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

In [2]:
```
df=pd.read_csv(r"C:\Users\Admin\Downloads\13_placement - 13_placement.csv")
df
```

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]:
```
df.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   int64  
 2   placed                1000 non-null   int64  
dtypes: float64(1), int64(2)
memory usage: 23.6 KB
```

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

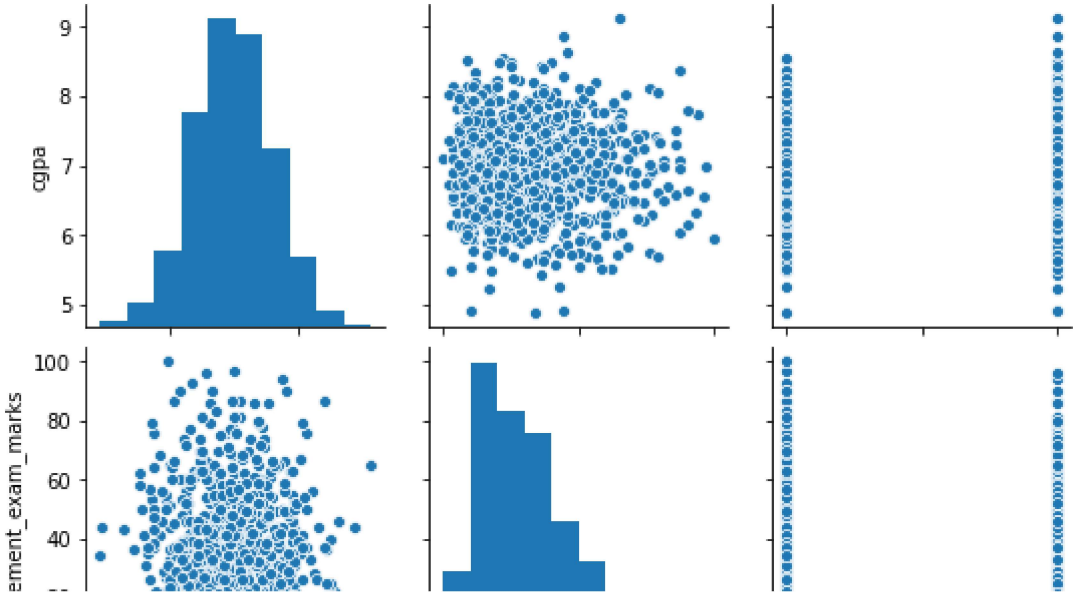
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 3 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   cgpa                  1000 non-null   float64
 1   placement_exam_marks  1000 non-null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

In [5]:
```
df.columns
```

Out[5]:
```
Index(['cgpa', 'placement_exam_marks', 'placed'], dtype='object')
```

