In [250…
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler , MinMaxScaler,LabelEncoder

from sklearn.linear_model import LinearRegression
```

In [251…
```
df = pd.read_csv("Jamboree_Admission.csv")
```

In [252…
```
df.head()
```

Out[252]:

	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

EDA

Problem Statment - Need to help Jamboree in understanding what factors are important in graduate admissions and how these factors are interrelated among themselves. It will also help predict one's chances of admission given the rest of the variables.

In [253…
```
df.shape
```

Out[253]:
```
(500, 9)
```

- 500 rows or records
- 9 columns or features

In [254…
```
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
```

- No null values, no categorical column

In [255…

```
df.nunique()
```

Out[255]:

```
Serial No.           500
GRE Score             49
TOEFL Score           29
University Rating      5
SOP                    9
LOR                    9
CGPA                 184
Research               2
Chance of Admit       61
dtype: int64
```

- Dropping the unique row Identifier as we don't want the model to build some understanding based on row numbers.

In [256…

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

In [257…

```
df
```

Out[257]:

	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
...	...	...	...	...	...	...	...	...
495	332	108	5	4.5	4.0	9.02	1	0.87
496	337	117	5	5.0	5.0	9.87	1	0.96
497	330	120	5	4.5	5.0	9.56	1	0.93
498	312	103	4	4.0	5.0	8.43	0	0.73
499	327	113	4	4.5	4.5	9.04	0	0.84

500 rows × 8 columns

In [258…

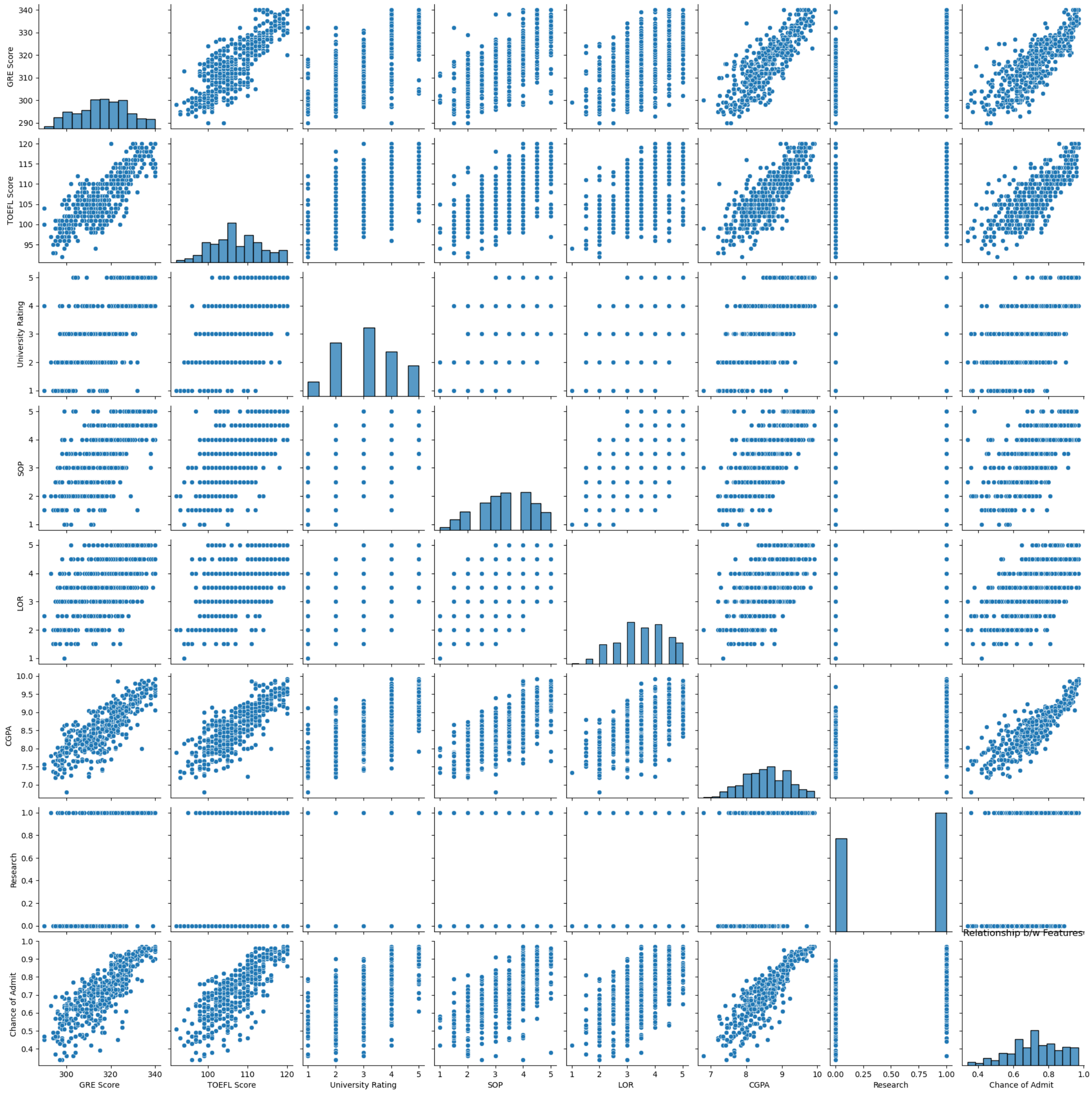
```
df.describe()
```

Out[258]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chan of Adm
count	500.000000	500.000000	500.000000	500.000000	500.00000	500.000000	500.000000	500.000
mean	316.472000	107.192000	3.114000	3.374000	3.48400	8.576440	0.560000	0.721
std	11.295148	6.081868	1.143512	0.991004	0.92545	0.604813	0.496884	0.141
min	290.000000	92.000000	1.000000	1.000000	1.00000	6.800000	0.000000	0.340
25%	308.000000	103.000000	2.000000	2.500000	3.00000	8.127500	0.000000	0.630
50%	317.000000	107.000000	3.000000	3.500000	3.50000	8.560000	1.000000	0.720
75%	325.000000	112.000000	4.000000	4.000000	4.00000	9.040000	1.000000	0.820
max	340.000000	120.000000	5.000000	5.000000	5.00000	9.920000	1.000000	0.970

In [259…

