Jamboree Education - Linear Regression

Context

Jamboree has helped thousands of students like you make it to top colleges abroad. Be it GMAT, GRE or SAT, their unique problem-solving methods ensure maximum scores with minimum effort. They recently launched a feature where students/learners can come to their website and check their probability of getting into the IVY league college. This feature estimates the chances of graduate admission from an Indian perspective.

How can you help here?

Your analysis will 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 [1]:
```
# 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_error
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 # to check the VIF
import statsmodels.stats.api as sms # to check the hetroscaditisity
from scipy.stats import shapiro # to check the normality

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

In [2]:
```
# importing the dataset
df= pd.read_csv('original_Jamboree_Admission.csv')
df.head()
```

Out[2]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	1	337	118	4	4.5	4.5	9.65	1	0.92
1	2	324	107	4	4.0	4.5	8.87	1	0.76
2	3	316	104	3	3.0	3.5	8.00	1	0.72
3	4	322	110	3	3.5	2.5	8.67	1	0.80
4	5	314	103	2	2.0	3.0	8.21	0	0.65

1.Basic Analysis

A.) Shape,Statistical summary

In [3]:
```
# information cheking
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
```

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 [4]:
```
df.drop(['Serial No.'],axis=1,inplace=True)
```

Chaning the naes to more short name

In [5]:
```
df=df.rename(columns={'Chance of Admit ' : 'Chance', 'LOR ':'LOR','University Rating':'UR','GRE Score':'GRE','TOEFL Sc
```

In [6]:
```
df.describe()
```

Out[6]:

	GRE	TOEFL	UR	SOP	LOR	CGPA	Research	Chance
count	500.000000	500.000000	500.000000	500.000000	500.00000	500.000000	500.000000	500.00000
mean	316.472000	107.192000	3.114000	3.374000	3.48400	8.576440	0.560000	0.72174
std	11.295148	6.081868	1.143512	0.991004	0.92545	0.604813	0.496884	0.14114
min	290.000000	92.000000	1.000000	1.000000	1.00000	6.800000	0.000000	0.34000
25%	308.000000	103.000000	2.000000	2.500000	3.00000	8.127500	0.000000	0.63000
50%	317.000000	107.000000	3.000000	3.500000	3.50000	8.560000	1.000000	0.72000
75%	325.000000	112.000000	4.000000	4.000000	4.00000	9.040000	1.000000	0.82000
max	340.000000	120.000000	5.000000	5.000000	5.00000	9.920000	1.000000	0.97000

In [7]:
```
# Converting the Chance into categorical valribale for further analysis
# converting all into categories like 30-40,40-50,60-70,50-60,70-80,80-90,90-100
df['chances%']=pd.cut(df['Chance'],bins=[i for i in np.arange(0.3,1.1,0.1)],labels=[f'{i}-{i+10}' for i in range(30,10
```

In [8]:
```
df
```

Out[8]:

	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 [9]:
```
# calculating the min and maximum values of each column for furthur analysis
df.groupby('chances%')[['GRE','TOEFL','UR','SOP','LOR','CGPA','Research','Chance']].agg(['min','max'])
```

Out[9]:

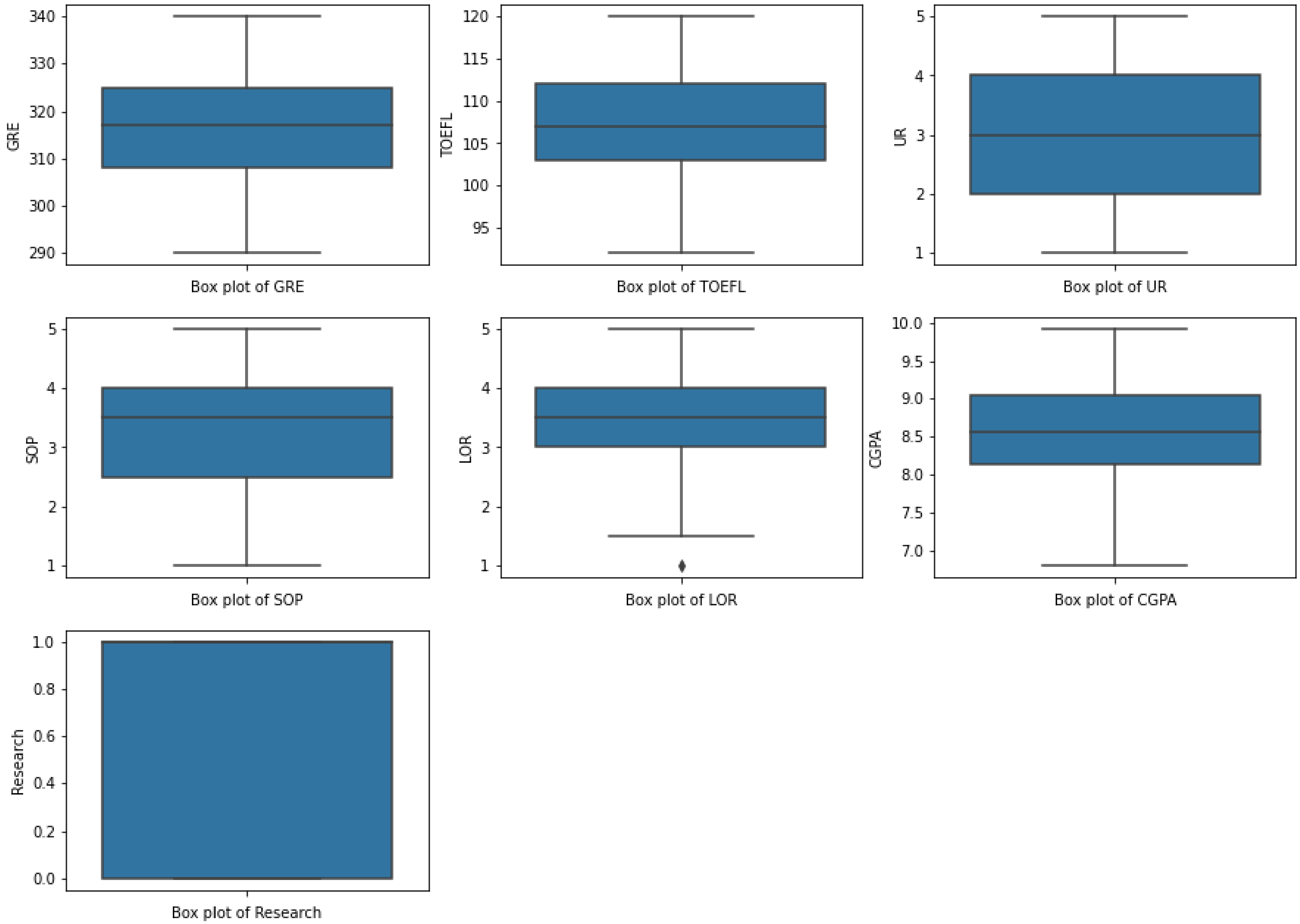
	GRE		TOEFL		UR		SOP		LOR		CGPA		Research		Chance	
	min	**max**	**min**	**max**	**min**	**max**	**min**	**max**	**min**	**max**	**min**	**max**	**min**	**max**	**min**	**max**
chances%																
30-40	295	315	96	105	1	3	2.0	5.0	1.5	3.5	6.80	8.03	0	1	0.34	0.39
40-50	290	323	93	110	1	4	1.0	4.0	1.0	3.5	7.20	8.60	0	1	0.42	0.50
50-60	295	325	92	112	1	4	1.0	4.5	1.5	4.5	7.23	8.92	0	1	0.51	0.60
60-70	293	327	95	115	1	5	1.5	5.0	1.5	5.0	7.40	9.22	0	1	0.61	0.70
70-80	300	334	98	116	1	5	1.5	5.0	2.0	5.0	7.89	9.16	0	1	0.71	0.80
80-90	312	340	104	120	2	5	2.0	5.0	1.5	5.0	8.44	9.70	0	1	0.81	0.90
90-100	320	340	110	120	4	5	3.0	5.0	3.5	5.0	9.06	9.92	1	1	0.91	0.97

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

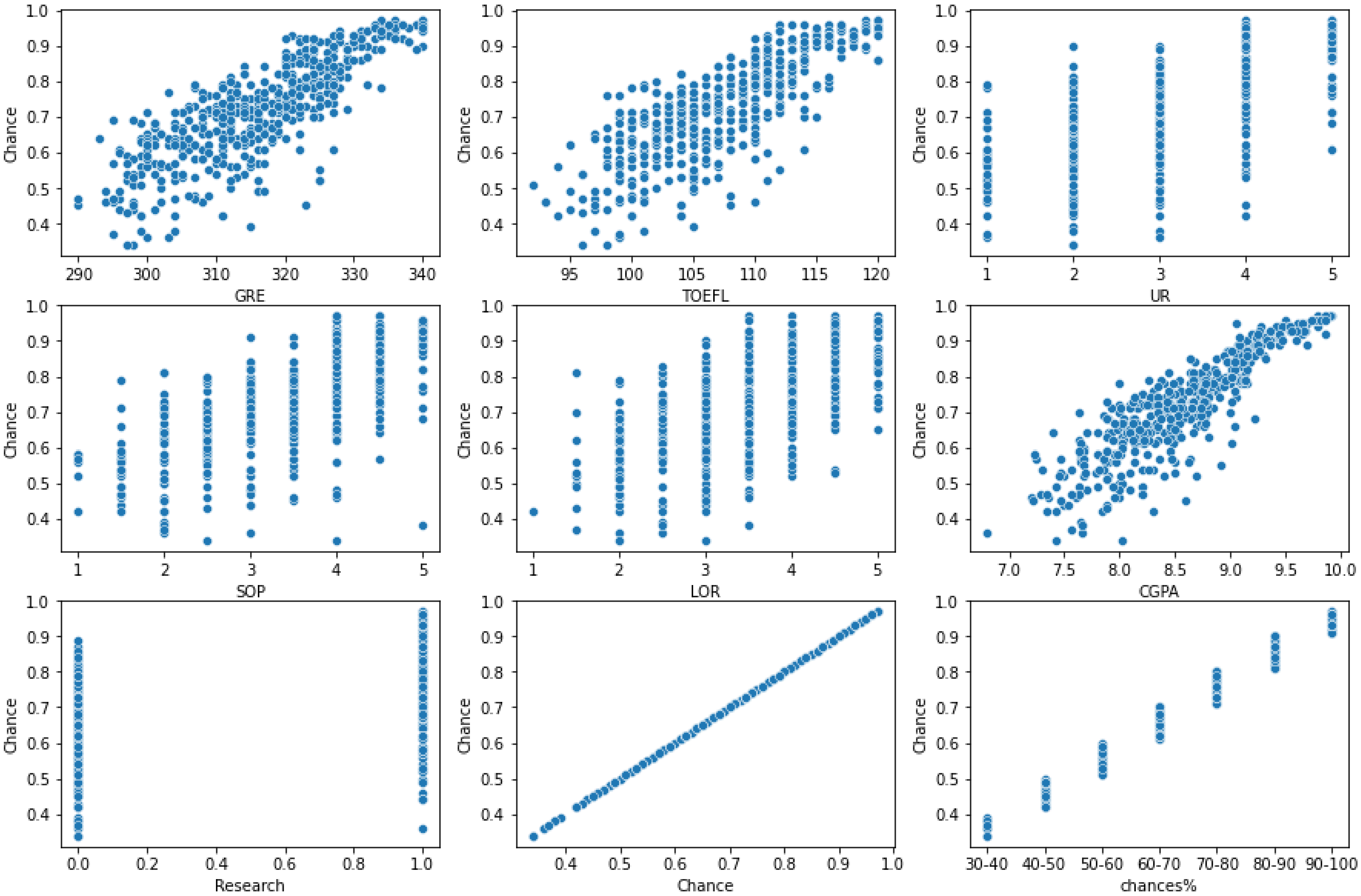
In [10]:
```
plt.figure(figsize=(15,15))
count=1
for i in ['GRE','TOEFL','UR','SOP','LOR','CGPA','Research']:
    plt.subplot(4,3,count)
    sns.boxplot(data=df, y=i)
    plt.xlabel(f'Box plot of {i}')
    count+=1
plt.show( )
```



- There are no outliers present in the Dataset

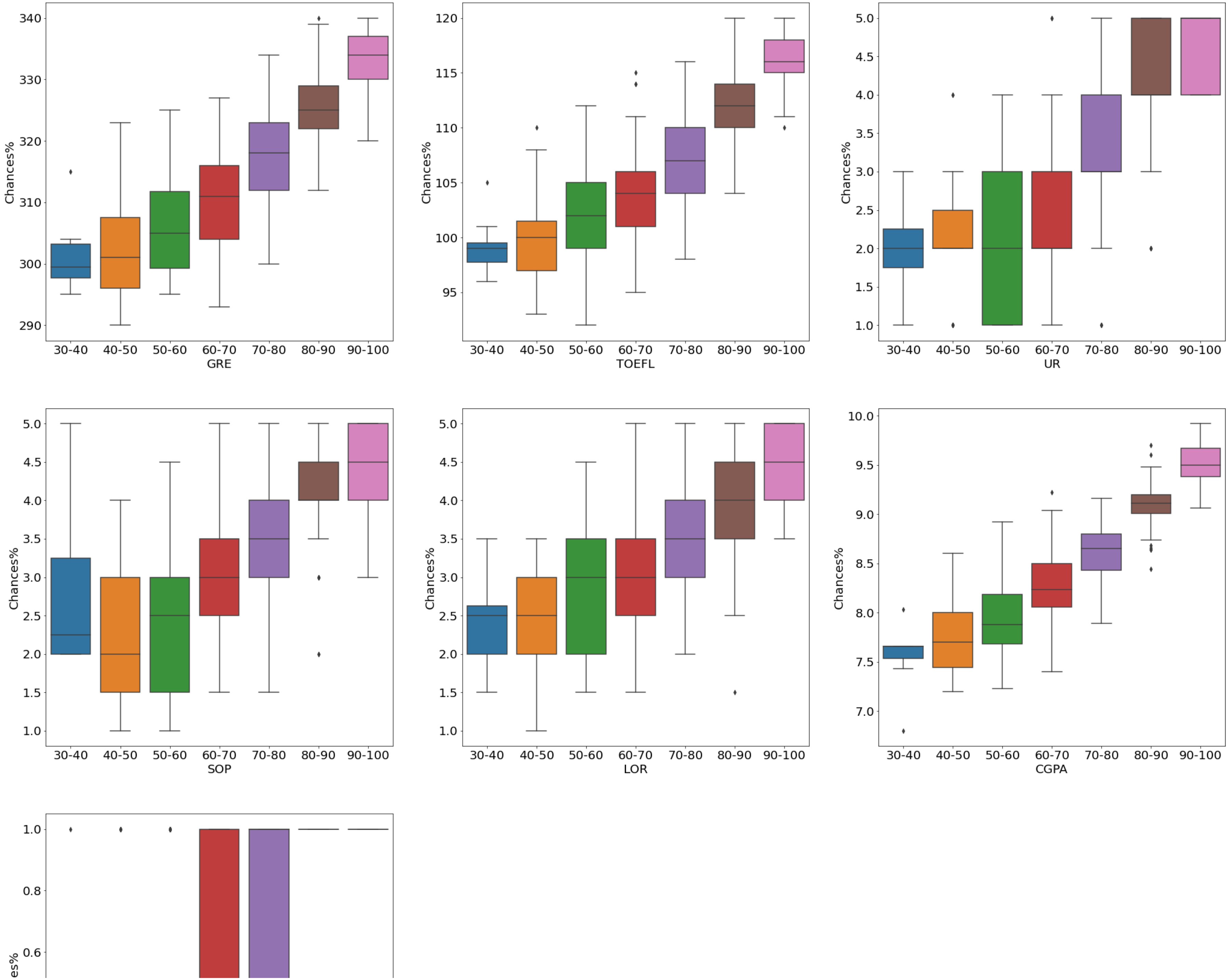
2. Univariate and Bivariate Plots

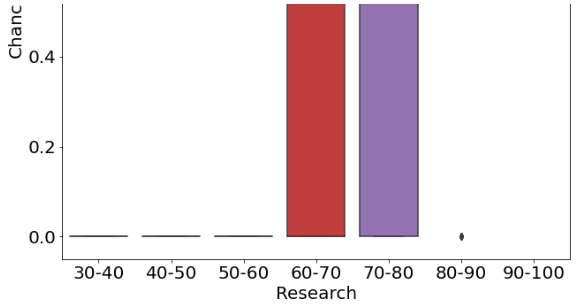
In [11]:
```
plt.figure(figsize=(15,10))
place=1
for i in df.columns:
    plt.subplot(3,3,place)
    sns.scatterplot(data=df,y='Chance',x=i)
    place+=1
plt.show()
```



1. Chances of getting selected are proportional to GRE and TOFEL score.
2. Chances of getting selected are proportionally steep with CGPA.

In [12]:
```
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()
```


Chances%
30-40
40-50
50-60
60-70
70-80
80-90
90-100
GRE
TOEFL
UR
SOP
LOR
CGPA
es%

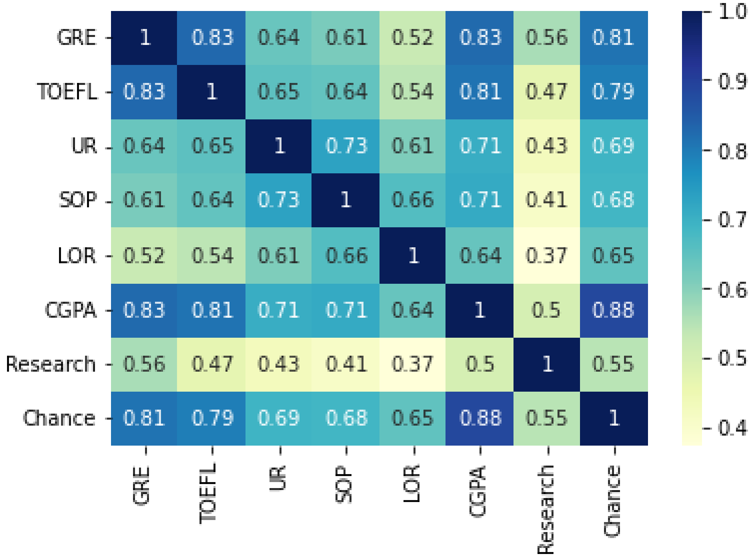


Results are moreover the same as previous

Checking the correlation

In [13]:
```
sns.heatmap(df.corr(),cmap='YlGnBu',annot=True)
```

Out[13]: <AxesSubplot:>



- CGPA and chances are most effect on chances of getting admission.

3. Data Preprocessing

A) Duplicates Values Check

In [14]:
```
df.drop_duplicates()
```

Out[14]:

	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 [15]:
```
df.info()
```

```
<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 [16]:
```
df_new=df.copy()
```

In [17]:
```
data = df_new.drop(['chances%'],axis=1)
```

In [18]:
```
x= data.drop(['Chance'],axis=1)
y=data['Chance']
```

In [19]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_state=10)
```

In [20]:
```
scaler=StandardScaler()
```

In [21]:
```
scaler.fit(x_train,y_train)
```

Out[21]:
```
StandardScaler()
```

In [22]:
```
x_tr_sc = pd.DataFrame(scaler.transform(x_train),columns=x_train.columns)
```

In [23]:
```
x_te_sc = pd.DataFrame(scaler.transform(x_test),columns=x_test.columns)
```

4. Model Building

A) Model Building with Linear Regression model

In [24]:
```
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)))
```

In [25]:
```

model=LinearRegression()
```

In [26]:
```
# Training with train data
model.fit(x_tr_sc,y_train)
```

Out[26]:
```
LinearRegression()
```

In [27]:
```
# r2 score of model on train data
model.score(x_tr_sc,y_train)
```

Out[27]:
```
0.8255906992873271
```

In [28]:
```
# r2 score of model on test data
model.score(x_te_sc,y_test)
```

Out[28]:
```
0.797091259637587
```

In [29]:
```
model.coef_
```

Out[29]:
```
array([0.02736491, 0.01104752, 0.00579047, 0.0059662 , 0.01397439,
       0.07204503, 0.01086388])
```

In [30]:
```
model.intercept_
```

Out[30]:
```
0.7184
```

In [31]:
```
y_pred = model.predict(x_te_sc)
```

In [32]:
```
pd.DataFrame(data=[[x] for x in model.coef_],index=x_tr_sc.columns,columns=['VIF'])
```

Out[32]:

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

In [33]:
```
print(f'Model Score on Train Data : {model.score(x_tr_sc,y_train)}')
print(f'Model Score on Test Data: {model.score(x_te_sc,y_test)}')
print(f'R2_Score : {r2_score(y_test,y_pred)}')
print(f'Mean Squared Error : {mean_squared_error(y_test,y_pred)}')
print(f'Mean Absolute Error : {mean_absolute_error(y_test,y_pred)}')
print(f'Root Mean Squared Error : {np.sqrt(mean_squared_error(y_test,y_pred))}')
print(f'Adjusted R2_Score :{adj_r2_score(x_te_sc,y_test,model.score(x_te_sc,y_test))}')
```

```
Model Score on Train Data : 0.8255906992873271
Model Score on Test Data: 0.797091259637587
R2_Score : 0.797091259637587
Mean Squared Error : 0.003506869730596181
Mean Absolute Error : 0.0406666639824264
Root Mean Squared Error : 0.05921882918967734
Adjusted R2_Score :0.7816525511317512
```

B) Model Training with ordinary least squares

In [34]:
```
x_tr_sc_sm = sm.add_constant(x_tr_sc)
y_tr = np.array(y_train)
```

In [35]:
```
x_te_sc_sm = sm.add_constant(x_te_sc)
y_te = np.array(y_test)
```

In [36]:
```
sm_model = sm.OLS(y_tr,x_tr_sc_sm).fit()
print(sm_model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.826
Model:                            OLS   Adj. R-squared:                  0.822
Method:                 Least Squares   F-statistic:                     265.1
Date:                Thu, 24 Aug 2023   Prob (F-statistic):          2.29e-144
Time:                        17:09:21   Log-Likelihood:                 559.41
No. Observations:                 400   AIC:                            -1103.
Df Residuals:                     392   BIC:                            -1071.
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):              194.225
Skew:                          -1.122   Prob(JB):                     6.68e-43
Kurtosis:                       5.572   Cond. No.                         5.62
==============================================================================

