LinearRegression

In [1]:

```
import numpy as np
import pandas as pd
```

data collection

In [2]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as pp
import seaborn as sb
```

In [3]:

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

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
...	...	...	...
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

first 10 rows

In [4]:

```
df.head(10)
```

Out[4]:

	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

In [5]:

```
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 [6]:

```
df.describe()
```

Out[6]:

	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 [7]:

```
df.columns
```

Out[7]:

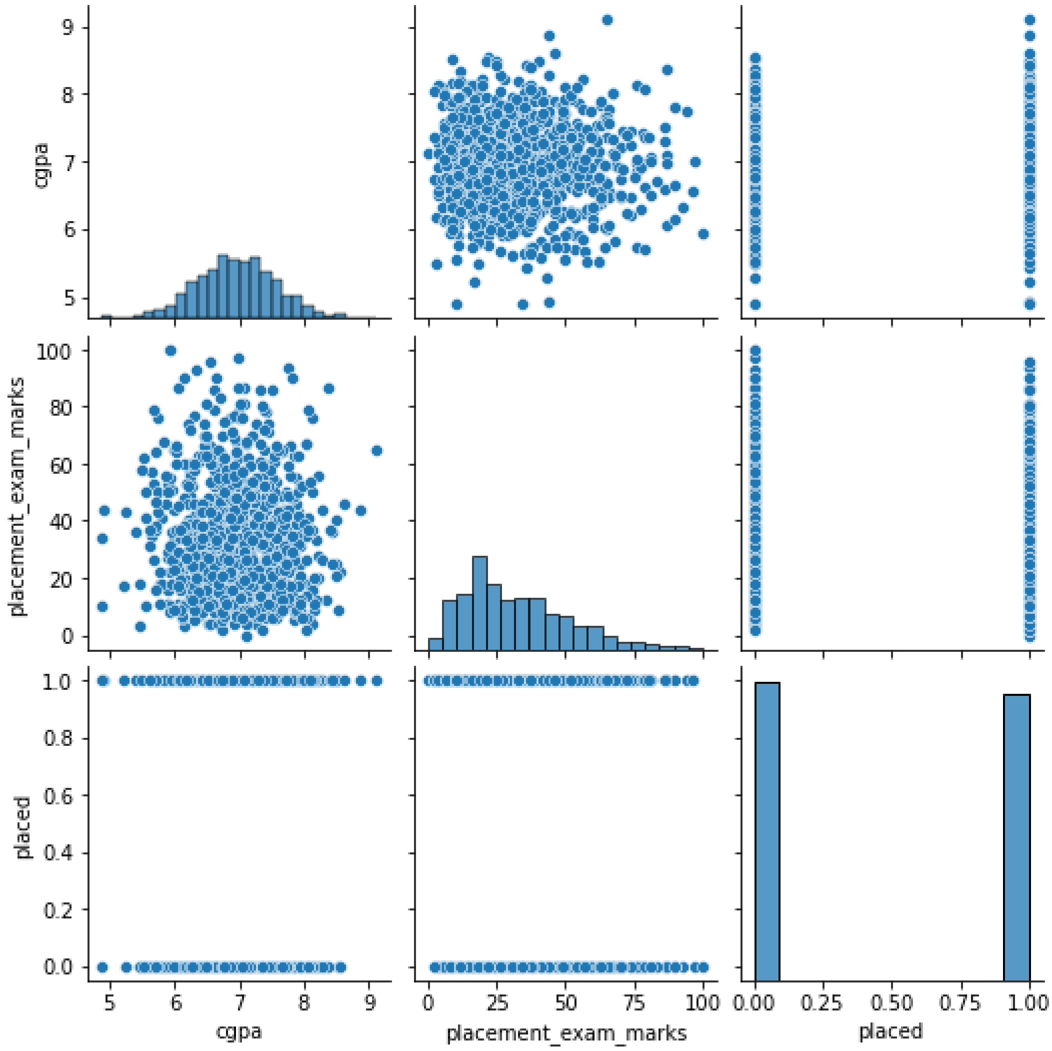
```
Index(['cgpa', 'placement_exam_marks', 'placed'], dtype='object')
```

In [8]:

```
sb.pairplot(df)
```

Out[8]:

```
<seaborn.axisgrid.PairGrid at 0x214a22a0640>
```



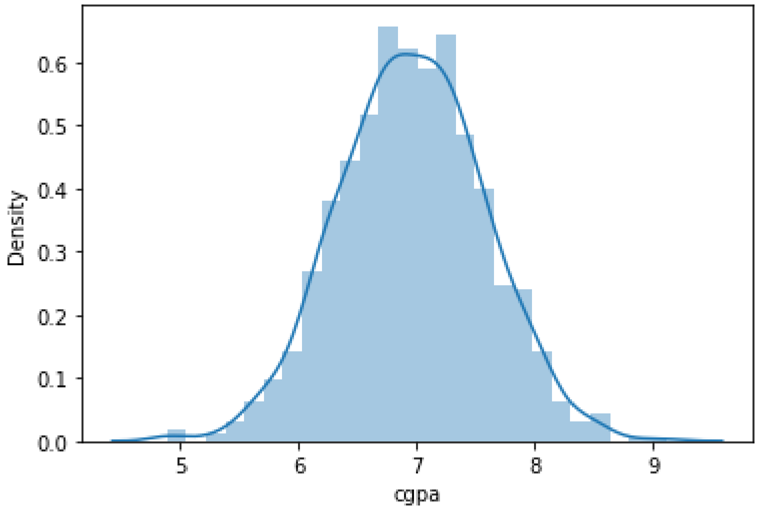
In [9]:

```
sb.distplot(df["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[9]:

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



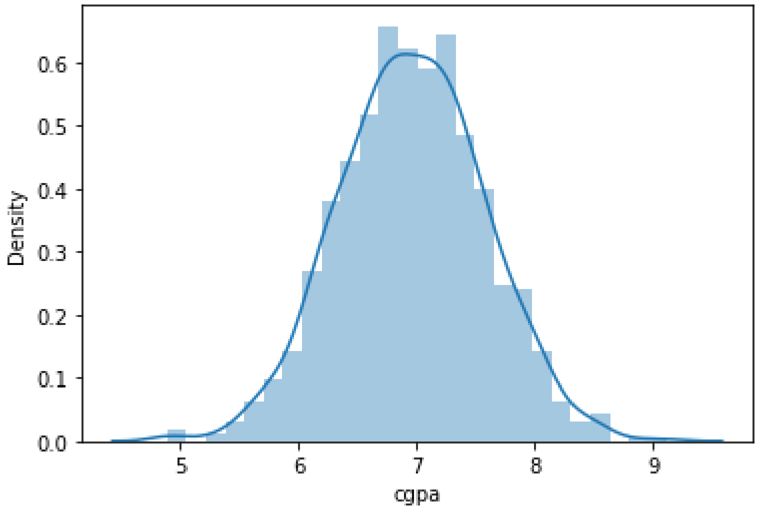
In [10]:

```
sb.distplot(df["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'>
```



In [11]:

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

Out[11]:

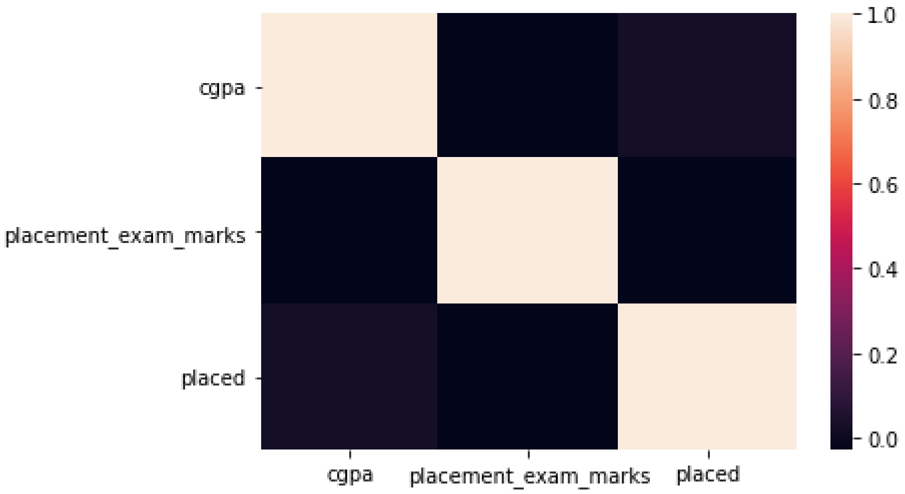
```
     cgpa  placement_exam_marks  placed
  0  7.19                  26.0       1
  1  7.46                  38.0       1
  2  7.54                  40.0       1
  3  6.42                   8.0       1
  4  7.23                  17.0       0
...   ...                   ...     ...
995  8.87       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

In [12]:

```
sb.heatmap(df1.corr())
```

Out[12]:

```
<AxesSubplot:>
```



model building

In [13]:

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

In [14]:

```
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)
```

linear regression

In [15]:

```
from sklearn.linear_model import LinearRegression

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

Out[15]:

```
LinearRegression()
```

In [16]:

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

```
1.7763568394002505e-15
```

In [17]:

```
coef = pd.DataFrame(lr.coef_,x.columns,columns=['Co_efficient'])
coef
```

Out[17]:

	Co_efficient
cgpa	1.000000e+00
placement_exam_marks	-6.555029e-17
placed	7.889805e-18

In [18]:

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

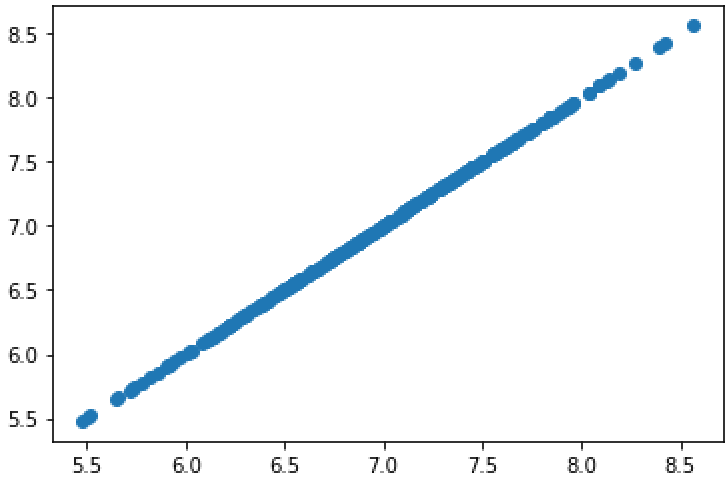
```
1.0
```

In [19]:

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

Out[19]:

```
<matplotlib.collections.PathCollection at 0x214a88c4340>
```



lasso and ridge regression

In [20]:

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

Out[20]:

```
1.0
```

In [21]:

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

Out[21]:

```
1.0
```

In [22]:

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

In [23]:

```
r = Ridge(alpha=10)
r.fit(x_train,y_train)
r.score(x_test,y_test)
r.score(x_train,y_train)
```

Out[23]:

```
0.9987018452597064
```

In [24]:

```
l = Lasso(alpha=10)
l.fit(x_train,y_train)
l.score(x_test,y_test)
l.score(x_train,y_train)
```

Out[24]:

```
0.0
```

elasticnet

In [25]:

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

Out[25]:

```
ElasticNet()
```

In [26]:

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

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

In [27]:

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

```
6.951371428571429
```

In [28]:

```
predictions = e.predict(x_test)
predictions
```

```
Out[28]:
```

```
array([6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
       6.95137143, 6.95137143, 6.95137143, 6.95137143, 6.95137143,
```

In [29]:

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

```
-0.0029228809314267858
```

In [30]:

```
from sklearn import metrics
```

mean absolute error

In [31]:

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

```
Mean Absolute Error: 0.5001177142857143
```

mean squared error

In [32]:

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

```
Mean Squared Error: 0.3712978812244898
```

root mean squared error

In [33]:

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

```
Root Mean Squared Error 0.6093421709325055
```

In [ ]: