DATA COLLECTION

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:

```
a=pd.read_csv(r"C:\Users\user\Downloads\13_placement.csv")
a
```

Out[2]:

	cgpa	placement_exam_marks	placed
0	7.19	26.0	1
1	7.46	38.0	1
2	7.54	40.0	1
3	6.42	8.0	1
4	7.23	17.0	0
...	...	...	...
995	8.87	44.0	1
996	9.12	65.0	1
997	4.89	34.0	0
998	8.62	46.0	1
999	4.90	10.0	1

1000 rows × 3 columns

In [3]:

```
a.head(5)
```

Out[3]:

	cgpa	placement_exam_marks	placed
0	7.19	26.0	1
1	7.46	38.0	1
2	7.54	40.0	1
3	6.42	8.0	1
4	7.23	17.0	0

DATA CLEANING AND PRE-PROCESSING

In [4]:

```
a.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 3 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   cgpa                  1000 non-null   float64
 1   placement_exam_marks  1000 non-null   float64
 2   placed                1000 non-null   int64  
dtypes: float64(2), int64(1)
memory usage: 23.6 KB
```

In [5]:

```
a.describe()
```

Out[5]:

	cgpa	placement_exam_marks	placed
count	1000.000000	1000.000000	1000.000000
mean	6.961240	32.225000	0.489000
std	0.615898	19.130822	0.500129
min	4.890000	0.000000	0.000000
25%	6.550000	17.000000	0.000000
50%	6.960000	28.000000	0.000000
75%	7.370000	44.000000	1.000000
max	9.120000	100.000000	1.000000

In [6]:

```
a.columns
```

Out[6]:

