Graduate Admission

Importing libraries

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
```

Loading the dataset

In [2]:

```
df= pd.read_csv("Admission_Predict.csv")
df
```

Out[2]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	1	337	118	4	4.5	4.5	9.65	1	0.92
1	2	324	107	4	4.0	4.5	8.87	1	0.76
2	3	316	104	3	3.0	3.5	8.00	1	0.72
3	4	322	110	3	3.5	2.5	8.67	1	0.80
4	5	314	103	2	2.0	3.0	8.21	0	0.65
...	...	...	...	...	...	...	...	...	...
395	396	324	110	3	3.5	3.5	9.04	1	0.82
396	397	325	107	3	3.0	3.5	9.11	1	0.84
397	398	330	116	4	5.0	4.5	9.45	1	0.91
398	399	312	103	3	3.5	4.0	8.78	0	0.67
399	400	333	117	4	5.0	4.0	9.66	1	0.95

400 rows × 9 columns

In [11]:

```
df.shape
```

Out[11]:

```
(400, 9)
```

In [8]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Serial No.         400 non-null    int64  
 1   GRE Score          400 non-null    int64  
 2   TOEFL Score        400 non-null    int64  
 3   University Rating  400 non-null    int64  
 4   SOP                400 non-null    float64
 5   LOR                400 non-null    float64
 6   CGPA               400 non-null    float64
 7   Research           400 non-null    int64  
 8   Chance of Admit    400 non-null    float64
dtypes: float64(4), int64(5)
memory usage: 28.2 KB
```

In [12]:

```
df.columns
```

Out[12]:

```
Index(['Serial No.', 'GRE Score', 'TOEFL Score', 'University Rating', 'SOP',
       'LOR ', 'CGPA', 'Research', 'Chance of Admit '],
      dtype='object')
```

In [13]:

```
df.dtypes
```

Out[13]:

```
Serial No.             int64
GRE Score              int64
TOEFL Score            int64
University Rating      int64
SOP                  float64
LOR                  float64
CGPA                 float64
Research               int64
Chance of Admit      float64
dtype: object
```

Checking null values

In [3]:

```
df.isnull().sum()
```

Out[3]:

```
Serial No.           0
GRE Score            0
TOEFL Score          0
University Rating    0
SOP                  0
LOR                  0
CGPA                 0
Research             0
Chance of Admit      0
dtype: int64
```

=> From the above results we can see that there are no null numbers in the dataframe

In [4]:

```
# Checking numerical features:

df.drop(columns=['Serial No.'], inplace= True)
df
```

Out[4]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	337	118	4	4.5	4.5	9.65	1	0.92
1	324	107	4	4.0	4.5	8.87	1	0.76
2	316	104	3	3.0	3.5	8.00	1	0.72
3	322	110	3	3.5	2.5	8.67	1	0.80
4	314	103	2	2.0	3.0	8.21	0	0.65
...	...	...	...	...	...	...	...	...
395	324	110	3	3.5	3.5	9.04	1	0.82
396	325	107	3	3.0	3.5	9.11	1	0.84
397	330	116	4	5.0	4.5	9.45	1	0.91
398	312	103	3	3.5	4.0	8.78	0	0.67
399	333	117	4	5.0	4.0	9.66	1	0.95

400 rows × 8 columns

In [5]:

```
df.describe()
```

Out[5]:

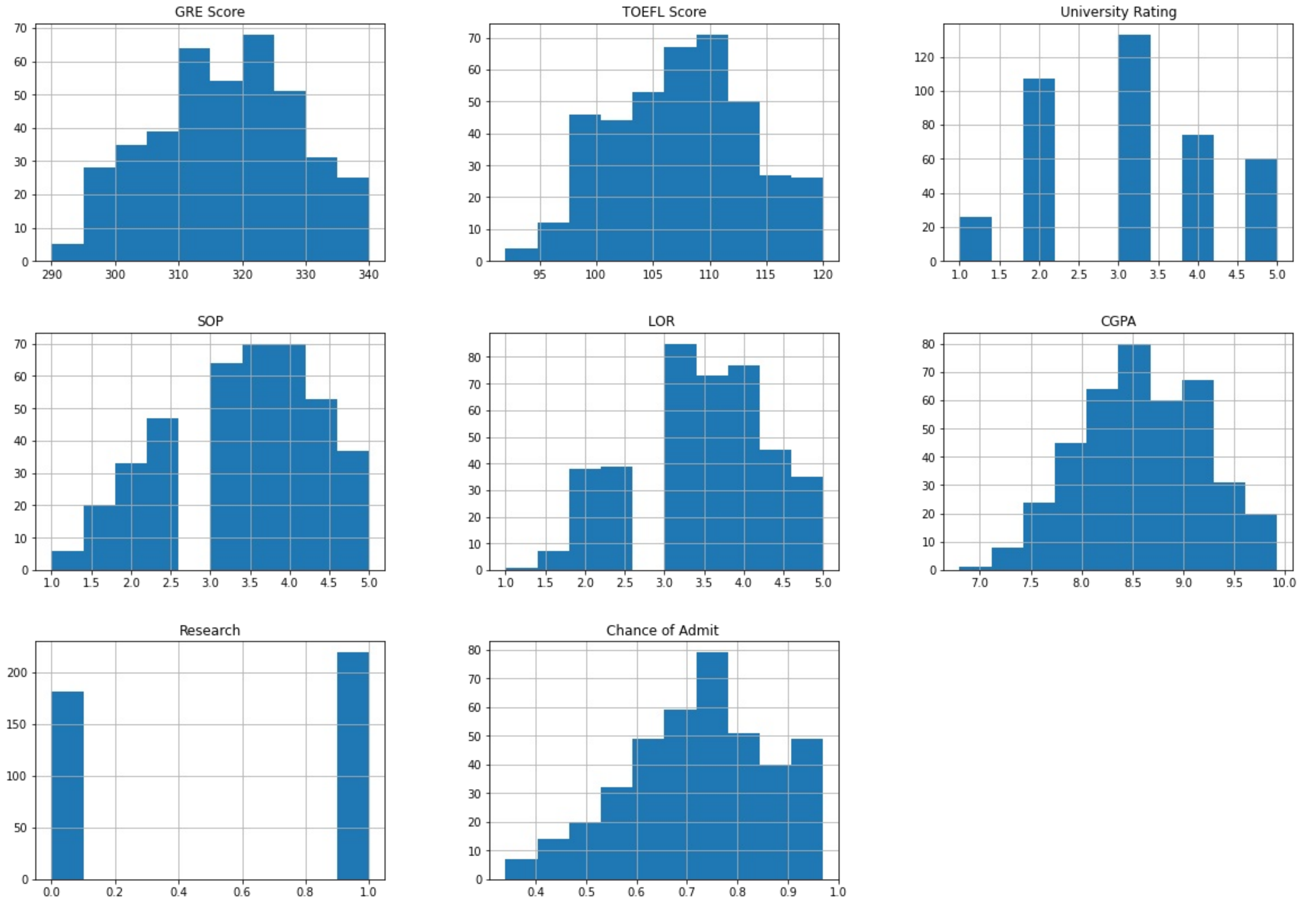
	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
count	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000
mean	316.807500	107.410000	3.087500	3.400000	3.452500	8.598925	0.547500	0.724350
std	11.473646	6.069514	1.143728	1.006869	0.898478	0.596317	0.498362	0.142609
min	290.000000	92.000000	1.000000	1.000000	1.000000	6.800000	0.000000	0.340000
25%	308.000000	103.000000	2.000000	2.500000	3.000000	8.170000	0.000000	0.640000
50%	317.000000	107.000000	3.000000	3.500000	3.500000	8.610000	1.000000	0.730000
75%	325.000000	112.000000	4.000000	4.000000	4.000000	9.062500	1.000000	0.830000
max	340.000000	120.000000	5.000000	5.000000	5.000000	9.920000	1.000000	0.970000

Explanatory Data Analysis (EDA)

Histogram

In [8]:

```
df.hist(figsize=(20,14))
plt.show()
```

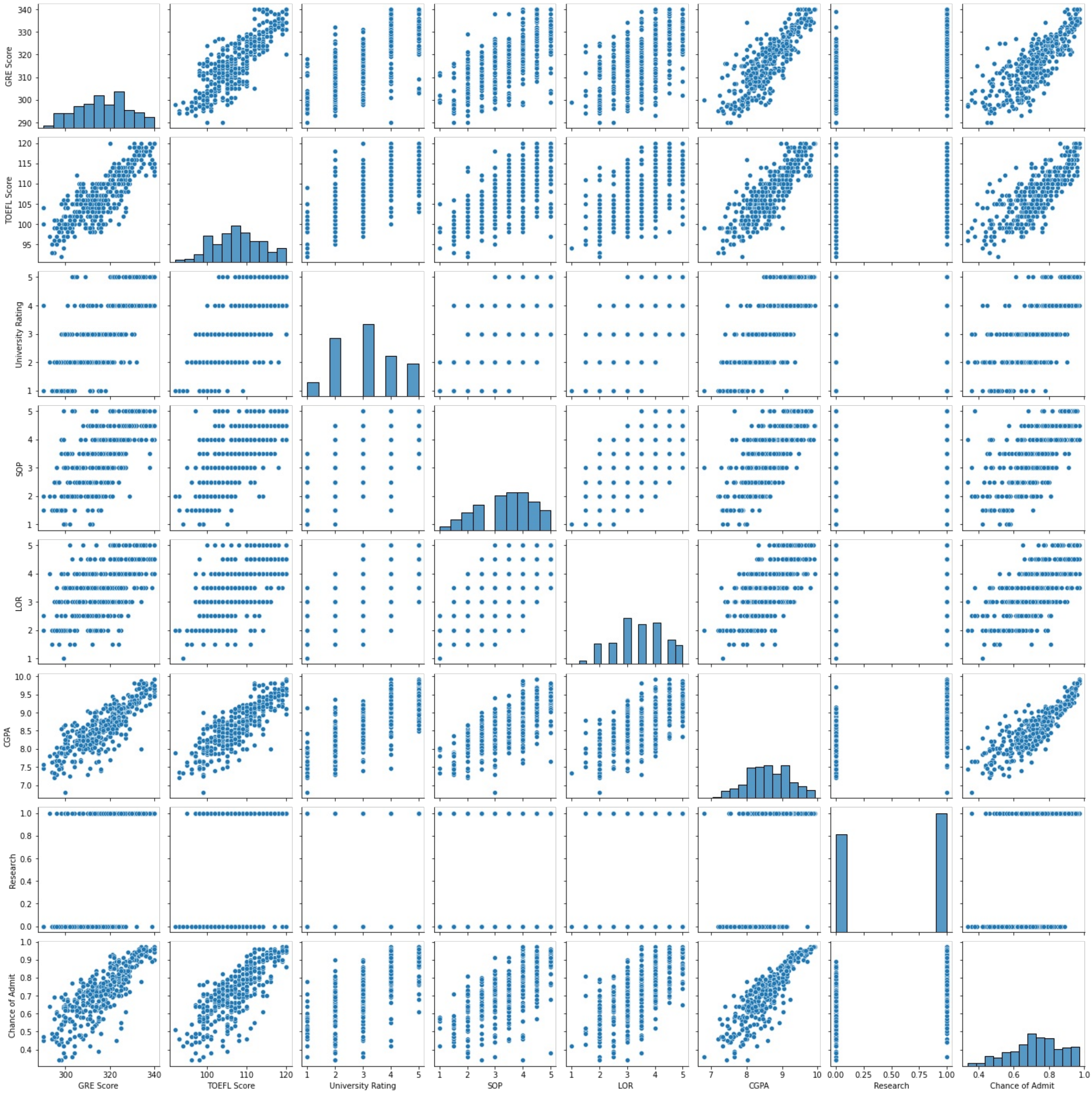


Pairplot

A pair plot, also known as a scatter plot matrix, is a visualization technique that allows you to examine the relationships between pairs of variables in a dataset. It provides a grid of scatter plots where each variable is compared with every other variable, resulting in a comprehensive overview of the pairwise relationships.

In [10]:

```
sns.pairplot(df)
plt.show()
```



In [12]:

```
corr_matrix=df.corr()
#calculate the correlation
```

Heatmap

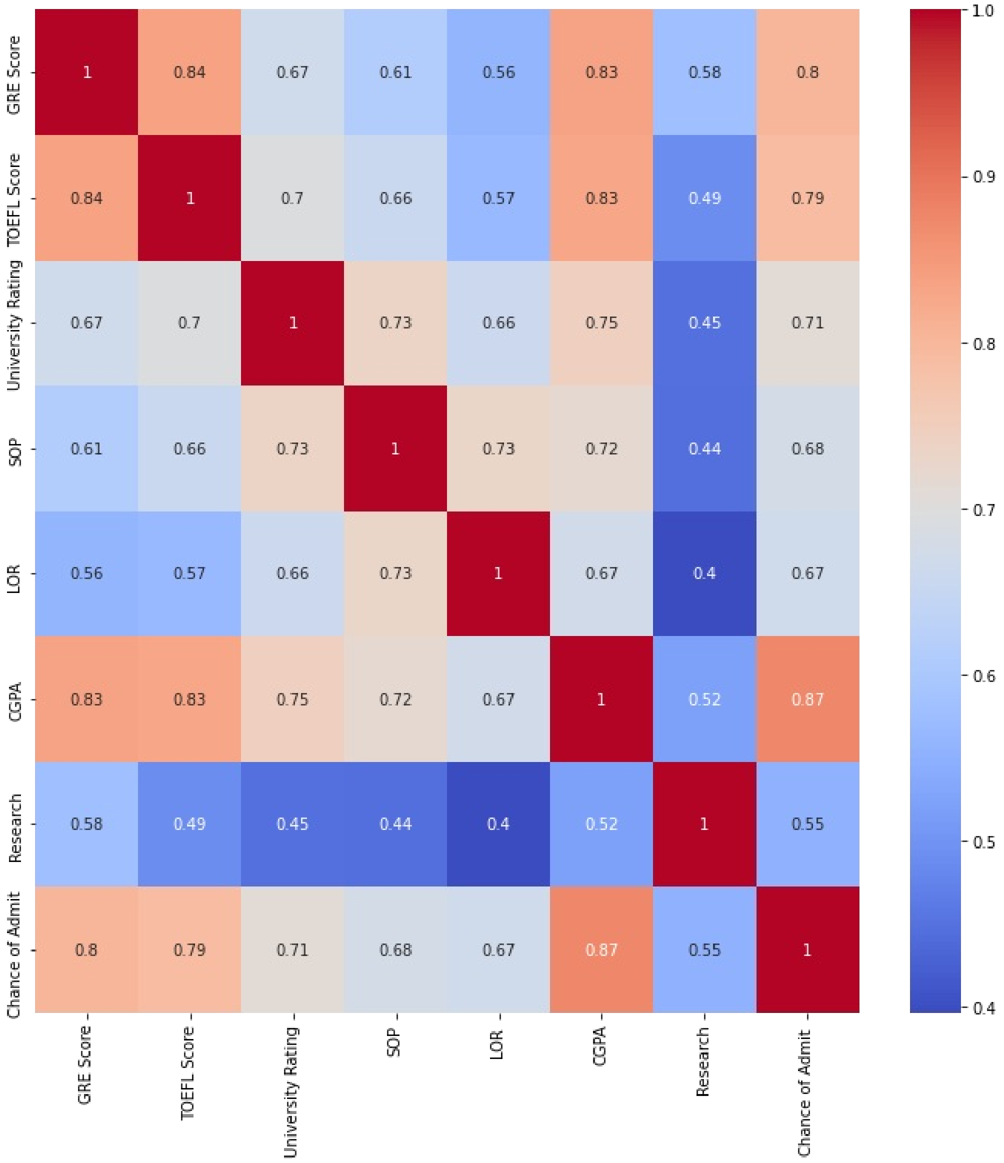
A heatmap is a graphical representation of data in the form of a color-coded matrix. It is commonly used to visualize the correlation matrix : It provide a visual summary of the correlation structure of your data

In [23]:

```
plt.figure(figsize=(12,12))
sns.heatmap(corr_matrix, annot=True , cmap="coolwarm")
```

Out[23]:

```
<AxesSubplot:>
```



	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
GRE Score	1	0.84	0.67	0.61	0.56	0.83	0.58	0.8
TOEFL Score	0.84	1	0.7	0.66	0.57	0.83	0.49	0.79
University Rating	0.67	0.7	1	0.73	0.66	0.75	0.45	0.71
SOP	0.61	0.66	0.73	1	0.73	0.72	0.44	0.68
LOR	0.56	0.57	0.66	0.73	1	0.67	0.4	0.67
CGPA	0.83	0.83	0.75	0.72	0.67	1	0.52	0.87
Research	0.58	0.49	0.45	0.44	0.4	0.52	1	0.55
Chance of Admit	0.8	0.79	0.71	0.68	0.67	0.87	0.55	1

Trainig and testing the data

In [29]:

```
x= df.drop('Chance of Admit ', axis=1)
y=df['Chance of Admit ']
```

In [32]:

```
from sklearn.model_selection import train_test_split
x_train,x_test, y_train, y_test=train_test_split(x,y,test_size=0.2, random_state =1)
```

In [33]:

```
x_train
```

Out[33]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
93	301	97	2	3.0	3.0	7.88	1
23	334	119	5	5.0	4.5	9.70	1
299	305	112	3	3.0	3.5	8.65	0
13	307	109	3	4.0	3.0	8.00	1
90	318	106	2	4.0	4.0	7.92	1
...	...	...	...	...	...	...	...
255	307	110	4	4.0	4.5	8.37	0
72	321	111	5	5.0	5.0	9.45	1
396	325	107	3	3.0	3.5	9.11	1
235	326	111	5	4.5	4.0	9.23	1
37	300	105	1	1.0	2.0	7.80	0

320 rows × 7 columns

In [38]:

```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
```

In [41]:

```
x_train_scaled= scaler.fit_transform(x_train)
x_test_scaled= scaler.fit_transform(x_test)
```

In [42]:

```
x_train_scaled
```

Out[42]:

```
array([[0.22      , 0.17857143, 0.25      , ..., 0.42857143, 0.25      ,
        1.        ],
       [0.88      , 0.96428571, 1.        , ..., 0.85714286, 0.91911765,
        1.        ],
       [0.3       , 0.71428571, 0.5       , ..., 0.57142857, 0.53308824,
        0.        ],
       ...,
       [0.7       , 0.53571429, 0.5       , ..., 0.57142857, 0.70220588,
        1.        ],
       [0.72      , 0.67857143, 1.        , ..., 0.71428571, 0.74632353,
        1.        ],
       [0.2       , 0.46428571, 0.        , ..., 0.14285714, 0.22058824,
        0.        ]])
```

In [47]:

```
import tensorflow
from tensorflow import  keras
from keras import Sequential
from keras.layers import Dense
```

In [48]:

```
#creating a neural network model using the Keras library with a Sequential API.
model = Sequential()
model.add(Dense(7, activation='relu', input_dim=7))#The activation='relu' parameter sets the activation functio
model.add(Dense(1, activation='linear'))#The activation function for this layer is set to linear, which means t
```

In [50]:

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense (Dense)               (None, 7)                 56        
                                                                 
 dense_1 (Dense)             (None, 1)                 8         
                                                                 
=================================================================
Total params: 64
Trainable params: 64
Non-trainable params: 0
_________________________________________________________________
```

In [51]:

```
model.compile(loss='mean_squared_error', optimizer='adam')
```

In [57]:

```
model.fit(x_train_scaled ,y_train, epochs=10, batch_size=32)
```

```
Epoch 1/10
10/10 [==============================] - 0s 2ms/step - loss: 0.0137
Epoch 2/10
10/10 [==============================] - 0s 2ms/step - loss: 0.0128
Epoch 3/10
10/10 [==============================] - 0s 2ms/step - loss: 0.0119
Epoch 4/10
10/10 [==============================] - 0s 2ms/step - loss: 0.0112
Epoch 5/10
10/10 [==============================] - 0s 2ms/step - loss: 0.0107
Epoch 6/10
10/10 [==============================] - 0s 2ms/step - loss: 0.0101
Epoch 7/10
10/10 [==============================] - 0s 2ms/step - loss: 0.0097
Epoch 8/10
10/10 [==============================] - 0s 2ms/step - loss: 0.0093
Epoch 9/10
10/10 [==============================] - 0s 1ms/step - loss: 0.0089
Epoch 10/10
10/10 [==============================] - 0s 845us/step - loss: 0.0086
```

Out[57]:
```
<keras.callbacks.History at 0x15f5dafa4c0>
```

In [ ]:
```python
model.predict(x_test_scaled)
```

In [59]:
```python
y_predict=model.predict(x_test_scaled)
```

```
3/3 [==============================] - 0s 2ms/step
```

In [60]:
```python
from sklearn.metrics import r2_score
r2_score(y_test, y_predict)
```

Out[60]:
```
0.622914304892979
```

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js