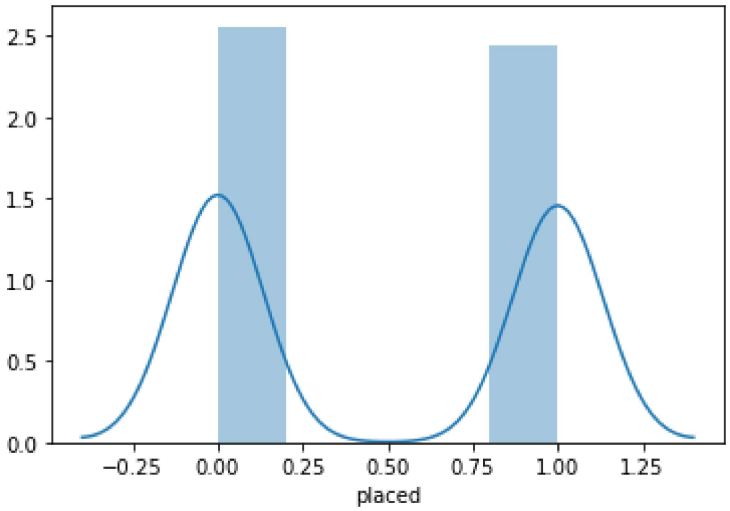
In [6]:
```
sns.pairplot(df)
```

Out[6]:
```
<seaborn.axisgrid.PairGrid at 0x1ca37530550>
```



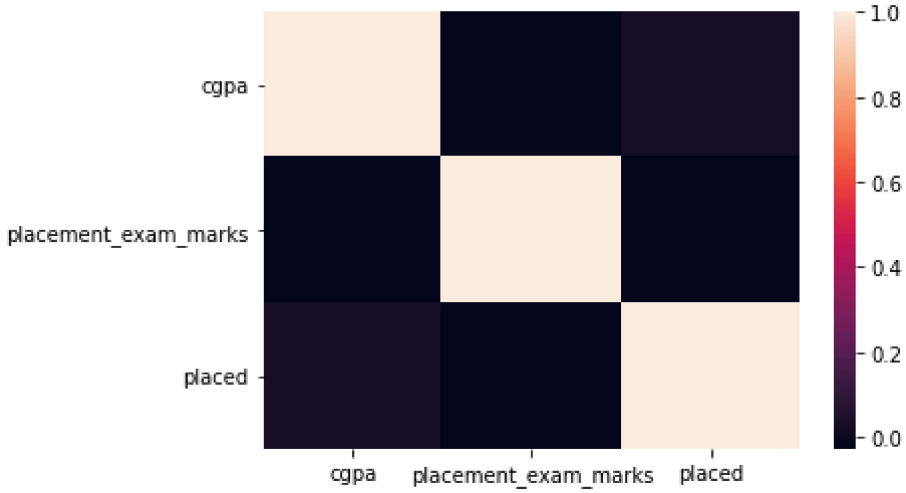
In [7]:
```
sns.distplot(df['placed'])
```

Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x1ca37a84a30>



In [8]:
```
sns.heatmap(df.corr())
```

Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x1ca37c58d00>



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

In [10]:
```
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 [11]:
```
from sklearn.linear_model import LinearRegression
lr= LinearRegression()
lr.fit(x_train,y_train)
```

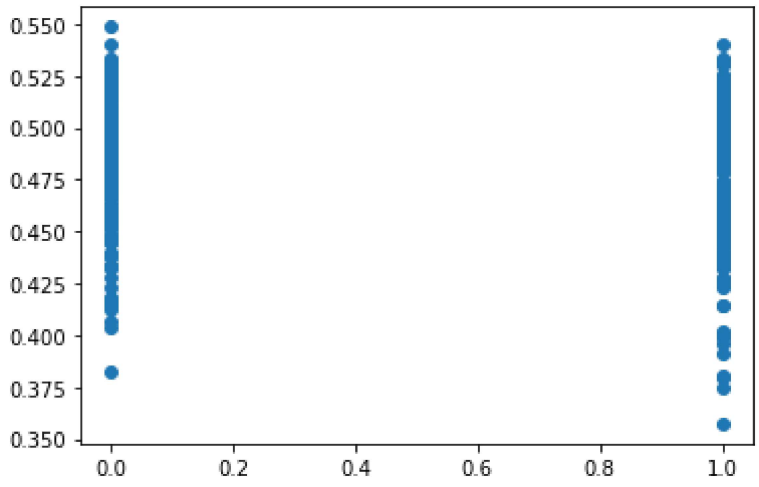
Out[11]: LinearRegression()

In [12]:
```
print(lr.intercept_)
```

```
[0.42722903]
```

In [13]:
```
prediction= lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[13]:
```
<matplotlib.collections.PathCollection at 0x1ca38736940>
```



In [14]:
```
print(lr.score(x_test,y_test))
```

```
-0.019464071417655937
```

In [15]:
```
print(lr.score(x_train,y_train))
```

```
0.005418009476347074
```

In [16]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [17]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[17]:
```
Ridge(alpha=10)
```

In [18]:
```
rr.score(x_test,y_test)
```

Out[18]:
```
-0.019498679246249928
```

In [19]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[19]:
```
Lasso(alpha=10)
```

In [20]:
```
la.score(x_test,y_test)
```

Out[20]:
```
-0.004836999734958658
```

In [21]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[21]:
```
ElasticNet()
```

In [22]:
```
print(en.intercept_)
```

```
[0.49474932]
```

In [23]:
```
print(en.predict(x_test))
```

```
[0.47928207 0.47129897 0.48975988 0.46880425 0.46730742 0.48377256
 0.44784862 0.47628841 0.47329474 0.46780636 0.46281693 0.48876199
 0.48277467 0.48676622 0.48227573 0.49125671 0.48377256 0.46231799
 0.4727958  0.48826305 0.46880425 0.47578946 0.49474932 0.48576833
 0.4842715  0.48526939 0.48776411 0.48676622 0.45982327 0.48526939
 0.4648127  0.47678735 0.48127784 0.47678735 0.48626728 0.48676622
 0.48926094 0.4727958  0.47978101 0.48626728 0.49025882 0.47129897
 0.48876199 0.47878312 0.47578946 0.48826305 0.4842715  0.49075777
 0.46531165 0.46581059 0.47529052 0.47728629 0.4842715  0.46980214
 0.48626728 0.47878312 0.47179791 0.48177678 0.47978101 0.49125671
 0.47628841 0.45583172 0.48975988 0.49025882 0.46780636 0.49025882
 0.48327362 0.4693032  0.45134123 0.48377256 0.47778524 0.48975988
 0.48377256 0.4807789  0.47529052 0.48626728 0.48327362 0.47828418
 0.46581059 0.46680848 0.49025882 0.48377256 0.48776411 0.46431376
 0.48227573 0.46730742 0.46880425 0.48676622 0.48975988 0.48826305
 0.47928207 0.48576833 0.4842715  0.47379369 0.46331587 0.48975988
 0.47778524 0.47529052 0.48027995 0.48027995 0.4807789  0.48826305
 0.47828418 0.47728629 0.48726516 0.48626728 0.46880425 0.48177678
 0.49275354 0.48576833 0.48177678 0.48127784 0.45533278 0.47778524
 0.47778524 0.46980214 0.48975988 0.47479157 0.48027995 0.48626728
 0.48826305 0.48377256 0.49075777 0.48926094 0.46331587 0.47728629
 0.46780636 0.47329474 0.45782749 0.47778524 0.49025882 0.48027995
 0.48277467 0.48676622 0.47778524 0.47529052 0.47678735 0.48776411
 0.45732855 0.49175566 0.47878312 0.48626728 0.47728629 0.47678735
 0.48277467 0.48726516 0.47379369 0.4807789  0.46032221 0.48576833
 0.48177678 0.48377256 0.47529052 0.48377256 0.45882538 0.48576833
 0.48526939 0.47828418 0.48526939 0.48477045 0.48726516 0.49025882
 0.49025882 0.48377256 0.47179791 0.47578946 0.47179791 0.48127784
 0.48876199 0.4727958  0.47080003 0.47578946 0.47529052 0.48277467
 0.48177678 0.48477045 0.48227573 0.47080003 0.45982327 0.4842715
 0.48975988 0.4807789  0.47429263 0.49075777 0.48177678 0.45533278
 0.48327362 0.47778524 0.48377256 0.47678735 0.47828418 0.4648127
 0.47379369 0.48377256 0.44685074 0.47978101 0.47628841 0.48826305
 0.48377256 0.47578946 0.48576833 0.47379369 0.48676622 0.48776411
 0.49025882 0.47778524 0.47529052 0.46680848 0.45882538 0.47628841
 0.48676622 0.47080003 0.47928207 0.45134123 0.47329474 0.46630953
 0.48127784 0.48327362 0.48576833 0.47578946 0.48975988 0.47329474
 0.46780636 0.46780636 0.46281693 0.46980214 0.48227573 0.48227573
 0.46730742 0.48227573 0.47728629 0.48177678 0.48726516 0.47429263
 0.48327362 0.47429263 0.48526939 0.47678735 0.49075777 0.4693032
 0.48826305 0.48477045 0.4648127  0.46331587 0.4727958  0.47030108
 0.4727958  0.47628841 0.49375143 0.47928207 0.48975988 0.46880425
 0.48676622 0.47728629 0.48177678 0.4807789  0.46980214 0.48526939
 0.48177678 0.48327362 0.48227573 0.47379369 0.48477045 0.47030108
 0.47828418 0.47978101 0.49025882 0.48626728 0.46381482 0.47080003
 0.49025882 0.46880425 0.47129897 0.49275354 0.48477045 0.48127784
 0.47628841 0.48876199 0.48477045 0.4727958  0.49075777 0.48576833
 0.48177678 0.47878312 0.48926094 0.46281693 0.48377256 0.48626728
 0.48926094 0.48177678 0.49025882 0.47628841 0.47179791 0.48626728
 0.46730742 0.47728629 0.48377256 0.48926094 0.49025882 0.48377256]
```

In [24]:
```
print(en.score(x_test,y_test))
```

```
-0.008315769992271349
```

Evaluation

In [25]:
```
from sklearn import metrics
```

In [26]:
```
print("Mean Absolute Error",metrics.mean_absolute_error(y_test,prediction))
```

```
Mean Absolute Error 0.50309738267642
```

In [27]:
```
print("Mean Squared Error",metrics.mean_squared_error(y_test,prediction))
```

```
Mean Squared Error 0.25468477979727305
```

In [28]:
```
print("Root Mean Squared Error:",np.sqrt(metrics.mean_squared_error(y_test,prec
```

```
Root Mean Squared Error: 0.5046630358935287
```

In [ ]:
```

```