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

```
df=pd.read_csv('Admission_Predict_Ver1.1.csv')
```

1. Define Problem Statement and perform Exploratory Data Analysis

a. Definition of problem

Jamboree has helped thousands of students to join in top colleges abroad. Be it GMAT, GRE or SAT, their unique problem-solving methods ensure maximum scores with minimum effort.

New feature launched where students/ learners can come to their website and check their probability of getting into the IVY league college.

Here our analysis will help Jamboree in understanding what factors are important in graduate admissions and how these factors are interrelated among themselves, by using various Data analytics & ML tools and libraries.

```
df.head()
```

	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

```
df.rename(columns={'Chance of Admit ': 'Chance of Admit','LOR ':'LOR'}, inplace=True)
```

b. Observations on shape of data, data types of all the attributes, conversion of categorical attributes to 'category' (If required), missing value detection, statistical summary.

```
df.shape
```

```
(500, 9)
```

```
df.ndim
```

```
2
```

```
df.dtypes
```

```
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
```

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

```
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
```

```
df.describe().T
```

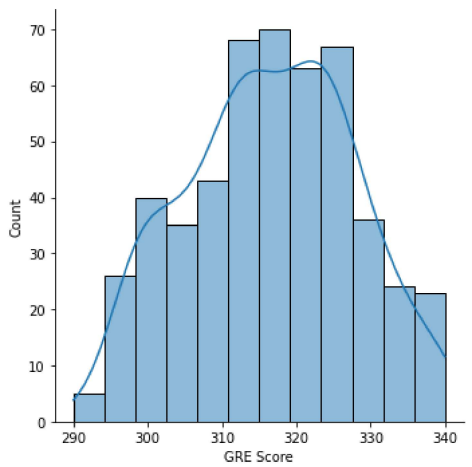
	count	mean	std	min	25%	50%	75%	max
Serial No.	500.0	250.50000	144.481833	1.00	125.7500	250.50	375.25	500.00
GRE Score	500.0	316.47200	11.295148	290.00	308.0000	317.00	325.00	340.00
TOEFL Score	500.0	107.19200	6.081868	92.00	103.0000	107.00	112.00	120.00
University Rating	500.0	3.11400	1.143512	1.00	2.0000	3.00	4.00	5.00
SOP	500.0	3.37400	0.991004	1.00	2.5000	3.50	4.00	5.00
LOR	500.0	3.48400	0.925450	1.00	3.0000	3.50	4.00	5.00
CGPA	500.0	8.57644	0.604813	6.80	8.1275	8.56	9.04	9.92
Research	500.0	0.56000	0.496884	0.00	0.0000	1.00	1.00	1.00
Chance of Admit	500.0	0.72174	0.141140	0.34	0.6300	0.72	0.82	0.97

```
df.info()
```

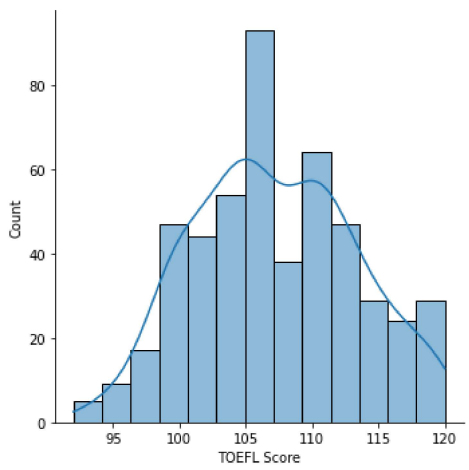
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Serial No.         500 non-null    int64  
 1   GRE Score          500 non-null    int64  
 2   TOEFL Score        500 non-null    int64  
 3   University Rating  500 non-null    int64  
 4   SOP                500 non-null    float64
 5   LOR                500 non-null    float64
 6   CGPA               500 non-null    float64
 7   Research           500 non-null    int64  
 8   Chance of Admit    500 non-null    float64
dtypes: float64(4), int64(5)
memory usage: 35.3 KB
```

c. Univariate Analysis (distribution plots of all the continuous variable(s) barplots/countplots of all the categorical variables)

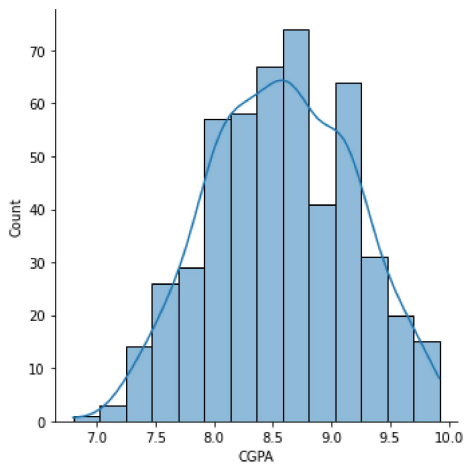
```
#plt.figure(figsize=(4,4))
#plt.title('', fontsize=12)
sns.displot(data=df,x='GRE Score',kde=True)
plt.show()
```



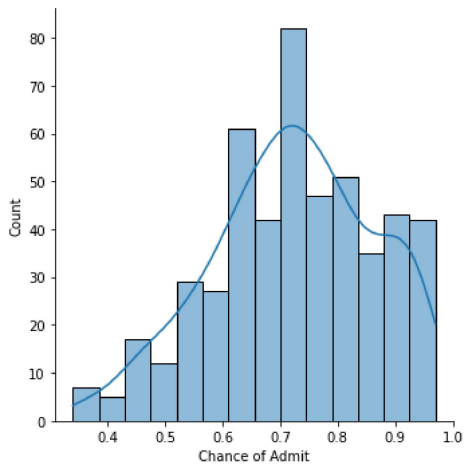

```
sns.displot(data=df,x='TOEFL Score',kde=True)
plt.show()
```



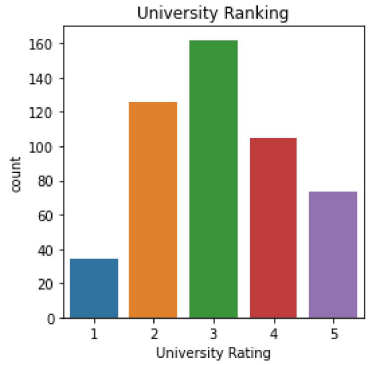

```
sns.displot(data=df,x='CGPA',kde=True)
plt.show()
```



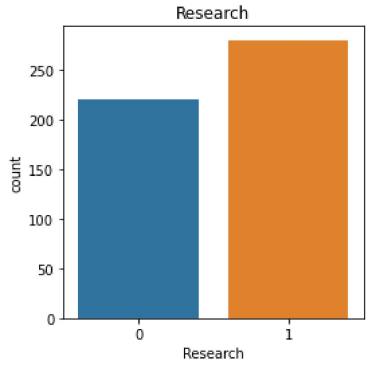

```
sns.displot(data=df,x='Chance of Admit',kde=True)
plt.show()
```




```
plt.figure(figsize=(4,4))
plt.title('University Ranking', fontsize=12)
sns.countplot(data=df,x='University Rating')
plt.show()
```

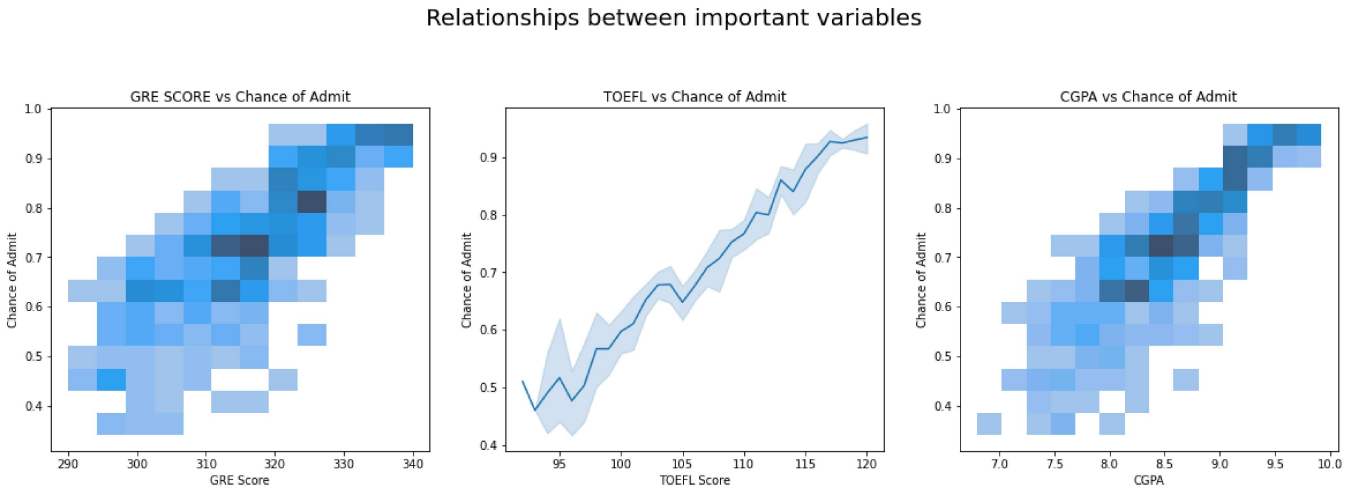



```
plt.figure(figsize=(4,4))
plt.title('Research', fontsize=12)
sns.countplot(data=df,x='Research')
plt.show()
```



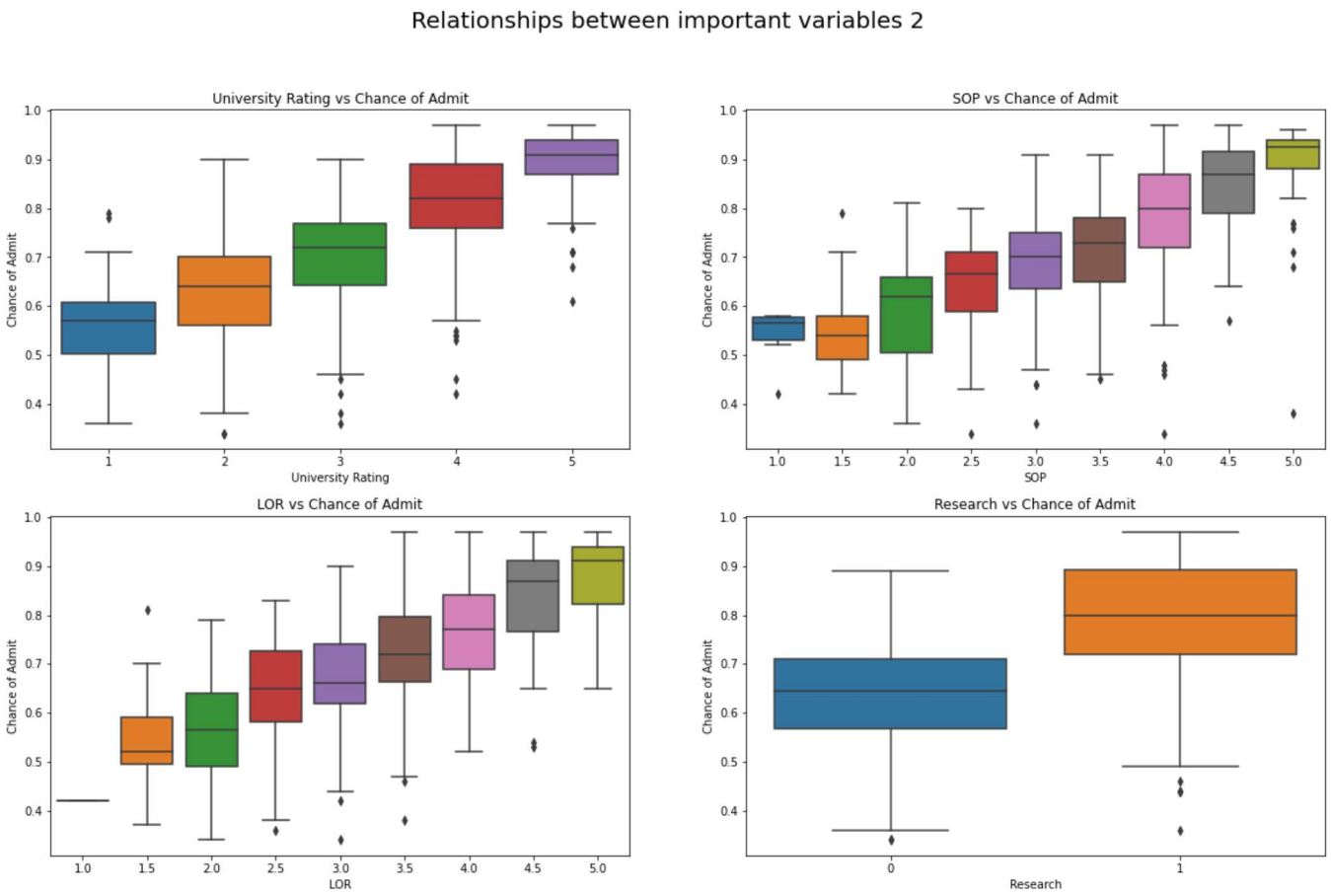
d. Bivariate Analysis

```
plt.figure(figsize=(20,12)).suptitle("Relationships between important variables",fontsize=20)
plt.subplot(2,3,1)
plt.title('GRE SCORE vs Chance of Admit')
sns.histplot(data=df,x='GRE Score', y='Chance of Admit')
plt.subplot(2,3,2)
plt.title('TOEFL vs Chance of Admit')
sns.lineplot(data=df,x='TOEFL Score', y='Chance of Admit')
plt.subplot(2,3,3)
plt.title('CGPA vs Chance of Admit')
sns.histplot(data=df,x='CGPA', y='Chance of Admit')
plt.show()
```

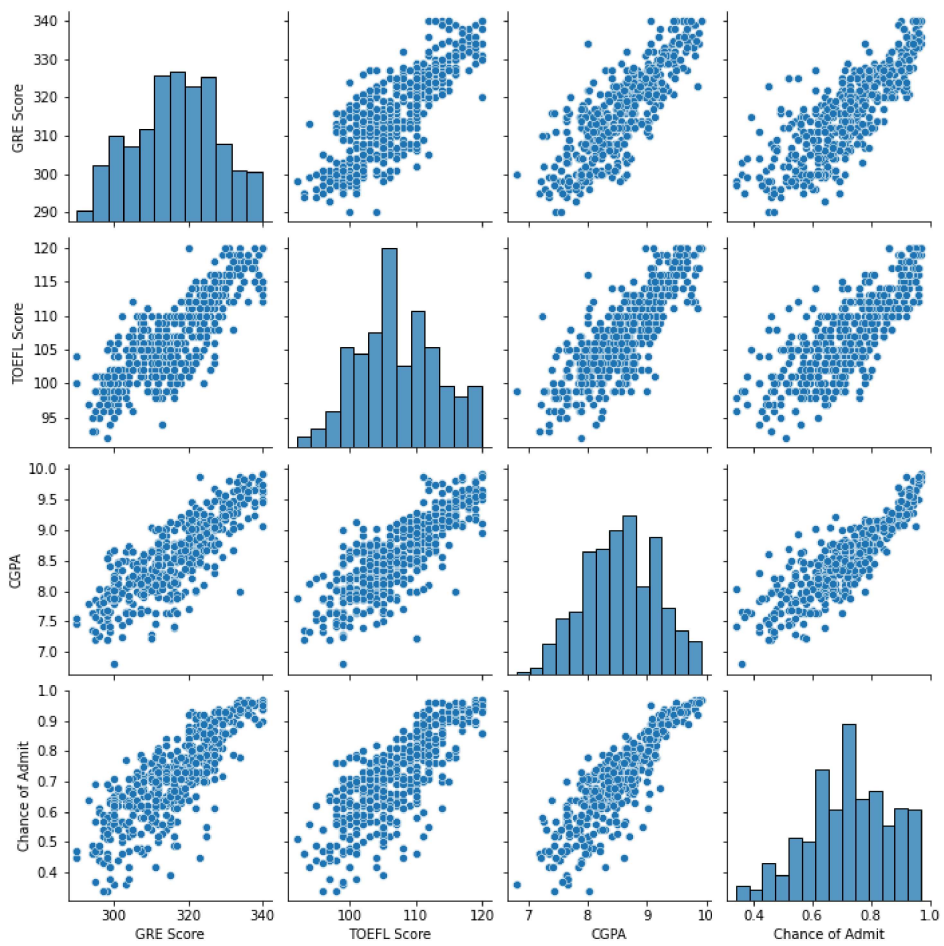



```
plt.figure(figsize=(20,12)).suptitle("Relationships between important variables 2",fontsize=20)
plt.subplot(2,2,1)
plt.title('University Rating vs Chance of Admit')
sns.boxplot(data=df,x='University Rating', y='Chance of Admit')
plt.subplot(2,2,2)
```

```
plt.title('SOP vs Chance of Admit')
sns.boxplot(data=df,x='SOP', y='Chance of Admit')
plt.subplot(2,2,3)
plt.title('LOR vs Chance of Admit')
sns.boxplot(data=df,x='LOR', y='Chance of Admit')
plt.subplot(2,2,4)
plt.title('Research vs Chance of Admit')
sns.boxplot(data=df,x='Research', y='Chance of Admit')
plt.show()
```



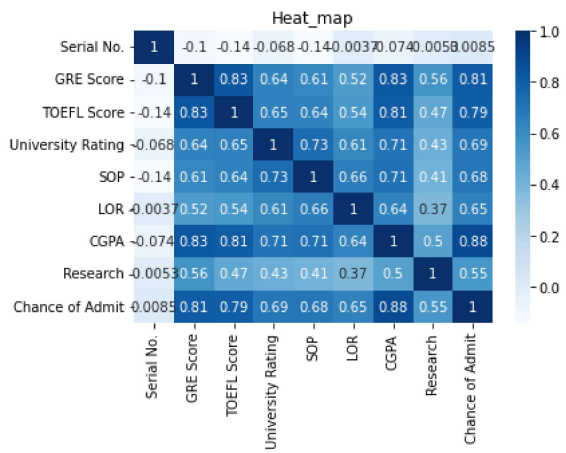

```
# pair plot
sns.pairplot(data=df[['GRE Score','TOEFL Score','CGPA','Chance of Admit']])
plt.show()
```




```
df.corr()
```

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
Serial No.	1.000000	-0.103839	-0.141696	-0.067641	-0.137352	-0.003694	-0.074289	-0.005332	0.008505
GRE Score	-0.103839	1.000000	0.827200	0.635376	0.613498	0.524679	0.825878	0.563398	0.810351
TOEFL Score	-0.141696	0.827200	1.000000	0.649799	0.644410	0.541563	0.810574	0.467012	0.792228
University Rating	-0.067641	0.635376	0.649799	1.000000	0.728024	0.608651	0.705254	0.427047	0.690132
SOP	-0.137352	0.613498	0.644410	0.728024	1.000000	0.663707	0.712154	0.408116	0.684137
LOR	-0.003694	0.524679	0.541563	0.608651	0.663707	1.000000	0.637469	0.372526	0.645365
CGPA	-0.074289	0.825878	0.810574	0.705254	0.712154	0.637469	1.000000	0.501311	0.882413
Research	-0.005332	0.563398	0.467012	0.427047	0.408116	0.372526	0.501311	1.000000	0.545871
Chance of Admit	0.008505	0.810351	0.792228	0.690132	0.684137	0.645365	0.882413	0.545871	1.000000

```
plt.title('Heat_map', fontsize=12)
sns.heatmap(df.corr(),cmap= "Blues", annot=True)
plt.show()
```



e. Insights based on EDA

Comments on range of attributes, outliers of various attributes:

1. GRE Score: This field shows GRE Score scored by Students, its range between 290-340.
2. TOEFL Score: It shows TOEFL--score scored by students, its range between 92-120.
3. University Rating: This field shows student's university rating, it ranges between 1 to 5.
4. SOP: This field shows Statement of Purpose of student who wants to join in IVY league College. It ranges between 1 to 5.
5. LOR: Letter of Recommendation Strength, which shows the students recommendation letter strength. It ranghes between 1 - 5.
6. CGPA: This field shows CGPA scores student's Bachelor course. It ranges between 6.0 to 10.0
7. Research: It shows research experience of the students. It range between 0 to 1.
8. Chance of Admit: It shows probability of student who get chance in IVY League college. It ranges between 0 to 1.

Comments on the distribution of the variables and relationship between them:

1. CGPA and Chance of Admit: Using these we can find above which CGPA, chance of Admit is high.
2. University rating and Chance of Admit: Using these we can find above which University Rating, chance of Admit is high. Like wi

Comments for each univariate and bivariate plots:

Univariate:
1. GRE Score(displot): This shows GRE Score distribution of students who aspire to join IVY league College.
2. TOEFL Score(displot): This shows TOEFL Score distribution of students who aspire to join IVY league College.
3. University Rating(countplot): This shows the ratings of college where students done their bachelors.
4. Research(countplot): This shows the research experience where students done their bachelors.
5. CGPA(displot): This shows CGPA Score distribution of students who aspire to join IVY league College.

Bivariate:
1. Histplot: This shows comparison between Chance of Admit and GRE Score, TOEFL Score, CGPA(here we used subplot).
2. Boxplot: This plot shows the distribution of quantitative data in a way that facilitates comparisons between Chance of Admit a

2. Data Preprocessing

2.1. Duplicate value check

```
df.duplicated().sum()
```

```
0
```

2.2. Missing value treatment

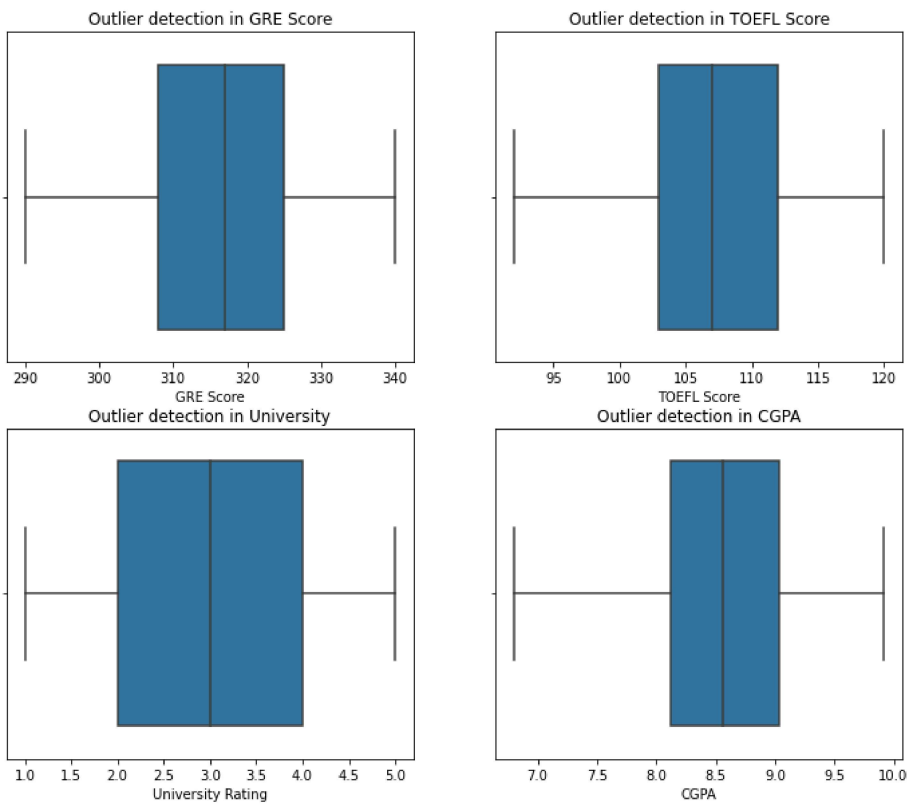
```
df.isnull().sum()
```

```
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
```

2.3. Outlier treatment

```
plt.figure(figsize=(12,10))
plt.subplot(2,2,1)
plt.title('Outlier detection in GRE Score', fontsize=12)
sns.boxplot(data=df,x='GRE Score')
plt.subplot(2,2,2)
plt.title('Outlier detection in TOEFL Score', fontsize=12)
sns.boxplot(data=df,x='TOEFL Score')
plt.subplot(2,2,3)
plt.title('Outlier detection in University', fontsize=12)
sns.boxplot(data=df,x='University Rating')
```

```
plt.subplot(2,2,4)
plt.title('Outlier detection in CGPA', fontsize=12)
sns.boxplot(data=df,x='CGPA')
plt.show()
```



2.4 Data preparation for modeling

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score

from statsmodels.stats.outliers_influence import variance_inflation_factor
from scipy import stats


#drop Serial_No
df = df.drop(columns=['Serial No.'], axis=1)


X=df[df.columns.drop('Chance of Admit')]
y=df['Chance of Admit']


std=StandardScaler()
X=std.fit_transform(X)


X_train,X_test,y_train, y_test=train_test_split(X,y,test_size=0.25,shuffle=True)


print(X_train.shape, X_test.shape)
print(y_train.shape, y_test.shape)
```

```
(375, 7) (125, 7)
(375,) (125,)
```

3.Model Building

```
def adjusted_r2(r2, p, n):
    adj_r2=1-((1-r2)*(n-1)/(n-p-1))
    return adj_r2
def metrics(y_true, y_pred, p=None):
    n=y_true.shape[0]
    mse = np.sum((y_true-y_pred)**2)/n
    rmse = np.sqrt(mse)
    mae = np.mean(np.abs(y_true - y_pred))
```

```
    score = r2_score(y_true, y_pred)
    adj_r2 = None
    if p is not None:
        adj_r2 = adjusted_r2(score, p, n)

    res = {
        "mean_absolute_error": round(mae,2),
        "rmse": round(rmse,2),
        "r2_score":round(score,2),
        "adj_r2": round(adj_r2,2)
    }
    return res


def train_model(X_train, y_train, X_test,y_test,cols):

    model=LinearRegression()
    model.fit(X_train, y_train)
    y_pred_train = model.predict(X_train)
    y_pred_test = model.predict(X_test)
    p = X_train.shape[1]
    train_res = get_metrics(y_train, y_pred_train, p)
    test_res = get_metrics(y_test, y_pred_test, p)

    print("Linear Regression Model")
def train_model(X_train, y_train, X_test,y_test,cols):

    model=LinearRegression()
    model.fit(X_train, y_train)
    y_pred_train = model.predict(X_train)
    y_pred_test = model.predict(X_test)
    p = X_train.shape[1]
    train_res = metrics(y_train, y_pred_train, p)
    test_res = metrics(y_test, y_pred_test, p)

    print("*"*15,"Linear Regression Model","*"*15)
    print(f"Train MAE: {train_res['mean_absolute_error']} Test MAE: {test_res['mean_absolute_error']}")
    print(f"Train_RSME: {train_res['rmse']} Test RMSE:{test_res['rmse']}")
    print(f"Train R2: {train_res['r2_score']} Test R2:{test_res['r2_score']}")
    print(f"Train Adj R2:{train_res['adj_r2']} Test Adj R2: {test_res['adj_r2']}")
    print('Intercept:',model.intercept_)

    coef_df = pd.DataFrame({'Column': cols, 'Coef': model.coef_})

    print(coef_df)

    return model

    print(f"Train_RSME: {train_res['rmse']} Test RMSE:{test_res['rmse']}")
    print(f"Train R2: {train_res['r2_score']} Test R2:{test_res['r2_score']}")
    print(f"Train Adj R2:{train_res['adj_r2']} Test Adj R2: {test_res['adj_r2']}")
    print('Intercept:',model.intercept_)

    coef_df = pd.DataFrame({'Column': cols, 'Coef': model.coef_})

    print(coef_df)

    return model



train_model(X_train, y_train, X_test, y_test, df.columns[:-1])
```

```
********** Linear Regression Model **********
Train MAE: 0.04 Test MAE: 0.05
Train_RSME: 0.06 Test RMSE:0.07
Train R2: 0.84 Test R2:0.74
Train Adj R2:0.84 Test Adj R2: 0.73
Intercept: 0.7236353277469838
              Column      Coef
0          GRE Score  0.024682
1        TOEFL Score  0.017138
2  University Rating  0.005577
3                SOP  0.002407
4                LOR  0.016309
5               CGPA  0.071764
6           Research  0.007703
LinearRegression()
```

4. Testing the assumptions of linear regression model

a. Multicollinearity check by VIF score

```
def vif(df1):
    vif_d = pd.DataFrame()
    vif_d['feature'] = df1.columns

    vif_d['VIF'] = [variance_inflation_factor(df1.values, i) for i in range(len(df1.columns))]

    return vif_d


res = vif(df.iloc[:, :-1])
res
```

	feature	VIF
0	GRE Score	1308.061089
1	TOEFL Score	1215.951898
2	University Rating	20.933361
3	SOP	35.265006
4	LOR	30.911476
5	CGPA	950.817985
6	Research	2.869493

```
# calculate VIF without GRE Score
res = vif(df.iloc[:,1:-1])
res
```

	feature	VIF
0	TOEFL Score	639.741892
1	University Rating	19.884298
2	SOP	33.733613
3	LOR	30.631503
4	CGPA	728.778312
5	Research	2.863301

```
# calculate VIF without TOEFL Score
res = vif(df.iloc[:,2:-1])
res
```

	feature	VIF
0	University Rating	19.777410
1	SOP	33.625178
2	LOR	30.356252
3	CGPA	25.101796
4	Research	2.842227

```
# from the above result SOP is high, we need to drop that.
res = vif(df.iloc[:,2:-1].drop(columns=['SOP']))
res
```

	feature	VIF
0	University Rating	15.140770
1	LOR	26.918495
2	CGPA	22.369655
3	Research	2.819171

```
# from the above result LOR is high, we need to drop that.
df1 = df.iloc[:,2:-1].drop(columns=['SOP','LOR'])
res = vif(df1)
res
```

	feature	VIF
0	University Rating	12.498400
1	CGPA	11.040746
2	Research	2.783179

```
# from the above result University Rating is high, we need to drop that.
df1 = df1.drop(columns=['University Rating'])
res = vif(df1)
res
```

	feature	VIF
0	CGPA	2.455008
1	Research	2.455008

```
# here we are going train the model using 2 important features only.
X = df[['CGPA', 'Research']]
sc = StandardScaler()
X = sc.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, shuffle=True )


train_model(X_train, y_train, X_test, y_test,['CGPA', 'Research'])
```

```
*************** Linear Regression Model ***************
Train MAE: 0.05 Test MAE: 0.05
Train_RSME: 0.06 Test RMSE:0.06
Train R2: 0.8 Test R2:0.78
Train Adj R2:0.8 Test Adj R2: 0.78
Intercept: 0.7201128939898352
     Column      Coef
0      CGPA  0.116841
1  Research  0.019531
LinearRegression()
```

b. Mean of residuals is nearly zero

From the Above results, Train_RMSE: 0.06 Test RMSE:0.06 RMSE is nearly zero.

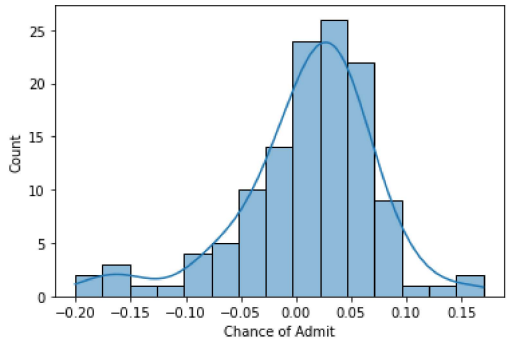
c. Linearity of variables

From the above analysis, we found that all the independent variables are linearly dependent on the chance of Admit. (Refer the below results)

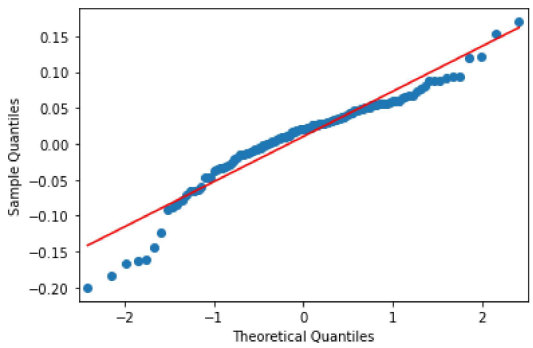
d. Normality of residuals

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, shuffle=True )


model=LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
residuals = (y_test - y_pred)
sns.histplot(residuals, kde=True)
plt.show()
```

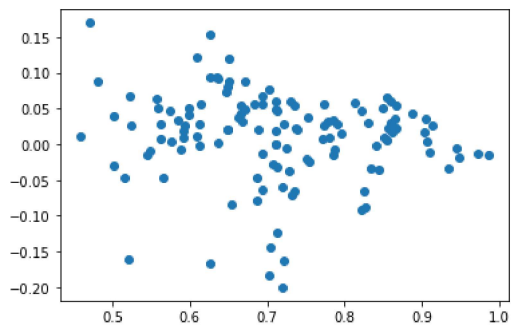


```
from statsmodels.graphics.gofplots import qqplot
qqplot(residuals, line='s')
plt.show()
```



e.Test for Homoscedasticity

```
plt.scatter(y_pred, residuals)
plt.show()
```



5. Model performance evaluation

a. Metrics checked - MAE, RMSE, R2, Adj R2

```
Test MAE: 0.05
Test RMSE:0.06
Test R2:0.78
Test Adj R2: 0.78
```

b. Train and test performances are checked

Result of Test and Train data were almost similar, there is no major variation.

```
Linear Regression Model
Train MAE: 0.05   Test MAE: 0.05
Train_RSME: 0.06  Test RMSE:0.06
Train R2: 0.8     Test R2:0.78
Train Adj R2:0.8  Test Adj R2: 0.78
```

c. Comments on the performance measures and if there is any need to improve the model or not

1. MAE is 0.05 – value is near zero, so model is good.
2. RMSE is 0.06 - value is near zero, hence model is good.
3. R2 is 0.78 – value is higher than 0.5, here R2 value is at desired level for the model.
4. Adj R2 is 0.78 – value is higher than 0.5, here Adj R2 value is at desired level for the model.