In [111]:
```
# importing the necessery Libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split,KFold
from sklearn.preprocessing import MinMaxScaler,StandardScaler
from sklearn.linear_model import Ridge,Lasso
from sklearn.pipeline import make_pipeline
from sklearn.metrics import r2_score,mean_squared_error,mean_absolute_erro
from statsmodels.stats.stattools import durbin_watson


import statsmodels.api as sm  # to train the model
from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.stats.api as sms # to check the hetroscaditisity
from scipy.stats import shapiro # to check the normality

plt.style.use('default')
```

executed in 6ms, finished 14:04:18 2024-02-19

In [112]:
```
# importing the dataset
df= pd.read_csv('original_Jamboree_Admission.csv')
df.head(15)
```

executed in 25ms, finished 14:04:19 2024-02-19

Out[112]:

	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
5	6	330	115	5	4.5	3.0	9.34	1	0.90
6	7	321	109	3	3.0	4.0	8.20	1	0.75
7	8	308	101	2	3.0	4.0	7.90	0	0.68
8	9	302	102	1	2.0	1.5	8.00	0	0.50
9	10	323	108	3	3.5	3.0	8.60	0	0.45
10	11	325	106	3	3.5	4.0	8.40	1	0.52
11	12	327	111	4	4.0	4.5	9.00	1	0.84
12	13	328	112	4	4.0	4.5	9.10	1	0.78
13	14	307	109	3	4.0	3.0	8.00	1	0.62
14	15	311	104	3	3.5	2.0	8.20	1	0.61

1.Basic Analysis

A.) Shape,Statistical summary

In [113]:
```
# information cheking
df.info()
```
executed in 9ms, finished 14:04:22 2024-02-19

```
<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
```

In [ ]:

1. There are zero null values
2. There are no missing Values
3. Shape of data is 500 x 8

```
 since the serial number will reduandent Column will not lead to any
informaion so we will drop this column
```

In [114]:
```
df.drop(['Serial No.'],axis=1,inplace=True)
```
executed in 4ms, finished 14:04:23 2024-02-19

Chaning the naes to more short name

In [115]:
```
df=df.rename(columns={'Chance of Admit ' : 'Chance', 'LOR ':'LOR','Univers
```
executed in 4ms, finished 14:04:24 2024-02-19

In [116]:
```
df.describe()
```
executed in 19ms, finished 14:04:24 2024-02-19

Out[116]:

	GRE	TOEFL	UR	SOP	LOR	CGPA	Research	
count	500.000000	500.000000	500.000000	500.000000	500.00000	500.000000	500.000000	5
mean	316.472000	107.192000	3.114000	3.374000	3.48400	8.576440	0.560000	
std	11.295148	6.081868	1.143512	0.991004	0.92545	0.604813	0.496884	
min	290.000000	92.000000	1.000000	1.000000	1.00000	6.800000	0.000000	
25%	308.000000	103.000000	2.000000	2.500000	3.00000	8.127500	0.000000	
50%	317.000000	107.000000	3.000000	3.500000	3.50000	8.560000	1.000000	
75%	325.000000	112.000000	4.000000	4.000000	4.00000	9.040000	1.000000	
max	340.000000	120.000000	5.000000	5.000000	5.00000	9.920000	1.000000	

In [117]:
```
# Converting the Chance into categorical valribale for further analysis
# converting all into categories like 30-40,40-50,60-70,50-60,70-80,80-90,
df['chances%']=pd.cut(df['Chance'],bins=[i for i in np.arange(0.3,1.1,0.1)
```
executed in 7ms, finished 14:04:25 2024-02-19

In [118]:
```
df
```
executed in 13ms, finished 14:04:25 2024-02-19

Out[118]:

	GRE	TOEFL	UR	SOP	LOR	CGPA	Research	Chance	chances%
0	337	118	4	4.5	4.5	9.65	1	0.92	90-100
1	324	107	4	4.0	4.5	8.87	1	0.76	70-80
2	316	104	3	3.0	3.5	8.00	1	0.72	70-80
3	322	110	3	3.5	2.5	8.67	1	0.80	70-80
4	314	103	2	2.0	3.0	8.21	0	0.65	60-70
...	...	...	...	...	...	...	...	...	...
495	332	108	5	4.5	4.0	9.02	1	0.87	80-90
496	337	117	5	5.0	5.0	9.87	1	0.96	90-100
497	330	120	5	4.5	5.0	9.56	1	0.93	90-100
498	312	103	4	4.0	5.0	8.43	0	0.73	70-80
499	327	113	4	4.5	4.5	9.04	0	0.84	80-90

500 rows × 9 columns

In [119]:

```
# calculating the min and maximum values of each column for furthur analys
df.groupby('chances%')[['GRE','TOEFL','UR','SOP','LOR','CGPA','Research','
```

executed in 24ms, finished 14:04:26 2024-02-19

Out[119]:

	GRE		TOEFL		UR		SOP		LOR		CGPA		Research		Ch
	min	max	min	max	min	max	min	max	min	max	min	max	min	max	mi
chances%															
30-40	295	315	96	105	1	3	2.0	5.0	1.5	3.5	6.80	8.03	0	1	0.3
40-50	290	323	93	110	1	4	1.0	4.0	1.0	3.5	7.20	8.60	0	1	0.4
50-60	295	325	92	112	1	4	1.0	4.5	1.5	4.5	7.23	8.92	0	1	0.5
60-70	293	327	95	115	1	5	1.5	5.0	1.5	5.0	7.40	9.22	0	1	0.6
70-80	300	334	98	116	1	5	1.5	5.0	2.0	5.0	7.89	9.16	0	1	0.7
80-90	312	340	104	120	2	5	2.0	5.0	1.5	5.0	8.44	9.70	0	1	0.8
90-100	320	340	110	120	4	5	3.0	5.0	3.5	5.0	9.06	9.92	1	1	0.9

