kaviyadevi 20106064

In [1]:
```
#to import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
#to import dataset
data1=pd.read_csv(r"C:\Users\user\Downloads\13_placement - 13_placement.csv")
data1
```

Out[2]:

	cgpa	placement_exam_marks	placed
0	7.19	26	1
1	7.46	38	1
2	7.54	40	1
3	6.42	8	1
4	7.23	17	0
...	...	...	...
995	8.87	44	1
996	9.12	65	1
997	4.89	34	0
998	8.62	46	1
999	4.90	10	1

1000 rows × 3 columns

In [3]:
```
#to display top 5 rows
data=data1.head()
data
```

Out[3]:

	cgpa	placement_exam_marks	placed
0	7.19	26	1
1	7.46	38	1
2	7.54	40	1
3	6.42	8	1
4	7.23	17	0

DATA CLEANING AND PREPROCESSING

In [4]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 3 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   cgpa                  5 non-null      float64
 1   placement_exam_marks  5 non-null      int64  
 2   placed                5 non-null      int64  
dtypes: float64(1), int64(2)
memory usage: 248.0 bytes
```

In [5]:
```
#to display summary of statistics
data.describe()
```

Out[5]:

	cgpa	placement_exam_marks	placed
count	5.0000	5.000000	5.000000
mean	7.1680	25.800000	0.800000
std	0.4437	13.645512	0.447214
min	6.4200	8.000000	0.000000
25%	7.1900	17.000000	1.000000
50%	7.2300	26.000000	1.000000
75%	7.4600	38.000000	1.000000
max	7.5400	40.000000	1.000000

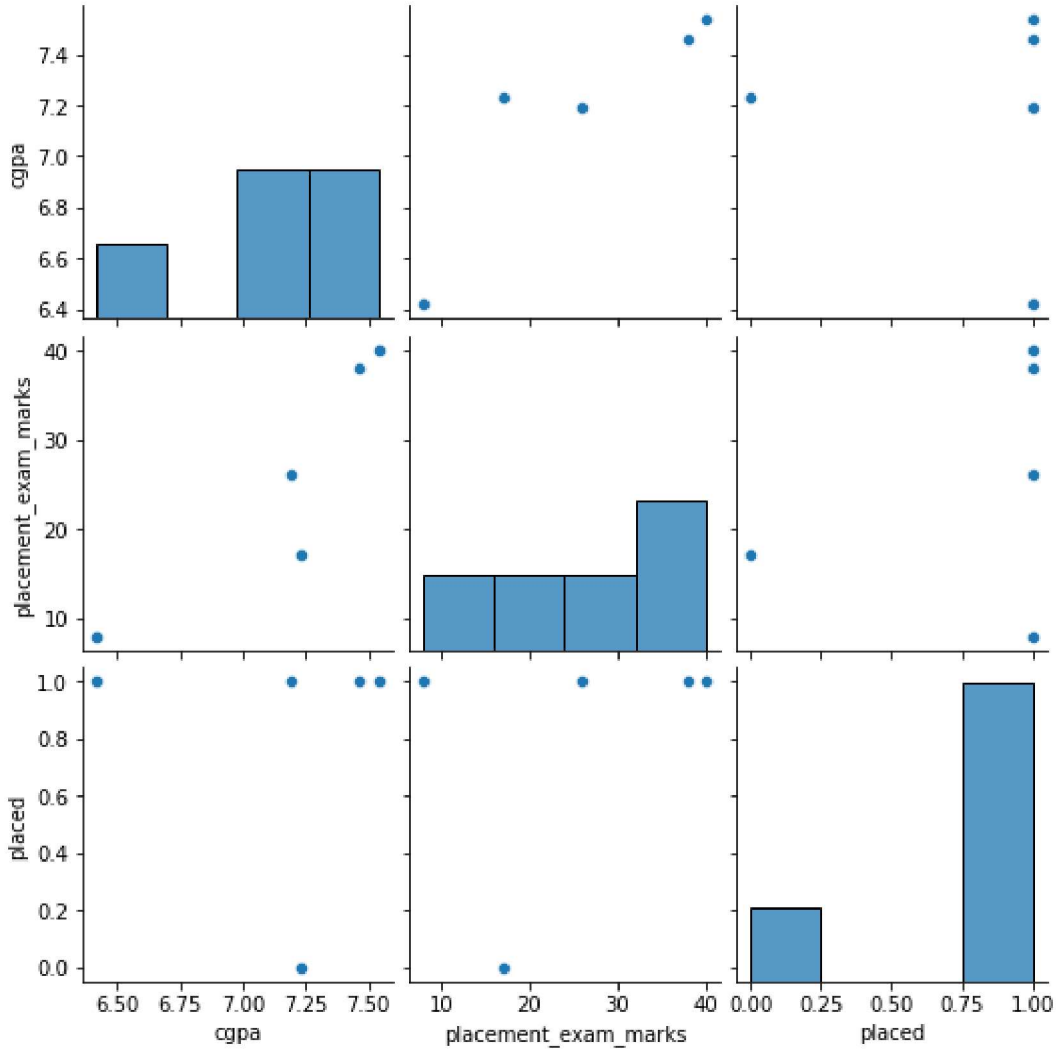
In [6]:
```
#to display the column heading
data.columns
```

Out[6]:
```
Index(['cgpa', 'placement_exam_marks', 'placed'], dtype='object')
```

EDA and DATA VISUALIZATION

In [7]:
```
sns.pairplot(data)
```

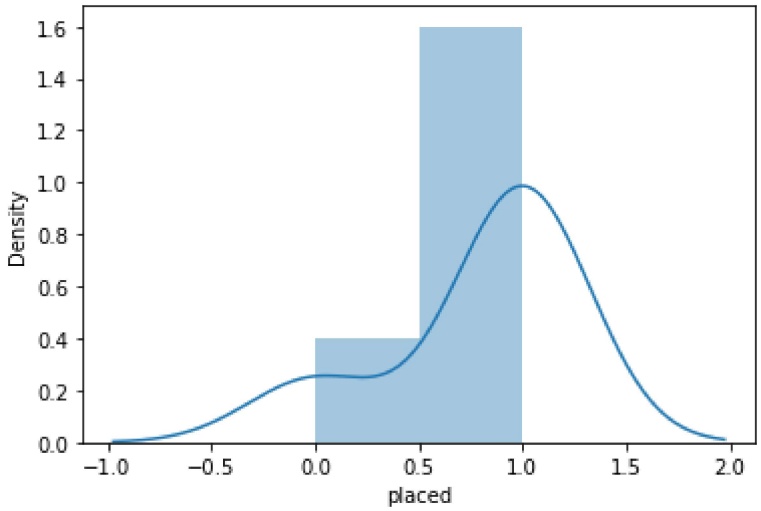
Out[7]:
```
<seaborn.axisgrid.PairGrid at 0x245a57f2040>
```



In [8]:
```
sns.distplot(data['placed'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

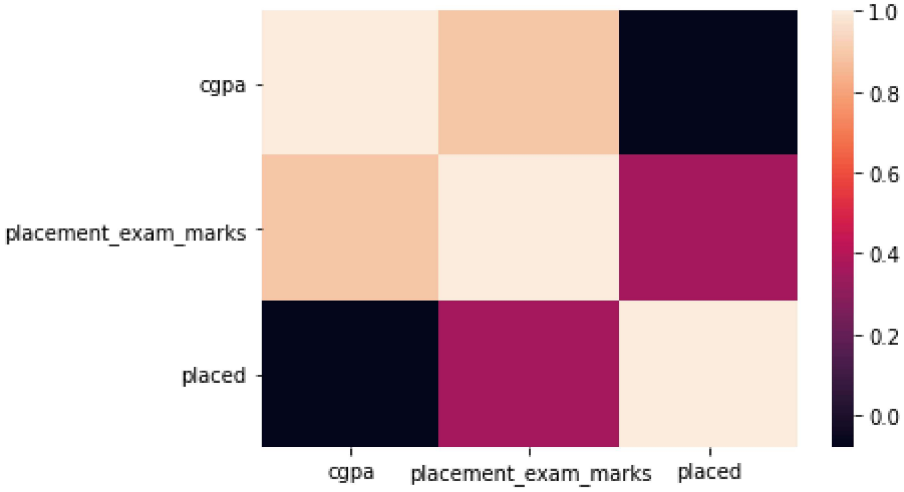
Out[8]: <AxesSubplot:xlabel='placed', ylabel='Density'>



In [9]:
```
df=data[['cgpa', 'placement_exam_marks', 'placed']]
```

In [10]:
```
sns.heatmap(df.corr())
```

Out[10]: <AxesSubplot:>



TRAINING MODEL

In [11]:
```
x=df[['cgpa', 'placement_exam_marks','placed']]
y=df[['placement_exam_marks']]
```

In [12]:
```
#to split my dataset into trainning and test
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
```

In [13]:
```
from sklearn.linear_model import LinearRegression

lr=LinearRegression()
lr.fit(x_train,y_train)
```

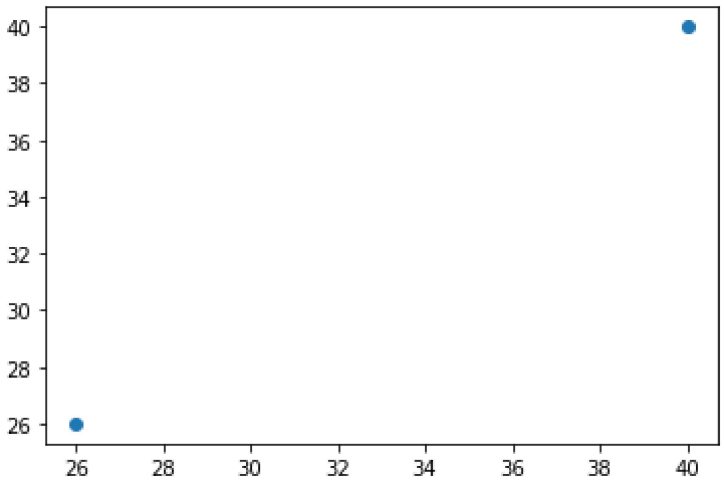
Out[13]:
```
LinearRegression()
```

In [14]:
```
#to find intercept
print(lr.intercept_)
```

```
[-0.18428491]
```

In [15]:
```
prediction = lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[15]:
```
<matplotlib.collections.PathCollection at 0x245a7ee4f40>
```



In [16]:
```
print(lr.score(x_test,y_test))
```

```
0.9999998315977877
```

RIDGE AND LASSO REGRESSION

In [17]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [18]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[18]:
```
Ridge(alpha=10)
```

In [19]:
```
rr.score(x_test,y_test)
```

Out[19]:
```
0.9983212463845949
```

In [20]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[20]:
```
Lasso(alpha=10)
```

In [21]:
```
la.score(x_test,y_test)
```

Out[21]:
```
0.9842221778953529
```

In [22]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[22]:
```
ElasticNet()
```

In [23]:
```
print(en.coef_)
```

```
[0.         0.99369085 0.        ]
```

In [24]:
```
print(en.predict(x_test))
```

```
[25.96845426 39.88012618]
```

In [25]:
```
print(en.score(x_test,y_test))
```

```
0.9998432156550386
```

In [26]:
```
from sklearn import metrics
```

In [27]:
```
print("Mean Absolute error",metrics.mean_absolute_error(y_test,prediction))
```

```
Mean Absolute error 0.0021738256449523874
```

In [28]:
```
print("Mean Squared error",metrics.mean_squared_error(y_test,prediction))
```

```
Mean Squared error 8.251708402157943e-06
```

In [29]:
```
print("Root Mean Absolute error",np.sqrt(metrics.mean_squared_error(y_test,predic
```

```
Root Mean Absolute error 0.0028725787025176423
```