Problem Statement ¶

Linear Regression

Import Libraries

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("placement.csv")
a
```

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

To display top 10 rows

In [3]:

```
c=a.head(15)
c
```

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
5	7.30	23	1
6	6.69	11	0
7	7.12	39	1
8	6.45	38	0
9	7.75	94	1
10	6.82	16	1
11	6.38	7	1
12	6.58	16	1
13	5.68	26	0
14	7.91	43	0

To find Missing values

In [4]:

```
c.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15 entries, 0 to 14
Data columns (total 3 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   cgpa                 15 non-null     float64
 1   placement_exam_marks  15 non-null     int64  
 2   placed               15 non-null     int64  
dtypes: float64(1), int64(2)
memory usage: 488.0 bytes
```

To display summary of statistics

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

To display column heading

In [6]:

```
a.columns
```

Out[6]:

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

Pairplot

In [7]:

```
s=a.dropna(axis=1)
s
```

Out[7]:

	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 [8]:

```
s.columns
```

Out[8]:

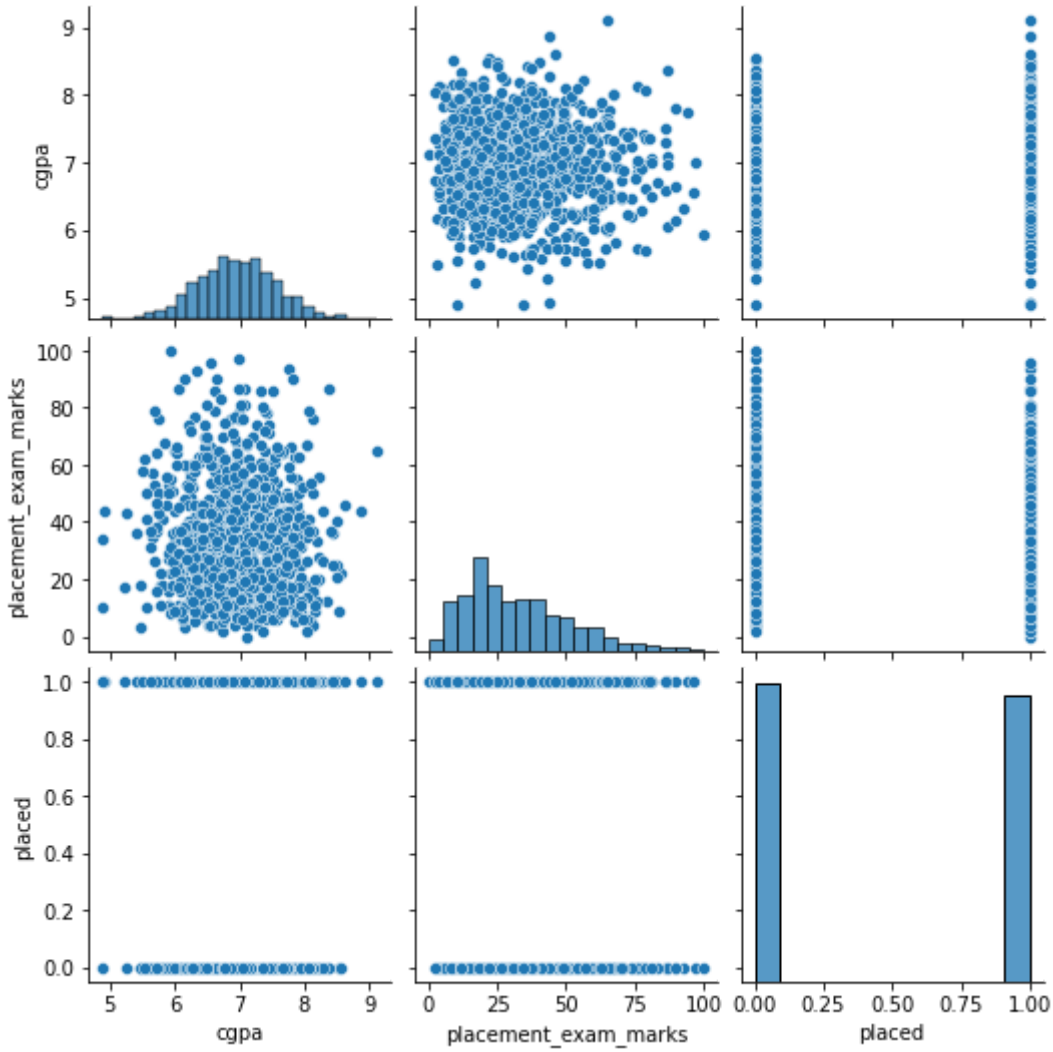
```
Index(['cgpa', 'placement_exam_marks', 'placed'], dtype='object')
```

In [9]:

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

Out[9]:

```
<seaborn.axisgrid.PairGrid at 0x1b66dbcb8b0>
```



Distribution Plot

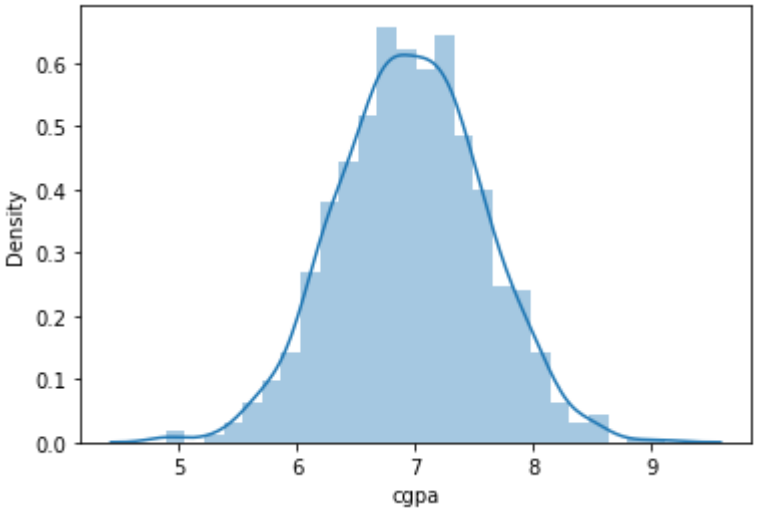
In [10]:

```
sns.distplot(a['cgpa'])
```

```
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[10]:

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



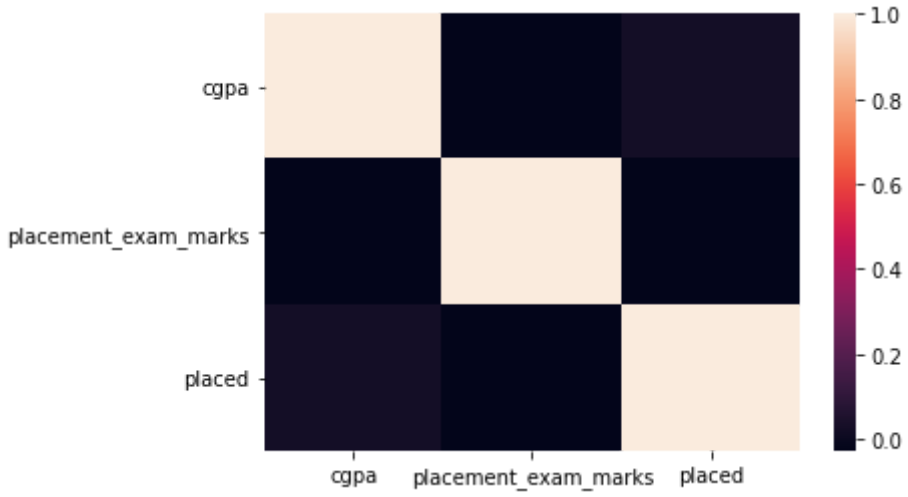
Correlation

In [11]:

```
b=s[['cgpa', 'placement_exam_marks', 'placed']]
sns.heatmap(b.corr())
```

Out[11]:

```
<AxesSubplot:>
```



Train the model - Model Building

In [12]:

```
g=s[['placement_exam_marks']]
h=s['placed']
```

To split dataset into training end test

In [13]:

```
from sklearn.model_selection import train_test_split
g_train,g_test,h_train,h_test=train_test_split(g,h,test_size=0.6)
```

To run the model

In [14]:

```
from sklearn.linear_model import LinearRegression
```

In [15]:

```
lr=LinearRegression()
lr.fit(g_train,h_train)
```

Out[15]:

```
LinearRegression()
```

In [16]:

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

```
0.4701124130322392
```

Coeffecient

In [17]:

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

Out[17]:

	Co-effecient
placement_exam_marks	0.000148

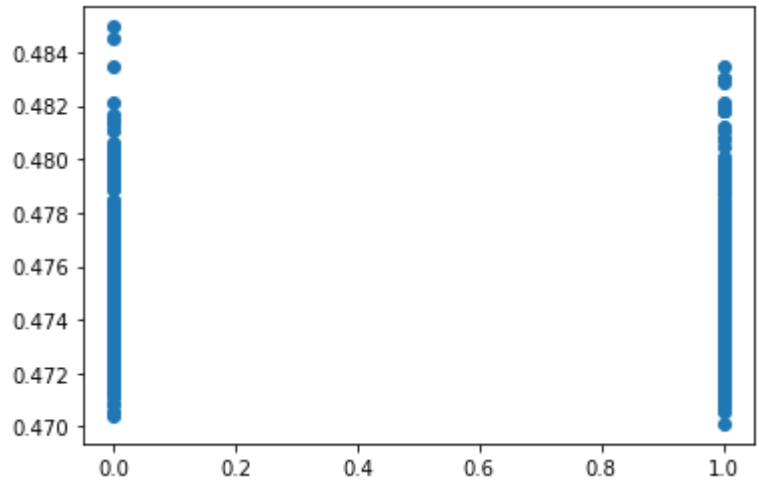
Best Fit line

In [18]:

```
prediction=lr.predict(g_test)
plt.scatter(h_test,prediction)
```

Out[18]:

```
<matplotlib.collections.PathCollection at 0x1b670394b50>
```



To find score

In [19]:

```
print(lr.score(g_test,h_test))
```

```
-0.0027563591066877002
```

Import Lasso and ridge

In [20]:

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

Ridge

In [21]:

```
ri=Ridge(alpha=5)
ri.fit(g_train,h_train)
```

Out[21]:

```
Ridge(alpha=5)
```

In [22]:

```
ri.score(g_test,h_test)
```

Out[22]:

```
-0.002756339138051178
```

In [23]:

```
ri.score(g_train,h_train)
```

Out[23]:

```
3.375219681300834e-05
```

Lasso

In [24]:

```
l=Lasso(alpha=6)
l.fit(g_train,h_train)
```

Out[24]:

```
Lasso(alpha=6)
```

In [25]:

```
l.score(g_test,h_test)
```

Out[25]:

```
-0.002177801975577376
```

In [27]:

```
ri.score(g_train,h_train)
```

Out[27]:

```
3.375219681300834e-05
```

ElasticNet

In [28]:

```
from sklearn.linear_model import ElasticNet
e=ElasticNet()
e.fit(g_train,h_train)
```

Out[28]:

```
ElasticNet()
```

Coeffecient,intercept

In [29]:

```
print(e.coef_)
```

```
[0.]
```

In [30]:

```
print(e.intercept_)
```

```
0.475
```

Prediction

In [31]:

```
d=e.predict(g_test)
d
```

Out[31]:

```
array([0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
       0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475, 0.475,
```

In [32]:

```
print(e.score(g_test,h_test))
```

```
-0.002178019795473576
```

Evaluation

In [33]:

```
from sklearn import metrics
print("Mean Absolute error:",metrics.mean_absolute_error(h_test,d))
```

```
Mean Absolute error: 0.4999166666666667
```

In [34]:

```
print("Mean Squared error:",metrics.mean_squared_error(h_test,d))
```

```
Mean Squared error: 0.2505416666666666
```

In [35]:

```
print("Mean Squared error:",np.sqrt(metrics.mean_squared_error(h_test,d)))
```

```
Mean Squared error: 0.5005413735813121
```

In [ ]: