In [1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount,
call drive.mount("/content/drive", force_remount=True).
```

1.DATA COLLECTION AND PREPARATION :-

IMPORTING THE REQUIRED LIBRARIES :-

In [2]:
```
import numpy as np
import pandas as pd
import os
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import svm
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
from sklearn.model_selection import cross_val_score
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import joblib
from sklearn.metrics import accuracy_score
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from tensorflow.keras import layers
import warnings
warnings.filterwarnings('ignore')
from sklearn.metrics import classification_report,confusion_matrix,RocCur
import pickle
```

READ THE DATASET :-

In [3]:
```
df = pd.read_csv(r'/content/drive/MyDrive/Colab Notebooks/collegePlace.cs
```

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

Out[4]:

	Age	Gender	Stream	Internships	CGPA	Hostel	HistoryOfBacklogs	PlacedOrNot
0	22	Male	Electronics And Communication	1	8	1	1	1
1	21	Female	Computer Science	0	7	1	1	1
2	22	Female	Information Technology	1	6	0	0	1
3	21	Male	Information Technology	0	8	0	1	1
4	22	Male	Mechanical	0	8	1	0	1

NUMERICAL ANALYSIS :-

In [5]:
```
df.describe()
```

Out[5]:

	Age	Internships	CGPA	Hostel	HistoryOfBacklogs	PlacedOrNot
count	2966.000000	2966.000000	2966.000000	2966.000000	2966.000000	2966.000000
mean	21.485840	0.703641	7.073837	0.269049	0.192178	0.552596
std	1.324933	0.740197	0.967748	0.443540	0.394079	0.497310
min	19.000000	0.000000	5.000000	0.000000	0.000000	0.000000
25%	21.000000	0.000000	6.000000	0.000000	0.000000	0.000000
50%	21.000000	1.000000	7.000000	0.000000	0.000000	1.000000
75%	22.000000	1.000000	8.000000	1.000000	0.000000	1.000000
max	30.000000	3.000000	9.000000	1.000000	1.000000	1.000000

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2966 entries, 0 to 2965
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   Age                2966 non-null   int64 
 1   Gender             2966 non-null   object
 2   Stream             2966 non-null   object
 3   Internships        2966 non-null   int64 
 4   CGPA               2966 non-null   int64 
 5   Hostel             2966 non-null   int64 
 6   HistoryOfBacklogs  2966 non-null   int64 
 7   PlacedOrNot        2966 non-null   int64 
dtypes: int64(6), object(2)
memory usage: 185.5+ KB
```

No missing values ...

HANDLING CATEGORICAL VALUES :-

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2966 entries, 0 to 2965
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   Age                2966 non-null   int64 
 1   Gender             2966 non-null   object
 2   Stream             2966 non-null   object
 3   Internships        2966 non-null   int64 
 4   CGPA               2966 non-null   int64 
 5   Hostel             2966 non-null   int64 
 6   HistoryOfBacklogs  2966 non-null   int64 
 7   PlacedOrNot        2966 non-null   int64 
dtypes: int64(6), object(2)
memory usage: 185.5+ KB
```

In [8]:
```python
df.drop('Hostel',axis=1,inplace=True)
```

In [9]:
```python
df.head()
```

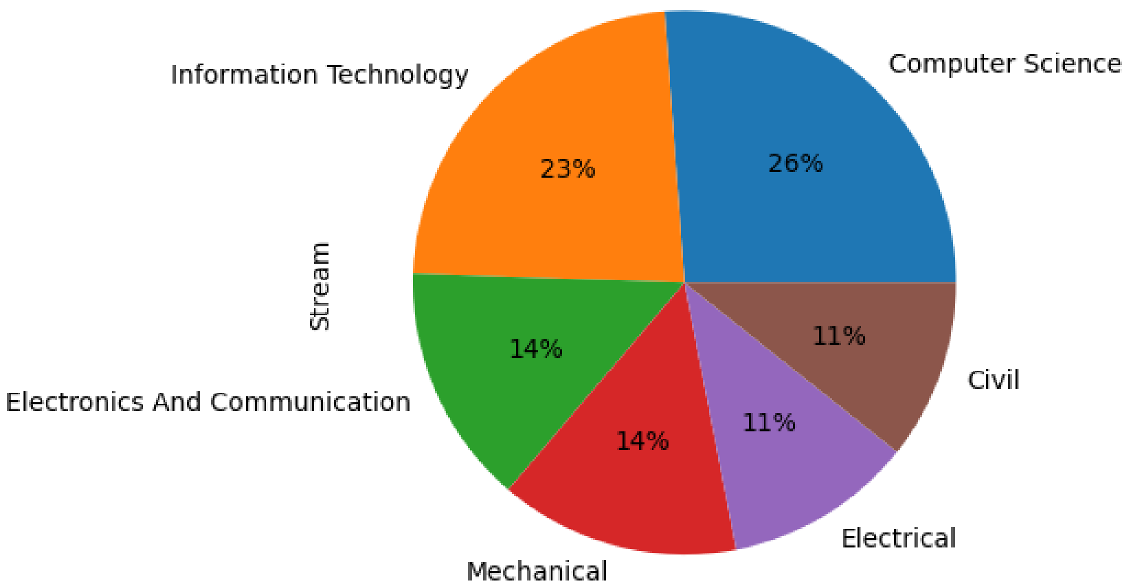
Out[9]:

	Age	Gender	Stream	Internships	CGPA	HistoryOfBacklogs	PlacedOrNot
0	22	Male	Electronics And Communication	1	8	1	1
1	21	Female	Computer Science	0	7	1	1
2	22	Female	Information Technology	1	6	0	1
3	21	Male	Information Technology	0	8	1	1
4	22	Male	Mechanical	0	8	0	1

EDA : Exploratory Data Analysis :-

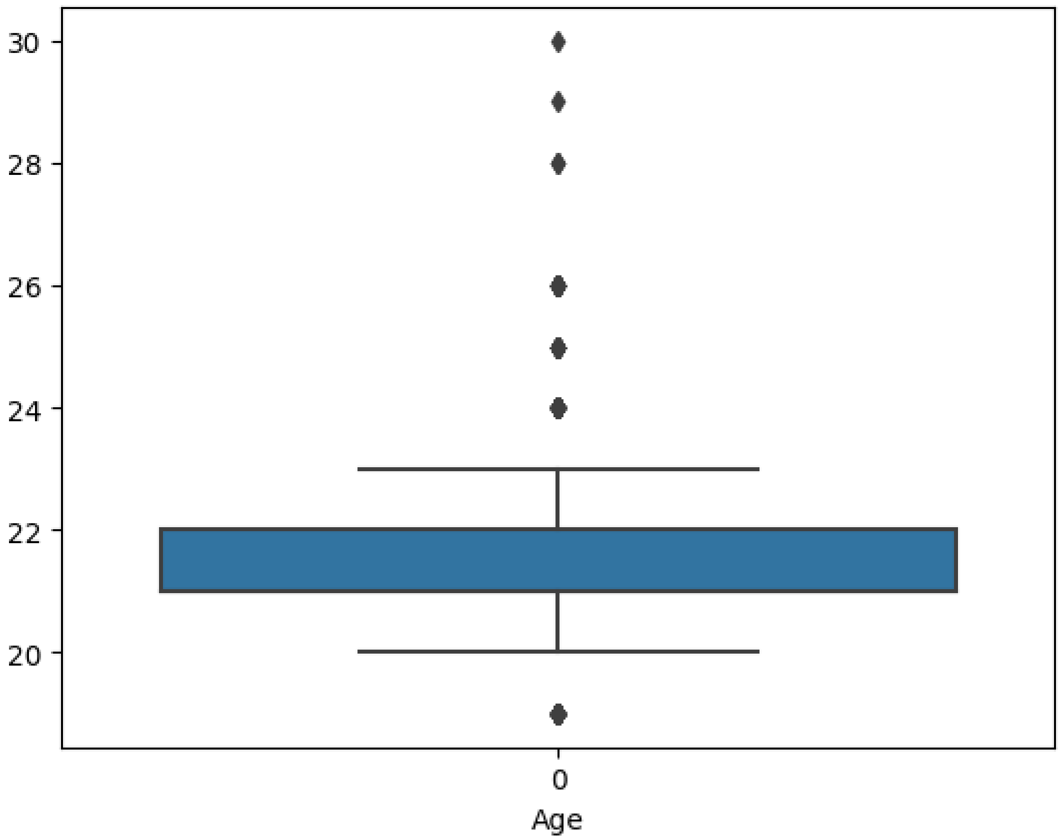
UNIVARIATE ANALYSIS :-

In [10]:
```python
df['Stream'].value_counts().plot(kind='pie',autopct='%1.0f%%')
plt.show()
```



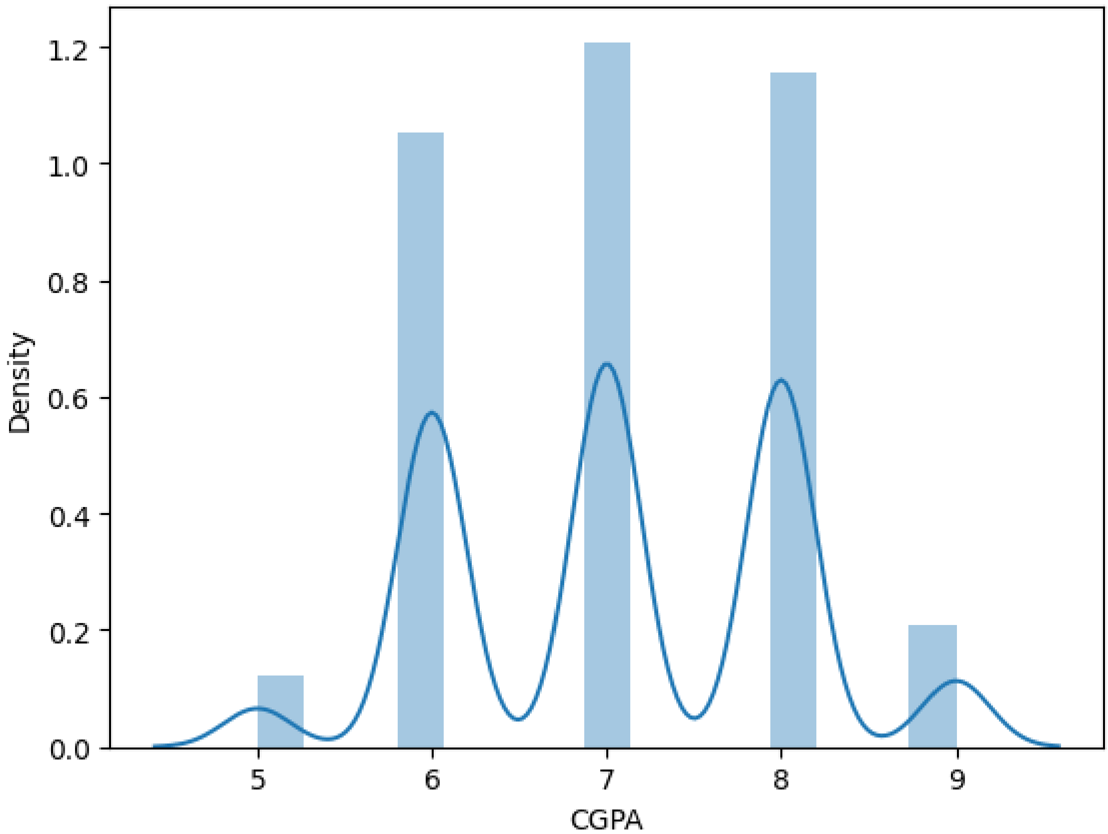
In [11]:
```python
plt.xlabel('Age')
sns.boxplot(df['Age'])
```

Out[11]: <Axes: xlabel='Age'>



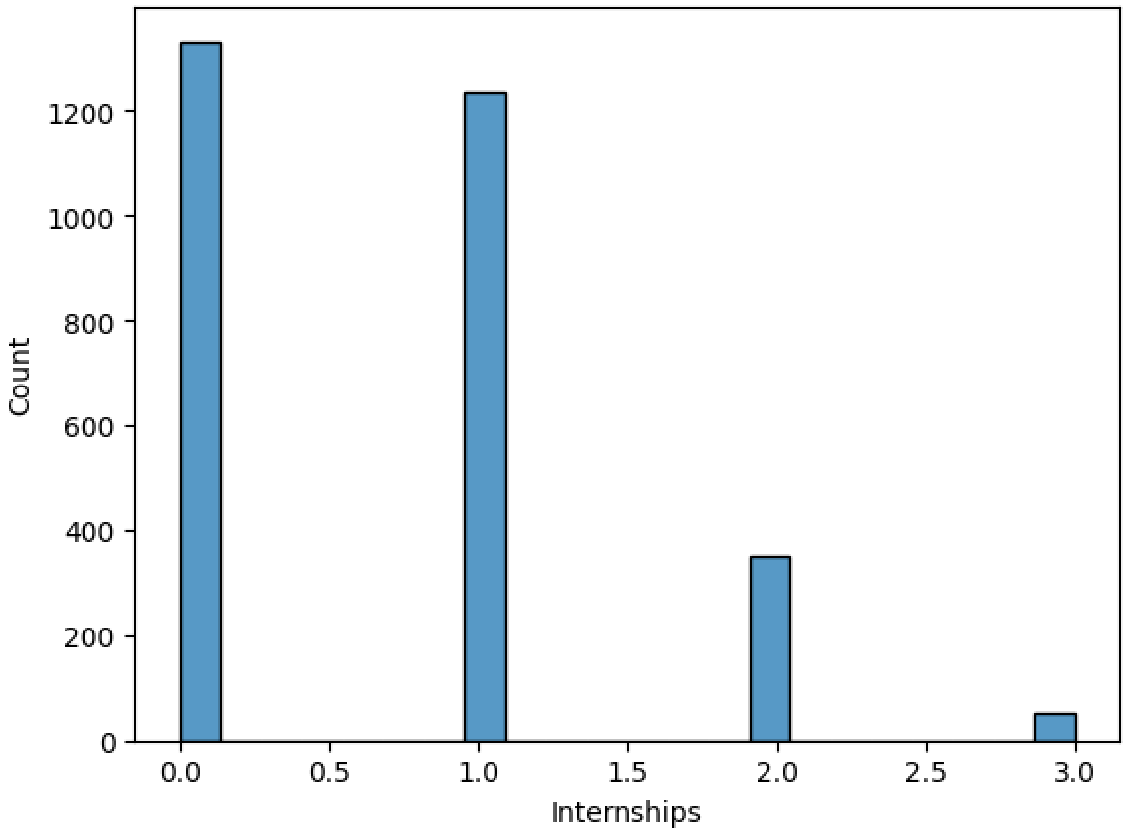
In [12]:
```
sns.distplot(df['CGPA'])
```

Out[12]: <Axes: xlabel='CGPA', ylabel='Density'>



In [13]:
```
sns.histplot(df,x='Internships')
```

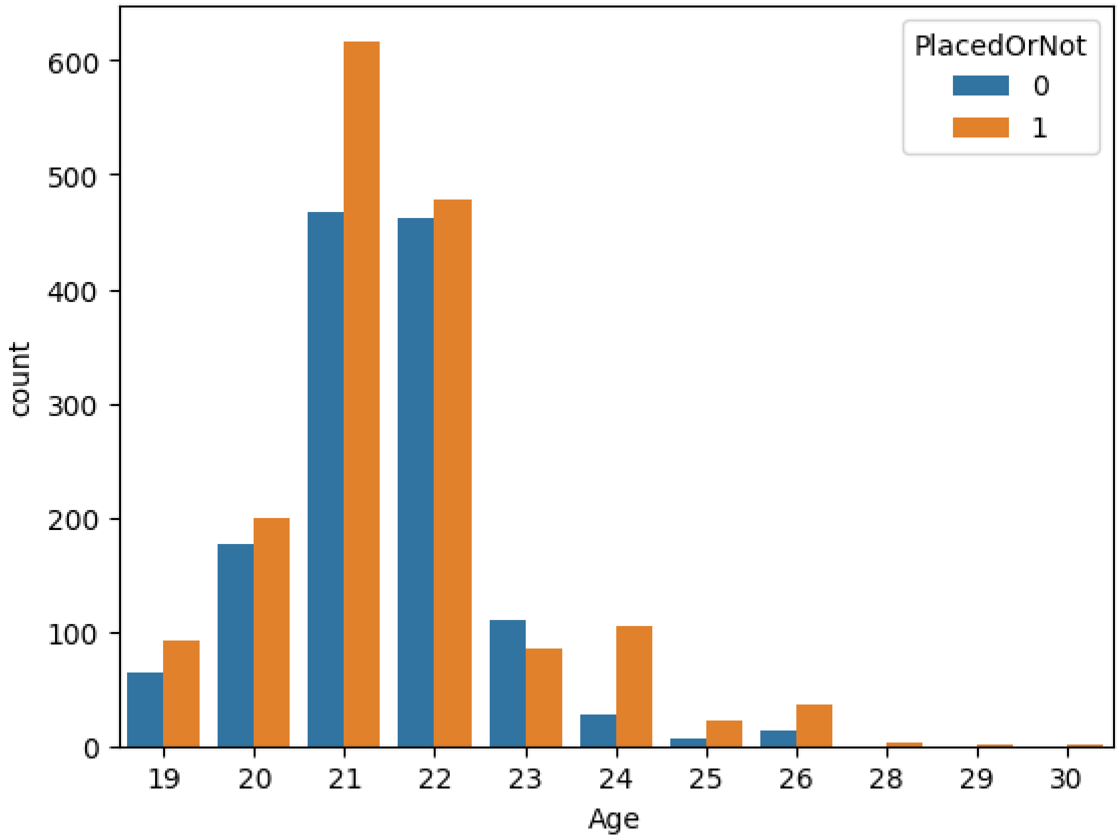
Out[13]: <Axes: xlabel='Internships', ylabel='Count'>



BIVARIATE ANALYSIS :-

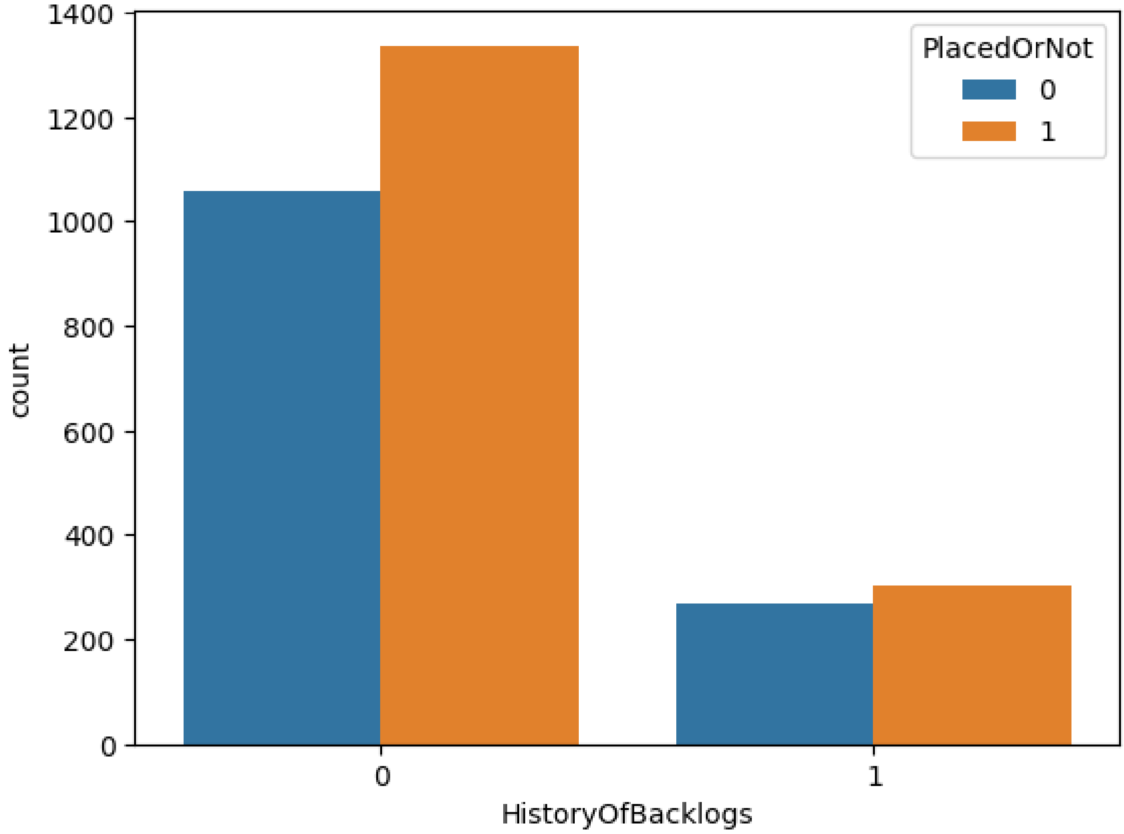
In [14]:
```
sns.countplot(df,x='Age',hue='PlacedOrNot')
```

Out[14]: <Axes: xlabel='Age', ylabel='count'>



In [15]:
```
sns.countplot(df,x='HistoryOfBacklogs',hue='PlacedOrNot')
```

Out[15]: <Axes: xlabel='HistoryOfBacklogs', ylabel='count'>

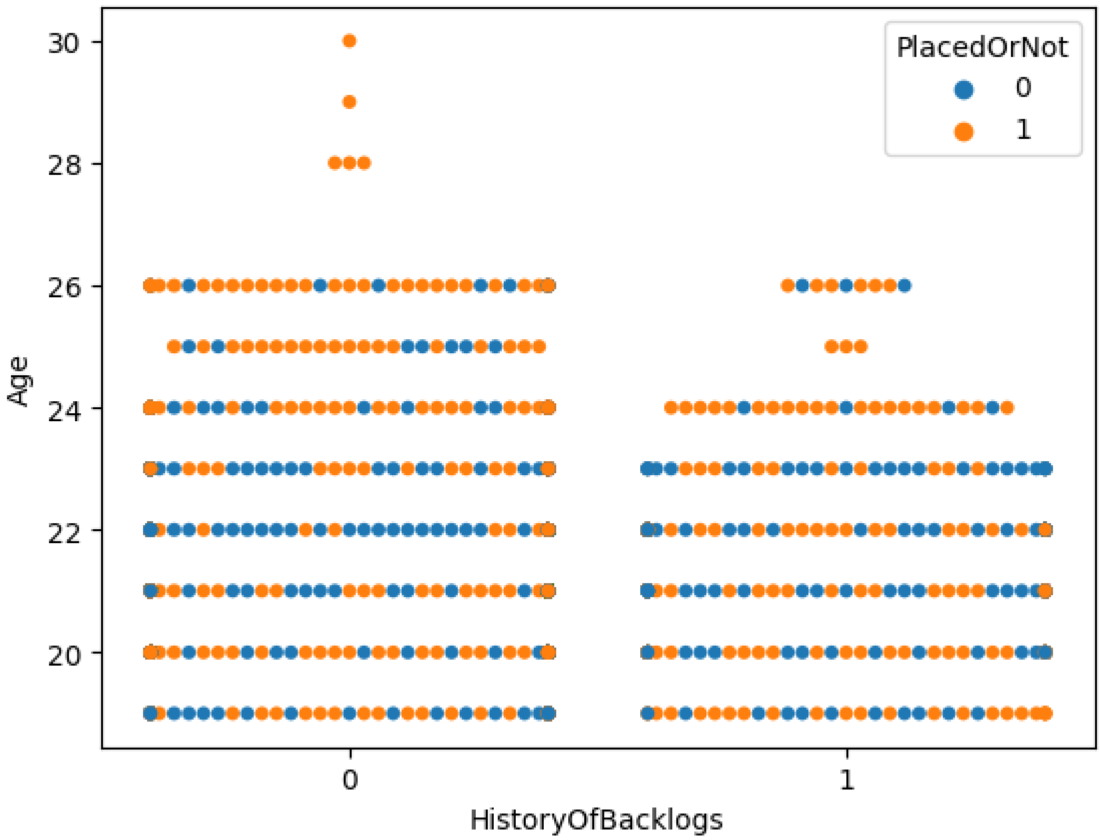


In [15]:

MULTIVARIATE ANALYSIS :-

In [16]:
```
sns.swarmplot(df,x='HistoryOfBacklogs',y='Age',hue='PlacedOrNot')
```

Out[16]: <Axes: xlabel='HistoryOfBacklogs', ylabel='Age'>



In [17]:
```
df['Gender'].value_counts()
```

Out[17]:
```
Male      2475
Female     491
Name: Gender, dtype: int64
```

In [18]:
```
df.Gender = df.Gender.replace({'Male':1,'Female':0})
```

In [19]:
```
df.Stream.value_counts()
```

Out[19]:
```
Computer Science                 776
Information Technology           691
Electronics And Communication    424
Mechanical                       424
Electrical                       334
Civil                            317
Name: Stream, dtype: int64
```

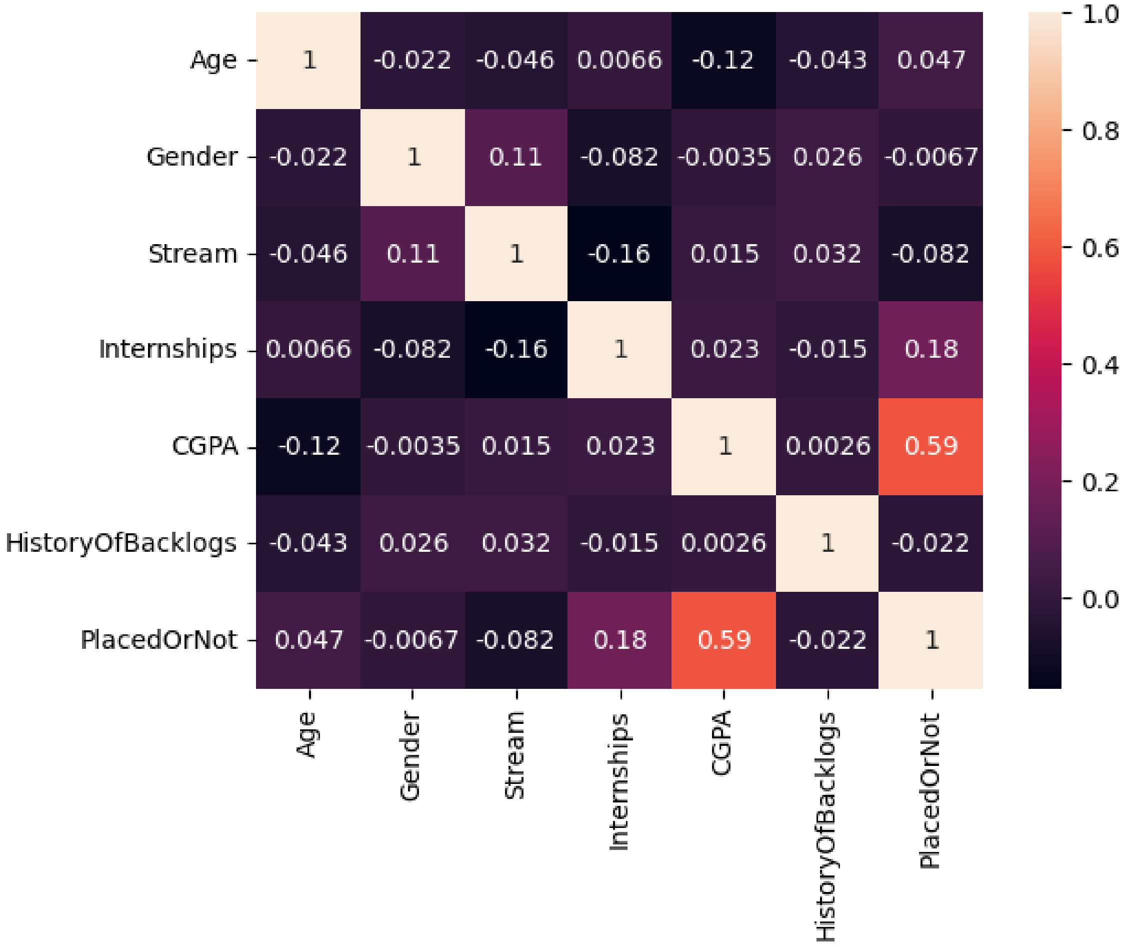
In [20]:
```
df['Stream'].replace({'Computer Science':0,'Information Technology':1,'El
```

In [21]:
```
df['CGPA'].value_counts()
```

Out[21]:
```
7    956
8    915
6    834
9    165
5     96
Name: CGPA, dtype: int64
```

In [22]:
```
sns.heatmap(df.corr(),annot=True)
```

Out[22]: <Axes: >



SEPARATING DEPENDENT AND INDEPENDENT VARIABLES :-

In [23]:
```
x = df.drop(columns ='PlacedOrNot', axis=1)
y = df['PlacedOrNot']
```

SCALING THE DATA:-

In [24]:
```
scaler = StandardScaler()
x = pd.DataFrame(scaler.fit_transform(x),columns=scaler.get_feature_names
print(x)
```

```
             Age    Gender    Stream  Internships      CGPA  HistoryOfBack
logs
0     0.388131  0.445403  0.040082     0.400445  0.957191             2.05
0246
1    -0.366752 -2.245158 -1.148743    -0.950773 -0.076310             2.05
0246
2     0.388131 -2.245158 -0.554331     0.400445 -1.109812            -0.48
7746
3    -0.366752  0.445403 -0.554331    -0.950773  0.957191             2.05
0246
4     0.388131  0.445403  0.634494    -0.950773  0.957191            -0.48
7746
...        ...       ...       ...          ...       ...
...
2961  1.143013  0.445403 -0.554331    -0.950773 -0.076310            -0.48
7746
2962  1.143013  0.445403  0.634494     0.400445 -0.076310            -0.48
7746
2963  0.388131  0.445403 -0.554331     0.400445 -0.076310            -0.48
7746
2964  0.388131  0.445403 -1.148743     0.400445 -0.076310            -0.48
7746
2965  1.143013  0.445403  1.823319    -0.950773  0.957191            -0.48
7746

[2966 rows x 6 columns]
```

In [25]:
```
pickle.dump(scaler,open('scaler.pkl','wb'))
```

SPLITTING THE DATA INTO TRAIN AND TEST

In [26]:
```
x_train, x_test , y_train, y_test = train_test_split(x,y, test_size=0.2,
print(x.shape, x_train.shape, x_test.shape )
```

```
(2966, 6) (2372, 6) (594, 6)
```

MODEL BUILDING :-

SVM MODEL :-

In [27]:
```
svc = svm.SVC(kernel ='linear')
svc.fit(x_train,y_train)

y_pred = svc.predict(x_train)
train_acc = accuracy_score(y_pred,y_train)
y_pred = svc.predict(x_test)
test_acc = accuracy_score(y_pred,y_test)
print('training data accuracy : ',train_acc)
print('testing data accuracy : ',test_acc)
```

```
training data accuracy :  0.7685497470489039
testing data accuracy :  0.7794612794612794
```

KNN MODEL :-

In [28]:
```
knn = KNeighborsClassifier()
knn.fit(x_train,y_train)

y_pred = knn.predict(x_train)
```

```
train_acc = accuracy_score(y_pred,y_train)
y_pred = knn.predict(x_test)
test_acc = accuracy_score(y_pred,y_test)
print('training data accuracy : ',train_acc)
print('testing data accuracy : ',test_acc)
```

```
training data accuracy :  0.8929173693086003
testing data accuracy :  0.8619528619528619
```

ARTIFICIAL NEURAL NETWORK :-

In [29]:
```
ann = Sequential()
ann.add(keras.layers.Dense(6,activation = 'selu'))
ann.add(keras.layers.Dense(24,activation = 'selu'))
ann.add(keras.layers.Dense(12,activation = 'selu'))

ann.add(keras.layers.Dense(1,activation='sigmoid'))
```

In [30]:
```
# compiting the model
loss_1 = tf.keras.losses.BinaryCrossentropy()
ann.compile(optimizer='Adam',loss=loss_1,metrics=['accuracy'])

# fitting the model-
ann.fit(x_train,y_train,batch_size =20 ,epochs=100)
```

```
Epoch 1/100
119/119 [==============================] - 1s 2ms/step - loss: 0.5866 -
accuracy: 0.6838
Epoch 2/100
119/119 [==============================] - 0s 2ms/step - loss: 0.4540 -
accuracy: 0.7745
Epoch 3/100
119/119 [==============================] - 0s 2ms/step - loss: 0.4310 -
accuracy: 0.7774
Epoch 4/100
119/119 [==============================] - 0s 2ms/step - loss: 0.4141 -
accuracy: 0.7846
Epoch 5/100
119/119 [==============================] - 0s 2ms/step - loss: 0.3880 -
accuracy: 0.8019
Epoch 6/100
119/119 [==============================] - 0s 2ms/step - loss: 0.3680 -
accuracy: 0.8280
Epoch 7/100
119/119 [==============================] - 0s 2ms/step - loss: 0.3534 -
accuracy: 0.8402
Epoch 8/100
119/119 [==============================] - 0s 2ms/step - loss: 0.3453 -
accuracy: 0.8364
Epoch 9/100
119/119 [==============================] - 0s 2ms/step - loss: 0.3395 -
accuracy: 0.8478
Epoch 10/100
119/119 [==============================] - 0s 2ms/step - loss: 0.3335 -
accuracy: 0.8491
Epoch 11/100
119/119 [==============================] - 0s 2ms/step - loss: 0.3314 -
accuracy: 0.8491
Epoch 12/100
119/119 [==============================] - 0s 2ms/step - loss: 0.3274 -
accuracy: 0.8516
Epoch 13/100
119/119 [==============================] - 0s 2ms/step - loss: 0.3243 -
accuracy: 0.8516
Epoch 14/100
119/119 [==============================] - 0s 2ms/step - loss: 0.3218 -
accuracy: 0.8550
Epoch 15/100
119/119 [==============================] - 0s 2ms/step - loss: 0.3188 -
accuracy: 0.8537
Epoch 16/100
119/119 [==============================] - 0s 2ms/step - loss: 0.3167 -
accuracy: 0.8571
Epoch 17/100
119/119 [==============================] - 0s 2ms/step - loss: 0.3142 -
accuracy: 0.8617
Epoch 18/100
119/119 [==============================] - 0s 2ms/step - loss: 0.3106 -
accuracy: 0.8596
Epoch 19/100
119/119 [==============================] - 0s 2ms/step - loss: 0.3101 -
accuracy: 0.8588
Epoch 20/100
119/119 [==============================] - 0s 3ms/step - loss: 0.3058 -
accuracy: 0.8647
```

```
Epoch 21/100
119/119 [==============================] - 0s 3ms/step - loss: 0.3047 -
accuracy: 0.8655
Epoch 22/100
119/119 [==============================] - 0s 3ms/step - loss: 0.3040 -
accuracy: 0.8630
Epoch 23/100
119/119 [==============================] - 0s 3ms/step - loss: 0.3027 -
accuracy: 0.8676
Epoch 24/100
119/119 [==============================] - 0s 3ms/step - loss: 0.3012 -
accuracy: 0.8651
Epoch 25/100
119/119 [==============================] - 0s 3ms/step - loss: 0.2984 -
accuracy: 0.8739
Epoch 26/100
119/119 [==============================] - 0s 3ms/step - loss: 0.2955 -
accuracy: 0.8702
Epoch 27/100
119/119 [==============================] - 0s 3ms/step - loss: 0.2973 -
accuracy: 0.8693
Epoch 28/100
119/119 [==============================] - 0s 3ms/step - loss: 0.2968 -
accuracy: 0.8756
Epoch 29/100
119/119 [==============================] - 0s 3ms/step - loss: 0.2942 -
accuracy: 0.8718
Epoch 30/100
119/119 [==============================] - 0s 3ms/step - loss: 0.2942 -
accuracy: 0.8723
Epoch 31/100
119/119 [==============================] - 0s 3ms/step - loss: 0.2905 -
accuracy: 0.8723
Epoch 32/100
119/119 [==============================] - 0s 3ms/step - loss: 0.2904 -
accuracy: 0.8714
Epoch 33/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2895 -
accuracy: 0.8739
Epoch 34/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2889 -
accuracy: 0.8731
Epoch 35/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2880 -
accuracy: 0.8744
Epoch 36/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2871 -
accuracy: 0.8790
Epoch 37/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2861 -
accuracy: 0.8714
Epoch 38/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2860 -
accuracy: 0.8744
Epoch 39/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2849 -
accuracy: 0.8723
Epoch 40/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2848 -
accuracy: 0.8748
```

```
Epoch 41/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2825 -
accuracy: 0.8761
Epoch 42/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2829 -
accuracy: 0.8735
Epoch 43/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2814 -
accuracy: 0.8777
Epoch 44/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2816 -
accuracy: 0.8782
Epoch 45/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2817 -
accuracy: 0.8761
Epoch 46/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2805 -
accuracy: 0.8727
Epoch 47/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2783 -
accuracy: 0.8773
Epoch 48/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2783 -
accuracy: 0.8790
Epoch 49/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2794 -
accuracy: 0.8786
Epoch 50/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2790 -
accuracy: 0.8769
Epoch 51/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2766 -
accuracy: 0.8756
Epoch 52/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2783 -
accuracy: 0.8782
Epoch 53/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2768 -
accuracy: 0.8752
Epoch 54/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2772 -
accuracy: 0.8752
Epoch 55/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2774 -
accuracy: 0.8756
Epoch 56/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2749 -
accuracy: 0.8777
Epoch 57/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2737 -
accuracy: 0.8790
Epoch 58/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2767 -
accuracy: 0.8744
Epoch 59/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2753 -
accuracy: 0.8756
Epoch 60/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2736 -
accuracy: 0.8761
```

```
Epoch 61/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2740 -
accuracy: 0.8773
Epoch 62/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2738 -
accuracy: 0.8786
Epoch 63/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2738 -
accuracy: 0.8773
Epoch 64/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2717 -
accuracy: 0.8853
Epoch 65/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2742 -
accuracy: 0.8752
Epoch 66/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2733 -
accuracy: 0.8794
Epoch 67/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2734 -
accuracy: 0.8777
Epoch 68/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2731 -
accuracy: 0.8773
Epoch 69/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2723 -
accuracy: 0.8794
Epoch 70/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2715 -
accuracy: 0.8782
Epoch 71/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2724 -
accuracy: 0.8786
Epoch 72/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2718 -
accuracy: 0.8803
Epoch 73/100
119/119 [==============================] - 0s 3ms/step - loss: 0.2710 -
accuracy: 0.8790
Epoch 74/100
119/119 [==============================] - 0s 3ms/step - loss: 0.2700 -
accuracy: 0.8756
Epoch 75/100
119/119 [==============================] - 0s 3ms/step - loss: 0.2684 -
accuracy: 0.8820
Epoch 76/100
119/119 [==============================] - 0s 3ms/step - loss: 0.2694 -
accuracy: 0.8794
Epoch 77/100
119/119 [==============================] - 0s 3ms/step - loss: 0.2675 -
accuracy: 0.8773
Epoch 78/100
119/119 [==============================] - 0s 3ms/step - loss: 0.2703 -
accuracy: 0.8723
Epoch 79/100
119/119 [==============================] - 0s 3ms/step - loss: 0.2691 -
accuracy: 0.8811
Epoch 80/100
119/119 [==============================] - 0s 3ms/step - loss: 0.2676 -
accuracy: 0.8811
```

```
Epoch 81/100
119/119 [==============================] - 0s 3ms/step - loss: 0.2686 -
accuracy: 0.8794
Epoch 82/100
119/119 [==============================] - 0s 3ms/step - loss: 0.2683 -
accuracy: 0.8773
Epoch 83/100
119/119 [==============================] - 0s 3ms/step - loss: 0.2662 -
accuracy: 0.8815
Epoch 84/100
119/119 [==============================] - 0s 3ms/step - loss: 0.2685 -
accuracy: 0.8803
Epoch 85/100
119/119 [==============================] - 0s 3ms/step - loss: 0.2676 -
accuracy: 0.8811
Epoch 86/100
119/119 [==============================] - 0s 3ms/step - loss: 0.2675 -
accuracy: 0.8811
Epoch 87/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2681 -
accuracy: 0.8777
Epoch 88/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2661 -
accuracy: 0.8820
Epoch 89/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2668 -
accuracy: 0.8777
Epoch 90/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2655 -
accuracy: 0.8773
Epoch 91/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2688 -
accuracy: 0.8777
Epoch 92/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2648 -
accuracy: 0.8820
Epoch 93/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2655 -
accuracy: 0.8836
Epoch 94/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2648 -
accuracy: 0.8790
Epoch 95/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2659 -
accuracy: 0.8811
Epoch 96/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2645 -
accuracy: 0.8849
Epoch 97/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2669 -
accuracy: 0.8798
Epoch 98/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2659 -
accuracy: 0.8777
Epoch 99/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2669 -
accuracy: 0.8794
Epoch 100/100
119/119 [==============================] - 0s 2ms/step - loss: 0.2644 -
accuracy: 0.8845
```

Out[30]: <keras.callbacks.History at 0x7f11bc0d7fd0>

In [31]:
```
#with the model

y_pred = ann.predict(x_train)
y_pred =(y_pred >0.5)
train_acc = accuracy_score(y_pred,y_train)

y_pred = ann.predict(x_test)
y_pred =(y_pred >0.5)
test_acc = accuracy_score(y_pred,y_test)

print('training data accuracy : ',train_acc)
print('testing data accuracy : ',test_acc)
```

```
75/75 [==============================] - 0s 2ms/step
19/19 [==============================] - 0s 2ms/step
training data accuracy :  0.8823777403035413
testing data accuracy :  0.8569023569023569
```

HYPER-PARAMETER TUNING :-

In [32]:
```
from sklearn.model_selection import RandomizedSearchCV
params = {'C': [0.1, 1, 10, 100],
              'kernel': ['rbf'],
              'gamma': [1, 0.1, 0.001],
              }

rscv = RandomizedSearchCV(svm.SVC(), params)
rscv.fit(x_train, y_train)

print("Best hyperparameters: ", rscv.best_params_)
```

```
Best hyperparameters:  {'kernel': 'rbf', 'gamma': 0.1, 'C': 10}
```

In [33]:
```
svc2 = svm.SVC(kernel ='rbf',gamma=0.1,C=100)
svc2.fit(x_train,y_train)

y_pred = svc2.predict(x_train)
train_acc = accuracy_score(y_pred,y_train)
y_pred = svc2.predict(x_test)
test_acc = accuracy_score(y_pred,y_test)
print('training data accuracy : ',train_acc)
print('testing data accuracy : ',test_acc)
```

```
training data accuracy :  0.8920741989881956
testing data accuracy :  0.8686868686868687
```

In [34]:
```
best_k = {'Regular':0}
best_score ={'Regular': 0}
for k in range(3,50,2):
  ##using Regular trainning set
  knn_temp =KNeighborsClassifier(n_neighbors=k)
  knn_temp.fit(x_train,y_train)
  knn_temp_pred=knn_temp.predict(x_test)
  score=metrics.accuracy_score(y_test, knn_temp_pred)*100
  if(score >= best_score['Regular'] and score<100):
    best_score['Regular'] = score
    best_k['Regular']=k
best_k['Regular']
```

Out[34]: 7

In [35]:
```
knn2 = KNeighborsClassifier(n_neighbors=best_k['Regular'])
knn2.fit(x_train,y_train)

y_pred = knn2.predict(x_train)
train_acc = accuracy_score(y_pred,y_train)
y_pred = knn2.predict(x_test)
test_acc = accuracy_score(y_pred,y_test)
print('training data accuracy : ',train_acc)
print('testing data accuracy : ',test_acc)
```

```
training data accuracy :  0.887858347386172
testing data accuracy :  0.8636363636363636
```

EVALUATING THE MODEL :-

In [36]:
```
def cl_res(name,model):
    y_pred = model.predict(x_test)
    if(name=='ANN'):
        y_pred = [0 if x<0.5 else 1 for x in y_pred]
    print(name,' :-\n-------------------------')
    print('accuracy score of ',name,' : ',accuracy_score(y_test,y_pred))
    print(classification_report(y_test,y_pred,target_names=['no delay','d
    print('confusion matrix : \n',confusion_matrix(y_test,y_pred))
    print('\n')
    # plt.subplot(121)
    plt.figure(figsize=(3,2))
    sns.heatmap(confusion_matrix(y_test,y_pred),annot=True)
    # plt.subplot(122)
    plt.figure(figsize=(1,1))
    RocCurveDisplay.from_predictions(y_test,y_pred)
    plt.show()
    print('\n\n')
```

In [37]:
```
models = {'SVM':svc,'SVM2':svc2,'KNN':knn,'knn2':knn2,'ANN':ann}
```

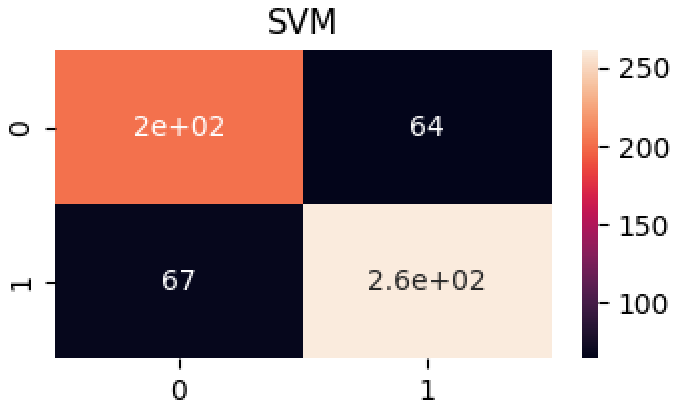
In [38]:
```
def prediction(model,input):
  output = model.predict(input)
  if isinstance(model,Sequential):
    output = output > 0.5
  return output
```

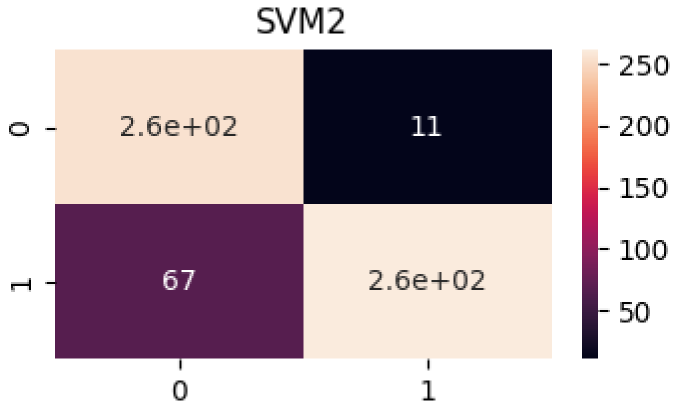
In [39]:
```
for name , model in models.items():
  y_pred = prediction(model,x_test)
  print('accuracy score of ',name,' : ',accuracy_score(y_test,y_pred))
```

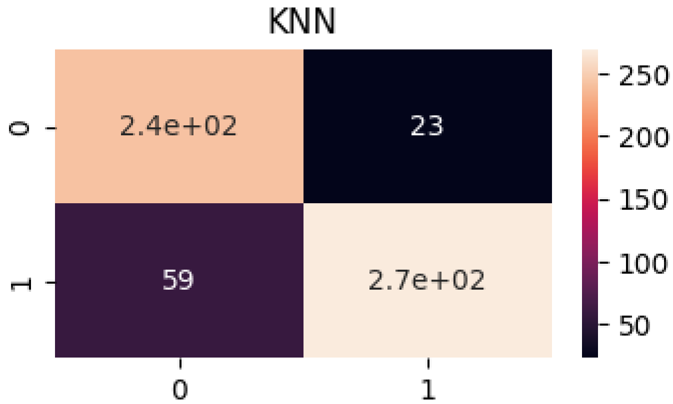
```
accuracy score of  SVM  :  0.7794612794612794
accuracy score of  SVM2  :  0.8686868686868687
accuracy score of  KNN  :  0.8619528619528619
accuracy score of  knn2  :  0.8636363636363636
19/19 [==============================] - 0s 1ms/step
accuracy score of  ANN  :  0.8569023569023569
```

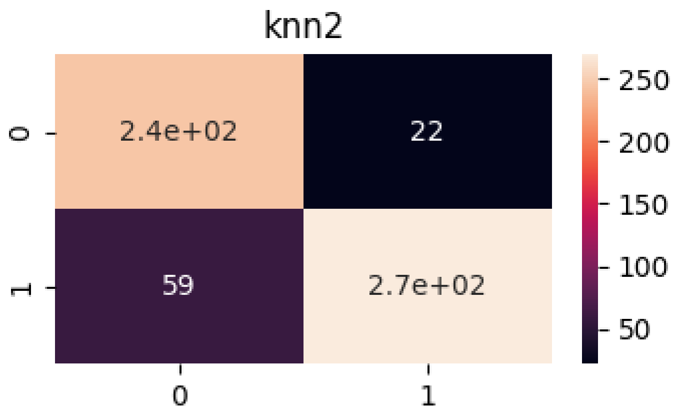
In [40]:
```
count = 1
for name , model in models.items():
  y_pred = prediction(model,x_test)
  plt.figure(figsize=(4,2))
  plt.title(name)
```

```
    count+=1
    sns.heatmap(confusion_matrix(y_test,y_pred),annot=True)
    plt.show()
```

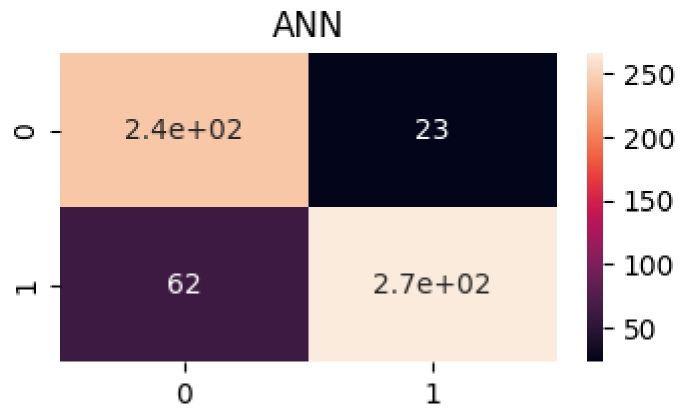






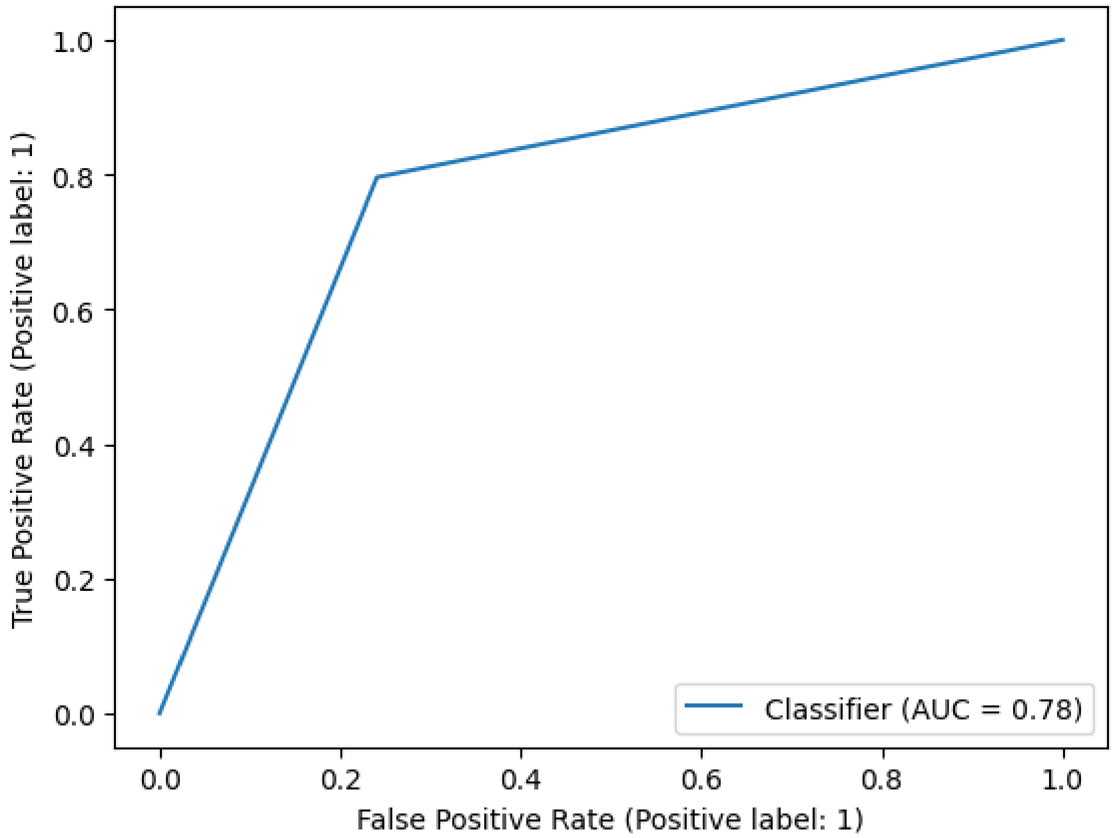



```
19/19 [==============================] - 0s 1ms/step
```

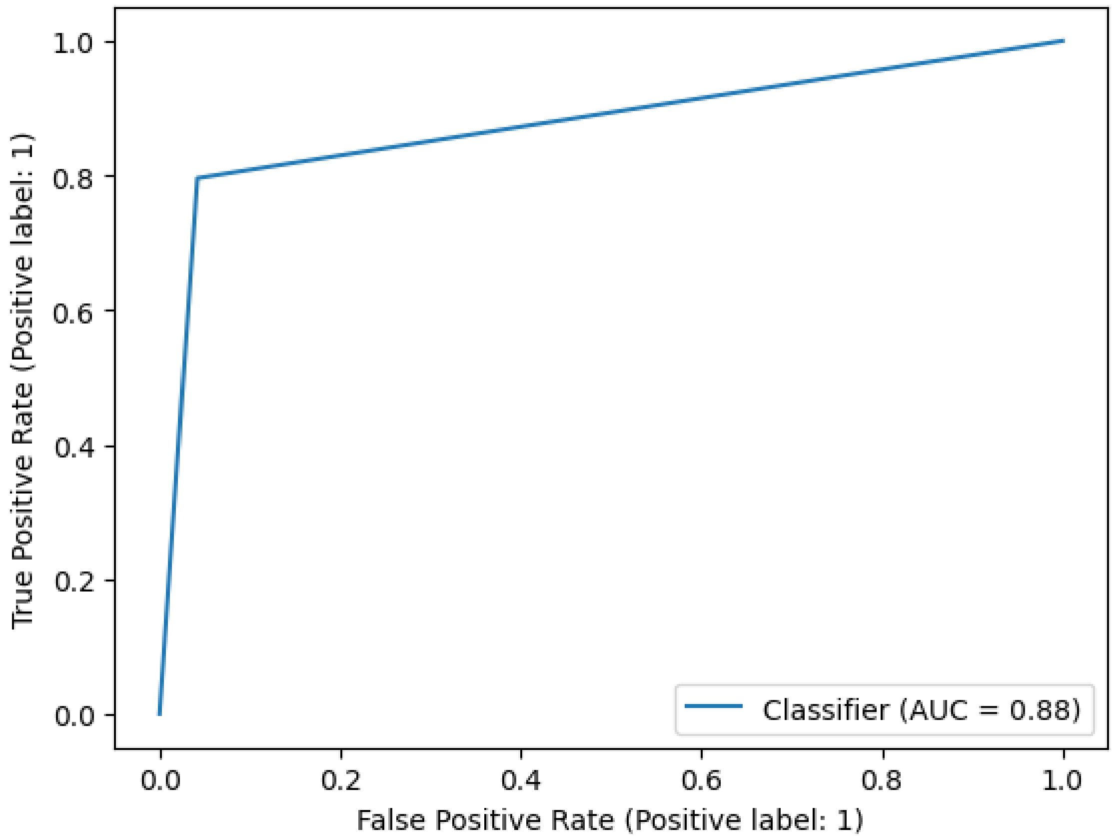


In [41]:
```
for name,model in models.items():
    y_pred = prediction(model,x_test)
    plt.figure(figsize=(2,1))
    print(name)
    RocCurveDisplay.from_predictions(y_test,y_pred)
```

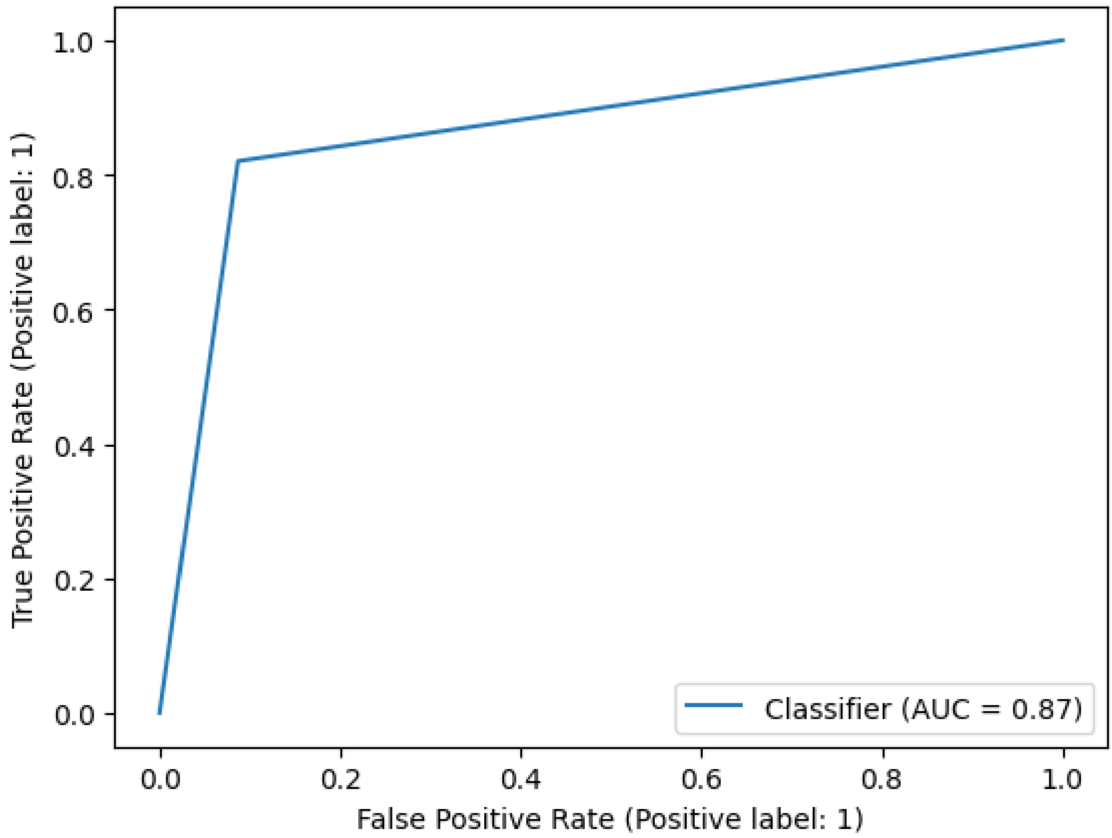
```
SVM
SVM2
KNN
knn2
19/19 [==============================] - 0s 1ms/step
ANN
<Figure size 200x100 with 0 Axes>
```



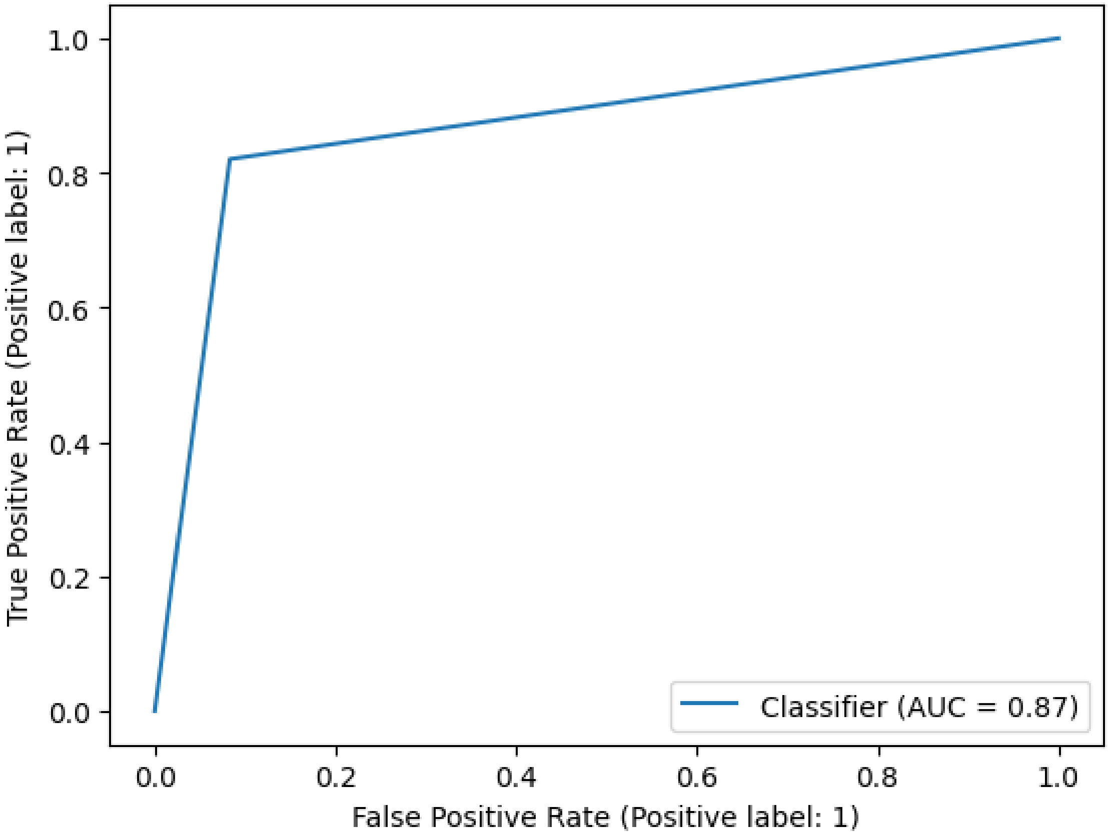

```
<Figure size 200x100 with 0 Axes>
```



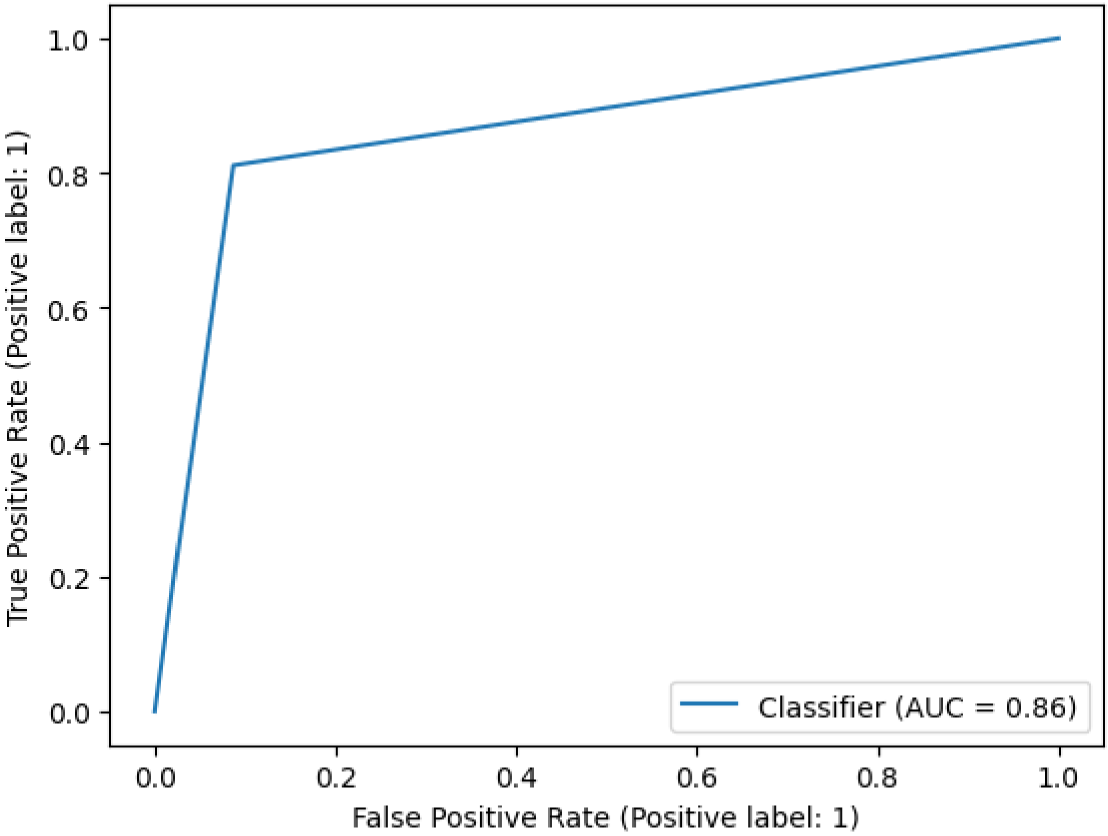
<Figure size 200x100 with 0 Axes>



<Figure size 200x100 with 0 Axes>




```
<Figure size 200x100 with 0 Axes>
```



SAVING THE MODEL :-

In [42]:
```
pickle.dump(svc2,open('svc.pkl','wb'))
```