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

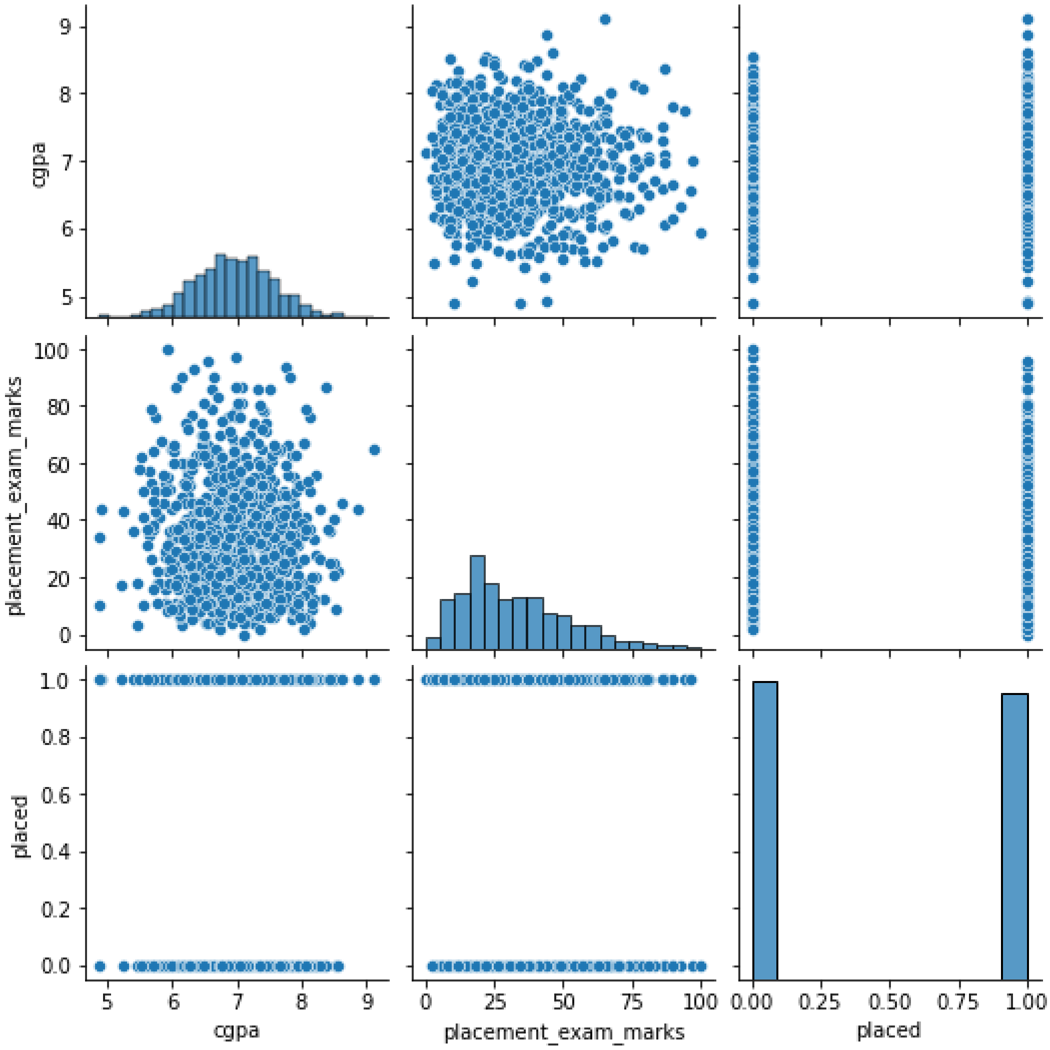
EDA and VISUALIZATION

In [7]:

```
sns.pairplot(a)
```

Out[7]:

```
<seaborn.axisgrid.PairGrid at 0x208a607dbe0>
```



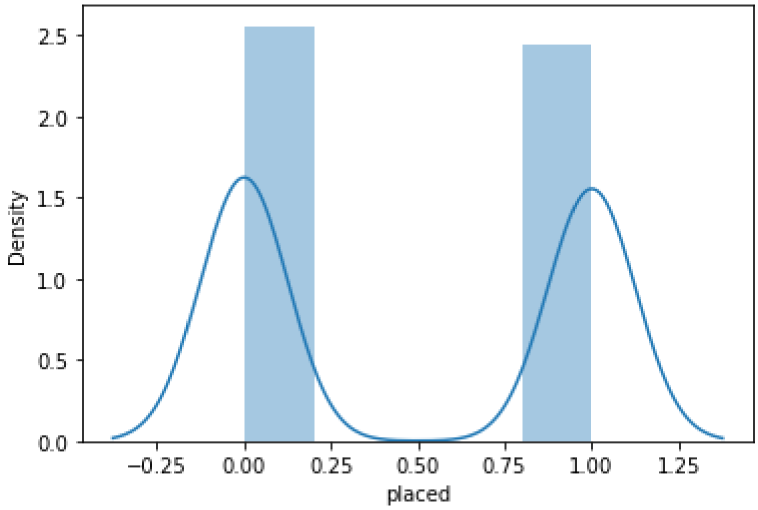
In [8]:

```
sns.distplot(a['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 fun
ction for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[8]:

```
<AxesSubplot:xlabel='placed', ylabel='Density'>
```



In [9]:

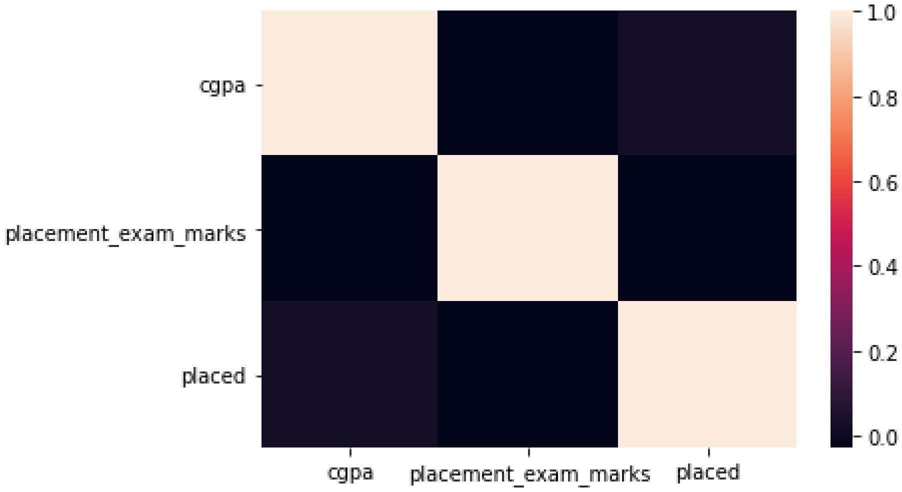
```
f=a[['cgpa', 'placement_exam_marks', 'placed']]
```

In [10]:

```
sns.heatmap(f.corr())
```

Out[10]:

```
<AxesSubplot:>
```



To train the model-model building

In [11]:

```
x=f[['cgpa', 'placement_exam_marks']]
y=f['placed']
```

In [12]:

```
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]:

```
print(lr.intercept_)
```

```
0.3274428091728991
```

In [15]:

```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[15]:

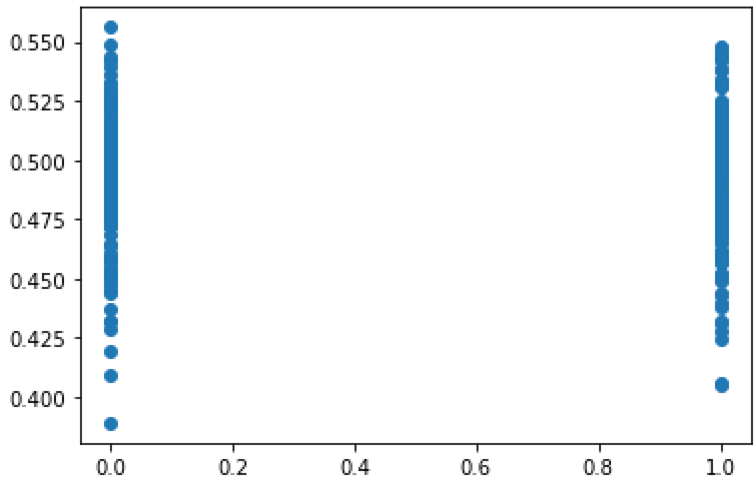
	Co-efficient
cgpa	0.028679
placement_exam_marks	-0.001092

In [16]:

```
prediction=lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[16]:

```
<matplotlib.collections.PathCollection at 0x208a87f3d90>
```



In [17]:

```
print(lr.score(x_test,y_test))
```

```
-0.0040838041602666575
```

In [18]:

```
lr.score(x_train,y_train)
```

Out[18]:

```
0.0030249247639947408
```

RIDGE REGRESSION

In [19]:

```
from sklearn.linear_model import Ridge,Lasso
```

In [20]:

```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[20]:

```
Ridge(alpha=10)
```

In [21]:

```
rr.score(x_test,y_test)
```

Out[21]:

```
-0.004012228951224106
```

LASSO REGRESSION

In [22]:

```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[22]:

```
Lasso(alpha=10)
```

In [23]:

```
la.score(x_test,y_test)
```

Out[23]:

```
-0.00026242310049706674
```

In [24]:

```
from sklearn.linear_model import ElasticNet
p=ElasticNet()
p.fit(x_train,y_train)
```

Out[24]:

```
ElasticNet()
```

In [25]:

```
print(p.coef_)
```

```
[ 0. -0.]
```

In [26]:

```
print(p.intercept_)
```

```
0.49142857142857144
```

In [27]:

```
print(p.predict(x_test))
```

```
[0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857
 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857 0.49142857]
```

In [28]:

```
prediction=p.predict(x_test)
print(p.score(x_test,y_test))
```

```
-0.00026242310049706674
```

In [29]:

```
from sklearn import metrics
```

In [30]:

```
print("Mean Absolytre Error:",metrics.mean_absolute_error(y_test,prediction))
```

```
Mean Absolytre Error: 0.4997142857142857
```

In [31]:

```
print("Mean Squared Error:",metrics.mean_squared_error(y_test,prediction))
```

```
Mean Squared Error: 0.2497877551020408
```

In [32]:

```
print("Root Mean Squared Error:",np.sqrt(metrics.mean_squared_error(y_test,prediction)))
```

```
Root Mean Squared Error: 0.49978771003501155
```

In [ ]: