CAMPUS PLACEMENT Task 2

```
import numpy as np
import pandas as pd
import os
import seaborn as sns
import matplotlib.pyplot as plt
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 warnings
warnings.filterwarnings('ignore')
```

Read the Dastaset

```
df = pd.read_csv(r"/content/collegePlace.csv")
df.head()
```

	Age	Gender	Stream	Internships	CGPA	Hostel	HistoryOfBacklogs	Placec
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	0	

```
df.shape
```

```
(2966, 8)
```

Data preperation

Handling missing values

```
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
```

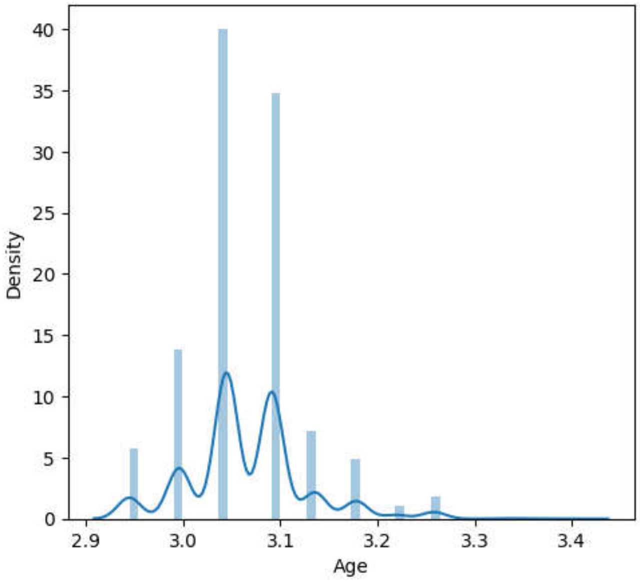
```
    dtypes: int64(6), object(2)
    memory usage: 185.5+ KB
```

```
df.isnull().sum()
```

```
    Age                  0
    Gender               0
    Stream               0
    Internships          0
    CGPA                 0
    Hostel               0
    HistoryOfBacklogs    0
    PlacedOrNot          0
    dtype: int64
```

Handling Outliers

```
def transformationplot(feature):
  plt.figure(figsize=(12,5))
  plt.subplot(1,2,1)
  sns.distplot(feature)
transformationplot(np.log(df['Age']))
```



Handling Categorial Values

```
df = df.replace(['Male'],[0])
df = df.replace(['Female'],[1])

df = df.replace(['Computer Science'],[0])
df = df.replace(['Information Technology'],[1])
df = df.replace(['Electronics And Communication'],[2])
df = df.replace(['Mechanical'],[3])
df = df.replace(['Electrical'],[4])
df = df.replace(['Civil'],[5])
```

```
df
```

	Age	Gender	Stream	Internships	CGPA	HistoryOfBacklogs	PlacedOrNot
0	22	0	2	1	8	1	1
1	21	1	0	0	7	1	1
2	22	1	1	1	6	0	1
3	21	0	1	0	8	1	1
4	22	0	3	0	8	0	1
...	...	...	...	...	...	...	...
2961	23	0	1	0	7	0	0
2962	23	0	3	1	7	0	0

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2966 entries, 0 to 2965
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Age                2966 non-null   int64
 1   Gender             2966 non-null   int64
 2   Stream             2966 non-null   int64
 3   Internships        2966 non-null   int64
 4   CGPA               2966 non-null   int64
 5   HistoryOfBacklogs  2966 non-null   int64
 6   PlacedOrNot        2966 non-null   int64
dtypes: int64(7)
memory usage: 162.3 KB
```

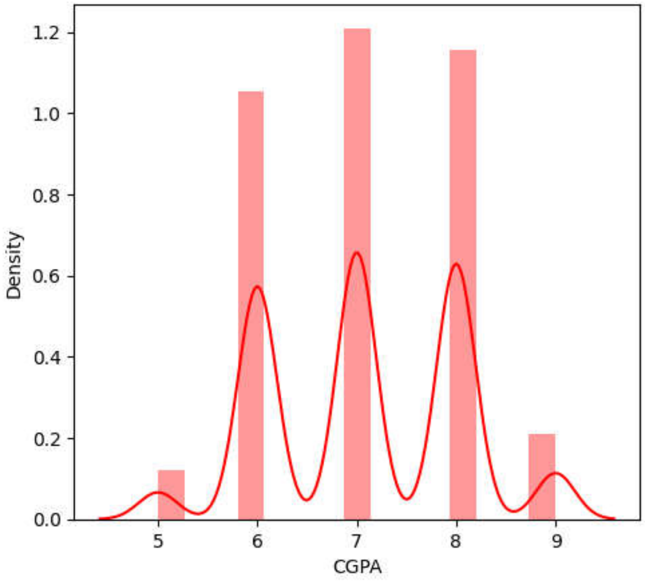
Explorary Data Analysis

Visual Analysis

1)Univariate analysis

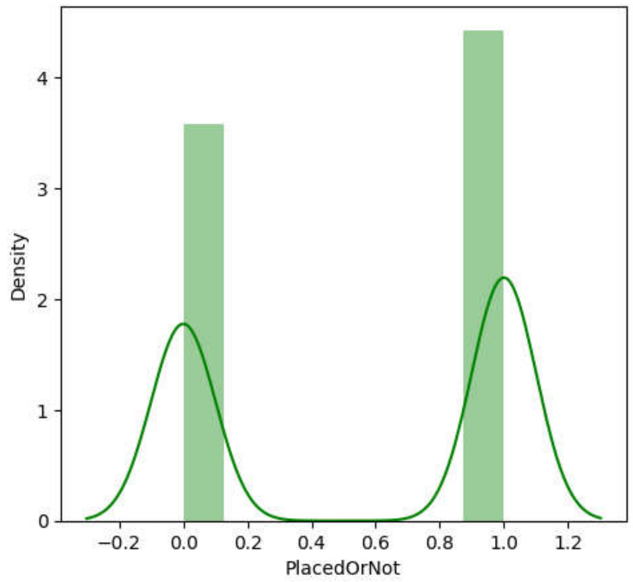
```
plt.figure(figsize=(12,5))
plt.subplot(121)
sns.distplot(df['CGPA'],color='r')
```

```
<Axes: xlabel='CGPA', ylabel='Density'>
```



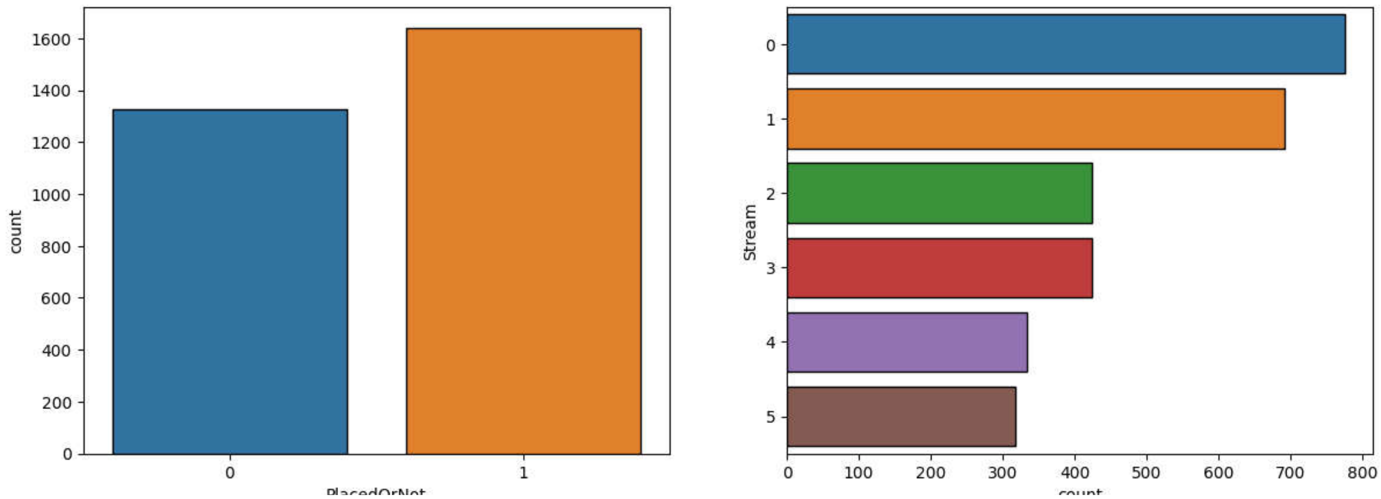
```
plt.figure(figsize=(12,5))
plt.subplot(121)
sns.distplot(df['PlacedOrNot'],color='g')
```

```
<Axes: xlabel='PlacedOrNot', ylabel='Density'>
```



2) Bivariate Analysis

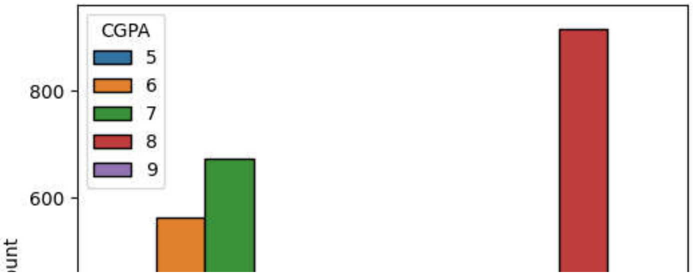
```
from matplotlib.offsetbox import martist
plt.figure(figsize=(30,5))
plt.subplot(1,4,1)
sns.countplot(x="PlacedOrNot",data=df, ec='black')
plt.subplot(1,4,2)
sns.countplot(y="Stream",data=df, ec='black')
plt.show()
```



Multivariate Analysis

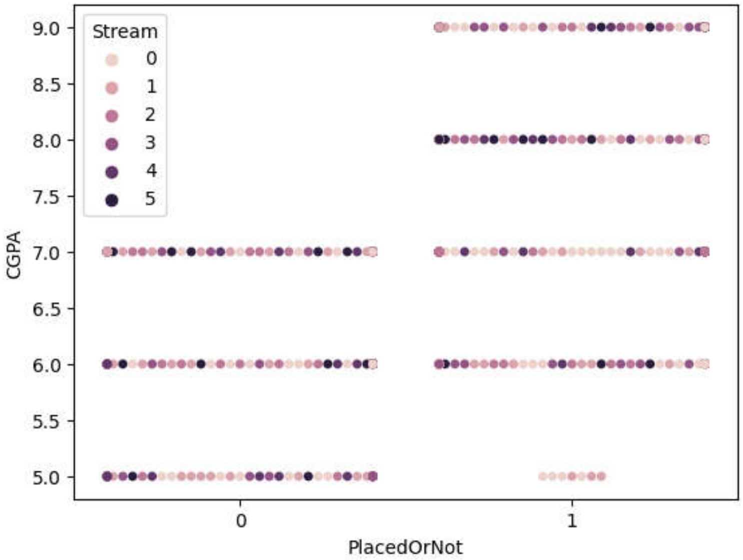
```
plt.figure(figsize=(20,5))
plt.subplot(131)
sns.countplot(x='PlacedOrNot', data=df, hue='CGPA', ec='black')
```

```
<Axes: xlabel='PlacedOrNot', ylabel='count'>
```



```
sns.swarmplot(x='PlacedOrNot',y='CGPA', hue='Stream', data=df)
```

```
<Axes: xlabel='PlacedOrNot', ylabel='CGPA'>
```



```
df.describe()
```

	Age	Gender	Stream	Internships	CGPA	HistoryOfBacklogs	PlacedOrNot
count	2966.000000	2966.000000	2966.000000	2966.000000	2966.000000	2966.000000	2966.000000
mean	21.485840	0.165543	1.932569	0.703641	7.073837	0.192178	0.552596
std	1.324933	0.371732	1.682618	0.740197	0.967748	0.394079	0.497310
min	19.000000	0.000000	0.000000	0.000000	5.000000	0.000000	0.000000
25%	21.000000	0.000000	0.000000	0.000000	6.000000	0.000000	0.000000
50%	21.000000	0.000000	2.000000	1.000000	7.000000	0.000000	1.000000
75%	22.000000	0.000000	3.000000	1.000000	8.000000	0.000000	1.000000
max	30.000000	1.000000	5.000000	3.000000	9.000000	1.000000	1.000000

Scaling the data

splitting the data into train and test

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

	Age	Gender	Stream	Internships	CGPA	HistoryOfBacklogs
0	22	0	2	1	8	1
1	21	1	0	0	7	1
2	22	1	1	1	6	0
3	21	0	1	0	8	1
4	22	0	3	0	8	0
...	...	...	...	...	...	...
2961	23	0	1	0	7	0
2962	23	0	3	1	7	0

```
y
```

```
0       1
1       1
2       1
3       1
4       1
       ..
2961    0
2962    0
2963    0
2964    0
2965    1
Name: PlacedOrNot, Length: 2966, dtype: int64
```

```
sc = StandardScaler()
x = sc.fit_transform(x)
x = pd.DataFrame(x)
```

```
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size= 0.11, stratify=y, random_state=42)
```

```
print(x_train.shape)

print(x_train.shape)
```

```
(2639, 6)
(327,)
```

Training the model in mulltiple algorithms

1.SVM model

```
from sklearn.svm import SVC
svm = SVC()
svm.fit(x_train,y_train)
SVC()
```

```
▾ SVC
SVC()
```

```
from sklearn import svm
classifier = svm.SVC()
x_test = np.array(x_test, dtype = float)
y_test = np.array(y_test, dtype = float)
classifier.fit(x_train, y_train)
SVC()

x_test_prediction = classifier.predict(x_test)
y_pred= accuracy_score(x_test_prediction,y_test)
y_pred
```

```
0.7767584097859327
```

KNN model

```
best_k = {"Regular":0}
best_score = {"Regular":0}
for k in range(3, 50, 2):
  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["Regualar"] = k


print("---Results---\nk: {}\nScore: {}".format(best_k, best_score))
knn = KNeighborsClassifier(n_neighbors=best_k["Regualar"])
knn.fit(x_train, y_train)
knn_pred = knn.predict(x_test)
testd = accuracy_score(knn_pred, y_test)
```

```
---Results---
k: {'Regular': 0, 'Regualar': 7}
Score: {'Regular': 88.37920489296636}
```

ANN

```
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from tensorflow.keras import layers


classifier = Sequential()

#add input layer and first hidden layer
classifier.add(keras.layers.Dense(6,activation = 'relu',input_dim = 6))
classifier.add(keras.layers.Dropout(0.50))

#add second hidden layer
classifier.add(keras.layers.Dense(6,activation = 'relu'))
classifier.add(keras.layers.Dropout(0,50))

#final or output layer
classifier.add(keras.layers.Dense(1,activation = 'sigmoid'))
```

```
#compiling the model
loss_1 = tf. keras.losses.BinaryCrossentropy()
classifier.compile(optimizer = 'Adam', loss= loss_1, metrics = ['accuracy'])


#fitting th model
classifier.fit(x_train, y_train, batch_size = 20, epochs = 100)
```

```
Epoch 1/100
132/132 [==============================] - 1s 2ms/step - loss: 1.7632 - accuracy: 0.5100
Epoch 2/100
132/132 [==============================] - 0s 3ms/step - loss: 0.8225 - accuracy: 0.5138
Epoch 3/100
132/132 [==============================] - 0s 2ms/step - loss: 0.7503 - accuracy: 0.5070
Epoch 4/100
132/132 [==============================] - 0s 2ms/step - loss: 0.7187 - accuracy: 0.5241
Epoch 5/100
132/132 [==============================] - 0s 2ms/step - loss: 0.7061 - accuracy: 0.5313
Epoch 6/100
132/132 [==============================] - 0s 2ms/step - loss: 0.7016 - accuracy: 0.5286
Epoch 7/100
132/132 [==============================] - 0s 2ms/step - loss: 0.6977 - accuracy: 0.5328
Epoch 8/100
132/132 [==============================] - 0s 2ms/step - loss: 0.6884 - accuracy: 0.5460
Epoch 9/100
132/132 [==============================] - 0s 2ms/step - loss: 0.6887 - accuracy: 0.5445
Epoch 10/100
```

```
132/132 [==============================] - 0s 2ms/step - loss: 0.6791 - accuracy: 0.5722
Epoch 11/100
132/132 [==============================] - 0s 3ms/step - loss: 0.6799 - accuracy: 0.5813
Epoch 12/100
132/132 [==============================] - 0s 4ms/step - loss: 0.6742 - accuracy: 0.5809
Epoch 13/100
132/132 [==============================] - 0s 2ms/step - loss: 0.6780 - accuracy: 0.5718
Epoch 14/100
132/132 [==============================] - 0s 2ms/step - loss: 0.6726 - accuracy: 0.5983
Epoch 15/100
132/132 [==============================] - 1s 4ms/step - loss: 0.6693 - accuracy: 0.5923
Epoch 16/100
132/132 [==============================] - 0s 2ms/step - loss: 0.6614 - accuracy: 0.6067
Epoch 17/100
132/132 [==============================] - 0s 2ms/step - loss: 0.6596 - accuracy: 0.6196
Epoch 18/100
132/132 [==============================] - 1s 4ms/step - loss: 0.6550 - accuracy: 0.6340
Epoch 19/100
132/132 [==============================] - 0s 2ms/step - loss: 0.6497 - accuracy: 0.6362
Epoch 20/100
132/132 [==============================] - 0s 2ms/step - loss: 0.6390 - accuracy: 0.6639
Epoch 21/100
132/132 [==============================] - 0s 2ms/step - loss: 0.6419 - accuracy: 0.6578
Epoch 22/100
132/132 [==============================] - 0s 2ms/step - loss: 0.6419 - accuracy: 0.6472
Epoch 23/100
132/132 [==============================] - 0s 2ms/step - loss: 0.6321 - accuracy: 0.6665
Epoch 24/100
132/132 [==============================] - 0s 2ms/step - loss: 0.6378 - accuracy: 0.6499
Epoch 25/100
132/132 [==============================] - 0s 2ms/step - loss: 0.6374 - accuracy: 0.6404
Epoch 26/100
132/132 [==============================] - 0s 2ms/step - loss: 0.6227 - accuracy: 0.6631
Epoch 27/100
132/132 [==============================] - 0s 2ms/step - loss: 0.6211 - accuracy: 0.6817
Epoch 28/100
132/132 [==============================] - 0s 2ms/step - loss: 0.6332 - accuracy: 0.6487
Epoch 29/100
132/132 [==============================] - 0s 2ms/step - loss: 0.6212 - accuracy: 0.6730
```

Model Deployment

Save the best model

```
import pickle

pickle.dump(knn,open("placement.pkl",'wb'))
model = pickle.load(open('placement.pkl','rb'))
```

```
input_data = [[22,0,2,1,8,1]]
```

```
prediction = knn.predict(input_data)
print(prediction)
if (prediction[0]==0):
  print('not placed')
else:
      print('placed')
```

```
placed
```

● ×