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

In [37]:
```
y_pred_sm = sm_model.predict(x_te_sc_sm)
```

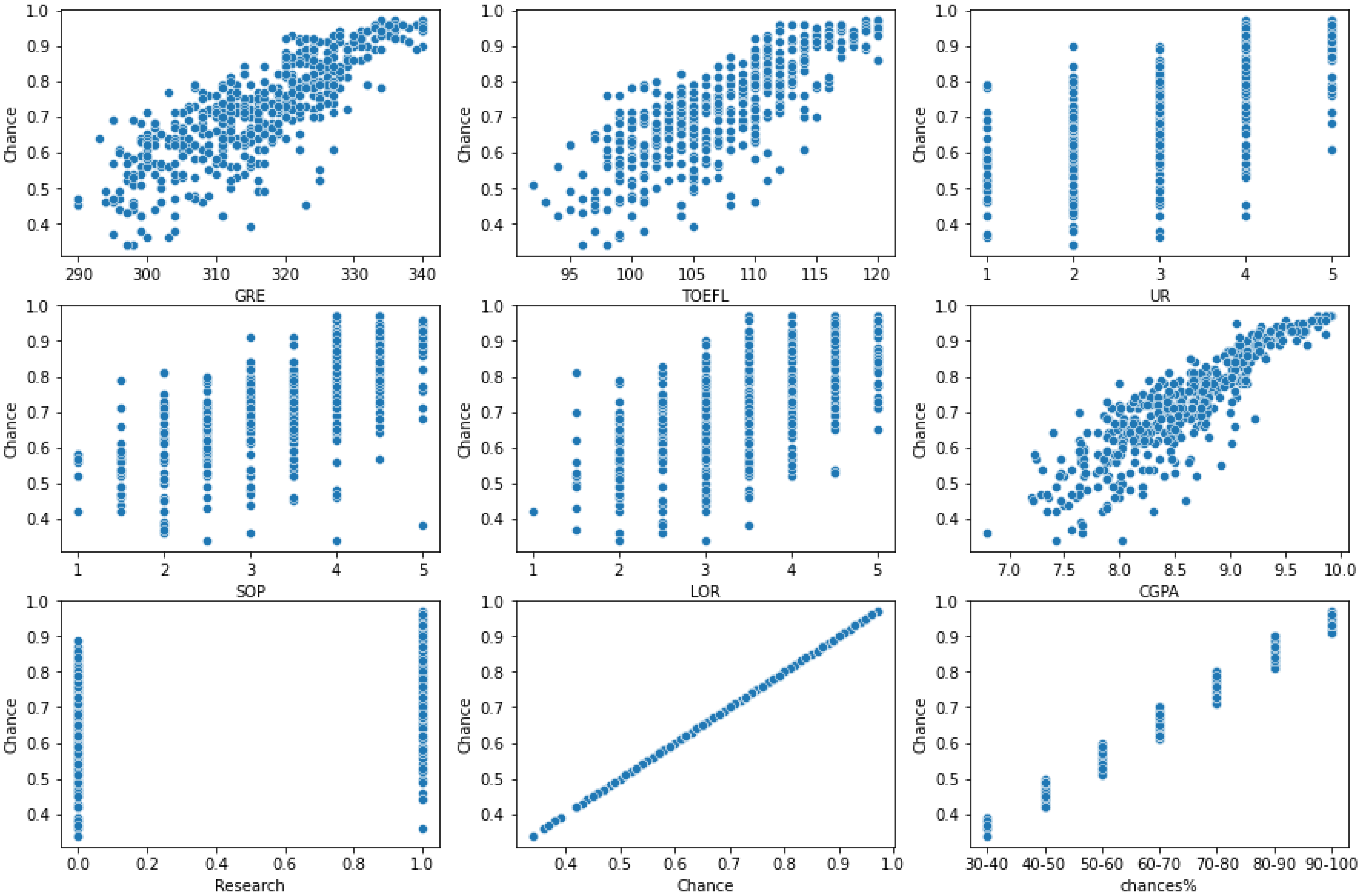
In [38]:
```
print(f'Model Score on Train Data : {sm_model.rsquared}')
print(f'Model Score on Test Data: {r2_score(y_test,y_pred)}')
print(f'R2_Score : {r2_score(y_test,y_pred)}')
print(f'Mean Squared Error : {mean_squared_error(y_test,y_pred_sm)}')
print(f'Mean Absolute Error : {mean_absolute_error(y_test,y_pred_sm)}')
print(f'Root Mean Squared Error : {np.sqrt(mean_squared_error(y_test,y_pred_sm))}')
print(f'Adjusted R2_Score :{sm_model.rsquared_adj}')
```

```
Model Score on Train Data : 0.825590699287327
Model Score on Test Data: 0.797091259637587
R2_Score : 0.797091259637587
Mean Squared Error : 0.003506869730596181
Mean Absolute Error : 0.0406666639824263
Root Mean Squared Error : 0.05921882918967734
Adjusted R2_Score :0.8224762474888864
```

5) Assumptions checking of Linear Regression

1.Assumption of Linearity

In [39]:
```
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()
```



- 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

2. Non multi-collinear features

In [40]:
```
vif=pd.DataFrame()
vif['Features'] = x_tr_sc.columns
vif['VIF'] = [round(variance_inflation_factor(x_tr_sc.values,i),2) for i in range(x_tr_sc.shape[1])]
vif
```

Out[40]:

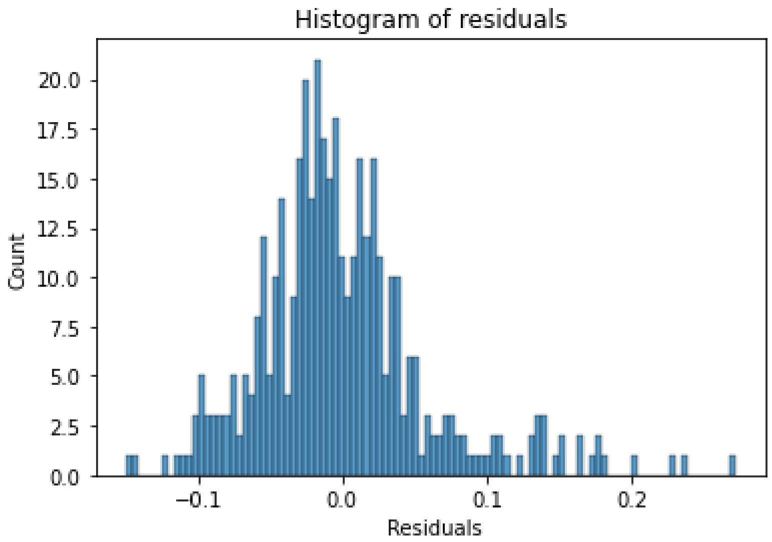
	Features	VIF
0	GRE	4.46
1	TOEFL	4.02
2	UR	2.53
3	SOP	2.85
4	LOR	2.00
5	CGPA	4.86
6	Research	1.51

- 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 [41]:
```
y_pred = sm_model.predict(x_tr_sc_sm)
errors = -y_tr + y_pred
```

In [42]:
```
sns.histplot(errors,bins=100)
plt.xlabel(" Residuals")
plt.title("Histogram of residuals")
plt.show()
```



- 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 [43]:
```
shapiro(errors)
```

Out[43]:
```
ShapiroResult(statistic=0.9310520887374878, pvalue=1.245478438265113e-12)
```

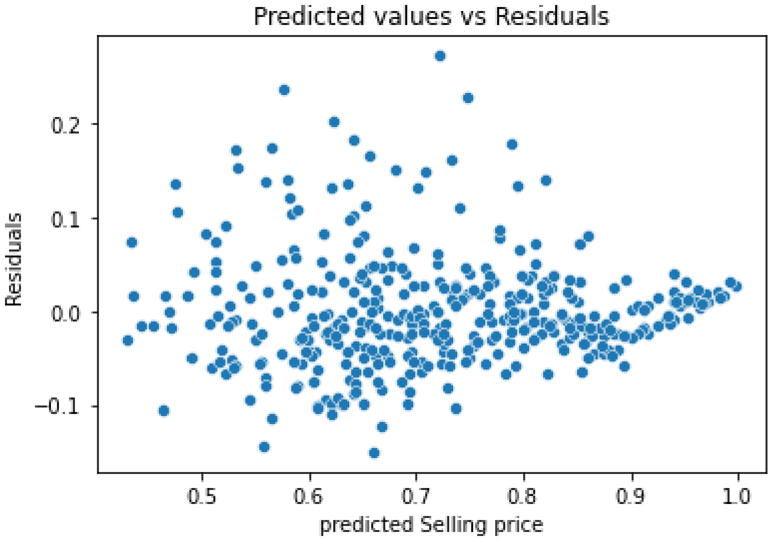
Conclusion :-

- 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 [44]:
```
sns.scatterplot(x=y_pred,y=errors)
plt.xlabel("predicted Selling price")
plt.ylabel("Residuals")
plt.title("Predicted values vs Residuals")
```

Out[44]:
```
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 [45]:
```
sms.het_goldfeldquandt(y_tr,x_tr_sc_sm)
```

Out[45]:
```
(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 [46]:
```
np.mean(errors)
```

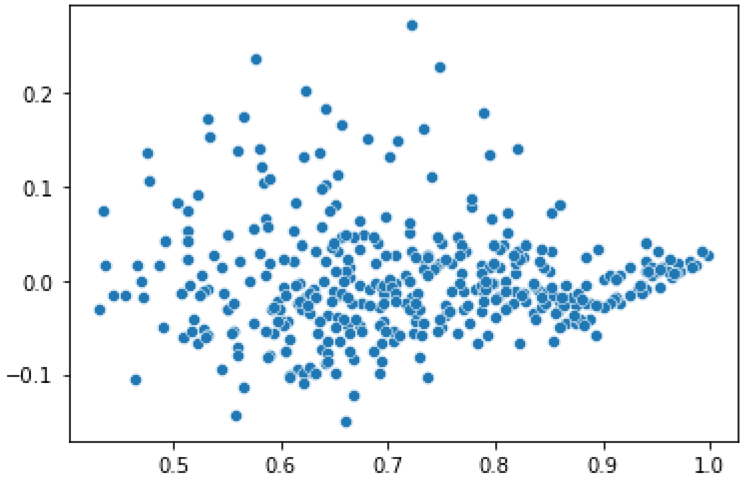
Out[46]:
```
4.3270942384765476e-16
```

Errors are normally distributed with a mean value of 0

Checking for the Linearity of variables

In [47]:
```
sns.scatterplot(x=y_pred,y=errors)
```

Out[47]:
```
<AxesSubplot:>
```



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 [48]:
```
np.mean(errors)
```

Out[48]:
```
4.3270942384765476e-16
```

Errors are normally distributed with a mean value of 0

In [49]:
```
durbin_watson(errors)
```

Out[49]:
```
1.9634445607660516
```

- 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

A) Lasso Regularization (L1 regularization)

Using K fold to find the perfet regularization rate

In [50]:
```
alpha_rr = [0.00001,0.0001,0.001,0.01,0.1,1,10]
k_fold =KFold(n_splits=5)
```

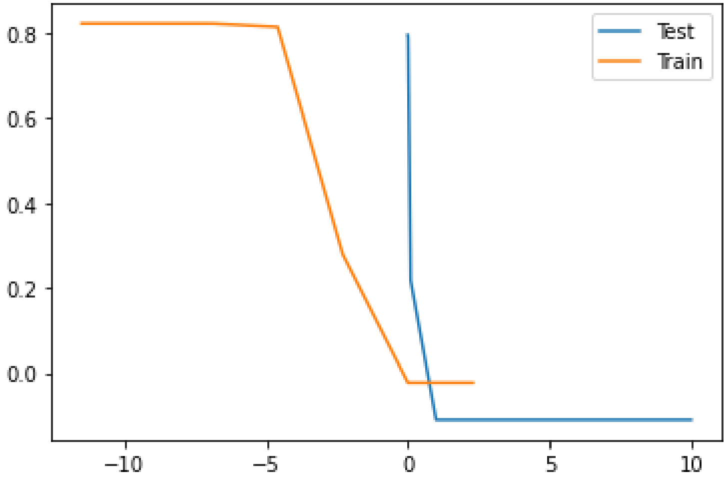
In [51]:
```
train_score = []
test_score = []
for alpha in alpha_rr:
    lasso_model =  Lasso(alpha=alpha)

    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, y_tra))
        valscore= adj_r2_score(x_val, y_val, polyreg_scaled.score(x_val, y_val))
        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))
```

In [52]:
```
np.argmax(train_score)
```

Out[52]:
```
0
```

In [53]:
```
np.argmax(test_score)
```
Out[53]: 1

In [54]:
```
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()
```



We will use Regularization rate = 0.0001

In [55]:
```
lasso =Lasso(alpha=0.0001)
```

In [56]:
```
lasso.fit(x_tr_sc,y_train)
```
Out[56]:
```
Lasso(alpha=0.0001)
```

In [57]:
```
lasso.score(x_tr_sc,y_train)
```
Out[57]: 0.825589883496988

In [58]:
```
lasso.score(x_te_sc,y_test)
```
Out[58]:
```
0.7970626409597299
```

In [59]:
```
y_pred_lasso = lasso.predict(x_te_sc)
```

In [60]:
```
print(f'Model Score on Train Data : {lasso.score(x_tr_sc,y_train)}')
print(f'Model Score on Test Data: {lasso.score(x_te_sc,y_test)}')
print(f'R2_Score : {r2_score(y_test,y_pred_lasso)}')
print(f'Mean Squared Error : {mean_squared_error(y_test,y_pred_lasso)}')
print(f'Mean Absolute Error : {mean_absolute_error(y_test,y_pred_lasso)}')
print(f'Root Mean Squared Error : {np.sqrt(mean_squared_error(y_test,y_pred_lasso))}')
print(f'Adjusted R2_Score :{adj_r2_score(x_te_sc,y_test,lasso.score(x_te_sc,y_test))}')
```

```
Model Score on Train Data : 0.8255898834969888
Model Score on Test Data: 0.7970626409597299
R2_Score : 0.7970626409597299
Mean Squared Error : 0.003507364346919399
Mean Absolute Error : 0.04068173024074727
Root Mean Squared Error : 0.05922300521688678
Adjusted R2_Score :0.7816217549457963
```

B) Ridge Regularization (L2 regularization)

In [61]:
```
ridge = Ridge(alpha=0.0001)
```

In [62]:
```
ridge.fit(x_tr_sc,y_train)
```
Out[62]:
```
Ridge(alpha=0.0001)
```

In [63]:
```
y_pred_ridge = ridge.predict(x_te_sc)
```

In [64]:
```
print(f'Model Score on Train Data : {ridge.score(x_tr_sc,y_train)}')
print(f'Model Score on Test Data: {ridge.score(x_te_sc,y_test)}')
print(f'R2_Score : {r2_score(y_test,y_pred_ridge)}')
print(f'Mean Squared Error : {mean_squared_error(y_test,y_pred_ridge)}')
print(f'Mean Absolute Error : {mean_absolute_error(y_test,y_pred_ridge)}')
print(f'Root Mean Squared Error : {np.sqrt(mean_squared_error(y_test,y_pred_ridge))}')
print(f'Adjusted R2_Score :{adj_r2_score(x_te_sc,y_test,lasso.score(x_te_sc,y_test))}')
```

```
Model Score on Train Data : 0.825590699287277
Model Score on Test Data: 0.7970912621195332
R2_Score : 0.7970912621195332
Mean Squared Error : 0.0035068696877007731
Mean Absolute Error : 0.04066666248696085
Root Mean Squared Error : 0.05921882882749988
Adjusted R2_Score :0.7816217549457963
```

Actionable Insights & Recommendations

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