```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression,LogisticRegression,Lasso,Ridge,ElasticNet
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
```

```
df = pd.read_csv('https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/839/original/Jamboree_Admission.csv')
```

```
df.ndim
# the data has only 2 dimensions
```

```
2
```

```
df.describe()
```

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Resea
count	500.000000	500.000000	500.000000	500.000000	500.000000	500.00000	500.000000	500.000
mean	250.500000	316.472000	107.192000	3.114000	3.374000	3.48400	8.576440	0.560
std	144.481833	11.295148	6.081868	1.143512	0.991004	0.92545	0.604813	0.496
min	1.000000	290.000000	92.000000	1.000000	1.000000	1.00000	6.800000	0.000
25%	125.750000	308.000000	103.000000	2.000000	2.500000	3.00000	8.127500	0.000
50%	250.500000	317.000000	107.000000	3.000000	3.500000	3.50000	8.560000	1.000
75%	375.250000	325.000000	112.000000	4.000000	4.000000	4.00000	9.040000	1.000

```
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
```

```
df.shape
# The data has 500 rows and 9 columns
```

```
(500, 9)
```

```
df.head()
# first few rows of the dataset
```

	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.isnull().sum()
# Null values are not present in the dataset
```

```
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[df.duplicated].count()
# There are no duplicated rows in the dataset
```

```
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.columns
# names of the columns
```

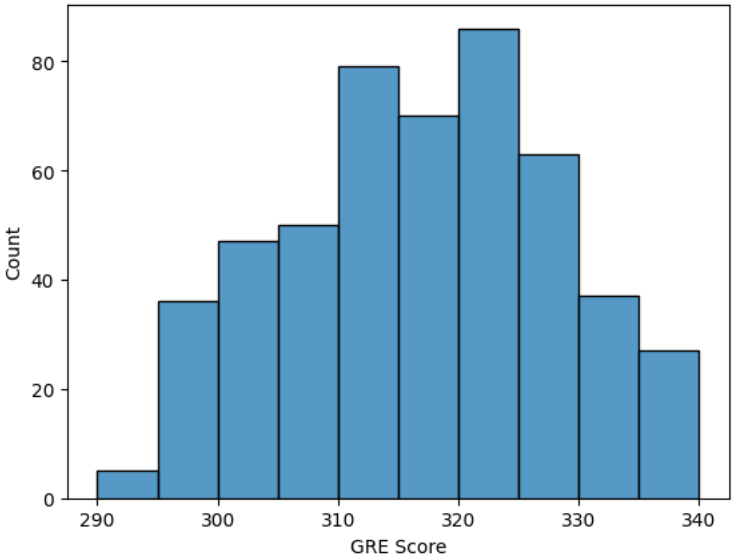
```
Index(['Serial No.', 'GRE Score', 'TOEFL Score', 'University Rating', 'SOP',
       'LOR ', 'CGPA', 'Research', 'Chance of Admit '],
      dtype='object')
```

```
df.dtypes
# datatypes of all the columns
```

```
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
```

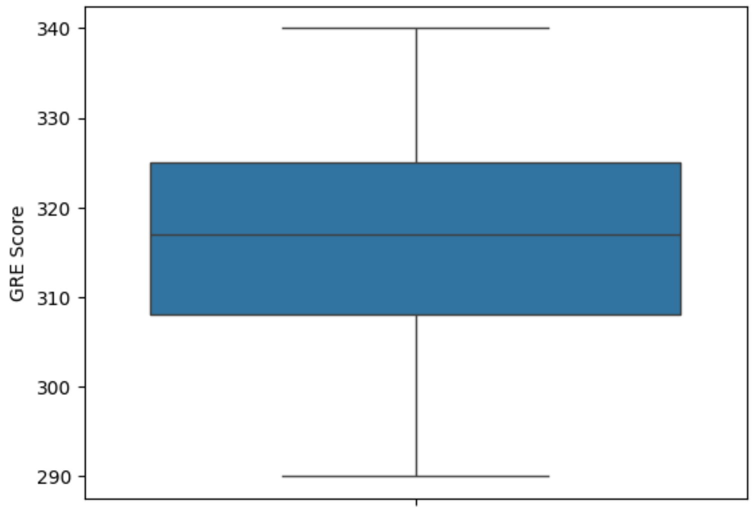
```
sns.histplot(df['GRE Score'],bins=10)
# Distribution of GRE scores
```

```
<Axes: xlabel='GRE Score', ylabel='Count'>
```



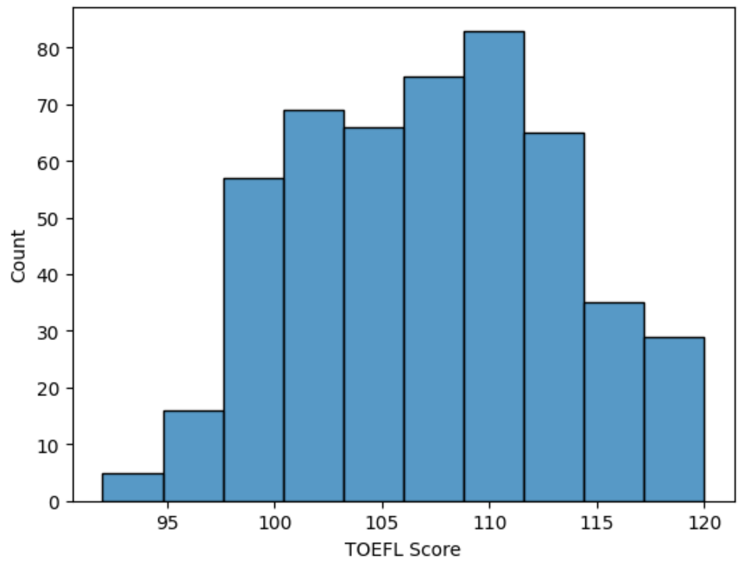

```
sns.boxplot(df['GRE Score'])
# No outliers are present
# Median is around 320
```

```
<Axes: ylabel='GRE Score'>
```



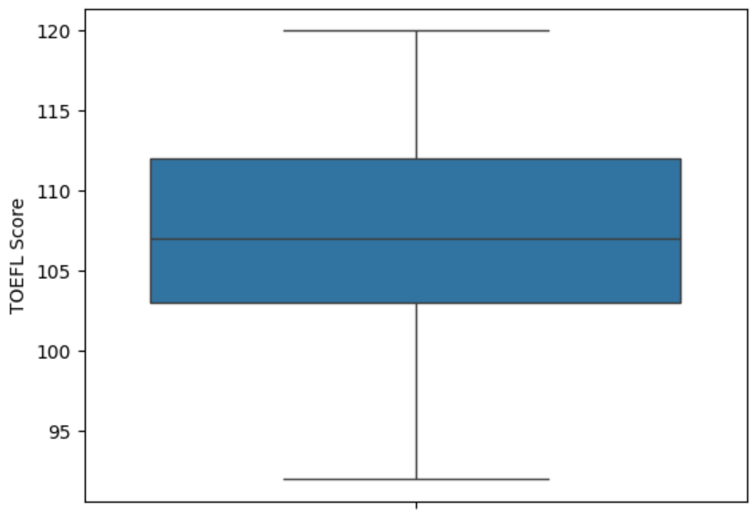

```
sns.histplot(df['TOEFL Score'],bins=10)
# Distribution of TOEFL scores
```

```
<Axes: xlabel='TOEFL Score', ylabel='Count'>
```



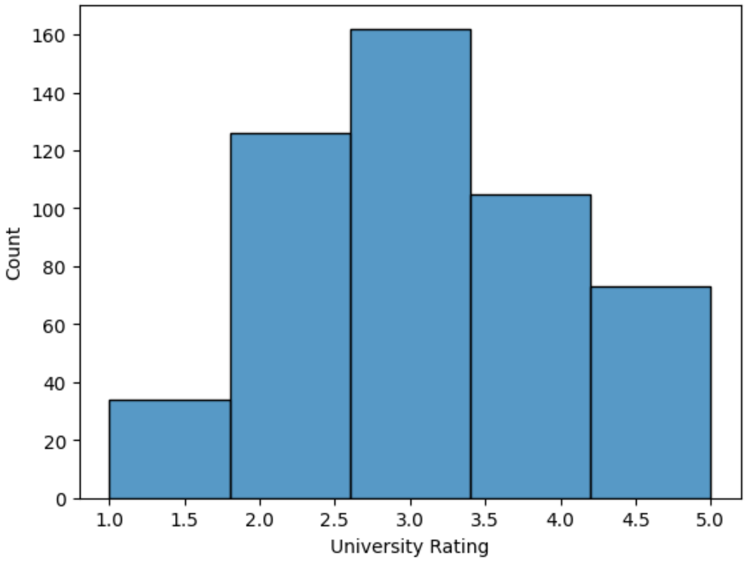

```
sns.boxplot(df['TOEFL Score'])
# no outliers are present
# Median is around 105-110
```

```
<Axes: ylabel='TOEFL Score'>
```



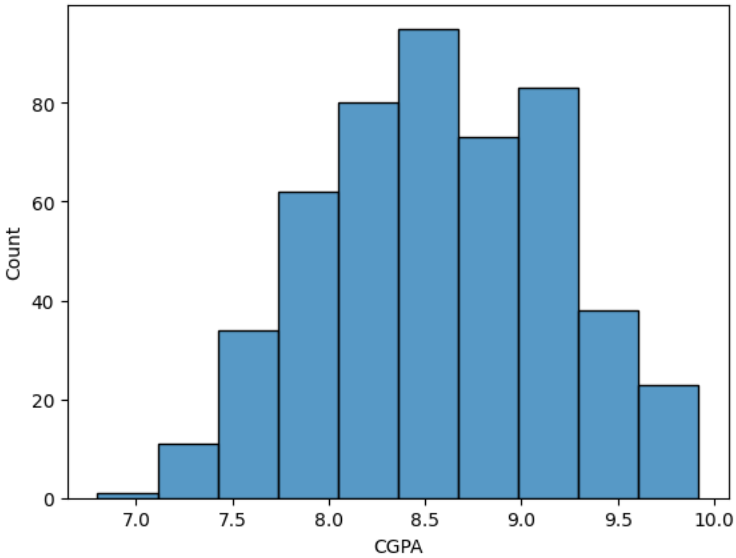

```
sns.histplot(df['University Rating'],bins=5)
# most of the universities have a rating 2 & 3
# only few universities have 1 rating
```

```
<Axes: xlabel='University Rating', ylabel='Count'>
```



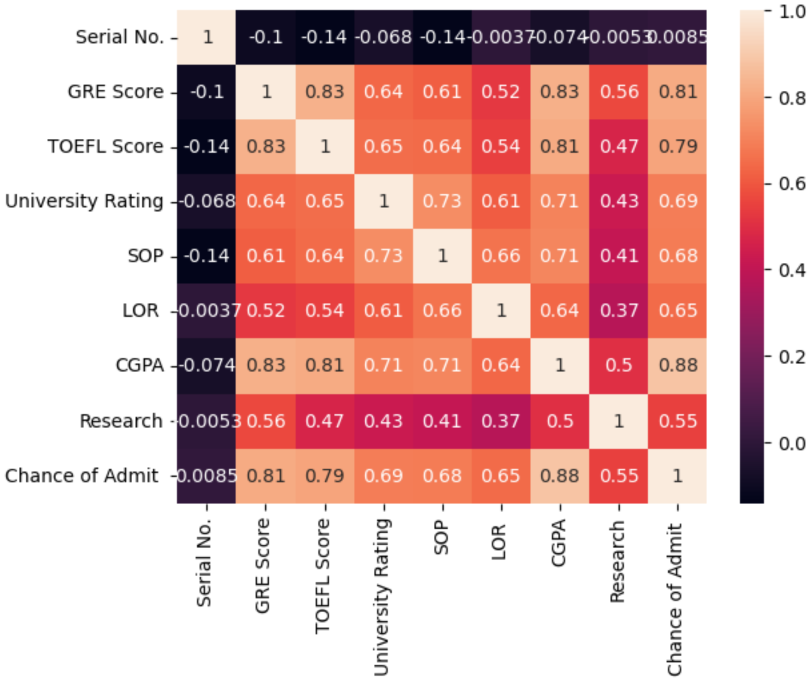

```
sns.histplot(df['CGPA'],bins=10)
# Most of the students have a CGPA around 8.5
```

```
<Axes: xlabel='CGPA', ylabel='Count'>
```



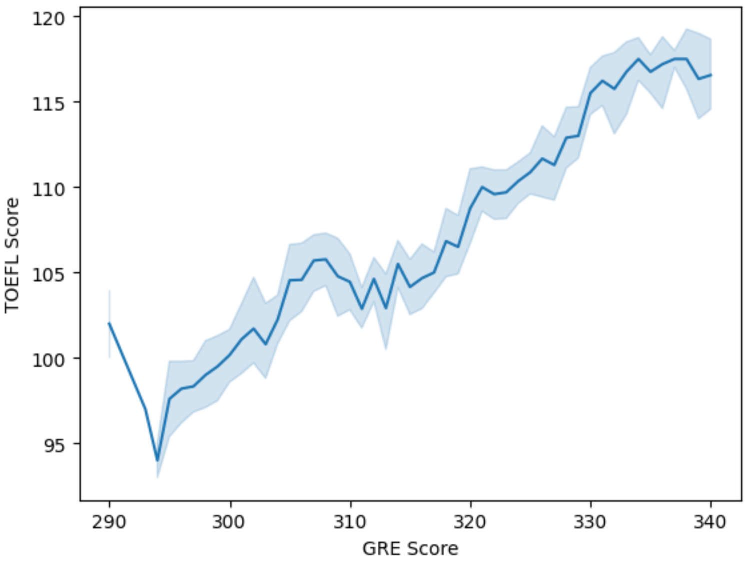

```
sns.heatmap(df.corr(),annot = True)
# correlation between columns

# we can see there is a strong +ve correlation between
# GRE SCORE and Chances of Admit
# GRE Score and CGPA
# GRE Score and TOEFL

# we can see there isn't much correlation between Research and other columns
```

<Axes: >



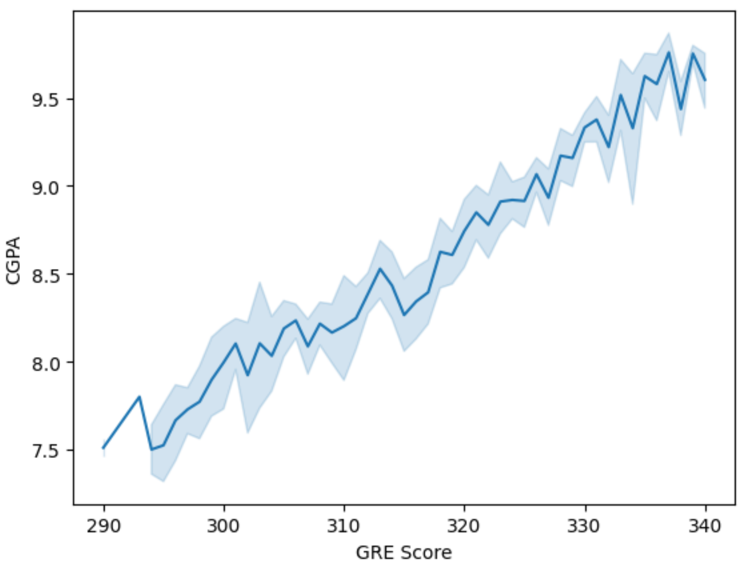

```
# Bivariate plot between GRE and TOEFL Scores
sns.lineplot(x='GRE Score',y='TOEFL Score',data=df)
# we observe students who tend to score high in GRE also score high in TOEFL and vice-versa
```

<Axes: xlabel='GRE Score', ylabel='TOEFL Score'>



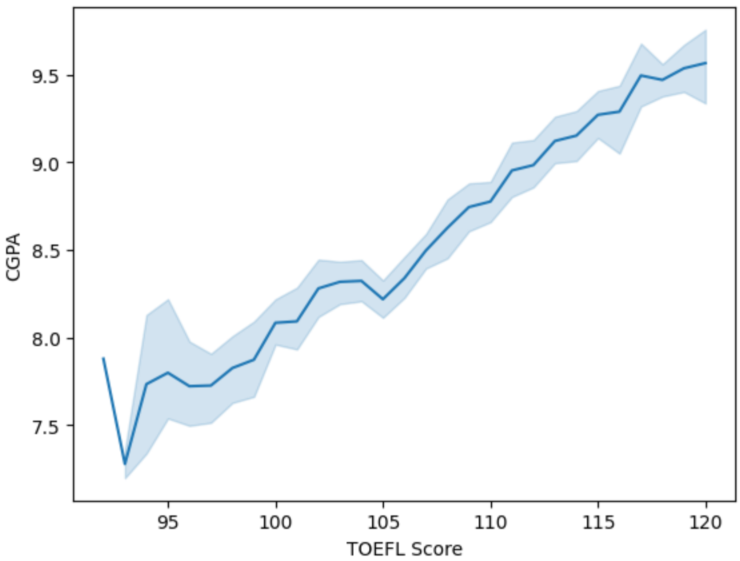