```
sns.pairplot(df)
plt.title('Relationship b/w Features')
plt.show();
```



- GRE , TOEFL , CGPA, Chance of Admit are positively correlated to each other
- We can see from the scatterplot that the values of university ranking, SOP, LOR and research are not continuous. We can convert these columns to categorical variables

Data Preprocessing

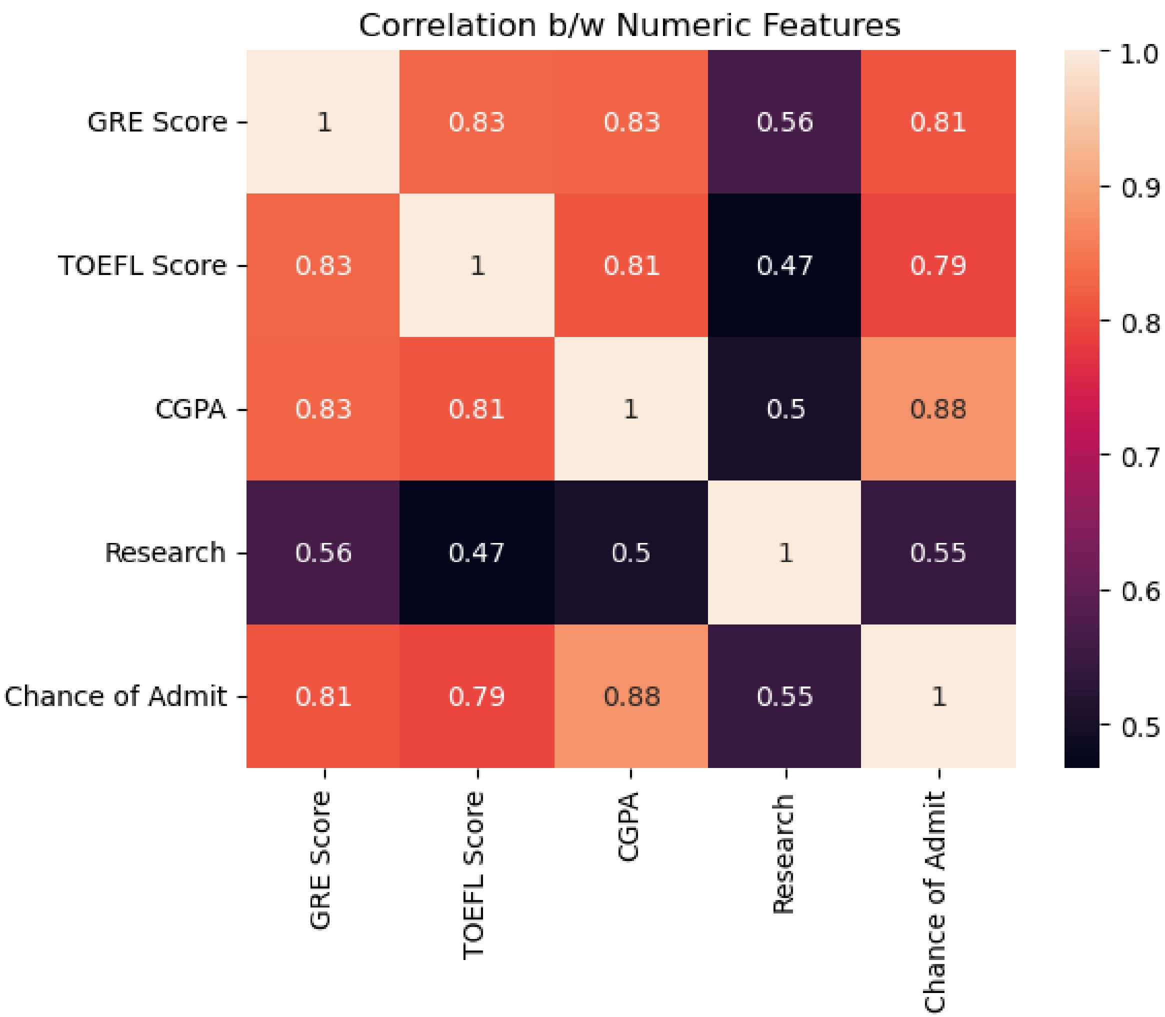
In [260…

```
df.rename(columns={'LOR ':'LOR', 'Chance of Admit ':'Chance of Admit'}, inplace=Tru
df[['University Rating', 'SOP', 'LOR']] = df[['University Rating', 'SOP', 'LOR']].a
df['Research'] = df['Research'].astype('bool')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   GRE Score          500 non-null    int64
 1   TOEFL Score        500 non-null    int64
 2   University Rating  500 non-null    category
 3   SOP                500 non-null    category
 4   LOR                500 non-null    category
 5   CGPA               500 non-null    float64
 6   Research           500 non-null    bool
 7   Chance of Admit    500 non-null    float64
dtypes: bool(1), category(3), float64(2), int64(2)
memory usage: 18.6 KB
```

In [261…

```
#Heatmap to analyse the correlation between numerical features and Chance of Admit
df_corr = df.corr(numeric_only=True)
sns.heatmap(df_corr, annot=True)
plt.title('Correlation b/w Numeric Features')
plt.show();
```



- Confirming the inferences from pairplot, the correlation matrix also shows that **exam scores (CGPA/GRE/TOEFL)** have a strong positive correlation with chance of admit
- Infact, they are also highly correlated amongst themselves

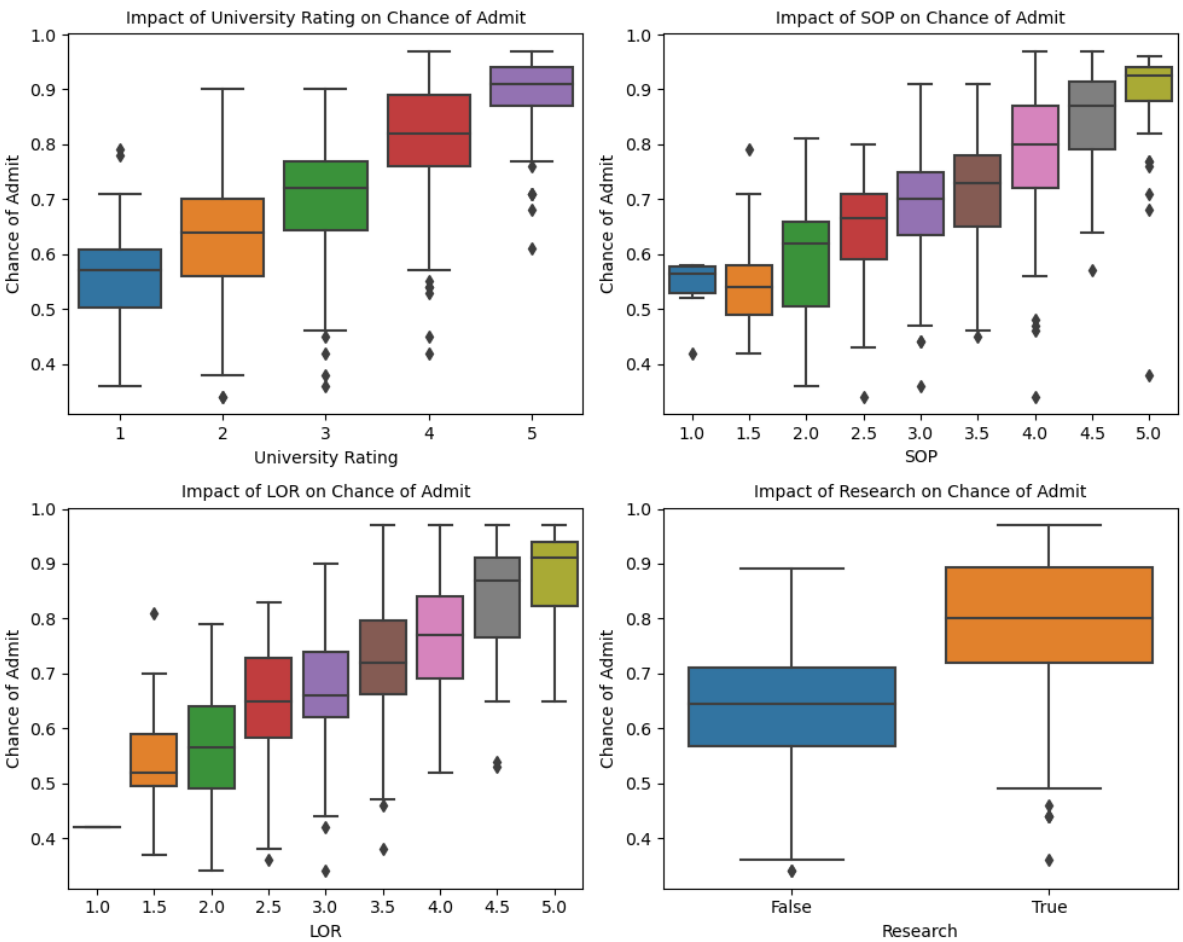
In [262…

```
# Boxplots to analyse the relationship between categorical variables and Chance of

cat_cols = df.select_dtypes(include=['bool','category']).columns.tolist()
```

```
plt.figure(figsize=(10,8))
i=1
for col in cat_cols:
  ax = plt.subplot(2,2,i)
  sns.boxplot(data = df, x=col, y='Chance of Admit')
  plt.title(f"Impact of {col} on Chance of Admit", fontsize=10)
  plt.xlabel(col)
  plt.ylabel('Chance of Admit')
  i+=1

plt.tight_layout()
plt.show();
```



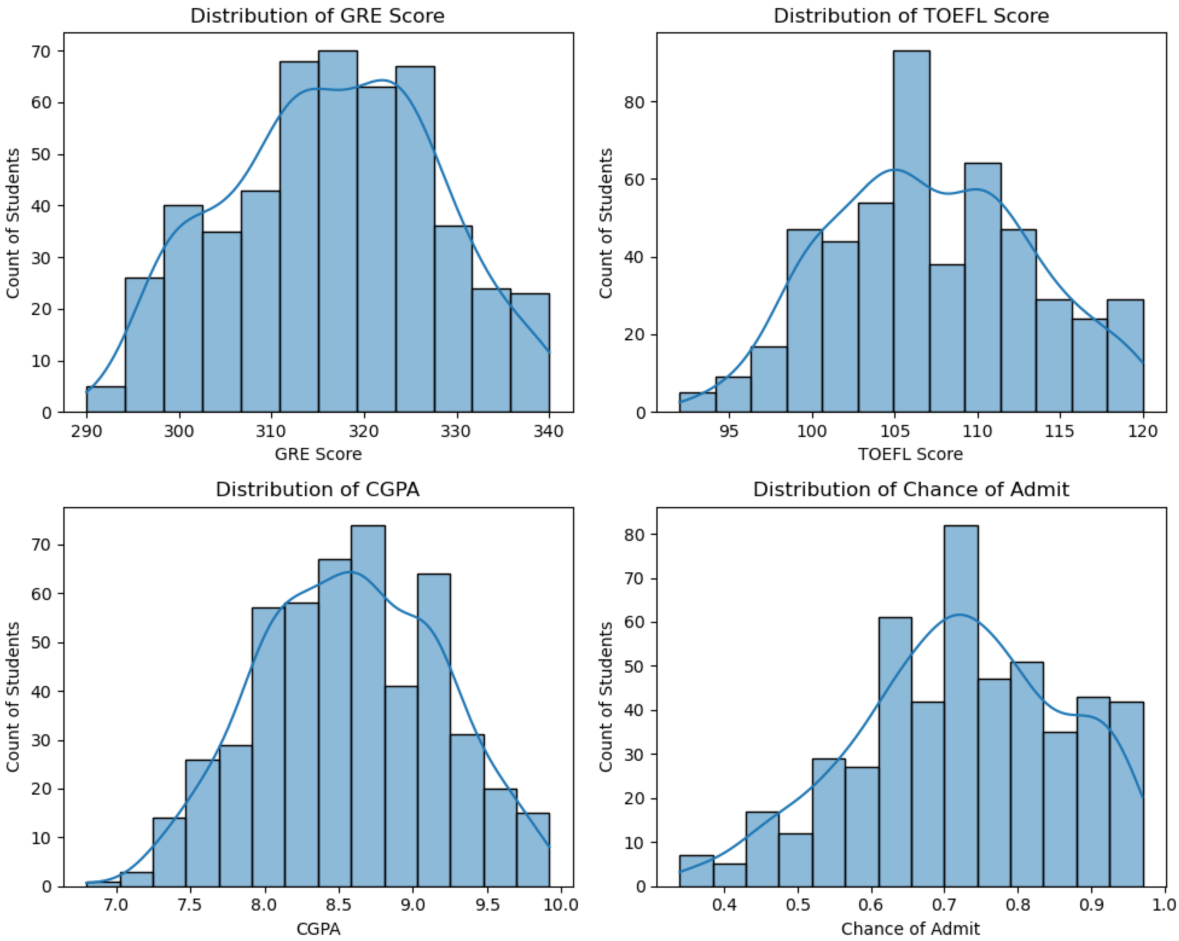
- As seen in the pairplot earlier, the categorical variables such as university ranking, research, quality of SOP and LOR also increase the chances of admit.

In [263…

```
# Distribution of continuous numerical features
numeric_cols = df.select_dtypes(include=['float','int']).columns.tolist()

plt.figure(figsize=(10,8))
i=1
for col in numeric_cols:
  ax=plt.subplot(2,2,i)
  sns.histplot(data=df[col], kde=True)
  plt.title(f'Distribution of {col}')
  plt.xlabel(col)
  plt.ylabel('Count of Students')
  i += 1

plt.tight_layout()
plt.show();
```



We can see the range of all the numerical attributes:

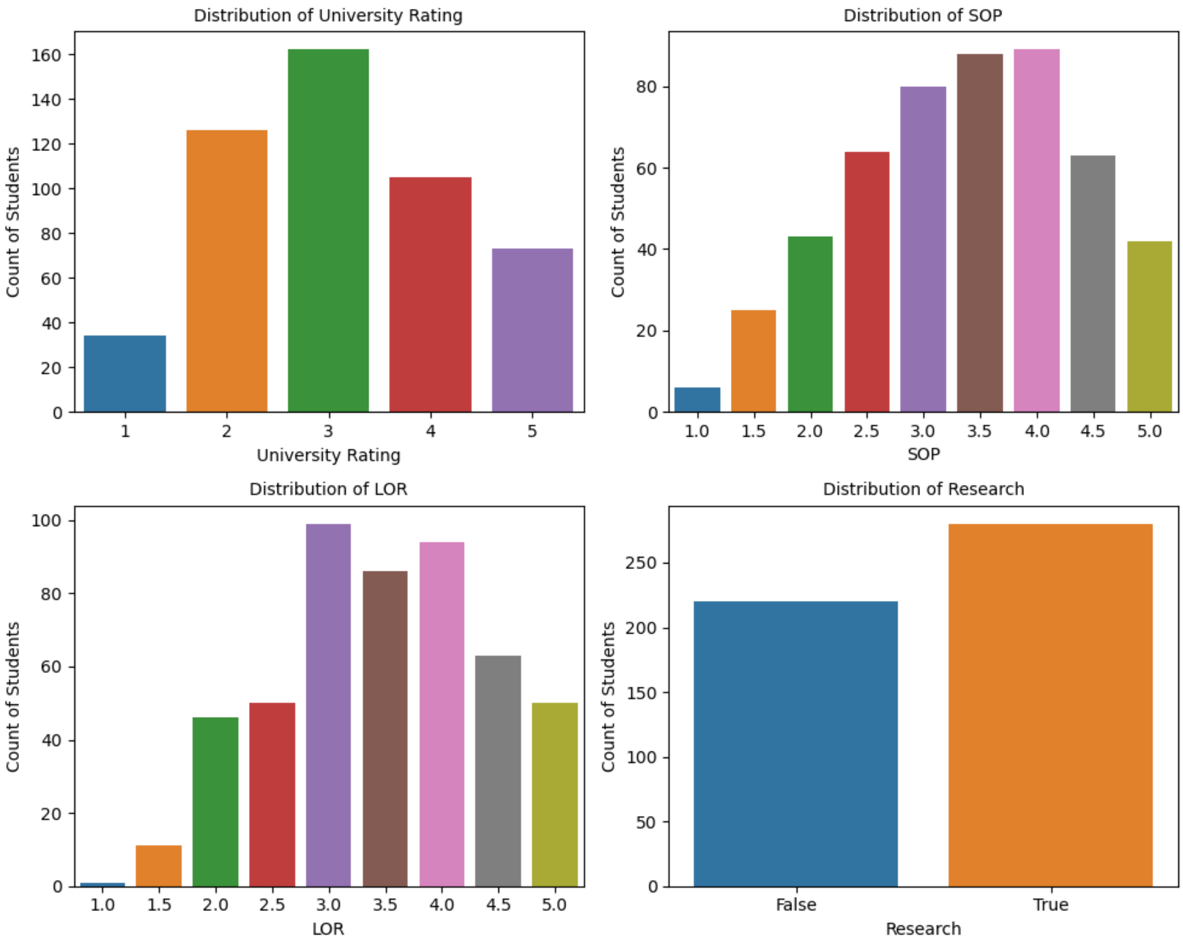
- GRE scores are between 290 and 340, with maximum students scoring in the range 310-330
- TOEFL scores are between 90 and 120, with maximum students scoring around 105
- CGPA ranges between 7 and 10, with maximum students scoring around 8.5
- Chance of Admit is a probability percentage between 0 and 1, with maximum students scoring around 70%-75%

In [264…

```
# Distribution of categorical variables
plt.figure(figsize=(10,8))
i=1

for col in cat_cols:
  ax = plt.subplot(2,2,i)
  sns.countplot(x=df[col])
  plt.title(f'Distribution of {col}', fontsize=10)
  plt.xlabel(col)
  plt.ylabel('Count of Students')
  i+=1

plt.tight_layout()
plt.show();
```



It can be observed that the most frequent value of categorical features is as following:

- University Rating: 3
- SOP: 3.5 & 4
- LOR: 3
- Research: True

Missing Values/Outliers/Duplicates Check

In [265…

```
#Check for missing values in all columns
df.isna().sum()
```

Out[265]:

```
GRE Score            0
TOEFL Score          0
University Rating    0
SOP                  0
LOR                  0
CGPA                 0
Research             0
Chance of Admit      0
dtype: int64
```

There are no missing values in the dataset

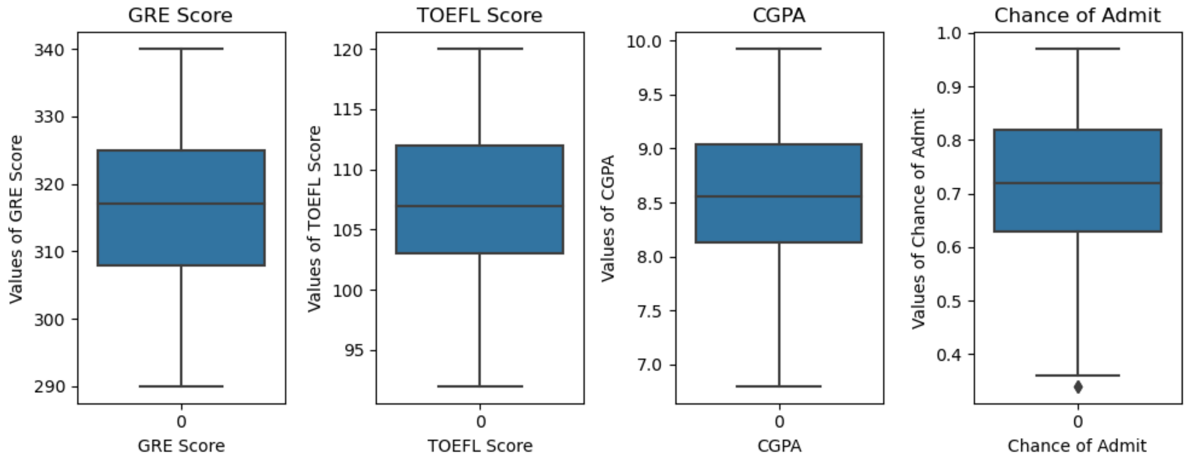
In [266…

```
# Check for outliers in numerical columns
plt.figure(figsize=(10,4))
i=1

for col in numeric_cols:
  ax = plt.subplot(1,4,i)
```

```
    sns.boxplot(df[col])
    plt.title(col)
    plt.xlabel(col)
    plt.ylabel(f'Values of {col}')
    i+=1

plt.tight_layout()
plt.show()
```



It can be observed that there are no outliers in the numeric columns.

In [267…

```
# Check for Duplicate rows
df[df.duplicated()].shape[0]
```

Out[267]: 0

There are no duplicate rows in the dataset

Model Building

Train-Test-Split

In [268…

```
numeric_cols.remove('Chance of Admit')
```

In [269…

```
# Separate predictor and target variables
x = df[numeric_cols + cat_cols]
y = df[['Chance of Admit']]
```

In [270…

```
x.head()
```

Out[270]:

	GRE Score	TOEFL Score	CGPA	University Rating	SOP	LOR	Research
0	337	118	9.65	4	4.5	4.5	True
1	324	107	8.87	4	4.0	4.5	True
2	316	104	8.00	3	3.0	3.5	True
3	322	110	8.67	3	3.5	2.5	True
4	314	103	8.21	2	2.0	3.0	False

In [271…
```
y.head()
```

Out[271]:

	Chance of Admit
0	0.92
1	0.76
2	0.72
3	0.80
4	0.65

Linear Regression model

In [272…
```
# Split the data into training and test data

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
                                                    random_state=42)


print(f'Shape of x_train: {x_train.shape}')
print(f'Shape of x_test: {x_test.shape}')
print(f'Shape of y_train: {y_train.shape}')
print(f'Shape of y_test: {y_test.shape}')
```

```
Shape of x_train: (400, 7)
Shape of x_test: (100, 7)
Shape of y_train: (400, 1)
Shape of y_test: (100, 1)
```

In [273…
```
x_train.head()
```

Out[273]:

	GRE Score	TOEFL Score	CGPA	University Rating	SOP	LOR	Research
428	316	103	8.74	2	2.0	4.5	False
490	307	105	8.12	2	2.5	4.5	True
53	324	112	8.10	4	4.0	2.5	True
336	319	110	8.79	3	3.0	2.5	False
154	326	108	8.89	3	3.0	3.5	False

In [274…
```
# Initialize a dictionary to store the label encoders
label_encoders = {}

# Loop through each categorical column and initialize the label encoder
for col in cat_cols:
    label_encoders[col] = LabelEncoder()
    x_train[col] = label_encoders[col].fit_transform(x_train[col])
    x_test[col] = label_encoders[col].fit_transform(x_test[col])
```

In [275…
```
x_cat_encoded = pd.concat([x_train, x_test])
x_cat_encoded.head(10)
```

Out[275]:

	GRE Score	TOEFL Score	CGPA	University Rating	SOP	LOR	Research
428	316	103	8.74	1	2	7	0
490	307	105	8.12	1	3	7	1
53	324	112	8.10	3	6	3	1
336	319	110	8.79	2	4	3	0
154	326	108	8.89	2	4	5	0
393	317	104	8.76	1	4	4	0
199	313	107	8.69	2	6	7	0
109	304	103	8.64	4	8	6	0
7	308	101	7.90	1	4	6	0
232	312	107	8.27	1	3	5	0

In [276…

```
#Initialising object of class MinMaxScaler() for Standardisation
scaler_x = MinMaxScaler()
```

In [277…

```
all_cols = x_train.columns
```

In [278…

```
#Transforming numeric columns of x_train and x_test
x_train[all_cols]=scaler_x.fit_transform(x_train[all_cols])
x_test[all_cols]=scaler_x.fit_transform(x_test[all_cols])
```

In [279…

```
x_train.head()
```

Out[279]:

	GRE Score	TOEFL Score	CGPA	University Rating	SOP	LOR	Research
428	0.489362	0.370370	0.621795	0.25	0.250	0.875	0.0
490	0.297872	0.444444	0.423077	0.25	0.375	0.875	1.0
53	0.659574	0.703704	0.416667	0.75	0.750	0.375	1.0
336	0.553191	0.629630	0.637821	0.50	0.500	0.375	0.0
154	0.702128	0.555556	0.669872	0.50	0.500	0.625	0.0

In [280…

```
x_test.head()
```

Out[280]:

	GRE Score	TOEFL Score	CGPA	University Rating	SOP	LOR	Research
129	0.86	0.928571	0.803030	1.00	1.000	1.000000	1.0
280	0.42	0.357143	0.534091	0.50	0.875	0.714286	1.0
440	0.30	0.428571	0.212121	0.25	0.375	0.000000	0.0
384	1.00	0.750000	0.950758	0.75	1.000	1.000000	1.0
225	0.12	0.250000	0.303030	0.25	0.375	0.285714	0.0

In [281…

```
#import and fit
model_lr = LinearRegression()
model_lr
```

Out[281]:

```
▾ LinearRegression
LinearRegression()
```

In [282…

```
# Fitting the model to the training data
model_lr.fit(x_train, y_train)
```

Out[282]:

```
▾ LinearRegression
LinearRegression()
```

In [283…

```
# Predicting values for the training and test data
y_pred_train = model_lr.predict(x_train)
y_pred_test = model_lr.predict(x_test)
```

In [284…

```
y_pred_train.shape
```

Out[284]:

```
(400, 1)
```

In [285…

```
x_train.shape
```

Out[285]:

```
(400, 7)
```

In [286…

```
from sklearn.metrics import r2_score
train_r2_score = r2_score(y_train,y_pred_train)
test_r2_score = r2_score(y_test,y_pred_test)
print(f"train_r2_score-----{train_r2_score}")
print(f"test_r2_score-----{test_r2_score}")
```

```
train_r2_score-----0.829322723369172
test_r2_score-----0.771360353542255
```

In [287…

```
# Evaluating the model using multiple loss functions
def model_evaluation(y_actual, y_forecast, model):
  n = len(y_actual)
  if len(model.coef_.shape)==1:
    p = len(model.coef_)
  else:
    p = len(model.coef_[0])
  MAE = np.round(mean_absolute_error(y_true=y_actual, y_pred=y_forecast),2)
  RMSE = np.round(mean_squared_error(y_true=y_actual,
                                     y_pred=y_forecast, squared=False),2)
  r2 = np.round(r2_score(y_true=y_actual, y_pred=y_forecast),2)
  adj_r2 = np.round(1 - ((1-r2)*(n-1)/(n-p-1)),2)
  return print(f"MAE: {MAE}\nRMSE: {RMSE}\nR2 Score: {r2}\nAdjusted R2: {adj_r2}")
```

In [288…

```
# Metrics for training data
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
model_evaluation(y_train.values, y_pred_train, model_lr)
```

```
MAE: 0.04
RMSE: 0.06
R2 Score: 0.83
Adjusted R2: 0.83
```

In [289…

```
#Metrics for test data
model_evaluation(y_test.values, y_pred_test, model_lr)
```

```
MAE: 0.05
RMSE: 0.07
R2 Score: 0.77
Adjusted R2: 0.75
```

Since there is negligible difference in the loss scores of training and test data, we can conclude that there is no overfitting of the model

- Mean Absolute Error of 0.04 shows that on an average, the absolute difference between the actual and predicted values of chance of admit is 4%
- Root Mean Square Err or of 0.06 means that on an average, the root of squared difference between the actual and predicted values is 6%
- R2 Score of 0.83 means that our model captures 8% variance in the data
- Adjusted R2 is an extension of R2 which shows how the number of features used changes the accuracy of the prediction

In [290…

```
# Model Coefficients

for feature,weight in zip(x_train.columns, model_lr.coef_[0]):
  print(f"Weight of {feature}: {np.round(weight,2)}")
```

```
Weight of GRE Score: 0.1
Weight of TOEFL Score: 0.08
Weight of CGPA: 0.35
Weight of University Rating: 0.02
Weight of SOP: 0.01
Weight of LOR: 0.07
Weight of Research: 0.02
```

In [291…

```
# Bias Term of the Model

model_lr.intercept_
```

Out[291]:

```
array([0.35435642])
```

- CGPA & GRE scores have the highest weight
- SOP, University rating, and research have the lowest weights

In [292…

```
import statsmodels.api as sm
x_train_sm = sm.add_constant(x_train) # to include and Learn bias
lr_sm=sm.OLS(y_train,x_train_sm)
fitted_model =  lr_sm.fit() # triggers the training process
# detailed summary
print(fitted_model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:        Chance of Admit   R-squared:                       0.829
Model:                            OLS   Adj. R-squared:                  0.826
Method:                 Least Squares   F-statistic:                     272.1
Date:                Mon, 25 Dec 2023   Prob (F-statistic):          3.33e-146
Time:                        20:54:51   Log-Likelihood:                 573.41
No. Observations:                 400   AIC:                            -1131.
Df Residuals:                     392   BIC:                            -1099.
Df Model:                           7
Covariance Type:            nonrobust
==================================================================================
===
                        coef    std err          t      P>|t|      [0.025      0.9
75]
----------------------------------------------------------------------------------
---
const                 0.3544      0.010     35.147      0.000       0.335       0.
374
GRE Score             0.1003      0.026      3.893      0.000       0.050       0.
151
TOEFL Score           0.0797      0.026      3.024      0.003       0.028       0.
131
CGPA                  0.3537      0.033     10.633      0.000       0.288       0.
419
University Rating     0.0194      0.016      1.185      0.237      -0.013       0.
052
SOP                   0.0084      0.020      0.428      0.669      -0.030       0.
047
LOR                   0.0744      0.018      4.131      0.000       0.039       0.
110
Research              0.0247      0.007      3.476      0.001       0.011       0.
039
==============================================================================
Omnibus:                       94.166   Durbin-Watson:                   1.943
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              231.309
Skew:                          -1.158   Prob(JB):                     5.92e-51
Kurtosis:                       5.918   Cond. No.                         23.1
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly s
pecified.
```

Testing Assumptions of Linear Regression Model

Multicolinearity Check

In [293…

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
vif = pd.DataFrame()
vif['Variable'] = x_train.columns
vif['VIF'] = [variance_inflation_factor(x_train.values, i) for i in range(x_train.s
vif
```

Out[293]:

	Variable	VIF
0	GRE Score	23.556132
1	TOEFL Score	26.315538
2	CGPA	39.256162
3	University Rating	10.850634
4	SOP	18.339464
5	LOR	15.034711
6	Research	3.304957

We see that almost all the variables (excluding research) have a very high level of colinearity. This was also observed from the correlation heatmap which showed strong positive correlation between GRE score, TOEFL score and CGPA.

Mean of Residuals

In [307…

```
residuals = y_test.values - y_pred_test
residuals.reshape((-1,))
print('Mean of Residuals: ', residuals.mean())
```

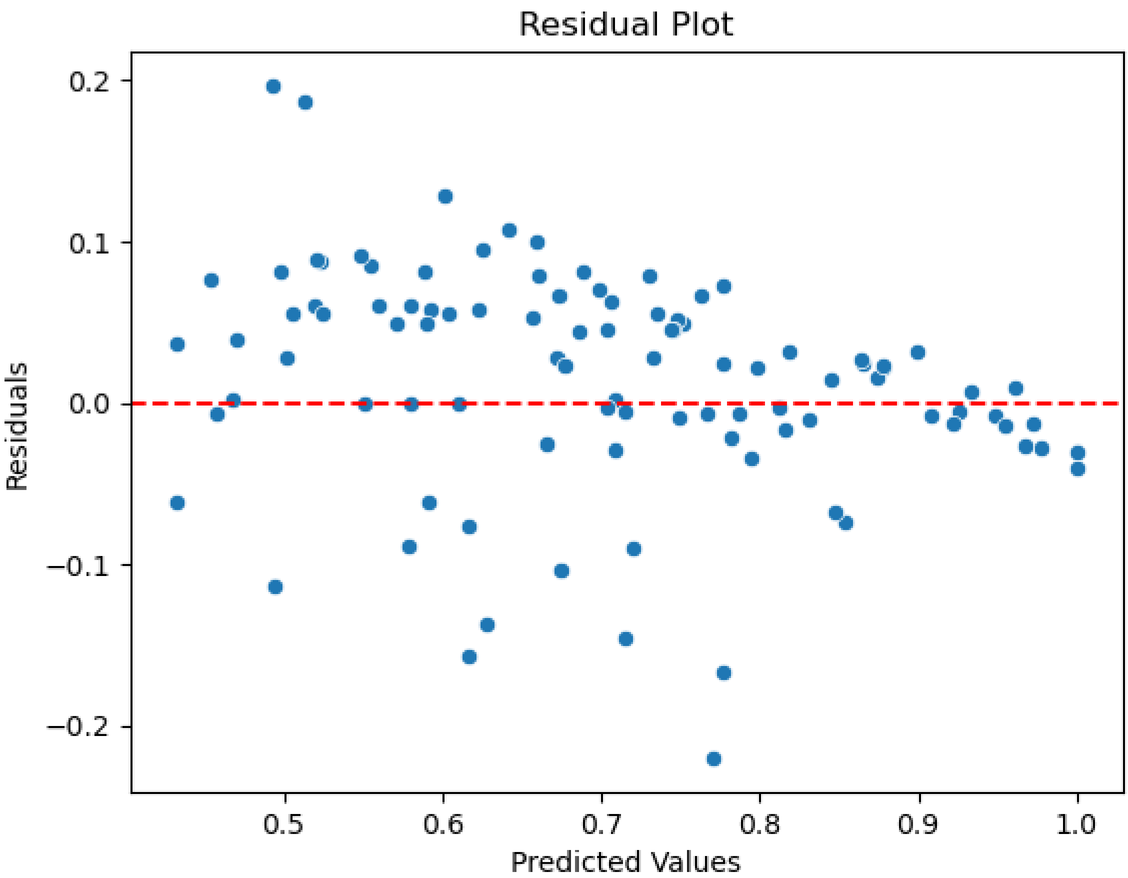
```
Mean of Residuals:  0.014298544942865601
```

Since the mean of residuals is close to 0, we can say that the model is unbiased

Linearity of variables

In [308…

```
sns.scatterplot(x = y_pred_test.reshape((-1,)), y=residuals.reshape((-1,)))
plt.title('Residual Plot')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.axhline(y=0, color='r', linestyle='--')
plt.show();
```



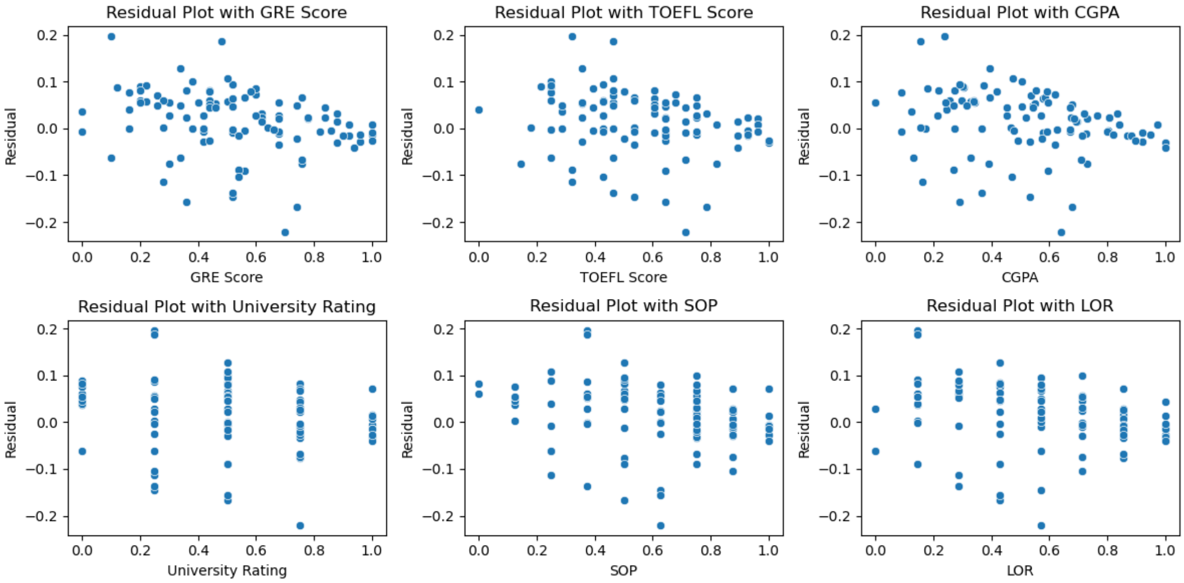
Since the residual plot shows no clear pattern or trend in residuals, we can conclude that linearity of variables exists

Homoscedasticity

In [309…

```
# Scatterplot of residuals with each independent variable to check for Homoscedasti
plt.figure(figsize=(12,6))
i=1
for col in x_test.columns[:-1]:
  ax = plt.subplot(2,3,i)
  sns.scatterplot(x=x_test[col].values.reshape((-1,)), y=residuals.reshape((-1,)))
  plt.title(f'Residual Plot with {col}')
  plt.xlabel(col)
  plt.ylabel('Residual')
  i+=1

plt.tight_layout()
plt.show();
```



In [314…

```
# Performing the Goldfeld-Quandt test to check for Homoscedasticity -
from statsmodels.compat import lzip
import statsmodels.stats.api as sms

name = ['F statistic', 'p-value']
test = sms.het_goldfeldquandt(y_train, x_train_sm)
lzip(name, test)
```

Out[314]:

```
[('F statistic', 1.077299427998724), ('p-value', 0.30323276479815664)]
```

From the goldfeld-quandt test:

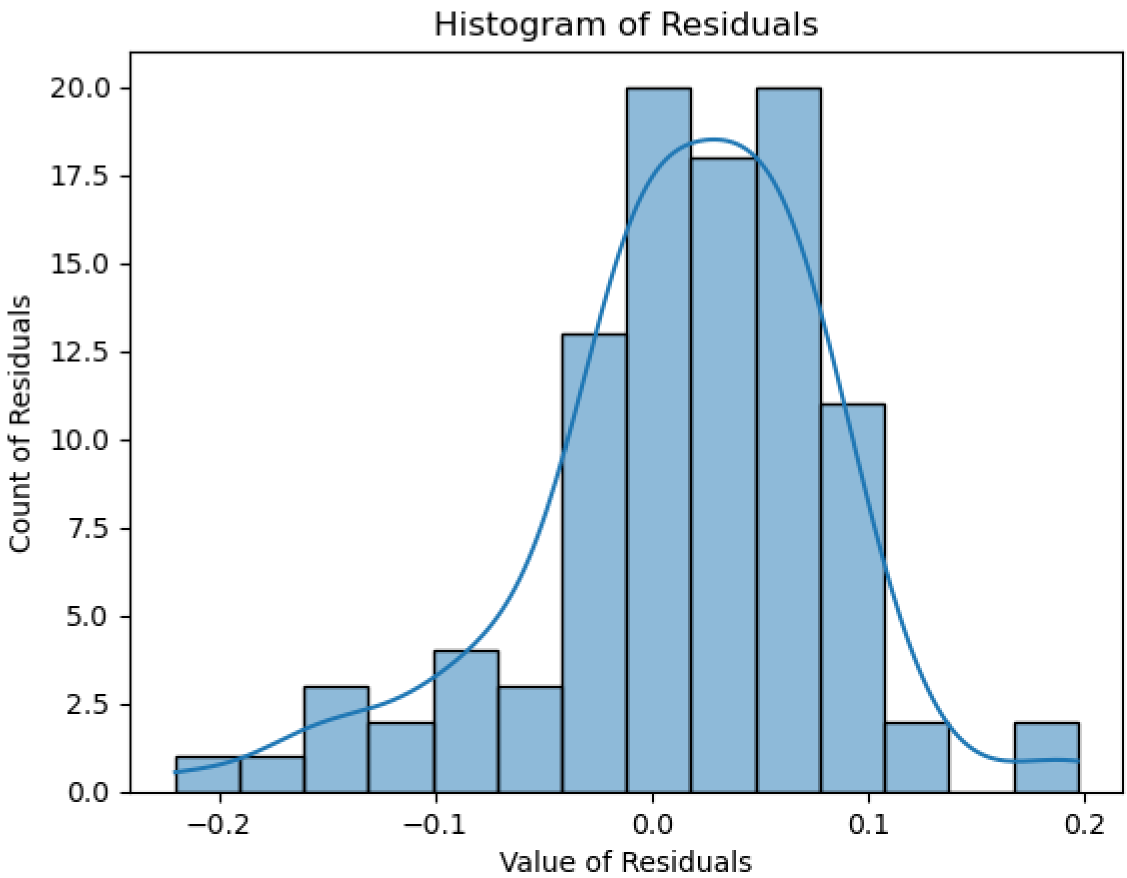
- F Statistic comes out to be 1.00 => Implying minimal difference in variance between groups
- p-value of 0.303 indicates that this difference is statistically significant at conventional levels of significance (e.g., 0.05).

Therefore, we accept the null hypothesis of homoscedasticity, and conclude that there is no strong evidence of heteroscedasticity in the data.

Normality of Residual

In [310…

```
#Histogram of Residuals
sns.histplot(residuals.reshape((-1,)), kde=True)
plt.title('Histogram of Residuals')
plt.xlabel('Value of Residuals')
plt.ylabel('Count of Residuals')
plt.show();
```



In [312…

```
from scipy import stats
res = stats.shapiro(residuals)
res.statistic
```

Out[312]:

```
0.9530214667320251
```

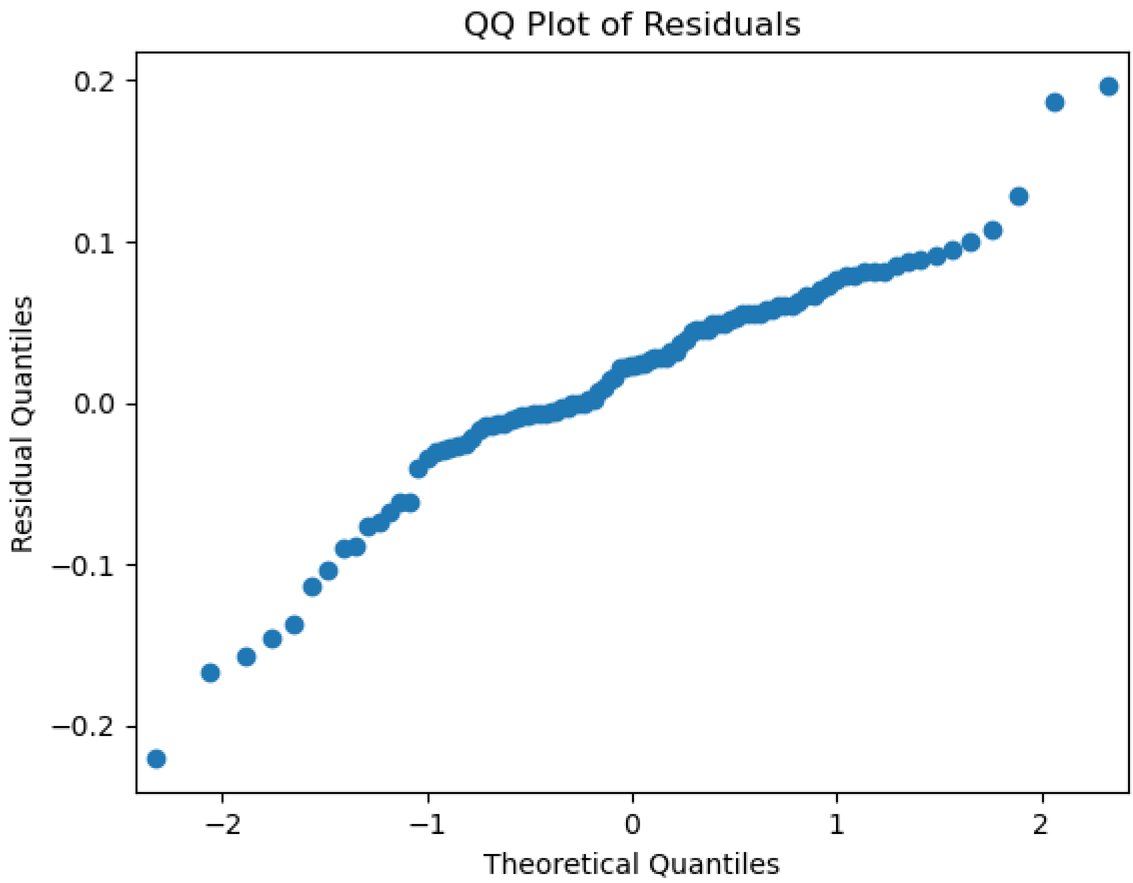
Closer the value to 1, more is the normality.

In this case, a value of 0.85 denotes a high level of normality for the error distribuiton

The histogram shows distribution of residuals is a normal distribution

In [311…

```
# QQ-Plot of residuals
sm.qqplot(residuals.reshape((-1,)))
plt.title('QQ Plot of Residuals')
plt.ylabel('Residual Quantiles')
plt.show();
```



The QQ plot shows that residuals are slightly deviating from the straight diagonal.

Model performance evaluation

Lasso and Ridge Regression

In [315…

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

In [316…

```
# Initialising instance of Ridge and Lasso classes
model_ridge = Ridge()
model_lasso = Lasso()
```

In [317…

```
# Fitting the models to training data
model_ridge.fit(x_train, y_train)
model_lasso.fit(x_train, y_train)
```

Out[317]:

```
▾ Lasso
Lasso()
```

In [318…

```
# Predicting values for train and test data

y_train_ridge = model_ridge.predict(x_train)
y_test_ridge = model_ridge.predict(x_test)

y_train_lasso = model_lasso.predict(x_train)
y_test_lasso = model_lasso.predict(x_test)
```

In [319…

```
# Evaluating Model Performance
print('Ridge Regression Training Accuracy\n')
model_evaluation(y_train.values, y_train_ridge, model_ridge)
print('\n\nRidge Regression Test Accuracy\n')
model_evaluation(y_test.values, y_test_ridge, model_ridge)
print('\n\nLasso Regression Training Accuracy\n')
model_evaluation(y_train.values, y_train_lasso, model_lasso)
print('\n\nLasso Regression Test Accuracy\n')
model_evaluation(y_test.values, y_test_lasso, model_lasso)
```

```
Ridge Regression Training Accuracy

MAE: 0.04
RMSE: 0.06
R2 Score: 0.83
Adjusted R2: 0.83


Ridge Regression Test Accuracy

MAE: 0.05
RMSE: 0.07
R2 Score: 0.78
Adjusted R2: 0.76


Lasso Regression Training Accuracy

MAE: 0.11
RMSE: 0.14
R2 Score: 0.0
Adjusted R2: -0.02


Lasso Regression Test Accuracy

MAE: 0.12
RMSE: 0.15
R2 Score: -0.0
Adjusted R2: -0.08
```

Identifying Best Model

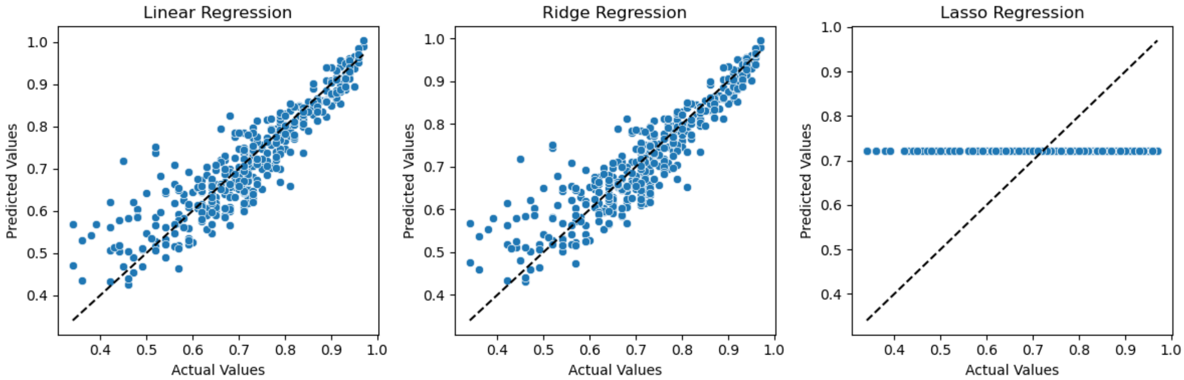
In [320…

```
# Actual v/s Predicted values for training data

actual_values = y_train.values.reshape((-1,))
predicted_values = [y_pred_train.reshape((-1,)), y_train_ridge.reshape((-1,)), y_tr
model = ['Linear Regression', 'Ridge Regression', 'Lasso Regression']

plt.figure(figsize=(12,4))
i=1
for preds in predicted_values:
  ax = plt.subplot(1,3,i)
  sns.scatterplot(x=actual_values, y=preds)
  plt.plot([min(actual_values),max(actual_values)], [min(actual_values),max(actual_
  plt.xlabel('Actual Values')
  plt.ylabel('Predicted Values')
  plt.title(model[i-1])
  i+=1

plt.tight_layout()
plt.show();
```



We can observe that both Linear Regression and Ridge Regression have similar accuracy while Lasso regression has oversimplified the model.

This is the reason that the r2 score of Lasso regression is 0. It doesn't capture any variance in the target variable. It has predicted the same value across all instances.

Insights & Recommendations

Insights:

- The distribution of target variable (chances of admit) is left-skewed
- Exam scores (CGPA/GRE/TOEFL) have a strong positive correlation with chance of admit. These variables are also highly correlated amongst themselves
- the categorical variables such as university ranking, research, quality of SOP and LOR also show an upward trend for chances of admit.
- From the model coefficients (weights), we can conclude that CGPA is the most significant predictor variable while SOP/University Rating are the least significant
- Both Linear Regression and Ridge Regression models, which are our best models, have captured upto 82% of the variance in the target variable (chance of admit). Due to high colinearity among the predictor variables, it is difficult to achieve better results.
- Other than no multicolinearity, the predictor variables have met the conditions required for Linear Regression - mean of residuals is close to 0, linearity of variables, normality of residuals and homoscedasticity is established.

Recommendations:

- Since all the exam scores are highly correlated, it is recommended to add more independent features for better prediction.
- Examples of other independent variables could be work experience, internships, mock interview performance, extracurricular activities or diversity variables

In [ ]: