	male
1	1	0
3	1	0
6	0	1
10	1	0
11	1	0
...	...	...
871	1	0
872	0	1
879	1	0
887	1	0
889	0	1

183 rows × 2 columns

6. Once we finish the preprocessing of data we can divide the data so we can predict the value of Y dataframe(Number of People who Survived).

In [22]:

```
X=titanic_data.drop("Survived",axis=1)
Y=titanic_data["Survived"]
```

Train Data:

Variation

There are many variations of finding the parameters to fit the model(using trial and error). The one that we have used here aims to fit the model with the data of the gender of the passengers in titanic to predict 'Who survived the sinking'. One other variation could be fitting the model with Pclass and "Number of spouses/Children".But according to the accuracy achieved the model doesn't perform well as compared to the accuracy when we consider the case of gender vs survivors.

1. Once we finish with the preprocessing of the data we proceed to train and test the model using the following commands. This model is based on logistic regression so we take the appropriate steps and we have a train test split of 80 :20.

In [23]:

```
from sklearn.model_selection import train_test_split
```

In [26]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_s
X.columns = X.columns.astype(str)
```

In [27]:

```
from sklearn.linear_model import LogisticRegression
```

In [28]:

```
model=LogisticRegression()
```

In [ ]:

```
model.fit(X_train, y_train)
```

In [ ]:

```
predictions = model.predict(X_test)
```

In [ ]:

```
from sklearn.metrics import accuracy_score
```

Once we have trained the model we proceed to check the accuracy of the model and we find that this model where we have taken the parameters of gender ,Spouse Children etc has the hishest accuracy among all the tested models with an accuracy of 76%.

In [ ]:

```
accuracy = accuracy_score(y_test, predictions)
print(f'Accuracy: {accuracy:.2f}')
```

```
Accuracy: 0.76
```

NETAJI SUBHAS UNIVERSITY OF TECHNOLOGY

MACHINE LEARNING

Lab Assignment 2

(CACSC17)

SUBMITTED BY: -
Name: *HARSH KUMAR*
Branch: CSAI
Roll no.:2021UCA1829

1. Reading Data From Wheather_data file

In [1]:

```
import pandas as pd
data=pd.read_csv("F:/DATA/Wheather_data_Bias_correction_ucl.csv")
print(data.columns)
data.head()
```

```
Index(['station', 'Date', 'Present_Tmax', 'Present_Tmin', 'LDAPS_RH
min',
       'LDAPS_RHmax', 'LDAPS_Tmax_lapse', 'LDAPS_Tmin_lapse', 'LDAP
S_WS',
       'LDAPS_LH', 'LDAPS_CC1', 'LDAPS_CC2', 'LDAPS_CC3', 'LDAPS_CC
4',
       'LDAPS_PPT1', 'LDAPS_PPT2', 'LDAPS_PPT3', 'LDAPS_PPT4', 'la
t', 'lon',
       'DEM', 'Slope', 'Solar radiation', 'Next_Tmax', 'Next_Tmi
n'],
      dtype='object')
```

Out[1]:

	station	Date	Present_Tmax	Present_Tmin	LDAPS_RHmin	LDAPS_RHmax	LDA
0	1.0	2013-06-30	28.7	21.4	58.255688	91.116364	
1	2.0	2013-06-30	31.9	21.6	52.263397	90.604721	
2	3.0	2013-06-30	31.6	23.3	48.690479	83.973587	
3	4.0	2013-06-30	32.0	23.4	58.239788	96.483688	
4	5.0	2013-06-30	31.4	21.9	56.174095	90.155128	

5 rows × 25 columns

In [2]:

```
X=data[["Present_Tmax","Present_Tmin","LDAPS_RHmin","LDAPS_RHmax","LDAPS_Tmax_lap
X.head()
```

Out[2]:

	Present_Tmax	Present_Tmin	LDAPS_RHmin	LDAPS_RHmax	LDAPS_Tmax_lapse
0	28.7	21.4	58.255688	91.116364	28.074101
1	31.9	21.6	52.263397	90.604721	29.850689
2	31.6	23.3	48.690479	83.973587	30.091292
3	32.0	23.4	58.239788	96.483688	29.704629
4	31.4	21.9	56.174095	90.155128	29.113934

In [3]:

```
Y=data["Next_Tmax"]
Y.head()
```

Out[3]:

```
0    29.1
1    30.5
2    31.1
3    31.7
4    31.2
Name: Next_Tmax, dtype: float64
```

1. Data Preprocessing

a. Checking Presence of missing values

In [4]:

```
P = X[X.isnull().any(axis=1)]
P
```

Out[4]:

	Present_Tmax	Present_Tmin	LDAPS_RHmin	LDAPS_RHmax	LDAPS_Tmax_la
225	NaN	NaN	70.051193	99.668961	27.872
271	NaN	NaN	72.196007	95.168205	28.097
300	NaN	NaN	95.027298	99.209839	24.078
450	NaN	NaN	60.891193	94.747780	29.195
464	NaN	NaN	52.795406	83.902847	31.480
...	...	...	...	...	
7579	NaN	NaN	38.403931	94.790405	29.929
7596	NaN	NaN	33.681381	91.842178	30.826
7605	NaN	NaN	24.100304	76.861076	28.999
7629	NaN	NaN	43.755058	83.340240	25.842
7707	NaN	NaN	44.392651	75.728195	22.223

145 rows × 12 columns

In [5]:

```
X=X.fillna(X.median())
X.isnull().sum()
```

Out[5]:

```
Present_Tmax        0
Present_Tmin        0
LDAPS_RHmin         0
LDAPS_RHmax         0
LDAPS_Tmax_lapse    0
LDAPS_Tmin_lapse    0
LDAPS_WS            0
LDAPS_LH            0
LDAPS_CC1           0
DEM                 0
Slope               0
Solar radiation     0
dtype: int64
```

b. Spliting the dataset into train and test

In [6]:

```
from sklearn.model_selection import train_test_split as tts
x_train,x_test,y_train,y_test=tts(X,Y,test_size=0.3,   random_state=0)
```

2. Training Linear Regression Model on training dataset.

a. Trainging the model

In [7]:

```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
```

In [8]:

```
import numpy as np
has_nan=np.isnan(y_train).any()
has_nan
```

Out[8]:

```
True
```

In [9]:

```
median_value=np.nanmedian(y_train)
y_train_filled=np.nan_to_num(y_train,nan=median_value)
lr.fit(x_train,y_train_filled)
c=lr.intercept_
m=lr.coef_
print(c,m)
```

```
2.8028489814203077 [ 1.17113560e-01  8.32234258e-02  5.61766007e-04
1.29643788e-02
  6.36884650e-01  1.08342397e-01 -1.45959847e-01  8.62170190e-03
 -1.62863501e+00 -4.31245675e-03  1.94049225e-01  1.07762379e-04]
```

In [10]:

```
y_pred_train=lr.predict(x_train)
y_pred_train
```

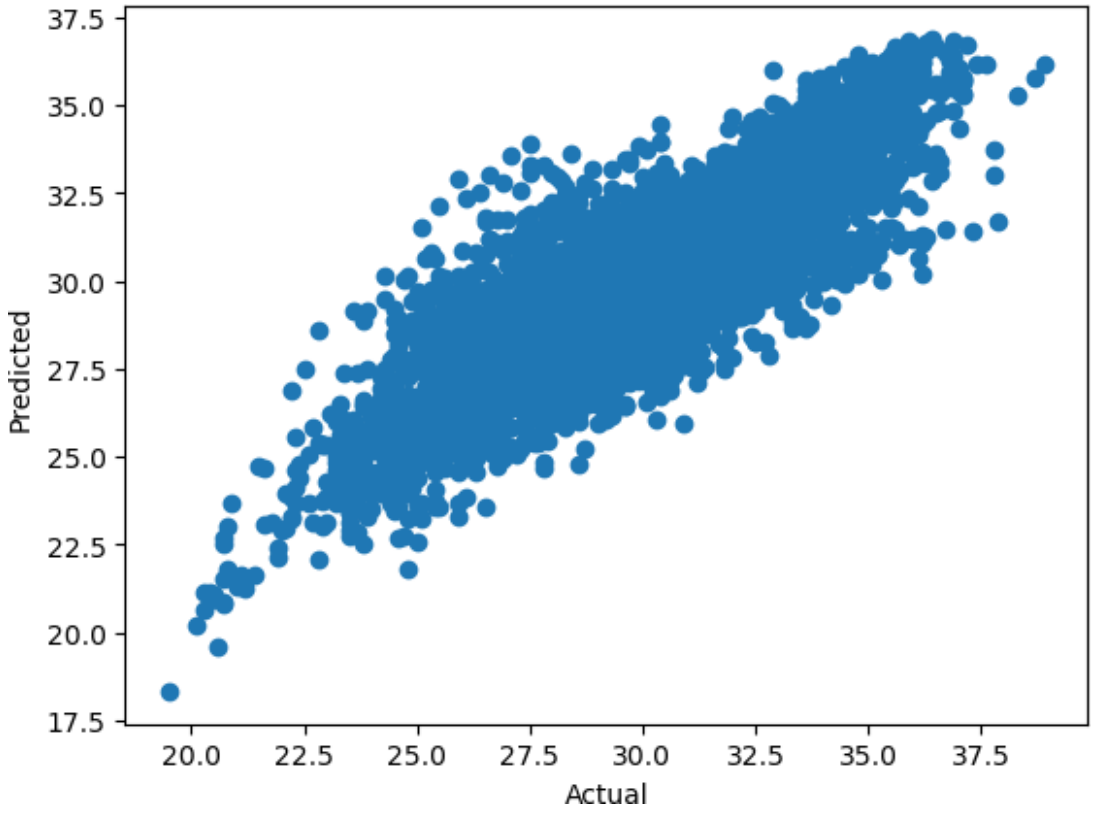
Out[10]:

```
array([33.89882449, 30.71516261, 29.35633252, ..., 31.93138721,
       29.90729827, 28.12681928])
```

b. Ploting the Linear regression of the given data set

In [11]:

```
import matplotlib.pyplot as plt
plt.scatter(y_train_filled,y_pred_train)
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.show()
```



In [12]:

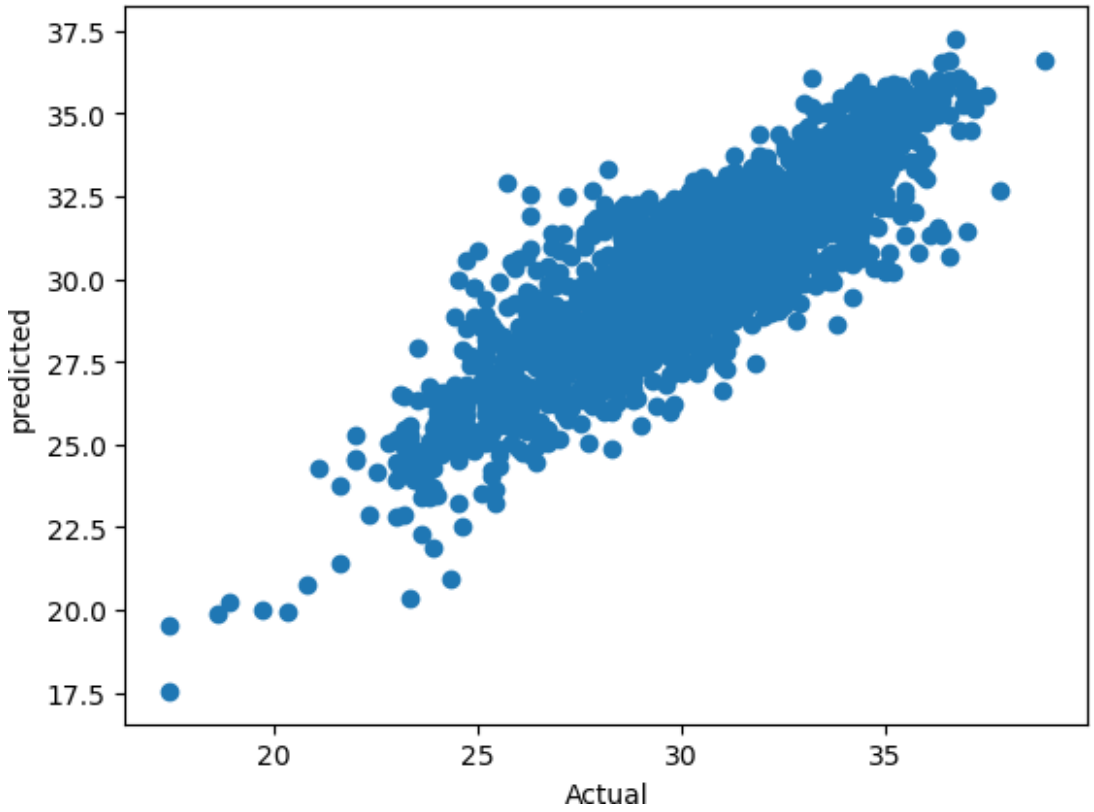
```
from sklearn.metrics import r2_score
r2_score(y_train_filled,  y_pred_train)
```

Out[12]:

```
0.7441053603934771
```

3. Testing the model

```
y_pred_tes=lr.predict(x_test)
plt.scatter(y_test,y_pred_tes)
plt.xlabel("Actual")
plt.ylabel("predicted")
plt.show()
```



In [14]:

```
has_nan=np.isnan(y_test).any()
has_nan
```

Out[14]:

```
True
```

In [15]:

```
median_value=np.nanmedian(y_test)
y_test_filled=np.nan_to_num(y_test,nan=median_value)
r2_score(y_test_filled,y_pred_tes)
```

Out[15]:

```
0.7555675015232703
```

NETAJI SUBHAS UNIVERSITY OF TECHNOLOGY

MACHINE LEARNING

Lab Assignment 1

(CACSC17)

SUBMITTED BY: -
Name: *HARSH KUMAR*
Branch: CSAI
Roll no.:2021UCA1829

Exercise:

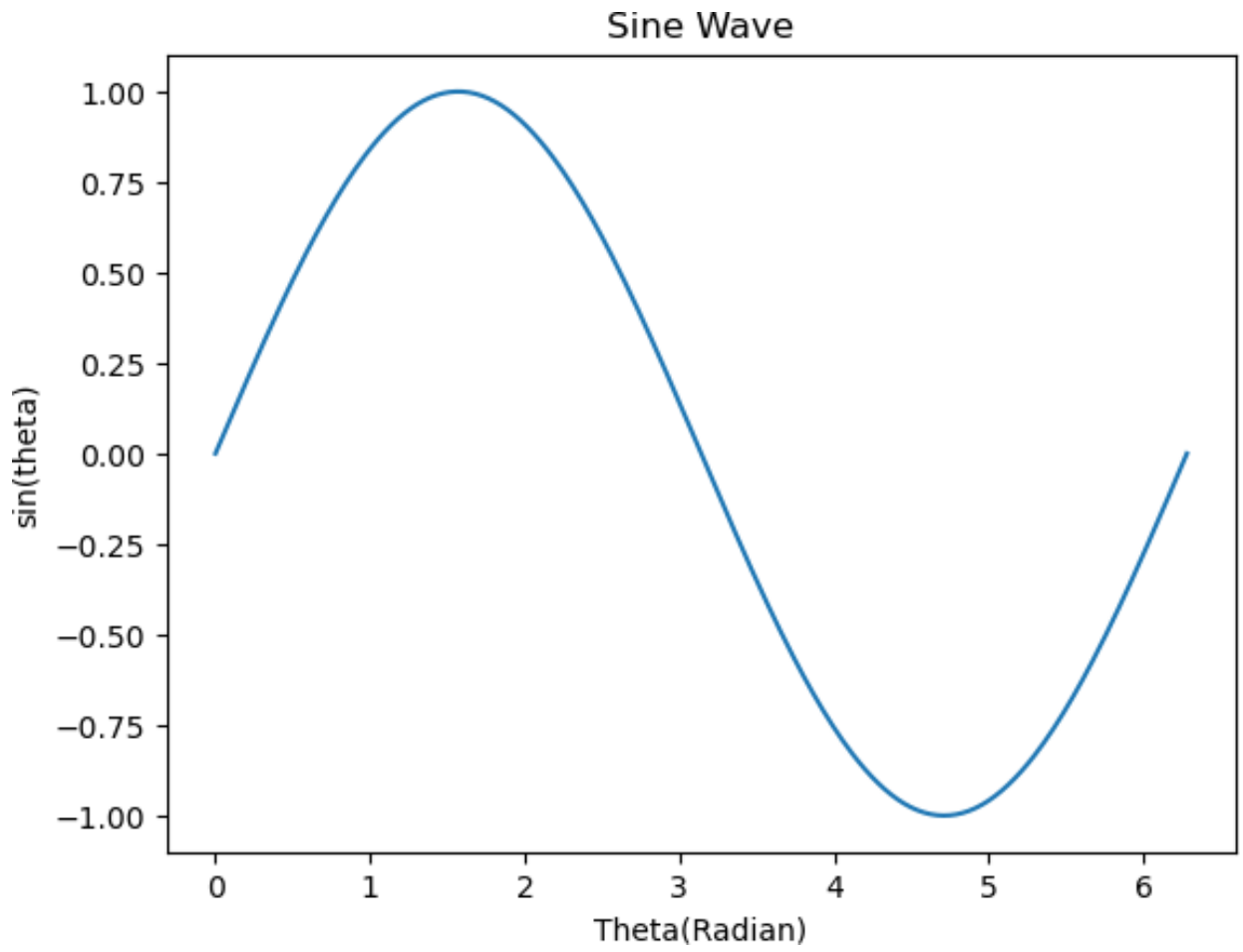
Q1. Slice elements from index 1 to index 5 from the following array: [1,2,3,4,5,6,7]

In [1]:
```
import numpy as np
arr =np.array([1, 2, 3, 4, 5, 6, 7])
print("original Array:",arr)
sliced_array = arr[1:6]
print("original  Array:",sliced_array)
```

```
original Array: [1 2 3 4 5 6 7]
original Array: [2 3 4 5 6]
```

Q2. Draw the sine wave using matplotlib "We shall now display a simple line plot of angle in radians vs. its sine value in Matplotlib."

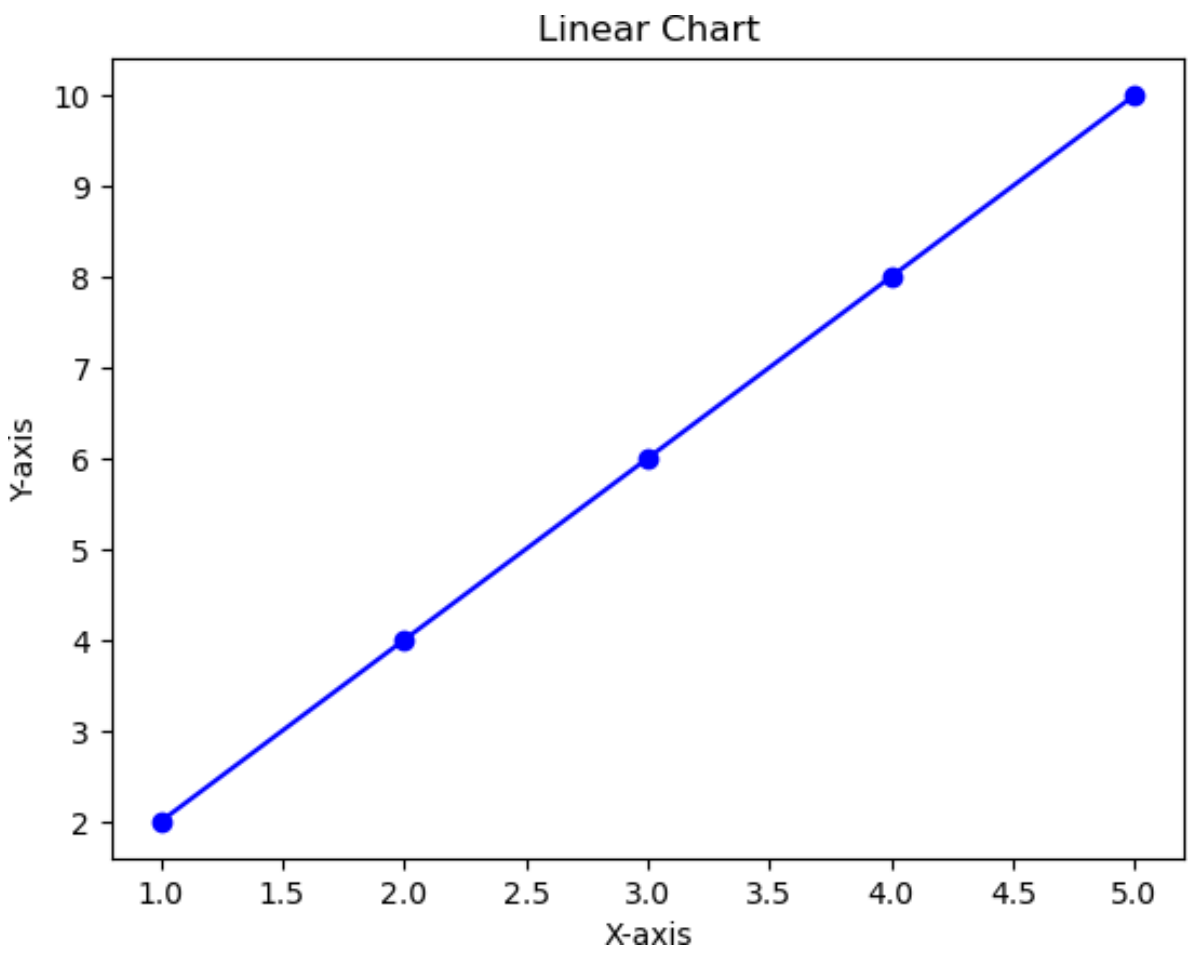
In [3]:
```
import numpy as np
import matplotlib.pyplot as plt
X=np.linspace(0, 2*np.pi, 300)
Y=np.sin(X)
plt.plot(X,Y)
plt.xlabel('Theta(Radian)')
plt.ylabel('sin(theta)')
plt.title('Sine  Wave')
plt.show()
```



Q3. Write programs to understand the use of Matplotlib for Working with Line Chart,Histogram, Bar Chart, Pie Charts.

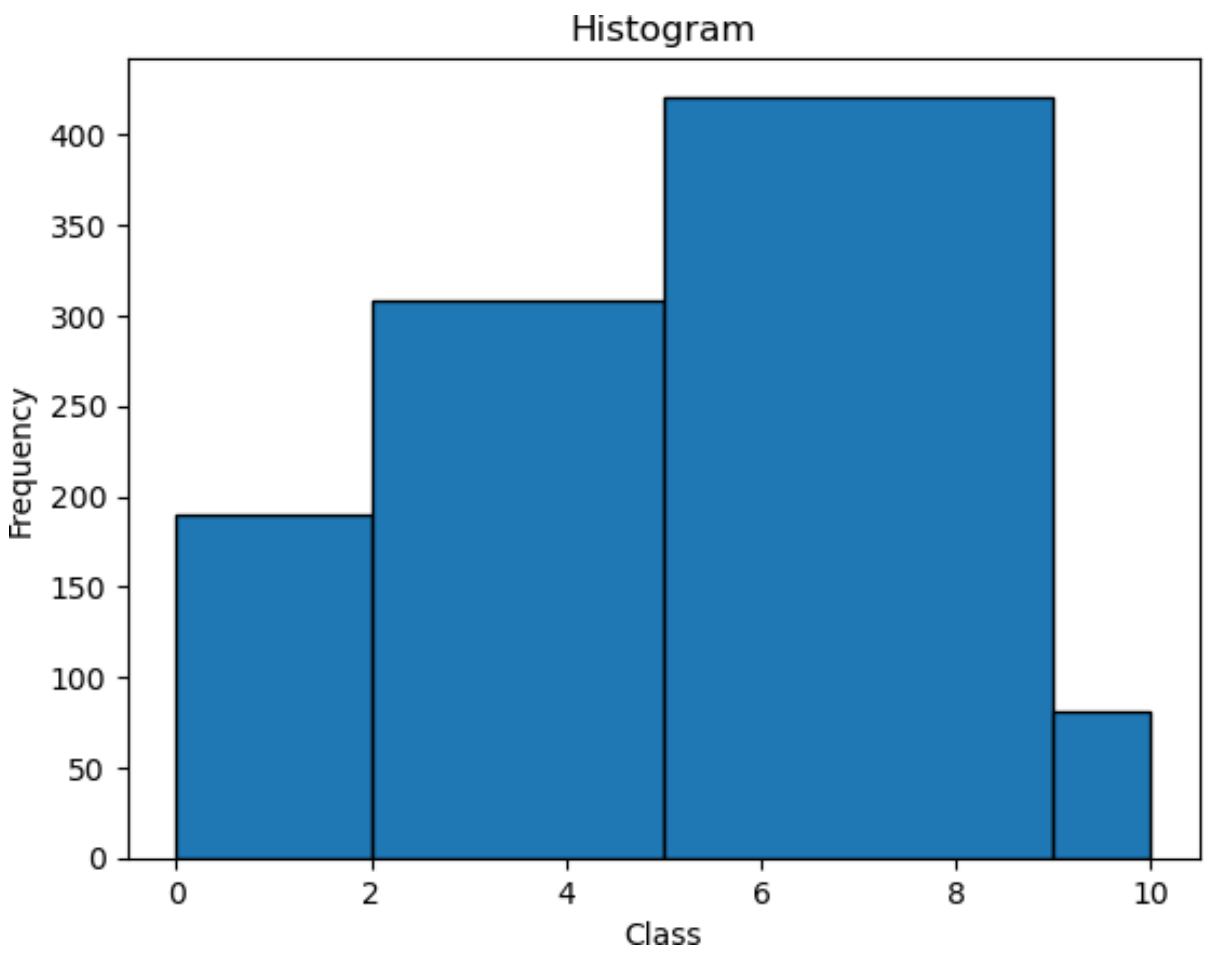
i. Line Chart

In [5]:
```
import matplotlib.pyplot as plt
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]
plt.plot(x, y, marker='o', color='b')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Linear  Chart')
plt.show()
```



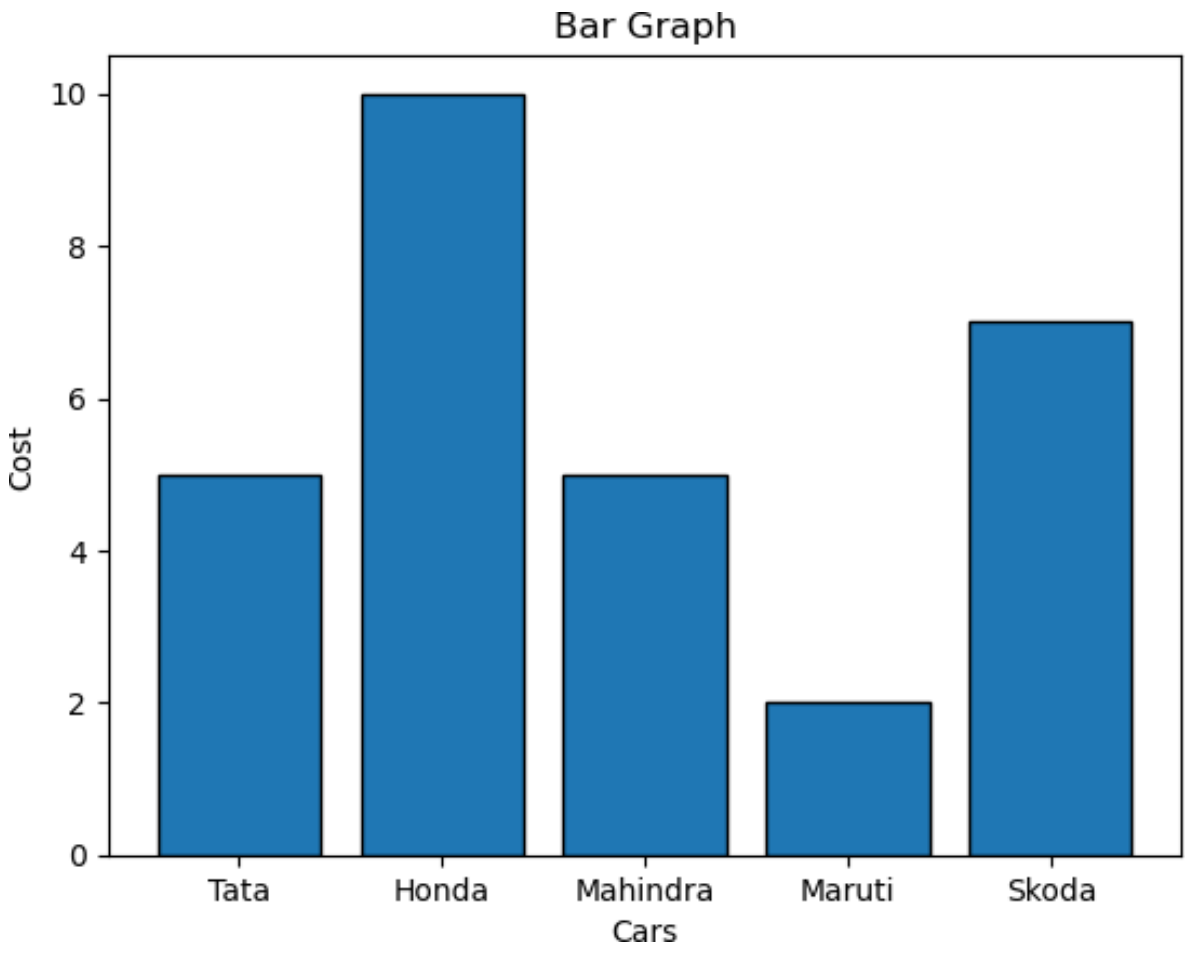
ii. Histogram

In [13]:
```
import numpy as np
import matplotlib.pyplot as plt
data = np.random.randint(0, 10, 1000)
plt.hist(data,[0,2,5,9,10], edgecolor='black')
plt.xlabel('Class')
plt.ylabel('Frequency')
plt.title('Histogram')
plt.show()
```



iii. Bar Chart

In [23]:
```
import numpy as np
import matplotlib.pyplot as plt
items=np.array(["Tata","Honda","Mahindra","Maruti","Skoda"])
value=np.array([5,10,5,2,7])
plt.bar(items, value,0.8,edgecolor="black")
plt.xlabel('Cars')
plt.ylabel('Cost')
plt.title('Bar  Graph')
plt.show()
```



iv. Pie Charts

In [44]:
```
import numpy as np
import matplotlib.pyplot as plt
items=np.array(["Tata","Honda","Mahindra","Maruti","Skoda"])
colors=np.array(["red","blue","green","yellow","pink"])
explode=np.array([0.2,0.2,0.2,0.2,0.2])
value=np.array([5,10,5,2,7])
plt.pie(value,  explode=explode,  labels=items,  colors=colors,shadow='right')
plt.title('Pie Chart')
plt.show()
```