# Problem statement

predicting the house price in USA.To create a model to help him estimate of what the house would sell for.

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("placement")
```

# To display top 10 rows

In [3]:
```
df.head(10)
```

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 |
| **5** | 7.30 | 23.0 | 1 |
| **6** | 6.69 | 11.0 | 0 |
| **7** | 7.12 | 39.0 | 1 |
| **8** | 6.45 | 38.0 | 0 |
| **9** | 7.75 | 94.0 | 1 |

# Data Cleaning And Pre-Processing

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

In [5]:
```
# Display the statistical summary
df.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]:
```
# To display the col headings
df.columns
```

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

In [7]:

```
cols=df.dropna(axis=1)
```

In [8]:

```
cols.columns
```

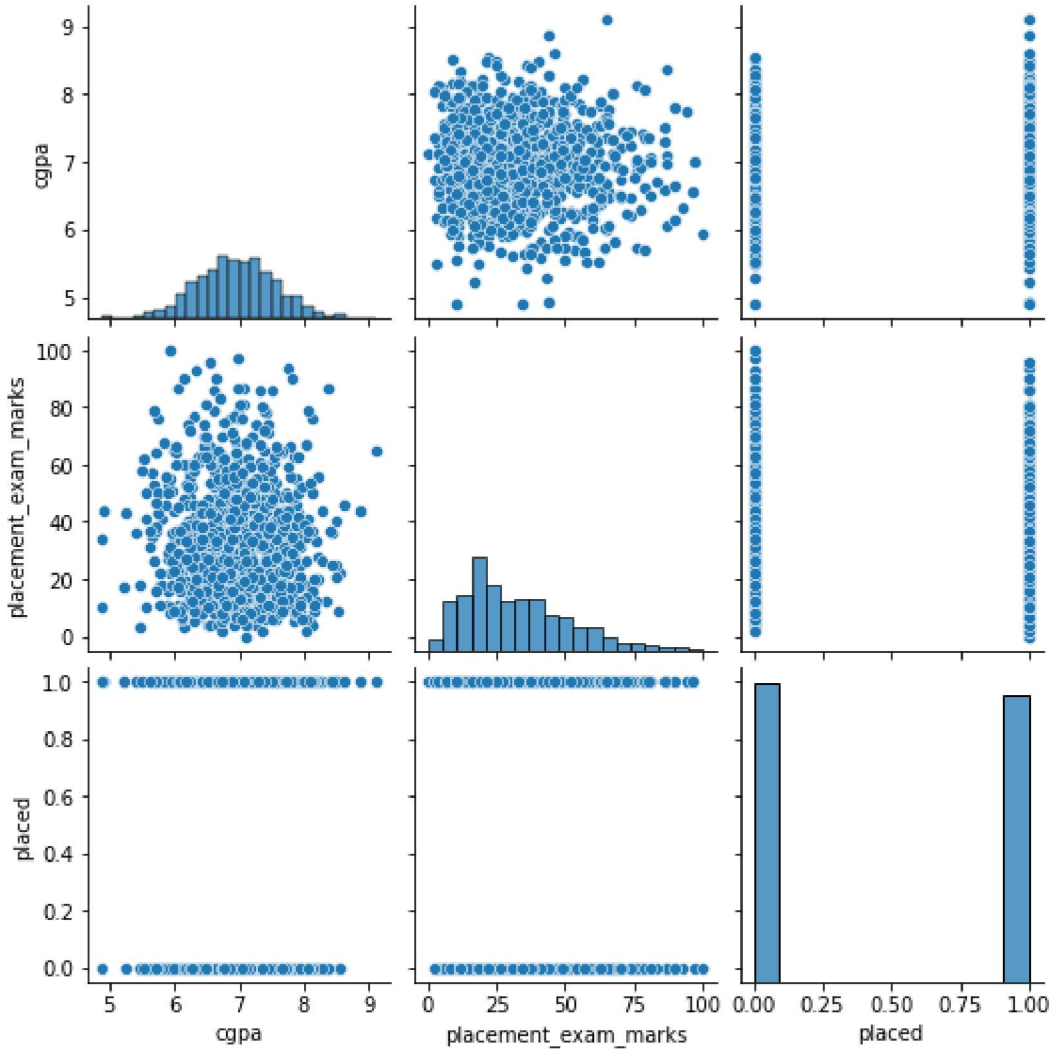
Out[8]:

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

# EDA and Visualization

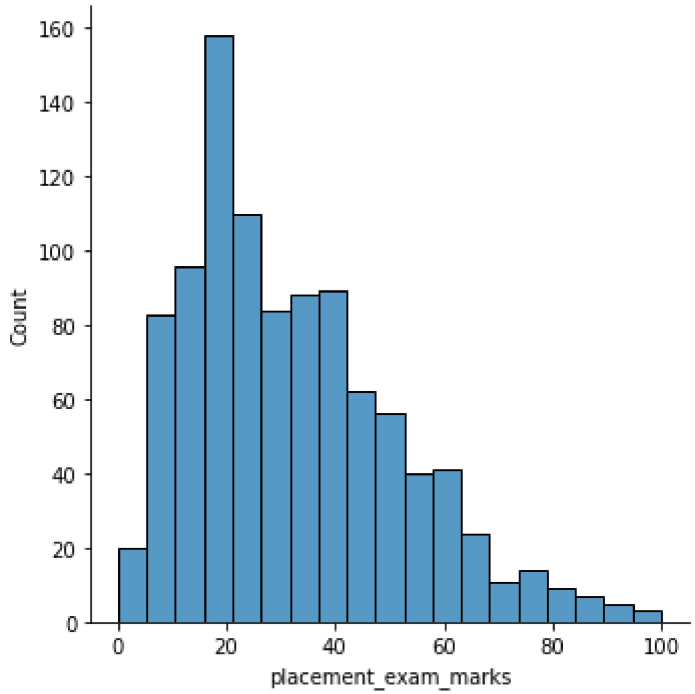
```
In [9]:  1  sns.pairplot(cols)
```

```
Out[9]: <seaborn.axisgrid.PairGrid at 0x2bf6f49e280>
```



In [10]:
```
sns.displot(df['placement_exam_marks'])
```

Out[10]: <seaborn.axisgrid.FacetGrid at 0x2bf711198b0>



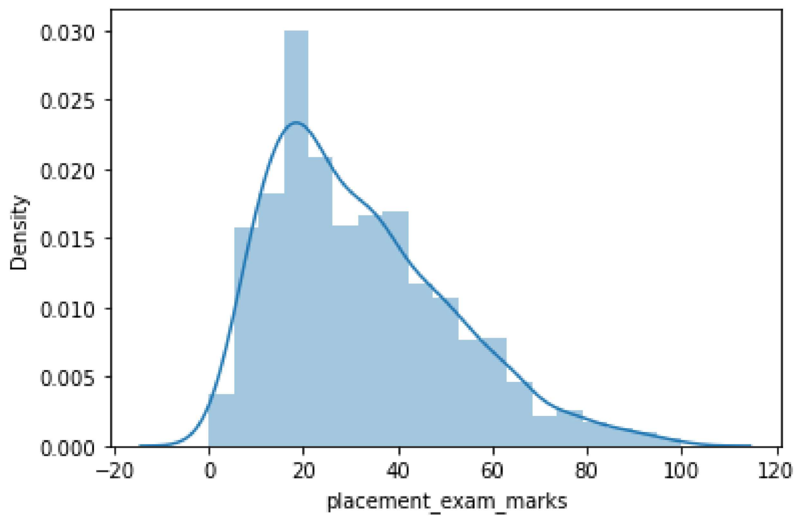
In [11]:

```
# We use displot in older version we get distplot use displot
sns.distplot(df['placement_exam_marks'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a dep
recated 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[11]:

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



In [12]:

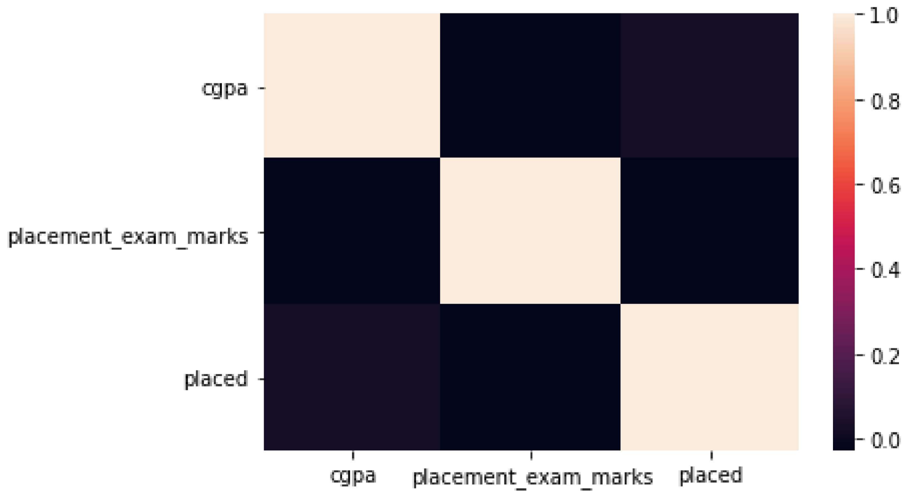
```
df1=cols[['cgpa', 'placement_exam_marks', 'placed']]
df1
```

Out[12]:

| | 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 [13]:
```
sns.heatmap(df1.corr())
```
Out[13]: <AxesSubplot:>




# To train the model - MODEL BUILD

Going to train linear regression model;We split our data into 2 variables x and y where x is independent var(input) and y is dependent on x(output), we could ignore address col as it is not required for our model

In [14]:
```
x=df1[['cgpa', 'placement_exam_marks', 'placed']]
y=df1[['placed']]
```

# To split the dataset into test data

In [15]:
```
# importing lib for splitting test data
from sklearn.model_selection import train_test_split
```

In [16]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3)
```

In [17]:
```
from sklearn.linear_model import LinearRegression

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

Out[17]:
```
LinearRegression()
```

In [18]:
```
print(lr.intercept_)
```

```
[-3.33066907e-16]
```

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

```
1.0
```

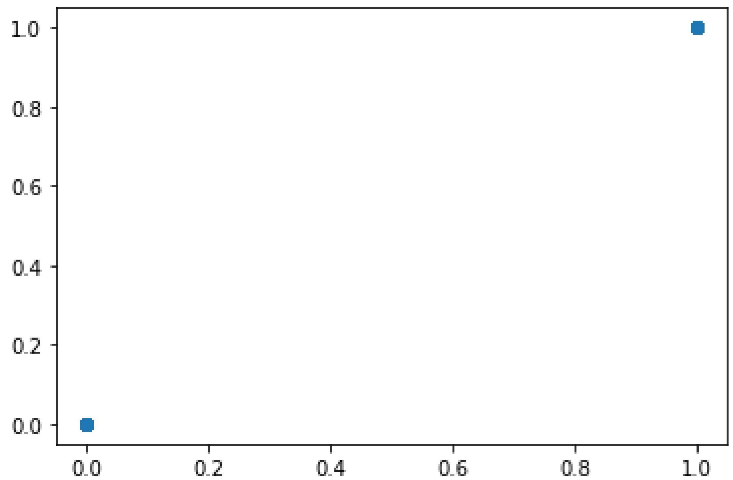
In [20]:
```
coeff=pd.DataFrame(lr.coef_)
coeff
```

Out[20]:

| | 0 | 1 | 2 |
|---|---|---|---|
| **0** | 1.196068e-16 | -6.880975e-18 | 1.0 |

In [21]:
```
pred = lr.predict(x_test)
plt.scatter(y_test,pred)
```

Out[21]: <matplotlib.collections.PathCollection at 0x2bf71ce5f70>




In [22]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [23]:
```
rr=Ridge(alpha=20)
rr.fit(x_train,y_train)
```

Out[23]: Ridge(alpha=20)

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

Out[24]: 0.9888145970795182

In [25]:
```
la=Lasso(alpha=20)
la.fit(x_train,y_train)
```

Out[25]: Lasso(alpha=20)

In [26]:
```
la.score(x_test,y_test)
```

Out[26]: -0.019886363636363535

# ELASTIC NET

In [30]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[30]:
```
ElasticNet()
```

In [31]:
```
print(en.coef_)
```

```
[ 0.          -0.00021605  0.        ]
```

In [32]:
```
print(en.intercept_)
```

```
[0.51692275]
```

In [33]:

```
prediction=en.predict(x_test)
prediction
```

```
Out[33]: array([0.51497833, 0.51303391, 0.51022531, 0.513466  , 0.51497833,
            0.51022531, 0.51130554, 0.51130554, 0.51216972, 0.51195368,
            0.51324996, 0.50698461, 0.50568833, 0.51368205, 0.50892903,
            0.513466  , 0.50871298, 0.51368205, 0.50914507, 0.51216972,
            0.51238577, 0.50439205, 0.5074167 , 0.50093531, 0.51000926,
            0.50568833, 0.49899089, 0.513466  , 0.51130554, 0.51022531,
            0.51454624, 0.51303391, 0.51433019, 0.51411414, 0.50439205,
            0.51195368, 0.50676856, 0.51541042, 0.51130554, 0.51216972,
            0.51238577, 0.51303391, 0.50590438, 0.50892903, 0.50698461,
            0.50806484, 0.51216972, 0.50892903, 0.51108949, 0.51303391,
            0.50395996, 0.50482414, 0.51130554, 0.51195368, 0.50633647,
            0.51087345, 0.50331182, 0.51108949, 0.50828089, 0.51411414,
            0.51411414, 0.50201554, 0.50720065, 0.51216972, 0.50331182,
            0.51519437, 0.50936112, 0.51562647, 0.51476228, 0.51303391,
            0.51260182, 0.50287973, 0.51324996, 0.50892903, 0.50849693,
            0.51605856, 0.5106574 , 0.51519437, 0.51173763, 0.50698461,
            0.50849693, 0.50849693, 0.50525624, 0.51519437, 0.51368205,
            0.51368205, 0.51108949, 0.51303391, 0.50266368, 0.51152158,
            0.4981267 , 0.50504019, 0.51303391, 0.50331182, 0.51411414,
            0.513466  , 0.51324996, 0.51238577, 0.50957717, 0.51324996,
            0.51130554, 0.51238577, 0.51476228, 0.5106574 , 0.51000926,
            0.50763275, 0.51108949, 0.51152158, 0.5106574 , 0.50914507,
            0.51368205, 0.51000926, 0.51368205, 0.5074167 , 0.51476228,
            0.51324996, 0.51562647, 0.51195368, 0.50504019, 0.51173763,
            0.51541042, 0.50871298, 0.50936112, 0.50763275, 0.51022531,
            0.49963903, 0.51497833, 0.513466  , 0.51087345, 0.51044135,
            0.51130554, 0.50504019, 0.50676856, 0.50914507, 0.51216972,
            0.50892903, 0.51454624, 0.50892903, 0.50806484, 0.51152158,
            0.5074167 , 0.50504019, 0.50914507, 0.51368205, 0.50698461,
            0.51022531, 0.50784879, 0.51324996, 0.504176  , 0.51130554,
            0.50007112, 0.50936112, 0.50979321, 0.50244763, 0.50676856,
            0.51022531, 0.50871298, 0.51173763, 0.50504019, 0.49618228,
            0.50914507, 0.51519437, 0.50849693, 0.51195368, 0.50007112,
            0.51605856, 0.5138981 , 0.51130554, 0.51454624, 0.50828089,
            0.50698461, 0.5046081 , 0.51087345, 0.50871298, 0.51044135,
            0.51541042, 0.51152158, 0.51281786, 0.50914507, 0.51497833,
            0.50957717, 0.51238577, 0.51368205, 0.51497833, 0.51497833,
            0.51497833, 0.50568833, 0.50439205, 0.51411414, 0.51152158,
            0.5013674 , 0.51216972, 0.50892903, 0.51476228, 0.51519437,
            0.51216972, 0.50806484, 0.50784879, 0.51173763, 0.51000926,
            0.51454624, 0.51087345, 0.51281786, 0.50568833, 0.5074167 ,
            0.50374391, 0.50568833, 0.50957717, 0.50849693, 0.51087345,
            0.50784879, 0.51324996, 0.50892903, 0.50936112, 0.51454624,
```

```
       0.50936112, 0.51260182, 0.51108949, 0.50763275, 0.50957717,
       0.51108949, 0.50179949, 0.51411414, 0.51173763, 0.51087345,
       0.50784879, 0.50287973, 0.50957717, 0.50892903, 0.51087345,
       0.50504019, 0.50979321, 0.51324996, 0.50028717, 0.51000926,
       0.51368205, 0.50957717, 0.50547228, 0.51411414, 0.50612042,
       0.50914507, 0.51000926, 0.51044135, 0.50849693, 0.51216972,
       0.51303391, 0.50892903, 0.51281786, 0.51022531, 0.51000926,
       0.504176  , 0.50979321, 0.51368205, 0.51433019, 0.51303391,
       0.51324996, 0.51195368, 0.5013674 , 0.50655252, 0.51497833,
       0.51281786, 0.51260182, 0.51130554, 0.51044135, 0.4981267 ,
       0.50806484, 0.50655252, 0.50395996, 0.50590438, 0.51022531,
       0.50806484, 0.49834275, 0.50957717, 0.51044135, 0.51303391,
       0.51000926, 0.50504019, 0.51433019, 0.51476228, 0.50093531,
       0.50676856, 0.50763275, 0.51260182, 0.49985507, 0.50720065,
       0.51303391, 0.51216972, 0.51195368, 0.50568833, 0.51216972,
       0.51281786, 0.51497833, 0.51368205, 0.50568833, 0.50504019,
       0.51368205, 0.50979321, 0.51087345, 0.50612042, 0.50806484])
```

In [34]:

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

```
-0.02088205440129043
```

# EVALUATION METRICS

In [35]:

```
from sklearn import metrics
```

In [36]:

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

```
Mean Absolute Error: 0.501432280822711
```

In [37]:

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

```
Mean Squared Error: 0.251545338204478
```

In [38]:

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

```
Root Mean Squared Error: 0.5015429574866723
```