```
sns.lineplot(x='GRE Score',y='CGPA',data=df)
# Students with high CGPA tend to score high in GRE
```

<Axes: xlabel='GRE Score', ylabel='CGPA'>



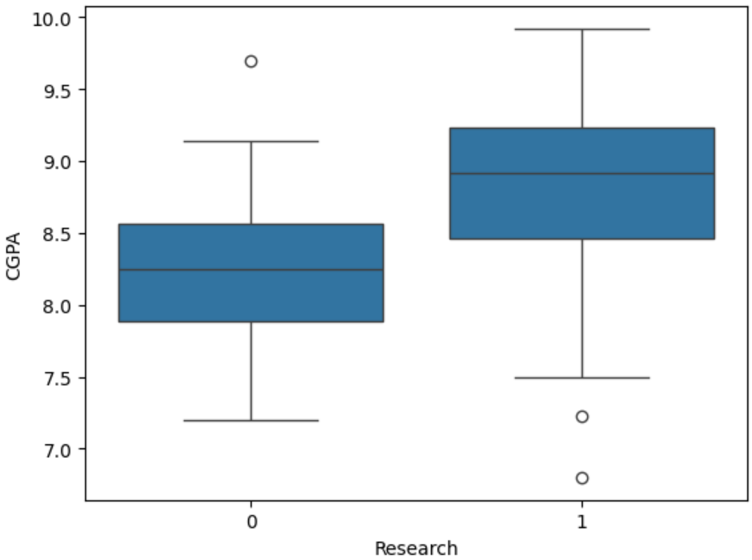
```
sns.lineplot(x='TOEFL Score',y='CGPA',data=df)
# Students with high CGPA tend to score high in TOEFL
```

```
<Axes: xlabel='TOEFL Score', ylabel='CGPA'>
```



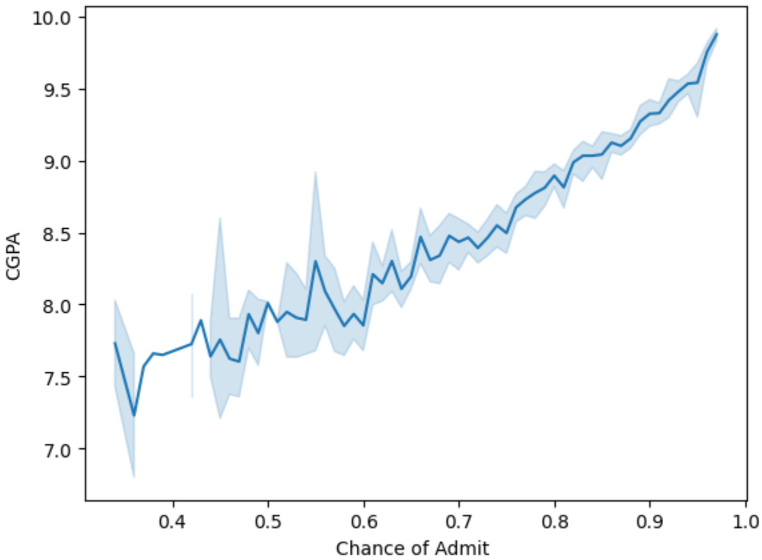

```
sns.boxplot(x='Research',y='CGPA',data=df)
# students who had some research experience have higher CGPA
```

```
<Axes: xlabel='Research', ylabel='CGPA'>
```



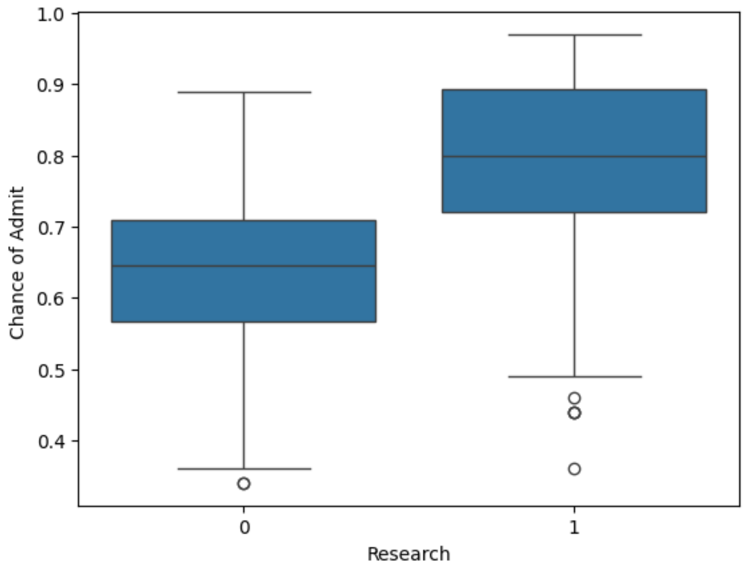

```
sns.lineplot(x='Chance of Admit ',y='CGPA',data=df)
# students who had higher CGPA have higher chances of admission
```

```
<Axes: xlabel='Chance of Admit ', ylabel='CGPA'>
```



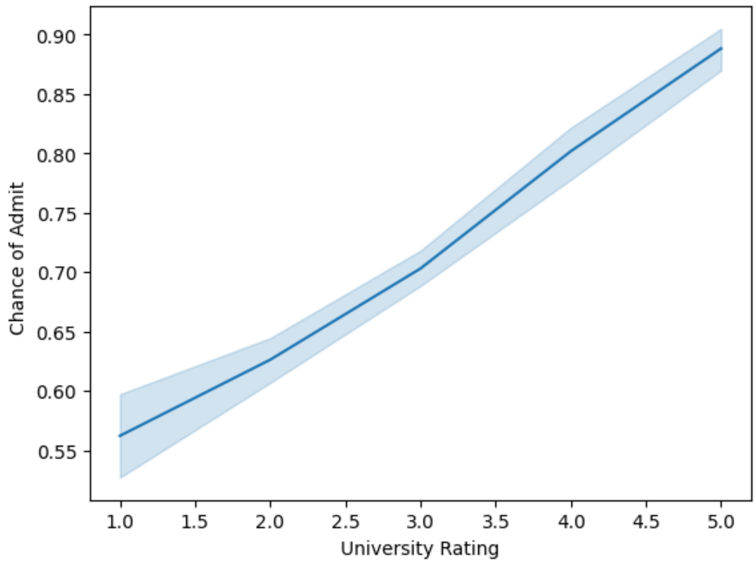

```
sns.boxplot(y='Chance of Admit ',x='Research',data=df)
# students who had research experience have higher chances of admission on an average
```

<Axes: xlabel='Research', ylabel='Chance of Admit '>




```
sns.lineplot(y='Chance of Admit ',x='University Rating',data=df)
# Students associated with higher ranked universities have higher changes of admission
```

<Axes: xlabel='University Rating', ylabel='Chance of Admit '>




```
# I want to drop Serial No. column as it doesn't help while training the model
df.drop(columns = 'Serial No.',inplace=True,axis=0)
```

```
df.head()
# we can see the column Serial No. has been dropped
```

	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

```
df.shape
# Yes the columns Serial No. has been dropped and so the column count is 8 while it was 9 earlier
```

```
(500, 8)
```

```
# Linear Regression

X_columns = df.columns[df.columns != 'Chance of Admit ']
X = df[X_columns]
Y = df['Chance of Admit ']
# Setting the Chance of Admit as Target variable and rest of the variables as independent variables
```

```
# Standardizing the dataset
# standardize the dataset
sc = StandardScaler()
X = sc.fit_transform(X)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
# Dividing the data into training and test data
# random_state=42 sets the random seed for reproducibility. It ensures that if you run the code multiple times, you'll get the same split
# test_size=0.2 specifies that 20% of the data should be used for testing, and the remaining 80% will be used for training.
```

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

```
(400, 7) (400,)
(100, 7) (100,)
```

```
# Linear Regression
linear_model = LinearRegression()
linear_model.fit(X_train, y_train)
linear_y_pred = linear_model.predict(X_test)

# Lasso Regression
lasso_model = Lasso(alpha=1.0)
lasso_model.fit(X_train, y_train)
lasso_y_pred = lasso_model.predict(X_test)

# Ridge Regression
ridge_model = Ridge(alpha=1.0)
ridge_model.fit(X_train, y_train)
ridge_y_pred = ridge_model.predict(X_test)

# Evaluate Linear Regression
linear_r2 = r2_score(y_test, linear_y_pred)
linear_mae = mean_absolute_error(y_test, linear_y_pred)
linear_mse = mean_squared_error(y_test, linear_y_pred)
n = len(X_test)
p = X_test.shape[1]
linear_adj_r2 = 1 - (1 - linear_r2) * ((n - 1) / (n - p - 1))

# Evaluate Lasso Regression
lasso_r2 = r2_score(y_test, lasso_y_pred)
lasso_mae = mean_absolute_error(y_test, lasso_y_pred)
lasso_mse = mean_squared_error(y_test, lasso_y_pred)
lasso_adj_r2 = 1 - (1 - lasso_r2) * ((n - 1) / (n - p - 1))

# Evaluate Ridge Regression
ridge_r2 = r2_score(y_test, ridge_y_pred)
ridge_mae = mean_absolute_error(y_test, ridge_y_pred)
ridge_mse = mean_squared_error(y_test, ridge_y_pred)
ridge_adj_r2 = 1 - (1 - ridge_r2) * ((n - 1) / (n - p - 1))

# Print results
print("Linear Regression:")
print(f"R-squared: {linear_r2}")
print(f"Adjusted R-squared: {linear_adj_r2}")
print(f"Mean Absolute Error: {linear_mae}")
print(f"Mean Squared Error: {linear_mse}\n")

print("Lasso Regression:")
print(f"R-squared: {lasso_r2}")
print(f"Adjusted R-squared: {lasso_adj_r2}")
print(f"Mean Absolute Error: {lasso_mae}")
print(f"Mean Squared Error: {lasso_mse}\n")

print("Ridge Regression:")
print(f"R-squared: {ridge_r2}")
print(f"Adjusted R-squared: {ridge_adj_r2}")
print(f"Mean Absolute Error: {ridge_mae}")
print(f"Mean Squared Error: {ridge_mse}")
```

```
Linear Regression:
R-squared: 0.8188432567829629
Adjusted R-squared: 0.8050595915381884
Mean Absolute Error: 0.042722654277053664
Mean Squared Error: 0.00370465539878841

Lasso Regression:
R-squared: -0.00724844132029312
Adjusted R-squared: -0.08388690968161971
Mean Absolute Error: 0.116268
Mean Squared Error: 0.020598230624999995

Ridge Regression:
R-squared: 0.8187987385531802
Adjusted R-squared: 0.8050116860517917
```

```
Mean Absolute Error: 0.04274636477332955
Mean Squared Error: 0.003705565796587465
```

```
# Printing the coefficients (weights) of each independent variable
# Assuming linear_model is your trained Linear Regression model and independent_vars is your list of independent variables
# Create an empty dictionary
weights_dict = {}

# Iterate over each independent variable and its corresponding coefficient
for i, var in enumerate(independent_vars):
    # Add the variable and its coefficient to the dictionary
    weights_dict[var] = coefficients[i]

# Sort the dictionary based on the values (coefficients)
sorted_weights = sorted(weights_dict.items(), key=lambda x: x[1], reverse=True)

# Print the sorted dictionary
for var, weight in sorted_weights:
    print(f"{var}: {weight}")
```

```
CGPA: 0.11252708444059893
Research: 0.024026787646206155
LOR : 0.01723798366142545
TOEFL Score: 0.0029958733715185655
University Rating: 0.0025687977435265965
GRE Score: 0.002434438394836755
SOP: 0.001813690012812433
```

```
# Maybe adjusting the alpha in Lasso Regression would have resulted in a better R2 score
# Testing Lasso Regression with another alpha
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
# Dividing the data into training and test data
# random_state=42 sets the random seed for reproducibility. It ensures that if you run the code multiple times, you'll get the same split
# test_size=0.2 specifies that 20% of the data should be used for testing, and the remaining 80% will be used for training.
```

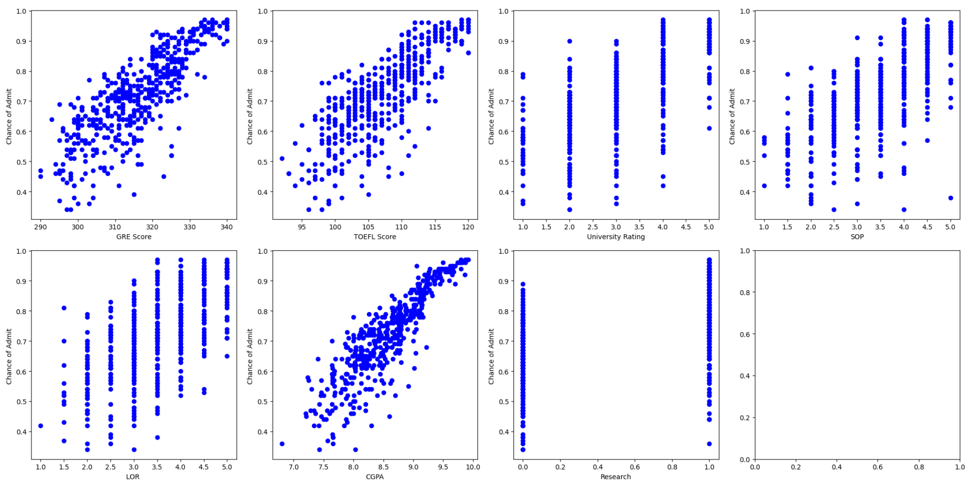
```
# Define a list of alpha values to try
alpha_values = [0.001, 0.01, 0.1, 1.0, 10.0]

# Iterate over alpha values
for alpha in alpha_values:
    # Create and fit Lasso Regression model
    lasso_model = Lasso(alpha=alpha)
    lasso_model.fit(X_train, y_train)

    # Make predictions
    y_pred = lasso_model.predict(X_test)

    # Calculate evaluation metrics
    r2 = r2_score(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    n = len(X_test)
    p = X_test.shape[1]
    lasso_adj_r2 = 1 - (1 - r2) * ((n - 1) / (n - p - 1))

    # Print results
    print(f"Lasso Regression with alpha={alpha}:")
    print(f"R-squared: {r2}")
    print(f"Adjusted R-squared: {lasso_adj_r2}")
    print(f"Mean Absolute Error: {mae}")
    print(f"Mean Squared Error: {mse}")
    print()
```

```
Lasso Regression with alpha=0.001:
R-squared: 0.8192766750510027
Adjusted R-squared: 0.8055259872831442
Mean Absolute Error: 0.04250059556488314
Mean Squared Error: 0.003695791995206996

Lasso Regression with alpha=0.01:
R-squared: 0.8147768068219199
Adjusted R-squared: 0.8006837377757616
Mean Absolute Error: 0.04260100848813942
Mean Squared Error: 0.003787814300491738

Lasso Regression with alpha=0.1:
R-squared: 0.2606300776476902
Adjusted R-squared: 0.20437367051218847
Mean Absolute Error: 0.09858647394745274
Mean Squared Error: 0.015120114912104738

Lasso Regression with alpha=1.0:
R-squared: -0.00724844132029312
Adjusted R-squared: -0.08388690968161971
Mean Absolute Error: 0.116268
Mean Squared Error: 0.020598230624999995
```

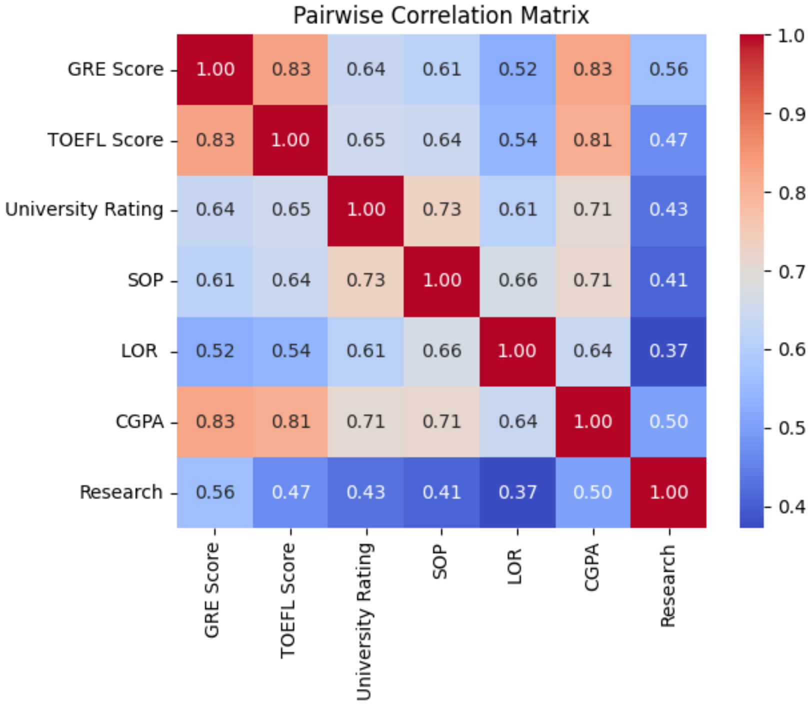
```
Lasso Regression with alpha=10.0:
R-squared: -0.00724844132029312
Adjusted R-squared: -0.08388690968161971
Mean Absolute Error: 0.116268
Mean Squared Error: 0.020598230624999995
```

```
# Observations
# alpha = 0.001 is the best Lasso Regression model since it has relatively better R2 score & better Ajusted R2 score
# Since model is not overfitting, Results for Linear, Ridge and Lasso are the same.
# R2_score and Adjusted_r2 are almost the same. Hence there are no unnecessary independent variables in the data.
```

```
# Assumtions Tests
# 1. Linearity Check : There should be linear check between dependent and independent variables
independent_vars = ['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR ', 'CGPA', 'Research']
dependent_var = 'Chance of Admit '
# Create subplots
fig, axes = plt.subplots(nrows=2, ncols=4, figsize=(20, 10))

# Flatten axes for easier iteration
axes = axes.flatten()

# Plot each independent variable against the dependent variable
for i, var in enumerate(independent_vars):
    ax = axes[i]
    ax.plot(df[var], df[dependent_var], marker='o', linestyle='None', color='b')
    ax.set_xlabel(var)
    ax.set_ylabel(dependent_var)

# Adjust layout
plt.tight_layout()

plt.show()
```

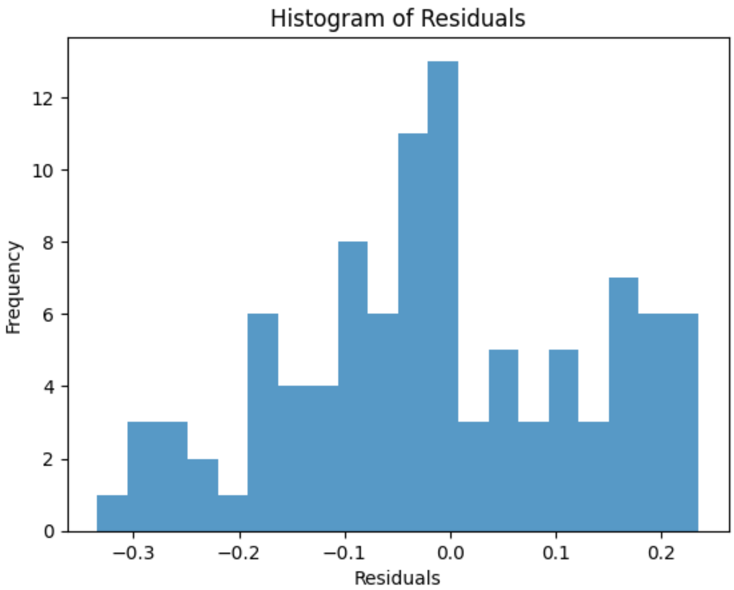


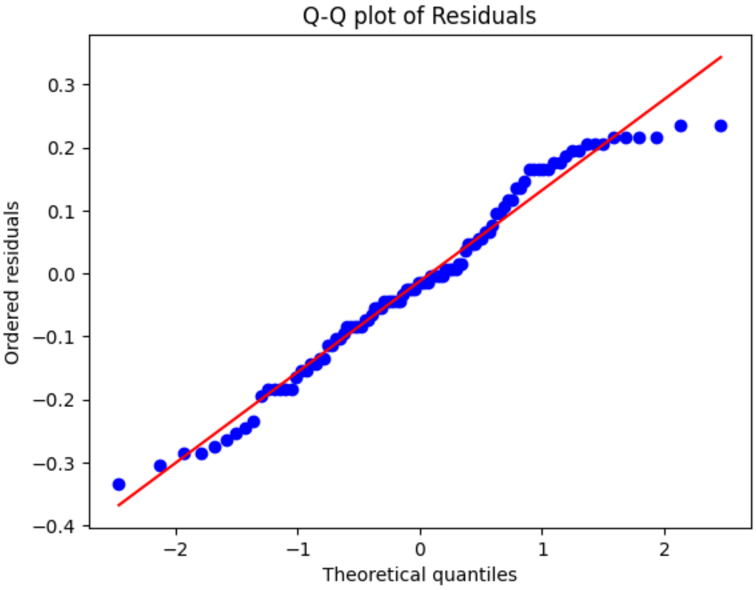

```
# No Multicolinearity
# Calculate pairwise correlations between independent variables
correlation_matrix = df[independent_vars].corr()

# Plot correlation matrix as a heatmap
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Pairwise Correlation Matrix")
plt.show()

# Calculate variance inflation factors (VIFs)
from statsmodels.stats.outliers_influence import variance_inflation_factor
vif_data = pd.DataFrame()
vif_data["Variable"] = independent_vars
vif_data["VIF"] = [variance_inflation_factor(df[independent_vars].values, i) for i in range(len(independent_vars))]
print(vif_data)
```



	Variable	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

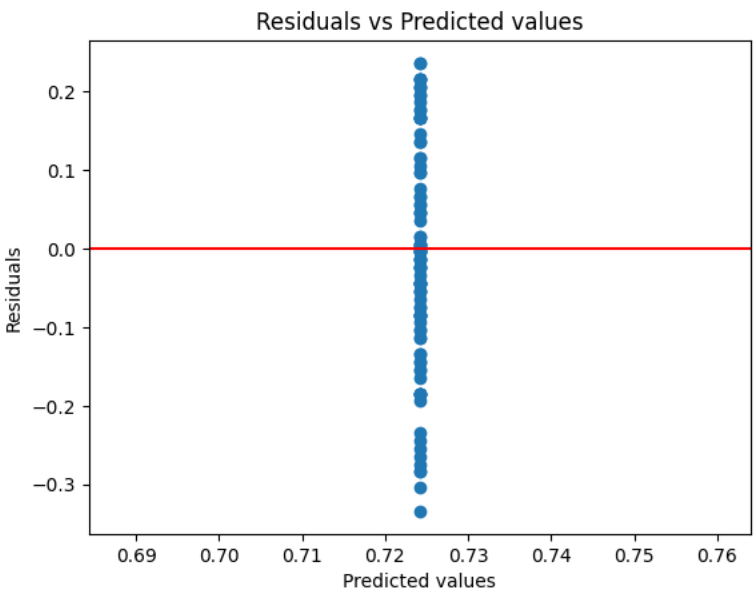
```
# 3. Residuals should be normal
import scipy.stats as stats

# Plot histogram of residuals
residuals = y_test - y_pred
plt.hist(residuals, bins=20, alpha=0.75)
plt.xlabel("Residuals")
plt.ylabel("Frequency")
plt.title("Histogram of Residuals")
plt.show()

# Q-Q plot of residuals
stats.probplot(residuals, dist="norm", plot=plt)
plt.xlabel("Theoretical quantiles")
plt.ylabel("Ordered residuals")
plt.title("Q-Q plot of Residuals")
plt.show()

# Shapiro-Wilk test for normality
_, p_value = stats.shapiro(residuals)
print("Shapiro-Wilk test p-value:", p_value)
```





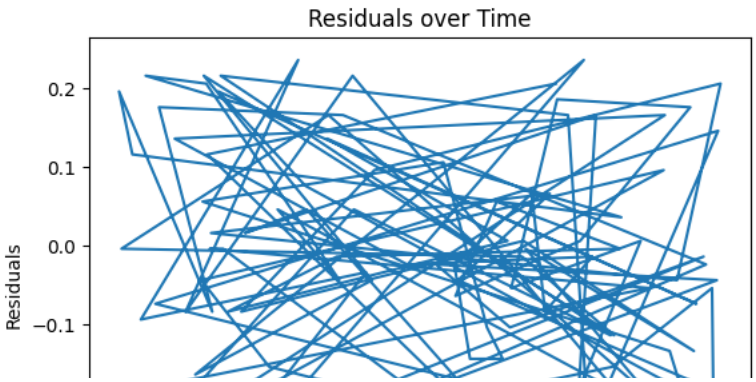

```
Shapiro-Wilk test p-value: 0.02883988432586193
```

```
# 4. No Heteroscedasticity
# Plot residuals against predicted values
residuals = y_test - y_pred
plt.scatter(y_pred, residuals)
plt.xlabel("Predicted values")
plt.ylabel("Residuals")
plt.title("Residuals vs Predicted values")
plt.axhline(y=0, color='r', linestyle='-')
plt.show()
```




```
# 5. No Autocorrelation
```

```
# For time-series data, plot residuals against time or index
plt.plot(residuals)
plt.xlabel("Index")
plt.ylabel("Residuals")
plt.title("Residuals over Time")
plt.show()

# Calculate autocorrelation function (ACF)
from statsmodels.graphics.tsaplots import plot_acf
plot_acf(residuals, lags=20)
plt.xlabel("Lag")
plt.ylabel("Autocorrelation")
plt.title("Autocorrelation Function (ACF) of Residuals")
plt.show()

# Durbin-Watson test for autocorrelation
from statsmodels.stats.stattools import durbin_watson
dw_statistic = durbin_watson(residuals)
print("Durbin-Watson statistic:", dw_statistic)
```



```
# Observations :
# 1. Feature Significance based on weights is
#     - CGPA, Research, LOR, TOEFL Score, University
Rating, GRE Score, SOP
#     This is clearly evident from the weights obtained
from the Linear Regression model
# 2. CGPA, Research, LOR have higher weightage when it
comes to estimating the Chance of Admit
# 3. There is a linear relation between GRE Score, TOEFL,
CGPA and Chance of Admit which is clearly evident from
#     the scatter plots
# 4. There is also some correlation between GRE Score,
TOEFL, CGPA which is evident from the HeatMap.


# Recommendations :
# 1. Jamboree Education can ask the candidates who are
planning to go abroad to pursue higher studies
# to focus highly on their CGPA since it has the highest
significance
# 2. Generally B.Tech studies do not concentrate on
Research only Master's and Ph.D candidates do. But if
# the candidate wants to pursue higher education abroad
he has to concentrate on Research so that the chance
# of getting admission is higher
# 3. It is good to have good Letters of Recommendation
which can create impact during admission. So, It is
#  prudent that students should maitain good rapport with
professors so that they can get compelling LORs.
# 4. Since SOP has the least amount of significance the
candidate should use Gen AI tools like ChatGPT, ATS
friendly tools
#   to prepare their Statement of Purpose.
# 5.Also Students should concentrate more on CGPA,
Research, LORs, TOEFL rather than on GRE
# 6.TOEFL vs GRE -  Since TOEFL has more weightage than
GRE Students should concentrate more on English
vocabulary, grammar
#   than on reasoning and mathematics.
```