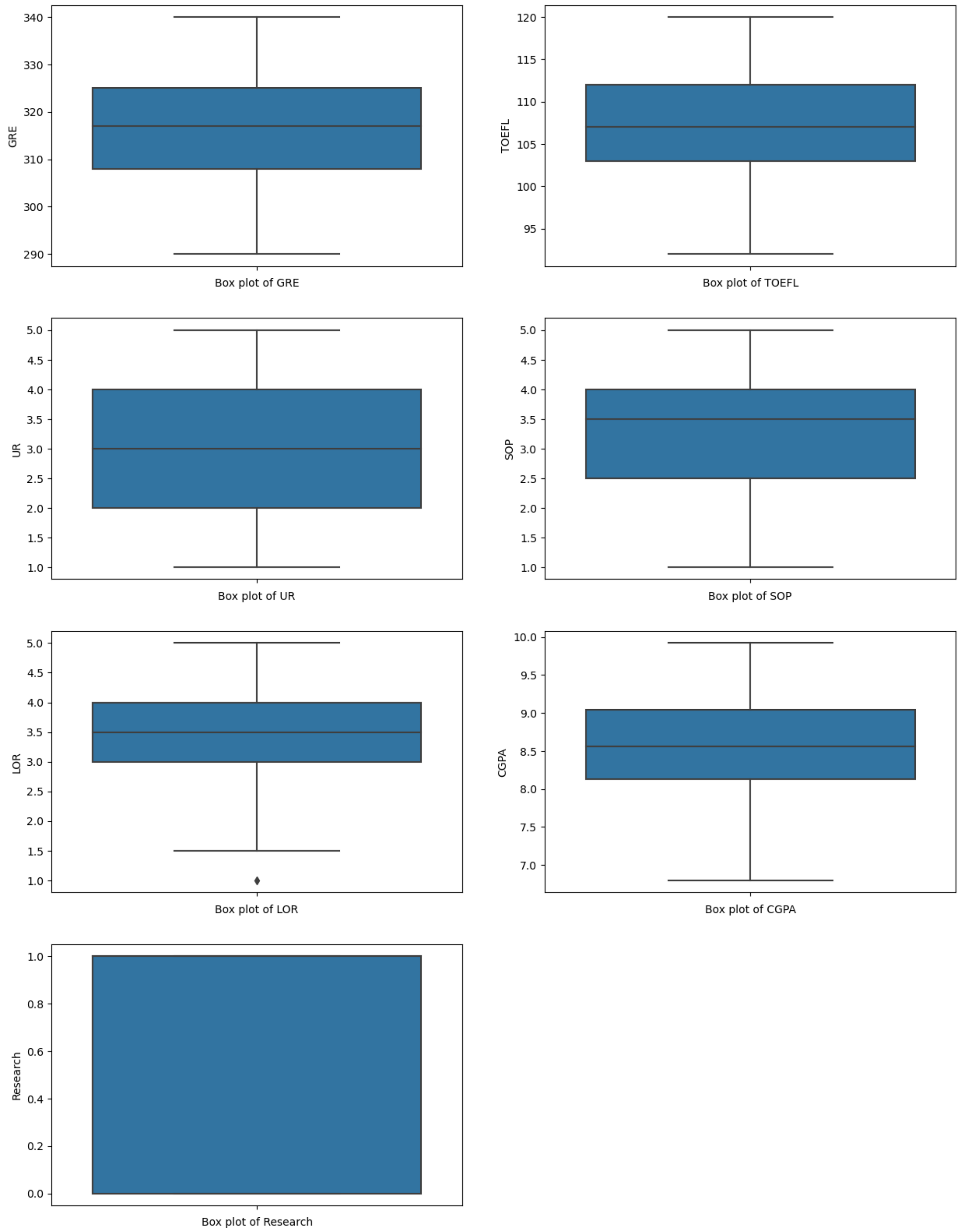
Results from this Summary

1. for GRE score above 300 there are 80 % chances.
2. Research work is must for increasing you chances above 90%
3. for 90% chances your CGPA must be above 9.

B) Checking of Outliers

In [122]:
```
plt.figure(figsize=(15,20))
count=1
for i in ['GRE','TOEFL','UR','SOP','LOR','CGPA','Research']:
    plt.subplot(4,2,count)
    sns.boxplot(data=df, y=i)
    plt.xlabel(f'Box plot of {i}')
    count+=1
plt.show( )
```

executed in 497ms, finished 14:05:12 2024-02-19

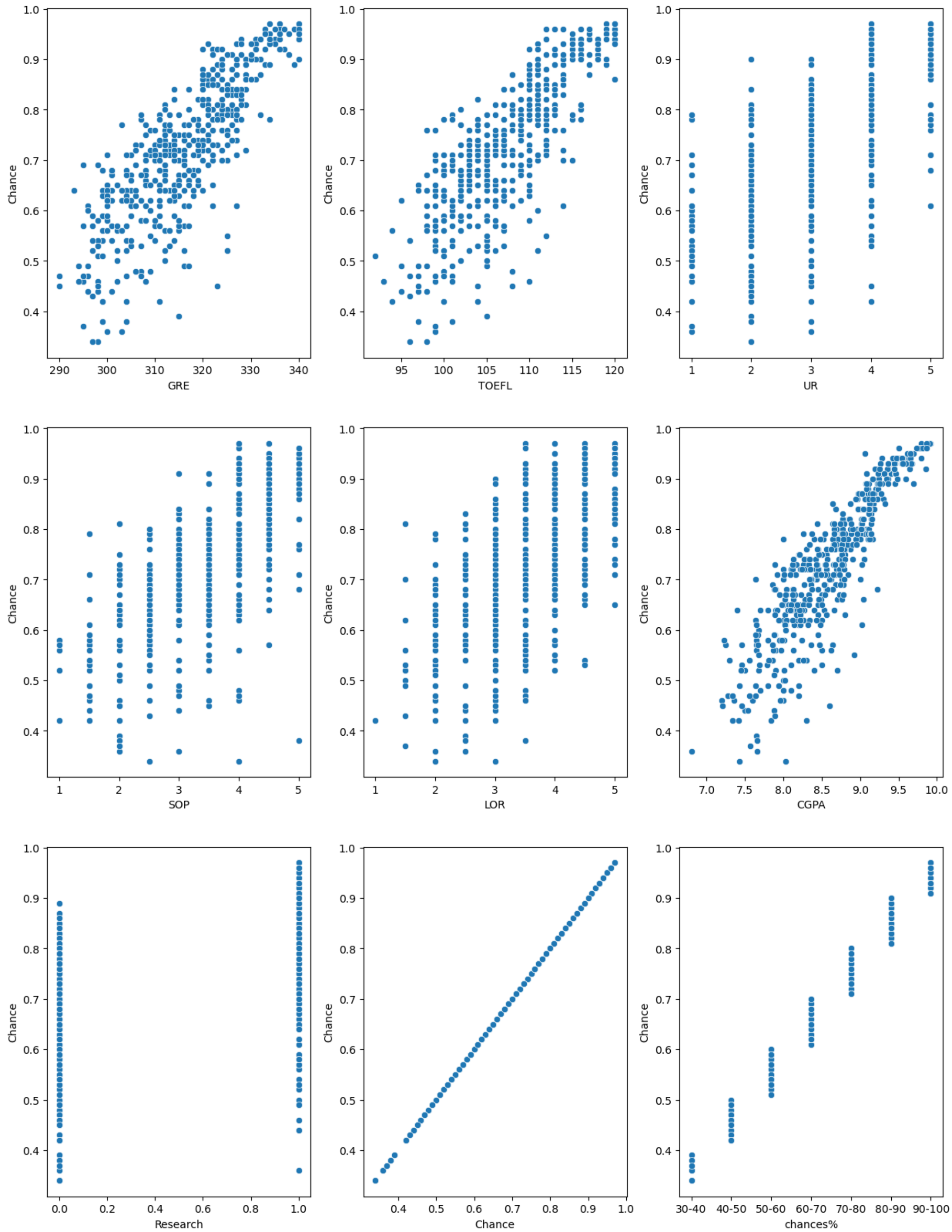


- There are no outliers present in the Dataset

2. Univariate and Bivariate Plots

In [125]:
```
plt.figure(figsize=(15,20))
place=1
for i in df.columns:
    plt.subplot(3,3,place)
    sns.scatterplot(data=df,y='Chance',x=i)
    place+=1
plt.show()
```

executed in 877ms, finished 14:06:10 2024-02-19

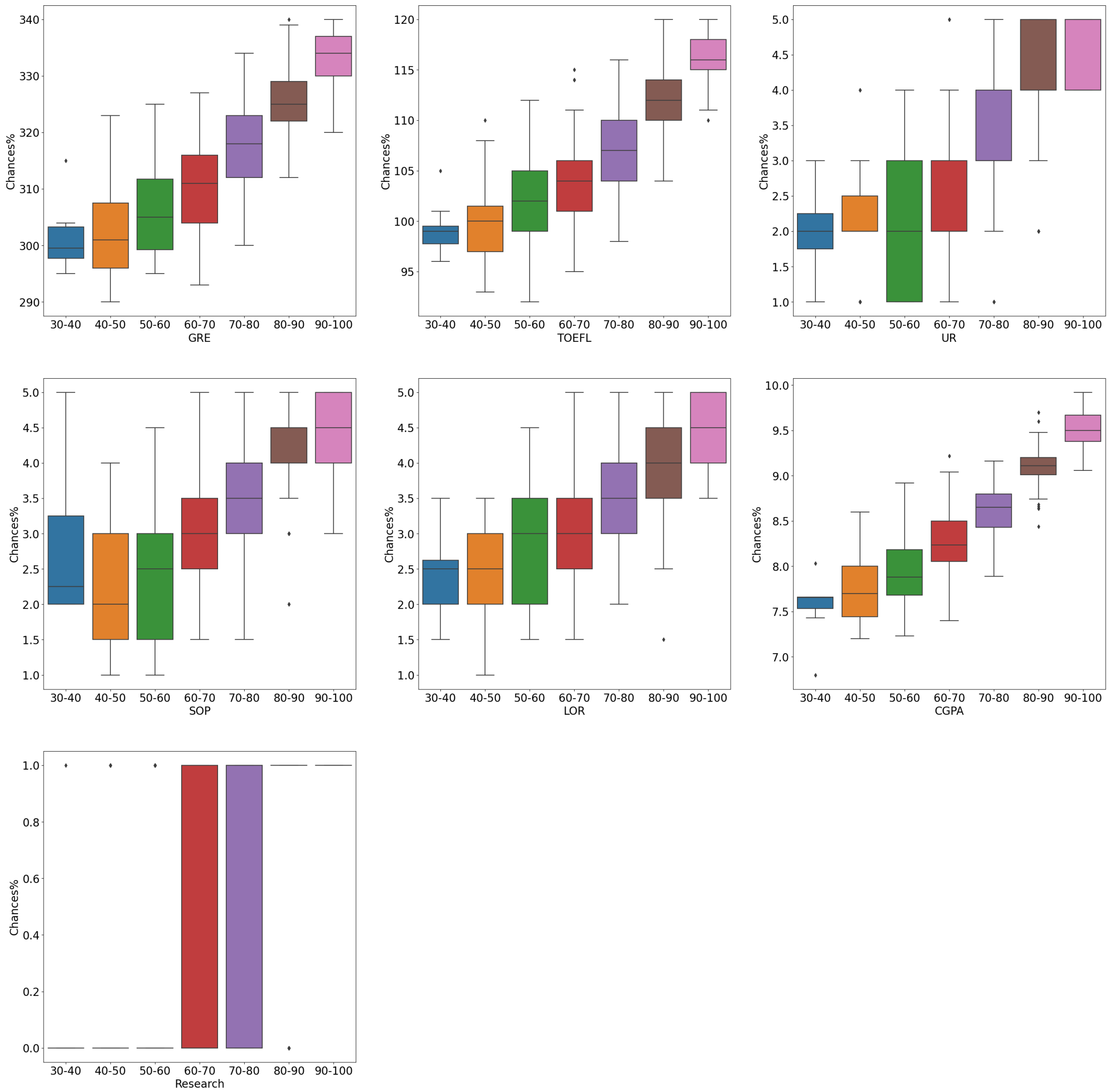


1. Chances of getting selected are proportional to GRE and TOFEL score.

In [64]:

```
plt.figure(figsize=(35,35))
place=1
for i in df.columns[:-2]:
    plt.subplot(3,3,place)
    ax = sns.boxplot(data=df,y=i,x='chances%')
    ax.set_xlabel(xlabel=i,fontsize = 20)
    ax.set_ylabel(ylabel='Chances%',fontsize = 20)
    plt.xticks(fontsize=20)
    plt.yticks(fontsize=20)
    place+=1
plt.show()
```

executed in 1.09s, finished 13:58:23 2024-02-19

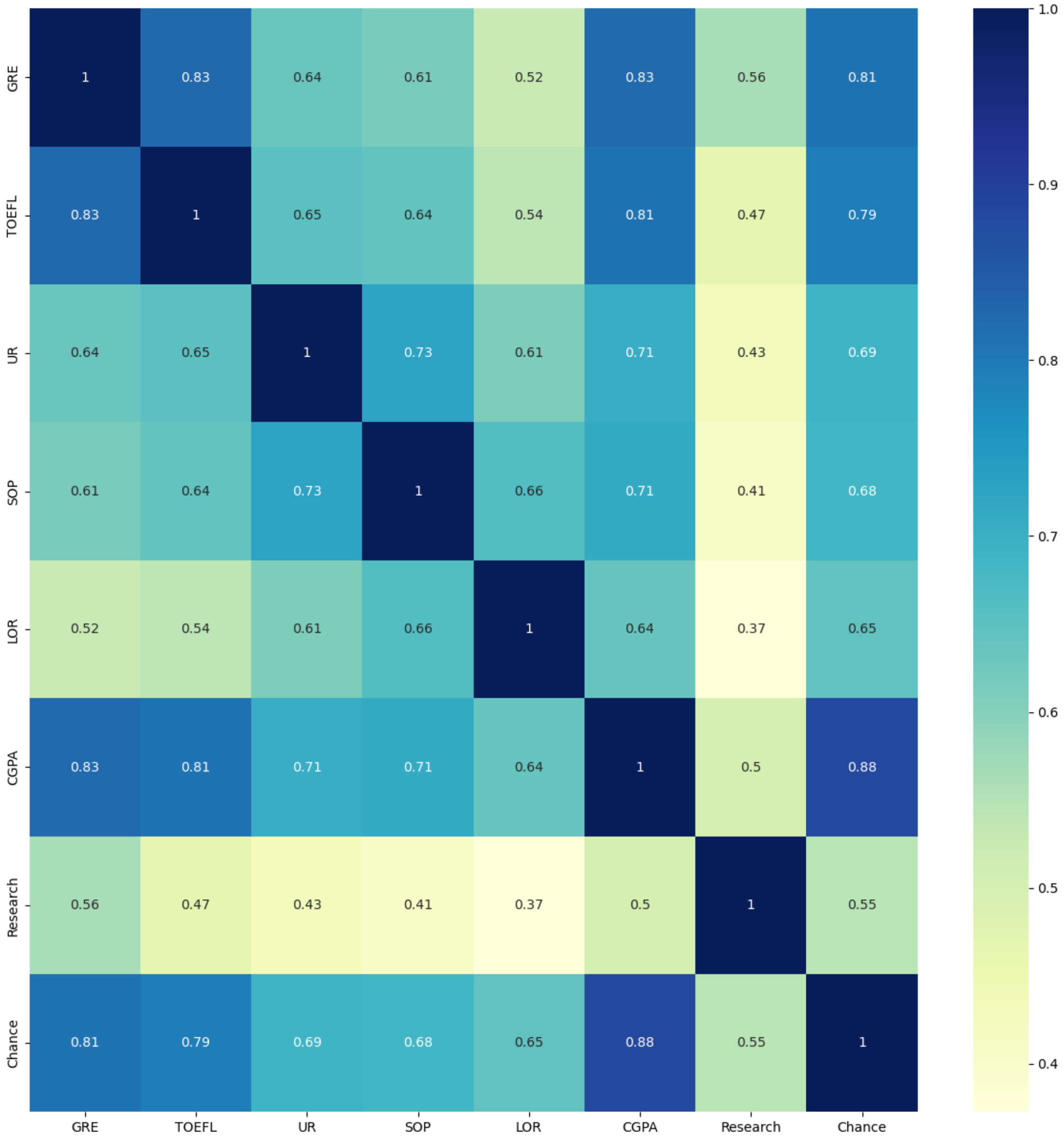


Results are moreover the same as previous

Checking the correlation

In [127]:
```
plt.figure(figsize=(15, 15))
sns.heatmap(df.select_dtypes(include='number').corr(),cmap='YlGnBu',annot=
```
executed in 320ms, finished 14:08:47 2024-02-19

Out[127]: <Axes: >



- CGPA and chances are most effect on chances of getting admission.
- after CGPA the GRE and TOFEL score has the most Chances

3. Data Preprocessing

A) Duplicates Values Check

In [66]:
```
df.drop_duplicates()
```
executed in 17ms, finished 13:58:24 2024-02-19

Out[66]:

	GRE	TOEFL	UR	SOP	LOR	CGPA	Research	Chance	chances%
0	337	118	4	4.5	4.5	9.65	1	0.92	90-100
1	324	107	4	4.0	4.5	8.87	1	0.76	70-80
2	316	104	3	3.0	3.5	8.00	1	0.72	70-80
3	322	110	3	3.5	2.5	8.67	1	0.80	70-80
4	314	103	2	2.0	3.0	8.21	0	0.65	60-70
...	...	...	...	...	...	...	...	...	...
495	332	108	5	4.5	4.0	9.02	1	0.87	80-90
496	337	117	5	5.0	5.0	9.87	1	0.96	90-100
497	330	120	5	4.5	5.0	9.56	1	0.93	90-100
498	312	103	4	4.0	5.0	8.43	0	0.73	70-80
499	327	113	4	4.5	4.5	9.04	0	0.84	80-90

500 rows × 9 columns

There are no Duplicates Present in Data

B) Missing values check and Treatment

In [67]:
```
df.info()
```
executed in 9ms, finished 13:58:24 2024-02-19

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 9 columns):
 #   Column    Non-Null Count  Dtype   
---  ------    --------------  -----   
 0   GRE       500 non-null    int64   
 1   TOEFL     500 non-null    int64   
 2   UR        500 non-null    int64   
 3   SOP       500 non-null    float64 
 4   LOR       500 non-null    float64 
 5   CGPA      500 non-null    float64 
 6   Research  500 non-null    int64   
 7   Chance    500 non-null    float64 
 8   chances%  500 non-null    category
dtypes: category(1), float64(4), int64(4)
memory usage: 32.2 KB
```

since all column has 500 values **There are no missing values present in data**

C) Feature Engineering and Data Preprocessing

In [68]:
```
df_new=df.copy()
```
executed in 4ms, finished 13:58:24 2024-02-19

In [69]:
```
data = df_new.drop(['chances%'],axis=1)
```
executed in 5ms, finished 13:58:24 2024-02-19

In [70]:
```
x= data.drop(['Chance'],axis=1)
y=data['Chance']
```
executed in 4ms, finished 13:58:24 2024-02-19

In [71]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_
```
executed in 10ms, finished 13:58:24 2024-02-19

In [72]:
```
scaler=StandardScaler()
```
executed in 4ms, finished 13:58:24 2024-02-19

In [73]:
```
scaler.fit(x_train,y_train)
```
executed in 7ms, finished 13:58:24 2024-02-19

Out[73]:
```
▾ StandardScaler i ? (https://scikit-learn.org/1.4/modules/generated/sklearn.preprocessing.StandardSc
StandardScaler()
```

In [74]:
```
x_tr_sc = pd.DataFrame(scaler.transform(x_train),columns=x_train.columns)
```
executed in 6ms, finished 13:58:24 2024-02-19

In [75]:
```
x_te_sc = pd.DataFrame(scaler.transform(x_test),columns=x_test.columns)
```
executed in 5ms, finished 13:58:24 2024-02-19

4. Model Building

A) Model Building with Linear Regression model

In [76]:

```
 #Print all the performance metrics for linear regression models
def get_metrics(x, y_true, model,r = None):
    """Calculate and print MAE, RMSE, R2, and Adjusted R2."""
    y_pred = model.predict(x)

    MAE = mean_absolute_error(y_true, y_pred)
    MSE = mean_squared_error(y_true, y_pred)
    RMSE = np.sqrt(MSE)
    R2 = r2_score(y_true, y_pred)
    adjusted_r2 = 1 - (1 - R2) * (len(y_pred) - 1) / (len(y_pred) - x.shap
    if r != None:
        print(f'-----------------Regression Type: {r}----------------')

    print(f'MAE:{MAE : 0.5f}')
    print(f'RMSE:{RMSE : 0.5f}')
    print(f'R2:{R2: 0.5f}')
    print(f'Adjusted R2:{adjusted_r2: 0.5f}')

    cols = list(np.array(x_te_sc.columns))
    coef_df = pd.DataFrame({"Column": cols, "Coef": model.coef_})

    print(f'Intercept: {model.intercept_}')
    print("Coefficients: ")
    print(coef_df)
    print("-"*50)
```

executed in 6ms, finished 13:58:24 2024-02-19

In [77]:

```

model=LinearRegression()
# Training with train data
model.fit(x_tr_sc,y_train)
```

executed in 20ms, finished 13:58:24 2024-02-19

Out[77]:

```
▾ LinearRegression ⓘ ?
(https://scikit-learn.org/1.4/modules/generated/sklearn.linear_model.LinearReg
LinearRegression()
```

In [78]:
```
get_metrics(x_tr_sc,y_train,model,'Linear')
```
executed in 8ms, finished 13:58:24 2024-02-19

```
-----------------Regression Type: Linear----------------
MAE: 0.04301
RMSE: 0.05976
R2: 0.82559
Adjusted R2: 0.82248
Intercept: 0.7184
Coefficients:
     Column      Coef
0       GRE  0.027365
1     TOEFL  0.011048
2        UR  0.005790
3       SOP  0.005966
4       LOR  0.013974
5      CGPA  0.072045
6  Research  0.010864
--------------------------------------------------
```

In [79]:
```
# r2 score of model on test data
get_metrics(x_te_sc,y_test,model,'Linear')
```
executed in 8ms, finished 13:58:24 2024-02-19

```
-----------------Regression Type: Linear----------------
MAE: 0.04067
RMSE: 0.05922
R2: 0.79709
Adjusted R2: 0.78165
Intercept: 0.7184
Coefficients:
     Column      Coef
0       GRE  0.027365
1     TOEFL  0.011048
2        UR  0.005790
3       SOP  0.005966
4       LOR  0.013974
5      CGPA  0.072045
6  Research  0.010864
--------------------------------------------------
```

In [80]:
```
pd.DataFrame(data=[[x] for x in model.coef_],index=x_tr_sc.columns,columns
```
executed in 7ms, finished 13:58:24 2024-02-19

Out[80]:

	VIF
GRE	0.027365
TOEFL	0.011048
UR	0.005790
SOP	0.005966
LOR	0.013974
CGPA	0.072045
Research	0.010864

B) Model Training with ordinary least squares

In [129]:

```
# adding a constant to the model of training
x_tr_sc_sm = sm.add_constant(x_tr_sc)
y_tr_sm = np.array(y_train)

# adding a constant to the model of testing
x_te_sc_sm = sm.add_constant(x_te_sc)
y_te_sm= np.array(y_test)


sm_model = sm.OLS(y_tr_sm,x_tr_sc_sm).fit()
print(sm_model.summary())
```

executed in 69ms, finished 14:13:24 2024-02-19

```
                            OLS Regression Results
==========================================================================
====
Dep. Variable:                      y   R-squared:
0.826
Model:                            OLS   Adj. R-squared:
0.822
Method:                 Least Squares   F-statistic:                     2
65.1
Date:                Mon, 19 Feb 2024   Prob (F-statistic):          2.29e
-144
Time:                        14:13:24   Log-Likelihood:                 55
9.41
No. Observations:                 400   AIC:                            -1
103.
Df Residuals:                     392   BIC:                            -1
071.
Df Model:                           7
Covariance Type:            nonrobust
==========================================================================
====
                 coef    std err          t      P>|t|      [0.025      0.
975]
--------------------------------------------------------------------------
----
const          0.7184      0.003    238.023      0.000       0.712
0.724
GRE            0.0274      0.006      4.295      0.000       0.015
0.040
TOEFL          0.0110      0.006      1.826      0.069      -0.001
0.023
UR             0.0058      0.005      1.205      0.229      -0.004
0.015
SOP            0.0060      0.005      1.172      0.242      -0.004
0.016
LOR            0.0140      0.004      3.272      0.001       0.006
0.022
CGPA           0.0720      0.007     10.828      0.000       0.059
0.085
Research       0.0109      0.004      2.927      0.004       0.004
0.018
==========================================================================
====
Omnibus:                       87.655   Durbin-Watson:
1.963
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               19
4.225
Skew:                          -1.122   Prob(JB):                     6.68
e-43
Kurtosis:                       5.572   Cond. No.
5.62
==========================================================================
====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is cor
rectly specified.
```

In [84]:

```
y_pred_sm = sm_model.predict(x_te_sc_sm)
```

executed in 4ms, finished 13:58:24 2024-02-19

In [85]:

```
print(f'Model Score on Train Data : {sm_model.rsquared}')
print(f'Model Score on Test Data: {r2_score(y_te_sm,y_pred_sm)}')
print(f'R2_Score : {r2_score(y_te_sm,y_pred_sm)}')
print(f'Mean Squared Error : {mean_squared_error(y_te_sm,y_pred_sm)}')
print(f'Mean Absolute Error : {mean_absolute_error(y_te_sm,y_pred_sm)}')
print(f'Root Mean Squared Error : {np.sqrt(mean_squared_error(y_te_sm,y_pr
print(f'Adjusted R2_Score :{sm_model.rsquared_adj}')
```

executed in 8ms, finished 13:58:24 2024-02-19

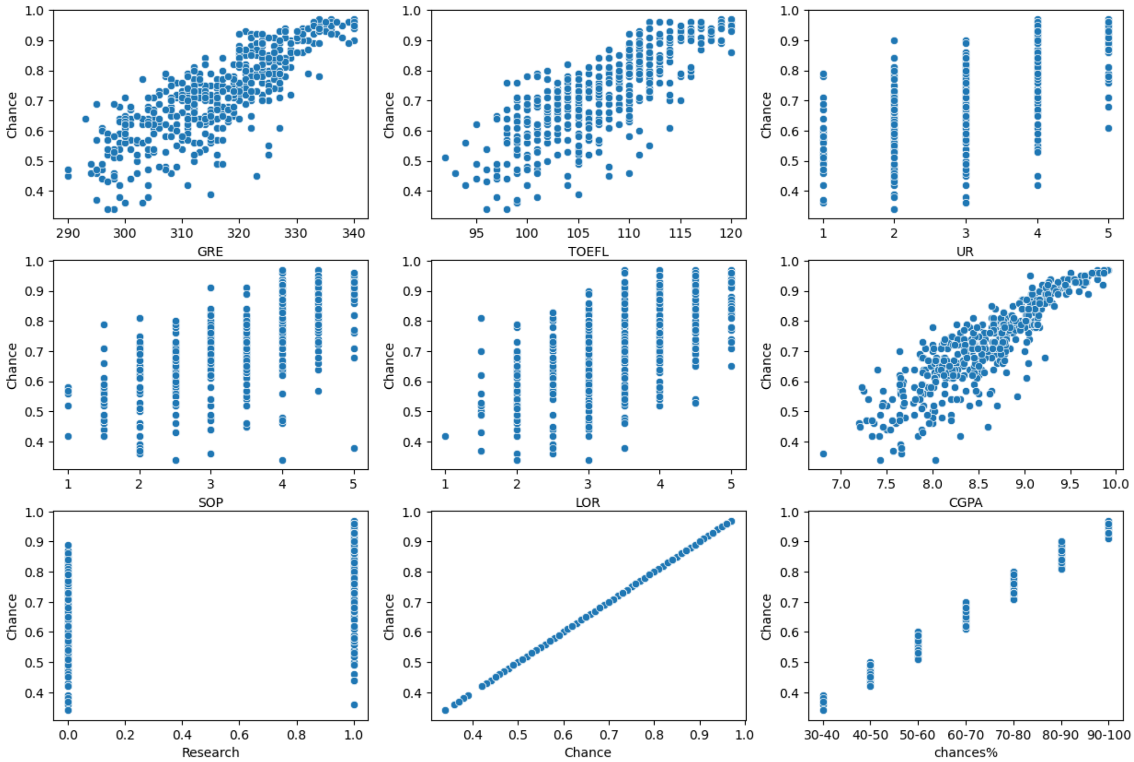
```
Model Score on Train Data : 0.8255906992873271
Model Score on Test Data: 0.797091259637587
R2_Score : 0.797091259637587
Mean Squared Error : 0.0035068697305961818
Mean Absolute Error : 0.04066666398242634
Root Mean Squared Error : 0.059218829189677344
Adjusted R2_Score :0.8224762474888865
```

5) Assumptions checking of Linear Regression

1.Assumption of Linearity

In [86]:
```
plt.figure(figsize=(15,10))
place=1
for i in df.columns:
    plt.subplot(3,3,place)
    sns.scatterplot(data=df,y='Chance',x=i)
    place+=1
plt.show()
```

executed in 1.03s, finished 13:58:25 2024-02-19



- TOEFL score, GRE score , CGPA are linear To the dependent variable of chances
- SOP and University ranking also approx linear to dependent variable
- Research does not look like its much effecting the relationship between chances and research but we will consider it

All the variables are linear to the dependent variable Hence assumption 1 is cleared

2. Non multi-collinear features

In [87]:
```
vif=pd.DataFrame()
vif['Features'] = x_tr_sc.columns
vif['VIF'] = [round(variance_inflation_factor(x_tr_sc.values,i),2) for i i
vif
```
executed in 17ms, finished 13:58:25 2024-02-19

Out[87]:

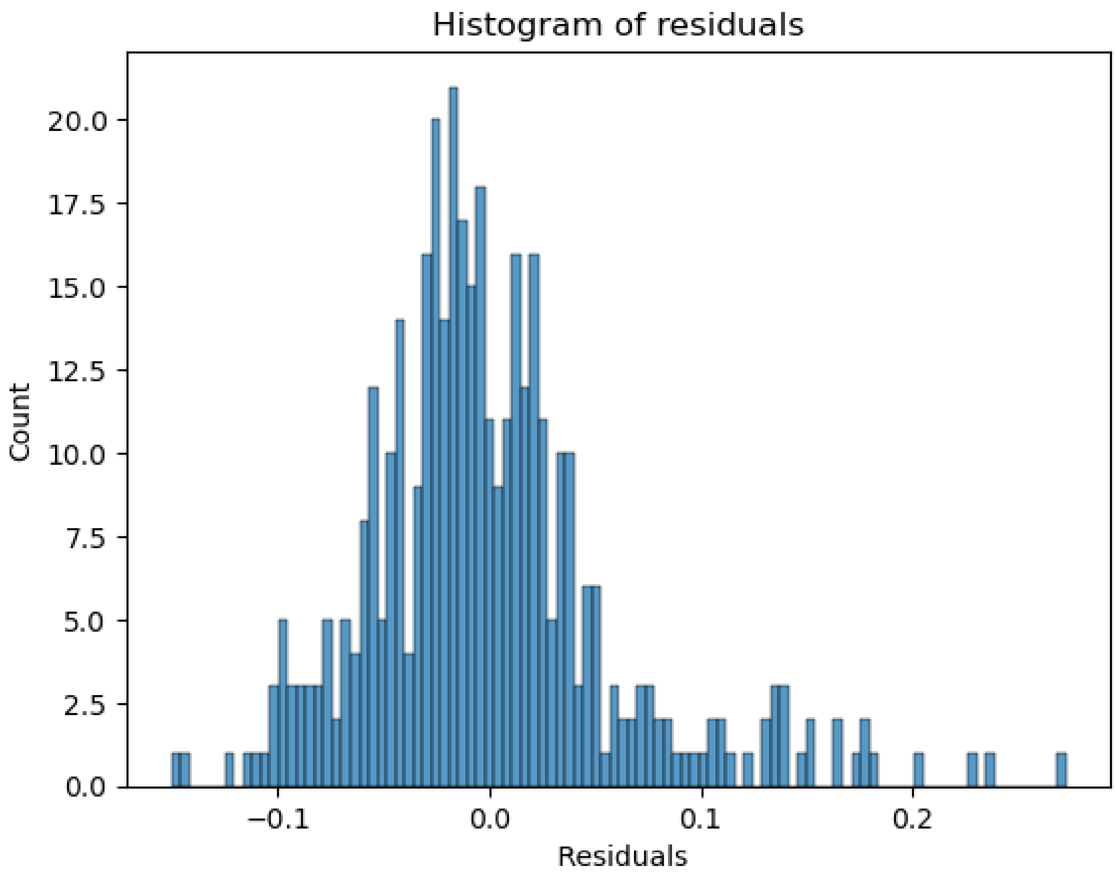
	Features	VIF
0	GRE	4.46
1	TOEFL	4.02
2	UR	2.53
3	SOP	2.85
4	LOR	2.00
5	CGPA	4.86
6	Research	1.51

- All the VIF for all the features having VIF < 5, so there are no Major Multi-collinear relations.

3. Assumptions of Errors are normally distributed

In [88]:
```
y_pred = sm_model.predict(x_tr_sc_sm)
errors = -y_tr_sm + y_pred
```
executed in 4ms, finished 13:58:25 2024-02-19

In [89]:
```
sns.histplot(errors,bins=100)
plt.xlabel(" Residuals")
plt.title("Histogram of residuals")
plt.show()
```
executed in 221ms, finished 13:58:25 2024-02-19



- Visually it looks like there is Normality in errors
- The histogram and Q-Q plot of residuals for Linear Regression show the following: The histogram indicates that the residuals are approximately normally distributed. The Q-Q plot shows that the points are mostly aligned along the straight line, further confirming the normality of residuals.

We will be confirming the Error normality by shapiro test

-Checking the normality for **Errors**

- we will select the significance level (alpha)=5%
- We will select the Null Hypothesis and alternate Hypothesis'
  - H0 = The Errors has the normal distribution
  - Ha = The Err has the not normal distribution

In [90]:
```
shapiro(errors)
```
executed in 5ms, finished 13:58:25 2024-02-19

Out[90]:
```
ShapiroResult(statistic=0.9310517311096191, pvalue=1.2453809684898065e-12)
```

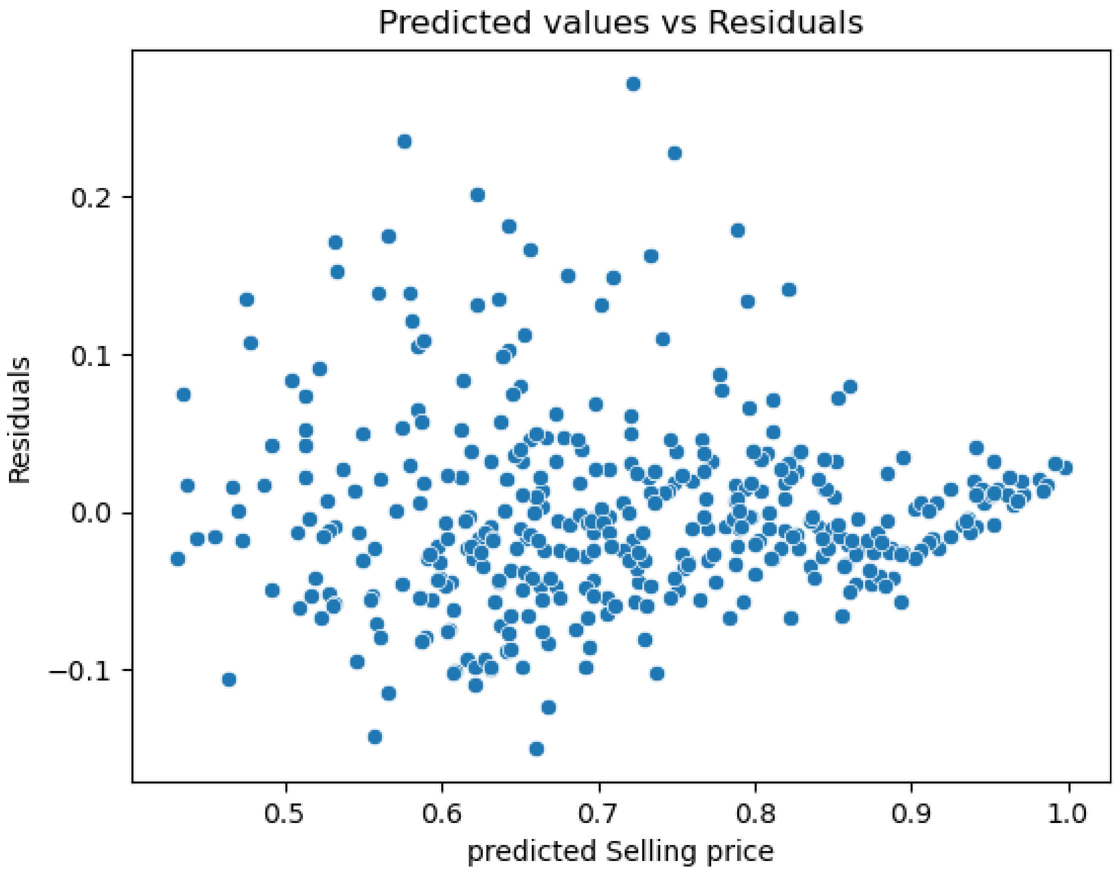
```
Conclusion :-
```

- According to the results, the p value for this test is extremely low and less than the level of significance.
- Thus, since we must reject the null hypothesis, the results are statistically significant. ... the evidence is sufficient to support the alternative hypothesis, which holds that **Errors not normally distributed**.

4.Heteroskedasticity should not exist

In [91]:
```
sns.scatterplot(x=y_pred,y=errors)
plt.xlabel("predicted Selling price")
plt.ylabel("Residuals")
plt.title("Predicted values vs Residuals")
```
executed in 147ms, finished 13:58:25 2024-02-19

Out[91]:
```
Text(0.5, 1.0, 'Predicted values vs Residuals')
```



- Notice that As we go from left to right,the spread of errors is almost constant

What can we understand from this constant Residuals?

- We can assume that heteroskedasticity does not exist in our data
- There are outliers present in the dataset

We can also use "Goldfeld-Quandt Test" to verify our assumptions

Using Goldfeld Quandt Test to check homoskedacity

- This test is used to test the presence of Heteroscedasticity in the given data
- The Goldfeld-Quandt test works by removing some number of observations located in the center of the dataset, then testing to see if the spread of residuals is different from

the resulting two datasets that are on either side of the central observations.

Null and Alternate Hypothesis of Goldfeld-Quandt Test

```
* Null Hypothesis: Heteroscedasticity is not present.
* Alternate Hypothesis: Heteroscedasticity is present.
```

In [92]:
```
sms.het_goldfeldquandt(y_tr_sm,x_tr_sc_sm)
```
executed in 6ms, finished 13:58:25 2024-02-19

Out[92]:
```
(1.042028868565688, 0.38787934387938844, 'increasing')
```

From the goldfeld-quandt test:

- F Statistic comes out to be 1.00 => Implying minimal difference in variance between groups
- p-value of 0.387indicates 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.

5. No Autocorrelation

Checking for The mean of residuals is nearly zero

In [93]:
```
np.mean(errors)
```
executed in 5ms, finished 13:58:25 2024-02-19

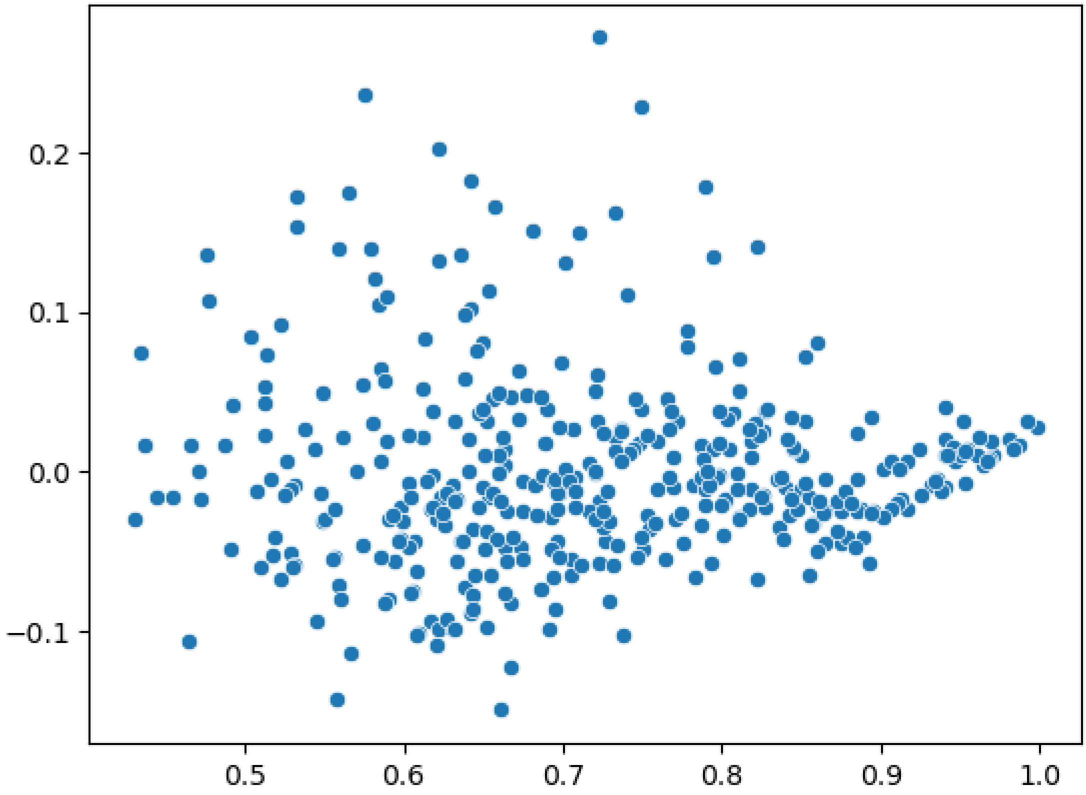
Out[93]:
```
4.260480856999038e-16
```

Errors are normally distributed with a mean value of 0

Checking for the Linearity of variables

In [94]:
```
sns.scatterplot(x=y_pred,y=errors)
```
executed in 123ms, finished 13:58:25 2024-02-19

Out[94]: <Axes: >



errors are linear and with y predicted

We will check the autocorrelation with durbin-watson statistics

- The Durbin Watson statistic is a test statistic used in statistics to detect autocorrelation in the residuals from a regression analysis.
- The Durbin Watson statistic will always assume a value between 0 and 4. A value of DW = 2 indicates that there is no autocorrelation.

The hypotheses followed for the Durbin Watson statistic:

H(0) = First-order autocorrelation does not exist.

H(1) = First-order autocorrelation exists

The assumptions of the test are:

Errors are normally distributed with a mean value of 0

In [95]:
```
np.mean(errors)
```
executed in 6ms, finished 13:58:25 2024-02-19

Out[95]: 4.260480856999038e-16

Errors are normally distributed with a mean value of 0

In [96]:
```
durbin_watson(errors)
```
executed in 4ms, finished 13:58:25 2024-02-19

Out[96]: 1.9634445607660518

- The test statistic is 1.9634. Since this is within the range of 1.5 and 2.5, we would consider `autocorrelation not to be problematic in this regression model` .

6) Trying the model with ridge and lasso regression

In [97]:
```
def Model_training(x_train,y_train,x_test,y_test, regression_type='Linear'
    if regression_type == 'Lasso':
        model = Lasso(alpha=0.0001)
    elif regression_type == 'Ridge':
        model = Ridge(alpha=1.0)
    else:
        model = LinearRegression()
    model.fit(x_train, y_train)
    #Compare scaled features
    if compareFeatures == True:
        imp = pd.DataFrame(list(zip(x_test.columns,np.abs(model.coef_))),
        columns=['feature', 'coeff'])
        sns.barplot(x='feature', y='coeff', data=imp)
    plt.xticks(rotation=90)

    print(f'Performace metrics for the train dataset: ')

    get_metrics(x_train, y_train, model , regression_type)
    print()
    print(f' Performace metrics for the test dataset: ')
    #print(f'-------------------------------------------')
    get_metrics(x_test, y_test, model , regression_type)
```
executed in 6ms, finished 13:58:25 2024-02-19

A) Lasso Regularization (L1 regularization)

Using K fold to find the perfet regularization rate

In [98]:
```
alpha_rr = [0.00001,0.0001,0.001,0.01,0.1,1,10]
k_fold =KFold(n_splits=5)
```
executed in 4ms, finished 13:58:25 2024-02-19

In [99]:
```
def adj_r2_score(x_test,y_test,R):
    return 1 - (1-R)*(len(y_test)-1)/(len(y_test)- x_test.shape[1]-1)
```
executed in 4ms, finished 13:58:25 2024-02-19

In [100]:

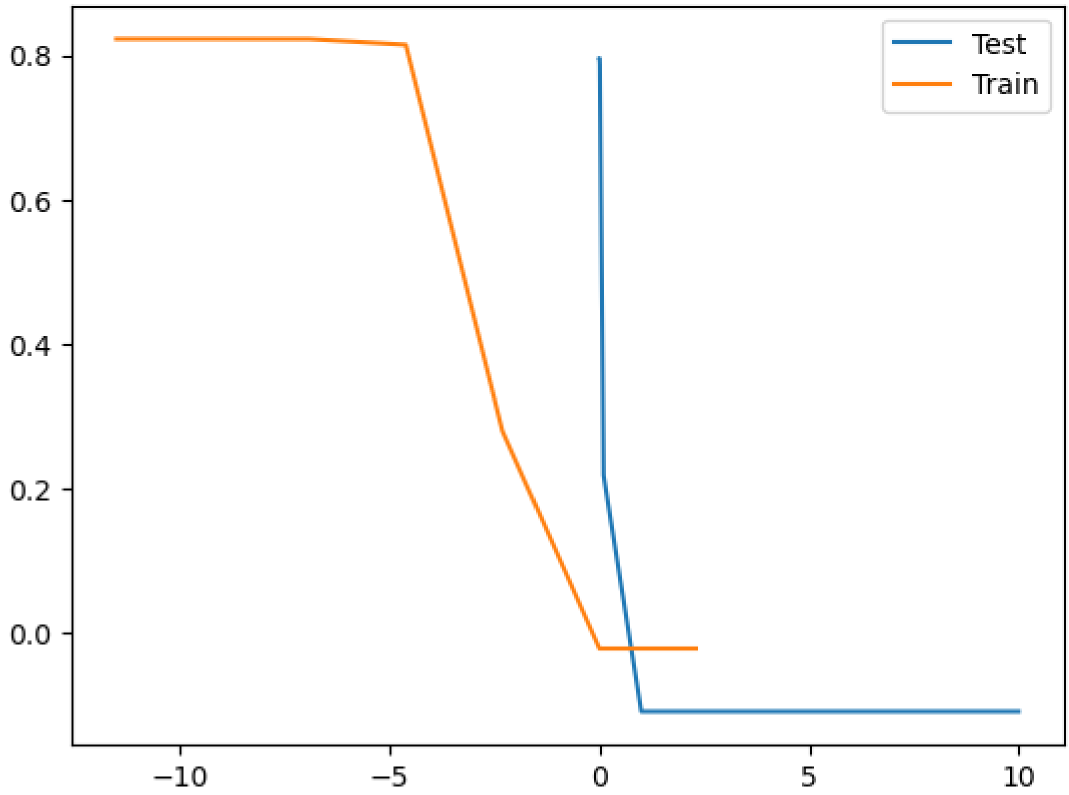
```
train_score = []
test_score = []
for alpha in alpha_rr:
    lasso_model =  Lasso(alpha=alpha)

    fold_train_score = []
    fold_test_score = []

    for train_index, val_index in k_fold.split(x_train):
        #print(train_index,val_index)

        x_tra, x_val = x_train.iloc[train_index], x_train.iloc[val_index]
        y_tra, y_val = y_train.iloc[train_index], y_train.iloc[val_index]

        polyreg_scaled = make_pipeline(scaler, lasso_model)
        polyreg_scaled.fit(x_tra, y_tra)
        trainscore = adj_r2_score(x_tra, y_tra, polyreg_scaled.score(x_tra
        valscore= adj_r2_score(x_val, y_val, polyreg_scaled.score(x_val, y
        fold_train_score.append(trainscore)
        fold_test_score.append(valscore)
    train_score.append(np.mean(fold_train_score))
    test_score.append(np.mean(fold_test_score))
```

executed in 189ms, finished 13:58:25 2024-02-19

In [101]:

```
plt.plot([10**x for x in range(-5,2)],test_score,label='Test')
plt.plot([np.log(10**x) for x in range(-5,2)],train_score,label='Train')
plt.legend(loc='upper right')
plt.show()
```

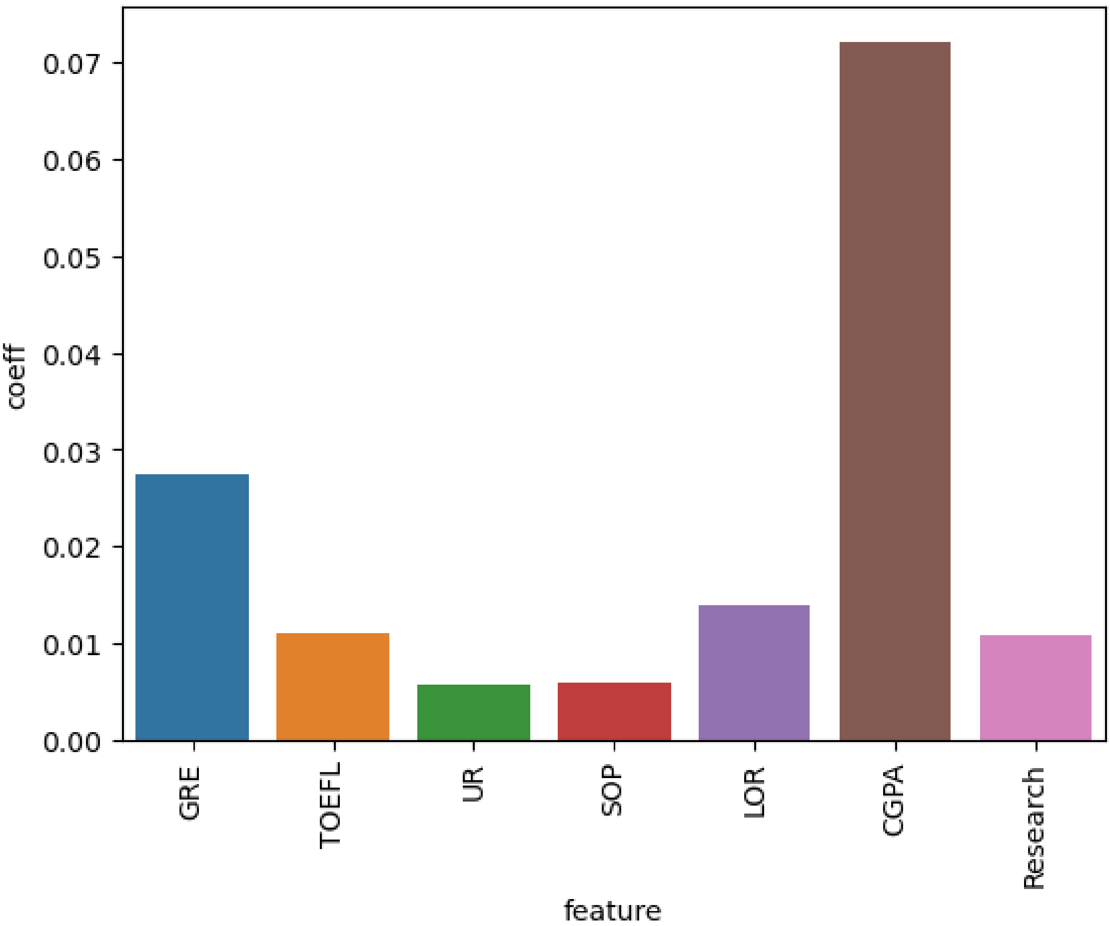
executed in 103ms, finished 13:58:26 2024-02-19



We will use Regularization rate = 0.0001

In [102]:
```
Model_training(x_tr_sc,y_train,x_te_sc,y_test, regression_type='Lasso', co
```
executed in 136ms, finished 13:58:26 2024-02-19

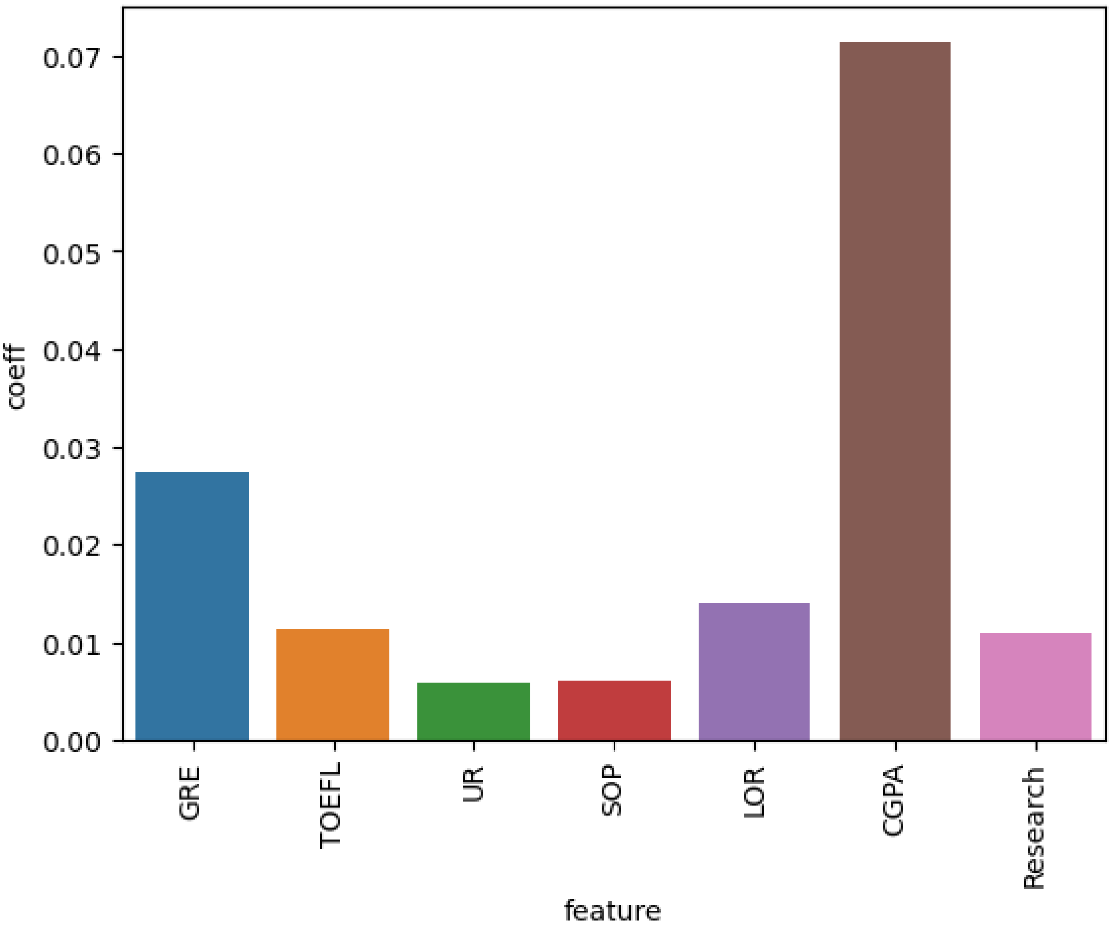
```
Performace metrics for the train dataset:
-----------------Regression Type: Lasso----------------
MAE: 0.04300
RMSE: 0.05976
R2: 0.82559
Adjusted R2: 0.82248
Intercept: 0.7184
Coefficients:
     Column      Coef
0       GRE  0.027357
1     TOEFL  0.011014
2        UR  0.005760
3       SOP  0.005949
4       LOR  0.013928
5      CGPA  0.072070
6  Research  0.010808
--------------------------------------------------

 Performace metrics for the test dataset:
-----------------Regression Type: Lasso----------------
MAE: 0.04068
RMSE: 0.05922
R2: 0.79706
Adjusted R2: 0.78162
Intercept: 0.7184
Coefficients:
     Column      Coef
0       GRE  0.027357
1     TOEFL  0.011014
2        UR  0.005760
3       SOP  0.005949
4       LOR  0.013928
5      CGPA  0.072070
6  Research  0.010808
--------------------------------------------------
```


0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
coeff
GRE
TOEFL
UR
SOP
LOR
CGPA
Research
feature

B) Ridge Regularization (L2 regularization)

In [103]:
```
Model_training(x_tr_sc,y_train,x_te_sc,y_test, regression_type='Ridge', co
```
executed in 134ms, finished 13:58:26 2024-02-19

```
Performace metrics for the train dataset:
-----------------Regression Type: Ridge----------------
MAE: 0.04301
RMSE: 0.05976
R2: 0.82559
Adjusted R2: 0.82247
Intercept: 0.7184
Coefficients:
     Column      Coef
0       GRE  0.027441
1     TOEFL  0.011304
2        UR  0.005887
3       SOP  0.006089
4       LOR  0.014034
5      CGPA  0.071381
6  Research  0.010901
--------------------------------------------------

 Performace metrics for the test dataset:
-----------------Regression Type: Ridge----------------
MAE: 0.04065
RMSE: 0.05922
R2: 0.79711
Adjusted R2: 0.78167
Intercept: 0.7184
Coefficients:
     Column      Coef
0       GRE  0.027441
1     TOEFL  0.011304
2        UR  0.005887
3       SOP  0.006089
4       LOR  0.014034
5      CGPA  0.071381
6  Research  0.010901
--------------------------------------------------
```



Insights from EDA:

1. **Strong Correlations :** The strong correlations between CGPA, GRE Score, and TOEFL Score suggest that academic performance and proficiency in English are closely related to each other. High GRE & TOEFL scores for high CGPA students might indicate that students who perform well academically tend to also score higher on standardized tests.

2. The strongest correlation is between CGPA and ”Chance of Admit’, so CGPA is the most important factor in deciding the admission of the candidate, followed by GRE score and TOEFL score.

3. Linear Regression has the highest R-squared score among the evaluated models, indicating a strong ability to predict the ‘Chance of Admit’ from the given features.

4. Ridge Regression also performs similarly to Linear Regression in this case, with a slight difference in R-squared score and RMSE. This similarity suggests that the regularization introduced by Ridge does not significantly alter the predictions for this particular dataset,possibly because multicollinearity is not a severe issue or the optimal alpha value is close to the one chosen.

5. Lasso Regression shows a lower R-squared score compared to the other models, which indicates that the level of regularization (controlled by alpha=0.0001) might be too strong.

Distributions:

6. The distributions of GRE Score, TOEFL Score, and CGPA are fairly normal but slightly left-skewed, indicating most Jamboree students have above-average scores.
7. The Chance of Admit distribution is also somewhat left-skewed, suggesting that most Jamboree students have a higher likelihood of admission.

Recommendations:

1. All assumptions of Linear Regression model are satisfied and we can safely use Linear Regression model.
2. Model trained has very less values of RMSE, MSE & Adjusted R2 score and give accurate prediction.
3. Company needs to collect more data for improving accuracy and reducing biasing of model.
4. More features shall be introduced for collected data.
5. Feature importance of Linear regression model tells us that CGPA score is most important factor followed by Research paper publishing.
6. Students needs to do more focus on their CGPA and Reserach paper publishing for improving chances of graduate admission.
7. University rating, SOP and GRE score have not much importance for getting admission