Problem Statement

A real estate agent want help to predict the house price for regions in USA.He gave us the dataset to work on to use linear regression model.Create a model that helps him to estimate of what the house would sell for

Import libraries

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

In [2]:
```
# To import dataset
df=pd.read_csv('placement csv')
df
```

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]:
```
# To display top 10 rows
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]:
```
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]:
```
df.columns
```

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

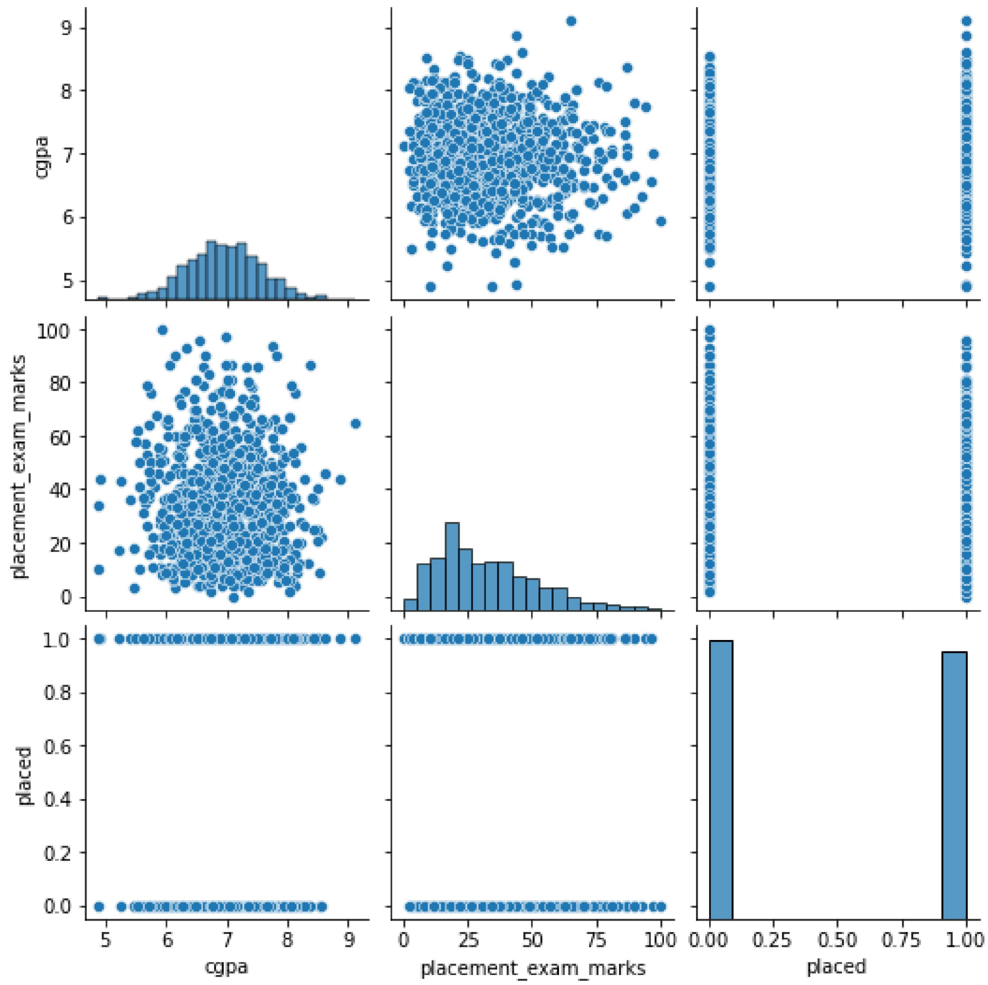
In [7]:
```
a = df.dropna(axis='columns')
a.columns
```

Out[7]:
```
Index(['cgpa', 'placement_exam_marks', 'placed'], dtype='object')
```

EDA and Visualization

In [8]:
```
sns.pairplot(a)
```

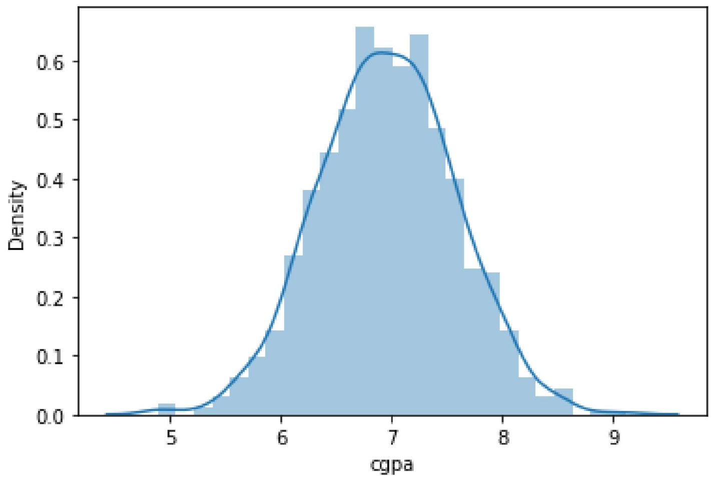
Out[8]:
```
<seaborn.axisgrid.PairGrid at 0x1a58f0471f0>
```



In [9]:
```
sns.distplot(a['cgpa'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarnin
g: `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 flexibil
ity) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

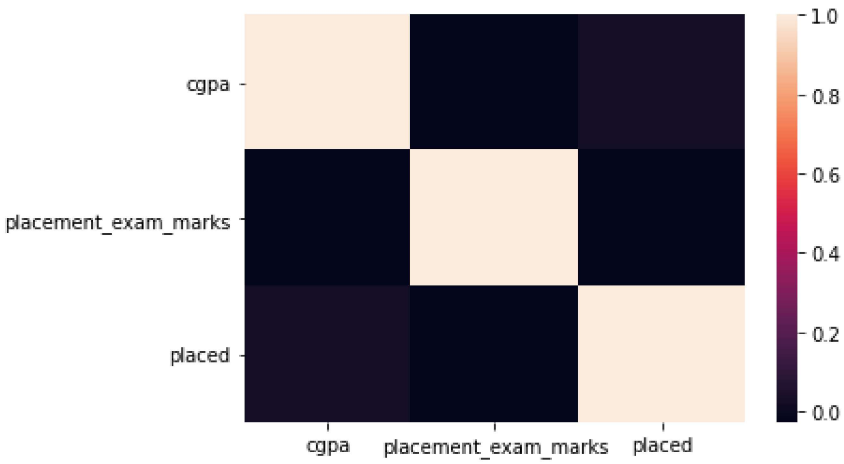
Out[9]: <AxesSubplot:xlabel='cgpa', ylabel='Density'>



In [10]:
```
a1=a[['cgpa', 'placement_exam_marks', 'placed']]
```

In [11]:
```
sns.heatmap(a1.corr())
```

Out[11]: <AxesSubplot:>



To Train the Model - Model Building

We are going to train Linear Regression model;We need to split out data into two variables x and y where x is independent variable (input) and y is dependent on x(output). We could ignore address column as it is not required for our model.

In [12]:
```
x=a1[[ 'placement_exam_marks', 'placed']]
y=a1['cgpa']
```

To split my dataset into training and test data

In [13]:
```
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 [14]:
```
from sklearn.linear_model import LinearRegression

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

Out[14]:
```
LinearRegression()
```

In [15]:
```
print(lr.intercept_)
```

```
7.001457794147576
```

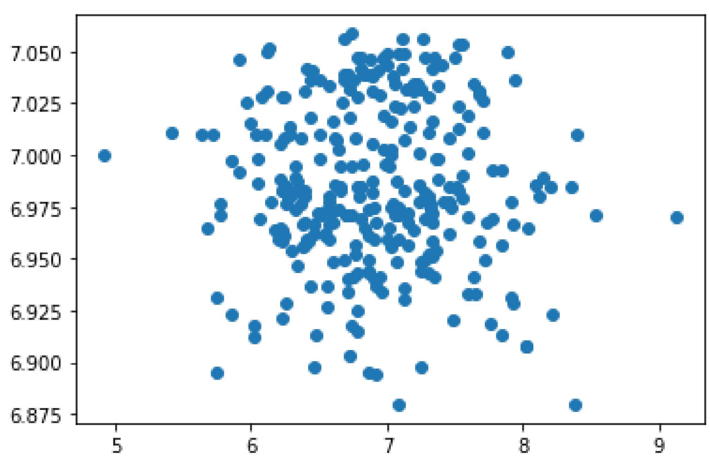
In [16]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[16]:

	Co-efficient
placement_exam_marks	-0.001399
placed	0.059951

In [17]:
```
prediction=lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[17]:
```
<matplotlib.collections.PathCollection at 0x1a5918019d0>
```



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

```
-0.027123722238079795
```

In [23]:
```
from sklearn.linear_model import ElasticNet
en = ElasticNet()
en.fit(x_train,y_train)
```

Out[23]:
```
ElasticNet()
```

In [24]:
```
print(en.coef_)
```

```
[-0.0001154  0.       ]
```

In [25]:
```
print(en.intercept_)
```

```
6.9896127652163065
```

In [27]:
```
print(en.predict(x_test))
```

```
[6.98753558 6.9862662  6.98707399 6.98718939 6.98672779 6.98857417
 6.98684319 6.98730479 6.98315042 6.98684319 6.98661239 6.98291963
 6.98684319 6.9854584  6.98799718 6.9860354  6.98649699 6.9861508
 6.98418901 6.98361202 6.98384282 6.98718939 6.985343   6.98915117
 6.98788178 6.98811258 6.98257343 6.98776638 6.98465061 6.9861508
 6.985343   6.98742019 6.98661239 6.98799718 6.98268883 6.98257343
 6.985343   6.9855738  6.98788178 6.98868957 6.98268883 6.985343
 6.98522761 6.9854584  6.98592    6.98141944 6.98430441 6.97957306
 6.9860354  6.98268883 6.98234263 6.98592    6.9858046  6.98661239
 6.97853447 6.98776638 6.98753558 6.98465061 6.98291963 6.98661239
 6.98476601 6.98938197 6.98695859 6.98418901 6.985343   6.98707399
 6.9861508  6.98453521 6.985343   6.98684319 6.98822798 6.98811258
 6.98915117 6.98661239 6.98107324 6.98476601 6.98338122 6.98522761
 6.98361202 6.98592    6.98522761 6.98834338 6.9861508  6.98511221
 6.98938197 6.9856892  6.98372742 6.9862662  6.98742019 6.98672779
 6.98753558 6.98430441 6.98765098 6.9860354  6.98315042 6.98857417
 6.98799718 6.98188104 6.98441981 6.98384282 6.98441981 6.98753558
 6.98730479 6.98488141 6.98742019 6.98742019 6.98845878 6.98857417
 6.98672779 6.98153484 6.9861508  6.98765098 6.98799718 6.98765098
 6.98845878 6.98465061 6.98857417 6.98788178 6.98822798 6.98407362
 6.98903577 6.98753558 6.98880497 6.98753558 6.98488141 6.98776638
 6.98222723 6.98453521 6.98868957 6.98522761 6.98718939 6.98707399
 6.98592    6.98326582 6.98072705 6.98730479 6.98026545 6.98811258
 6.98788178 6.98072705 6.97957306 6.98718939 6.98638159 6.98511221
 6.98845878 6.98811258 6.98845878 6.98130404 6.98649699 6.98684319
 6.98395822 6.9860354  6.9855738  6.98672779 6.98661239 6.98384282
 6.98199643 6.98834338 6.98765098 6.98684319 6.98811258 6.98788178
 6.98338122 6.98234263 6.98188104 6.98776638 6.98349662 6.98707399
 6.98903577 6.98799718 6.98857417 6.98095784 6.98684319 6.98338122
 6.985343   6.98245803 6.98845878 6.98084245 6.98730479 6.98430441
 6.98707399 6.98707399 6.98522761 6.98776638 6.98476601 6.98695859
 6.98361202 6.98407362 6.98822798 6.98592    6.9861508  6.98476601
 6.98742019 6.98465061 6.98742019 6.98707399 6.98361202 6.98718939
 6.98753558 6.98811258 6.98707399 6.98522761 6.98695859 6.98499681
 6.98407362 6.98799718 6.98695859 6.98315042 6.98718939 6.9860354
 6.98453521 6.98649699 6.98811258 6.98522761 6.98811258 6.98661239
 6.98280423 6.98649699 6.98211183 6.985343   6.98707399 6.98707399
 6.9861508  6.98765098 6.98822798 6.98326582 6.98441981 6.98638159
 6.98303503 6.98753558 6.98268883 6.98326582 6.98465061 6.98211183
 6.98418901 6.98453521 6.98188104 6.98084245 6.98834338 6.9856892
 6.98845878 6.98268883 6.98465061 6.98892037 6.9861508  6.98730479
 6.98845878 6.98672779 6.98892037 6.98592    6.98776638 6.98799718
 6.9854584  6.98742019 6.985343   6.98684319 6.9858046  6.98707399
 6.98326582 6.98845878 6.98499681 6.98291963 6.98476601 6.98834338
 6.98338122 6.98695859 6.98395822 6.98707399 6.98718939 6.9856892
 6.98684319 6.98776638 6.98649699 6.98903577 6.98395822 6.98730479
 6.98742019 6.98257343 6.98326582 6.98395822 6.98799718 6.9860354
 6.98476601 6.98107324 6.98684319 6.98234263 6.9854584  6.98430441
 6.98730479 6.98511221 6.9858046  6.98788178 6.98915117 6.98707399
 6.98868957 6.985343   6.985343   6.98638159 6.98153484 6.9855738 ]
```

In [28]:
```
print(en.score(x_test,y_test))
```

```
-0.018759520840909927
```

Evaluation Metrics

In [29]:
```
from sklearn import metrics
print("Mean Absolytre Error:",metrics.mean_absolute_error(y_test,prediction))
print("Mean Squared Error:",metrics.mean_squared_error(y_test,prediction))
print("Root Mean Squared Error:",np.sqrt(metrics.mean_squared_error(y_test,prediction)))
```

```
Mean Absolytre Error: 0.4846568836403954
Mean Squared Error: 0.37318670968221296
Root Mean Squared Error: 0.610890096238442
```