Jamboree Education - Linear Regression

About Jamboree

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.

Define Problem Statement

Business Problem

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.

Columns info:-

- Serial No. (Unique row ID)
- GRE Scores (out of 340)

- TOEFL Scores (out of 120)
- University Rating (out of 5)
- Statement of Purpose and Letter of Recommendation Strength (out of 5)
- Undergraduate GPA (out of 10)
- Research Experience (either 0 or 1)
- Chance of Admit (ranging from 0 to 1)

In [1]:
```
import math
from datetime import datetime

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

import seaborn as sns # All palettes -> https://r02b.github.io/seaborn_palettes/
sns.set_theme(style="whitegrid")

import matplotlib.pylab as pylab
params = {
    'figure.titlesize': 'xx-large',
    'legend.fontsize': 'x-large',
    'axes.labelsize': 'x-large',
    'axes.titlesize':'x-large',
    'xtick.labelsize':'x-large',
    'ytick.labelsize':'x-large',
    'axes.formatter.limits': (-10, 10)
}
pylab.rcParams.update(params)

import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
link = 'https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/839/original/Jamboree_Admission.cs
df = pd.read_csv(link, index_col = 'Serial No.')
df.head()
```

Out[2]:

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

Data Analysis & EDA

In [3]:
```
# checking shape
df.shape
```

Out[3]:
```
(500, 8)
```

In [4]:
```
#columns info
df.columns
```

Out[4]:
```
Index(['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR ', 'CGPA',
       'Research', 'Chance of Admit '],
      dtype='object')
```

In [5]:
```python
# Unique data for each column
df.nunique()
```

Out[5]:
```
GRE Score             49
TOEFL Score           29
University Rating      5
SOP                    9
LOR                    9
CGPA                 184
Research               2
Chance of Admit       61
dtype: int64
```

In [ ]:

In [6]:
```python
categorical_columns = ['University Rating', 'SOP', 'LOR ', 'Research']
numerical_columns = ['GRE Score', 'TOEFL Score', 'CGPA', 'Chance of Admit ']

categorical_columns, numerical_columns
```

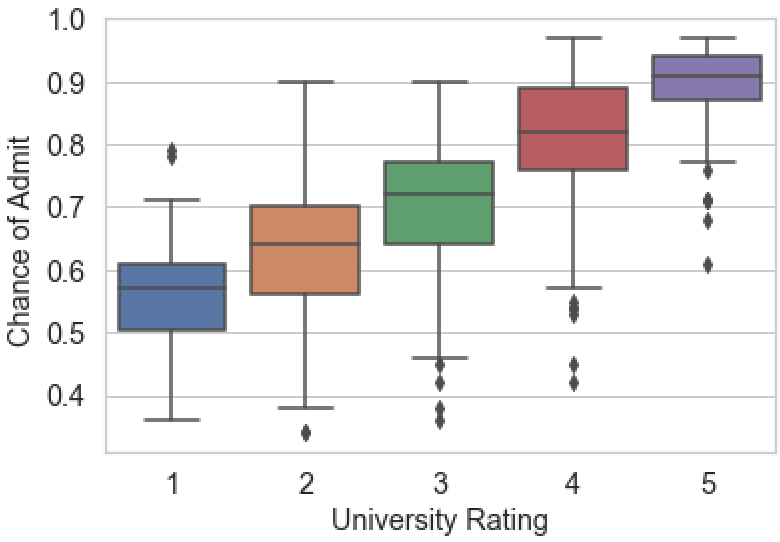
Out[6]:
```
(['University Rating', 'SOP', 'LOR ', 'Research'],
 ['GRE Score', 'TOEFL Score', 'CGPA', 'Chance of Admit '])
```

In [ ]:

In [7]:
```python
def describe_cat_column(column):
    print('-'*100)
    print(f"Unique values of {column} are : ", sorted(list(df[column].unique())))
    print('-'*100)

    print(df[column].value_counts())
    print('-'*100)

    sns.boxplot(x = column, y = 'Chance of Admit ', data=df)
```

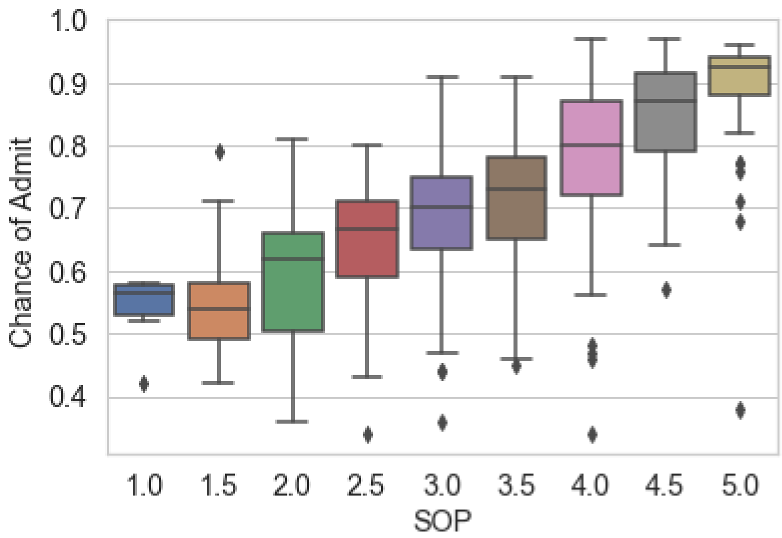
In [8]:
```python
describe_cat_column('University Rating')
```

```
----------------------------------------------------------------------------------------------------
Unique values of University Rating are :  [1, 2, 3, 4, 5]
----------------------------------------------------------------------------------------------------
University Rating
3    162
2    126
4    105
5     73
1     34
Name: count, dtype: int64
----------------------------------------------------------------------------------------------------
```



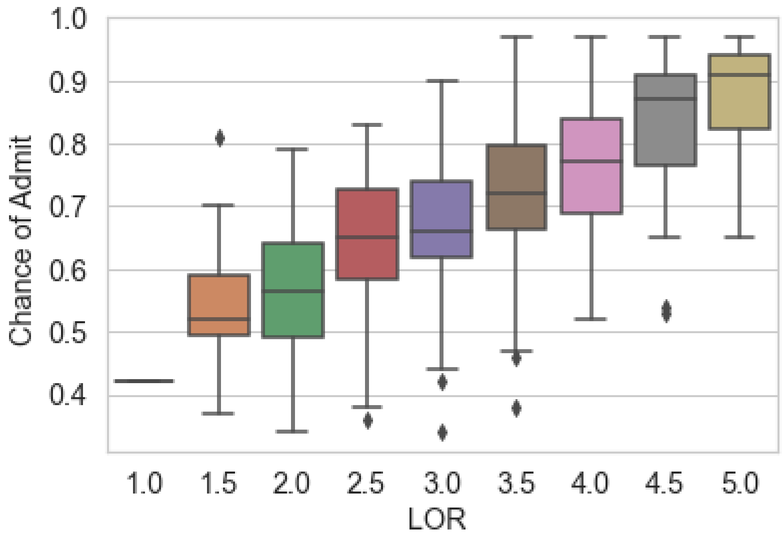
```
In [9]: describe_cat_column('SOP')
```

```
--------------------------------------------------------------------------------------------------
Unique values of SOP are :  [1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0]
--------------------------------------------------------------------------------------------------
SOP
4.0    89
3.5    88
3.0    80
2.5    64
4.5    63
2.0    43
5.0    42
1.5    25
1.0     6
Name: count, dtype: int64
--------------------------------------------------------------------------------------------------
```



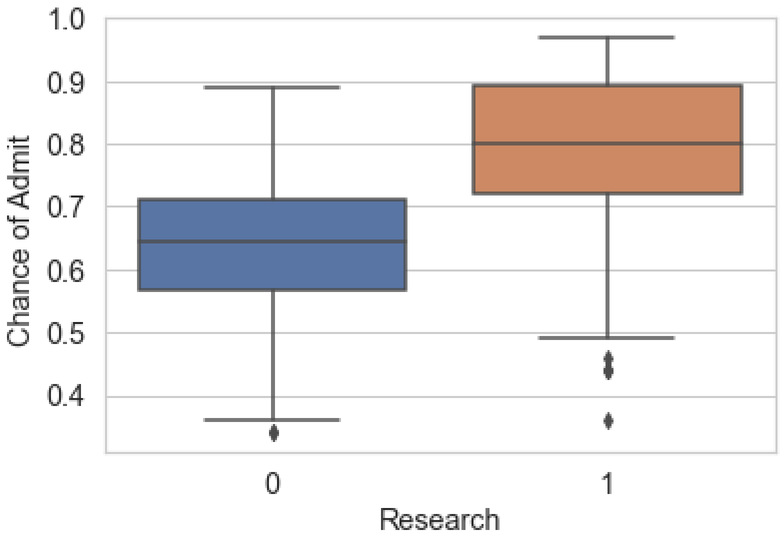
```
In [10]: describe_cat_column('LOR ')
```

```
----------------------------------------------------------------------------------------------------
Unique values of LOR  are :  [1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0]
----------------------------------------------------------------------------------------------------
LOR
3.0    99
4.0    94
3.5    86
4.5    63
2.5    50
5.0    50
2.0    46
1.5    11
1.0     1
Name: count, dtype: int64
----------------------------------------------------------------------------------------------------
```



In [11]:
```
describe_cat_column('Research')
```

```
----------------------------------------------------------------------------------------------------
Unique values of Research are :  [0, 1]
----------------------------------------------------------------------------------------------------
Research
1    280
0    220
Name: count, dtype: int64
----------------------------------------------------------------------------------------------------
```

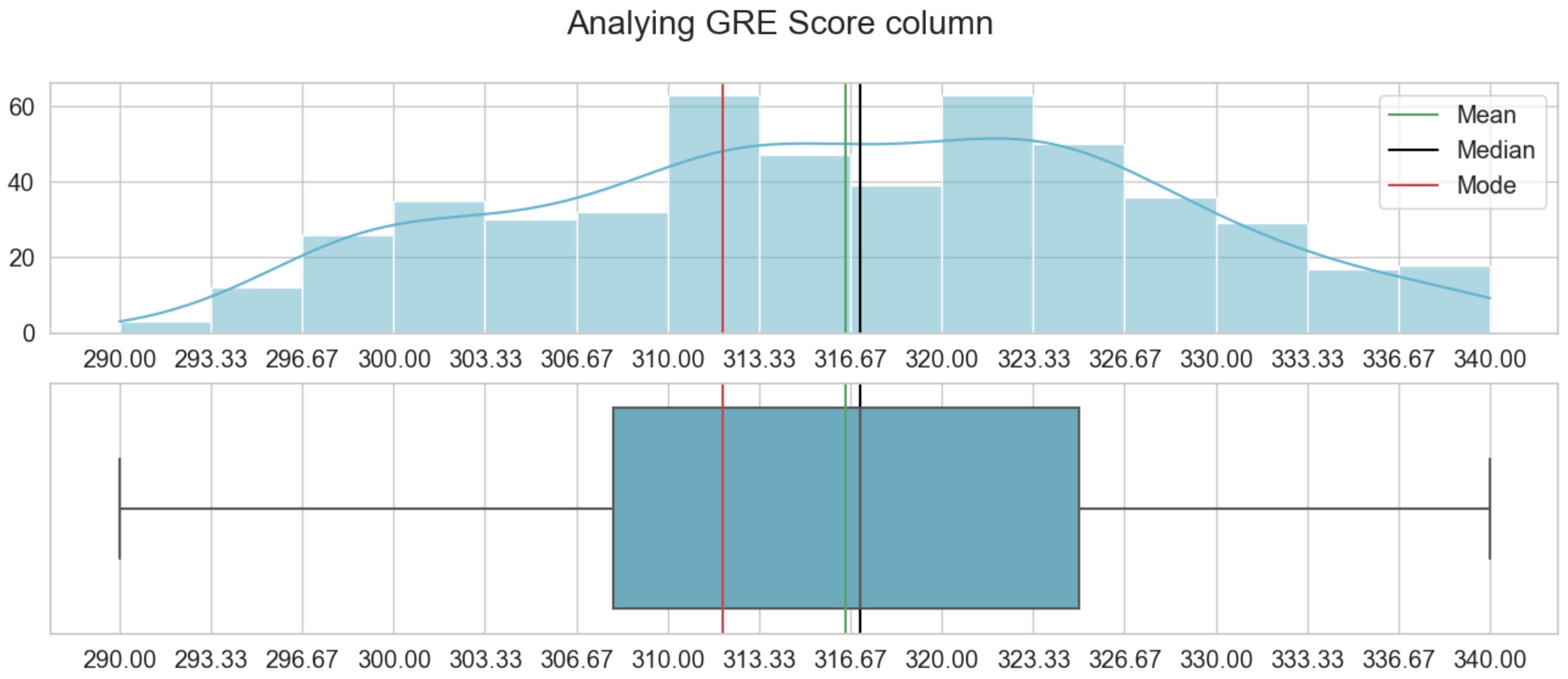


In [ ]:

In [12]:
```
def draw_plot_numerical(column, bins = 15, color='c'):
    fig = plt.figure(figsize = (16, 6), dpi=100)

    fig1 = fig.add_subplot(2,1,1)
    sns.histplot(x=df[column], kde='reg', bins=bins, color=color)
    plt.axvline(df[column].mean(), color="g", label='Mean')
    plt.axvline(df[column].median(), color="black", label = 'Median')
    plt.axvline(df[column].mode()[0], color="r", label = 'Mode')
    plt.legend()
    plt.ylabel('')
    plt.xlabel('')

    fig.add_subplot(2,1,2, sharex=fig1)
    sns.boxplot(x=df[column], color=color)

    plt.suptitle(f'Analying {column} column', fontsize=20)
    plt.axvline(df[column].mean(), color="g", label='Mean')
    plt.axvline(df[column].median(), color="black", label = 'Median')
    plt.axvline(df[column].mode()[0], color="r", label = 'Mode')
    plt.xlabel('')
    plt.xticks(np.linspace(df[column].min(), df[column].max(), num=bins+1))
    plt.show()

def describe_and_find_outliers(column):
    print('-'*100)
    print(df[column].describe().to_string())

    Q1 = df[column].quantile(.25)
    Q3 = df[column].quantile(.75)
    IQR = Q3 - Q1

    right = df[df[column] > (Q3 + 1.5 * IQR)]
    left = df[df[column] < (Q1 - 1.5 * IQR)]

    print('-'*100)
    if(len(left) > 0):
        print('\nOutliers on left extreme:-\n')
        print(f'Total {len(left)} outliers which are lesser than {Q1 - 1.5 * IQR}')
    else:
        print('No outliers on left extreme')
```

```
    print('-'*100)
    if(len(right) > 0) :
        print('\nOutliers on right extreme:-\n')
        print(f'Total {len(right)} outliers which are greater than {Q3 + 1.5 * IQR}')
    else:
        print('No outliers on right extreme')

    print('-'*100)

    return left, right
```

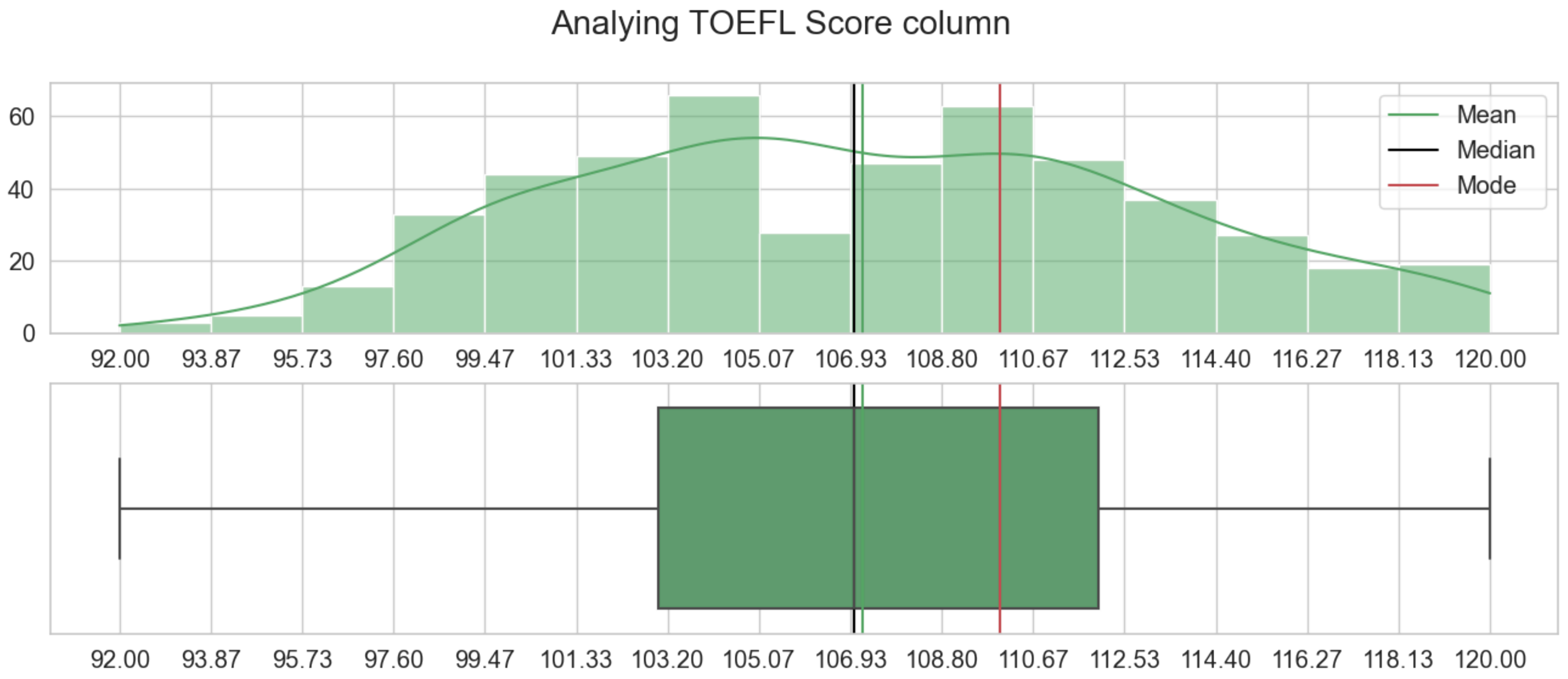
In [13]:
```
column = 'GRE Score'

draw_plot_numerical(column, 15)
temp_left_outliers, temp_right_outliers = describe_and_find_outliers(column)
```



```
----------------------------------------------------------------------------------------------------
count    500.000000
mean     316.472000
std       11.295148
min      290.000000
25%      308.000000
50%      317.000000
75%      325.000000
max      340.000000
----------------------------------------------------------------------------------------------------
No outliers on left extreme
----------------------------------------------------------------------------------------------------
No outliers on right extreme
----------------------------------------------------------------------------------------------------
```

In [ ]:

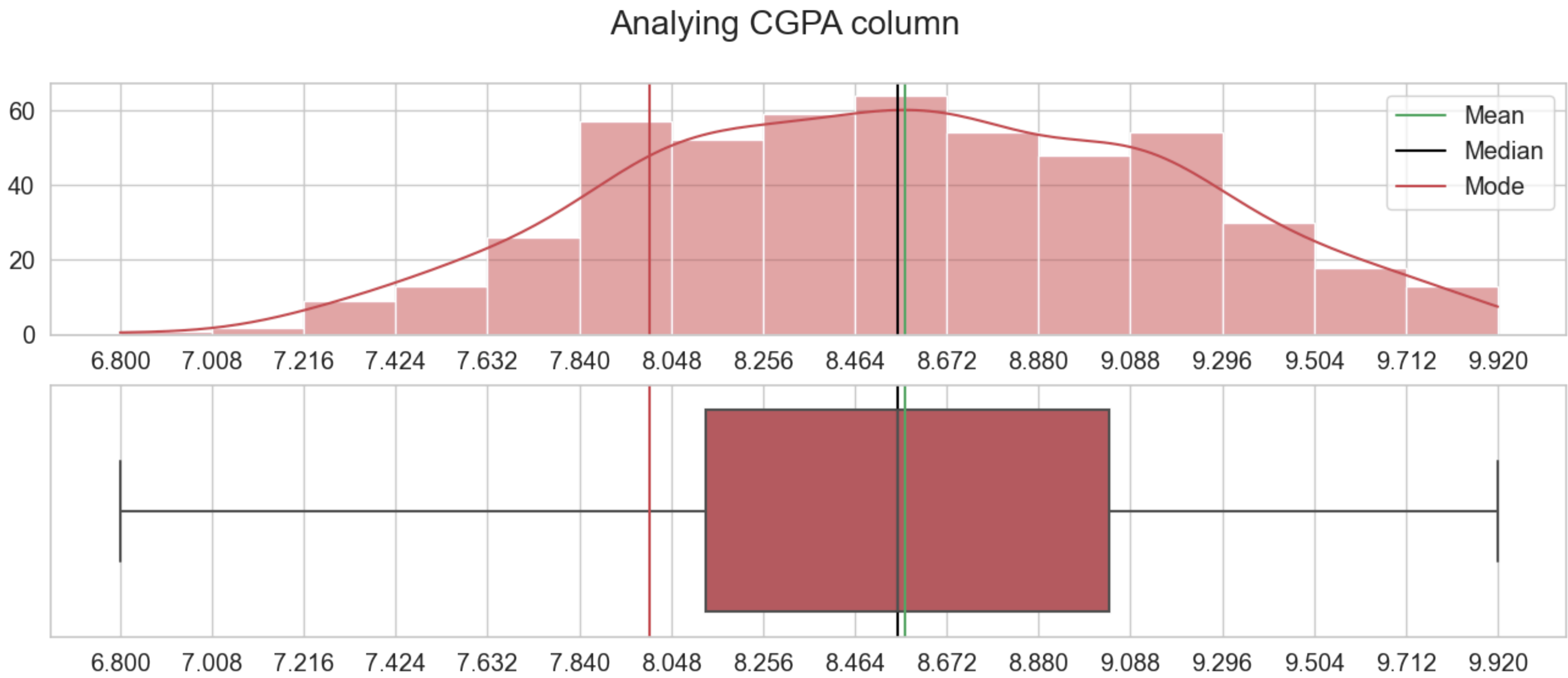
In [14]:
```
column = 'TOEFL Score'

draw_plot_numerical(column, 15, color='g')
humidity_left_outliers, humidity_right_outliers = describe_and_find_outliers(column)
```



```
--------------------------------------------------------------------------------------------------
count    500.000000
mean     107.192000
std        6.081868
min       92.000000
25%      103.000000
50%      107.000000
75%      112.000000
max      120.000000
--------------------------------------------------------------------------------------------------
No outliers on left extreme
--------------------------------------------------------------------------------------------------
No outliers on right extreme
--------------------------------------------------------------------------------------------------
```

In [ ]:

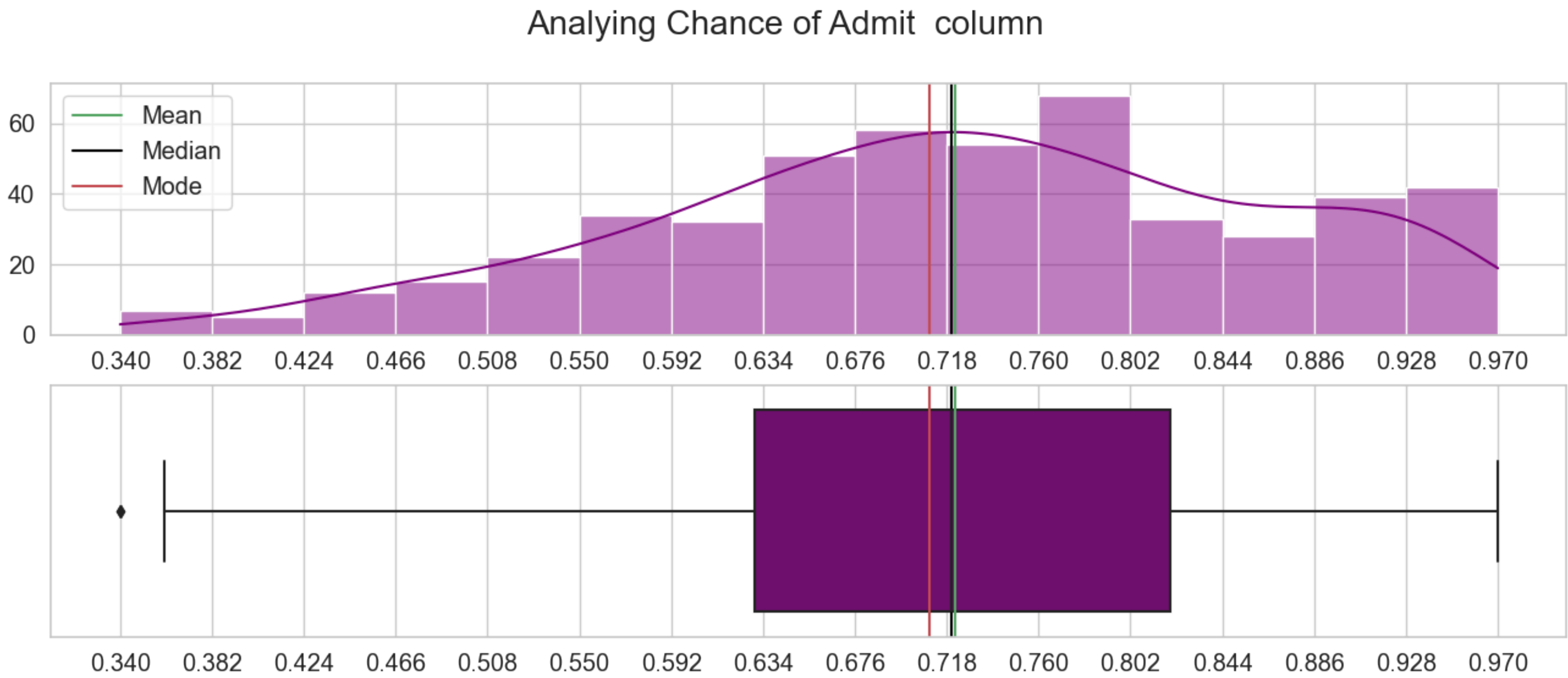
In [15]:
```
column = 'CGPA'

draw_plot_numerical(column, 15, color='r')
humidity_left_outliers, humidity_right_outliers = describe_and_find_outliers(column)
```



```
----------------------------------------------------------------------------------------------------
count    500.000000
mean       8.576440
std        0.604813
min        6.800000
25%        8.127500
50%        8.560000
75%        9.040000
max        9.920000
----------------------------------------------------------------------------------------------------
No outliers on left extreme
----------------------------------------------------------------------------------------------------
No outliers on right extreme
----------------------------------------------------------------------------------------------------
```

In [ ]:

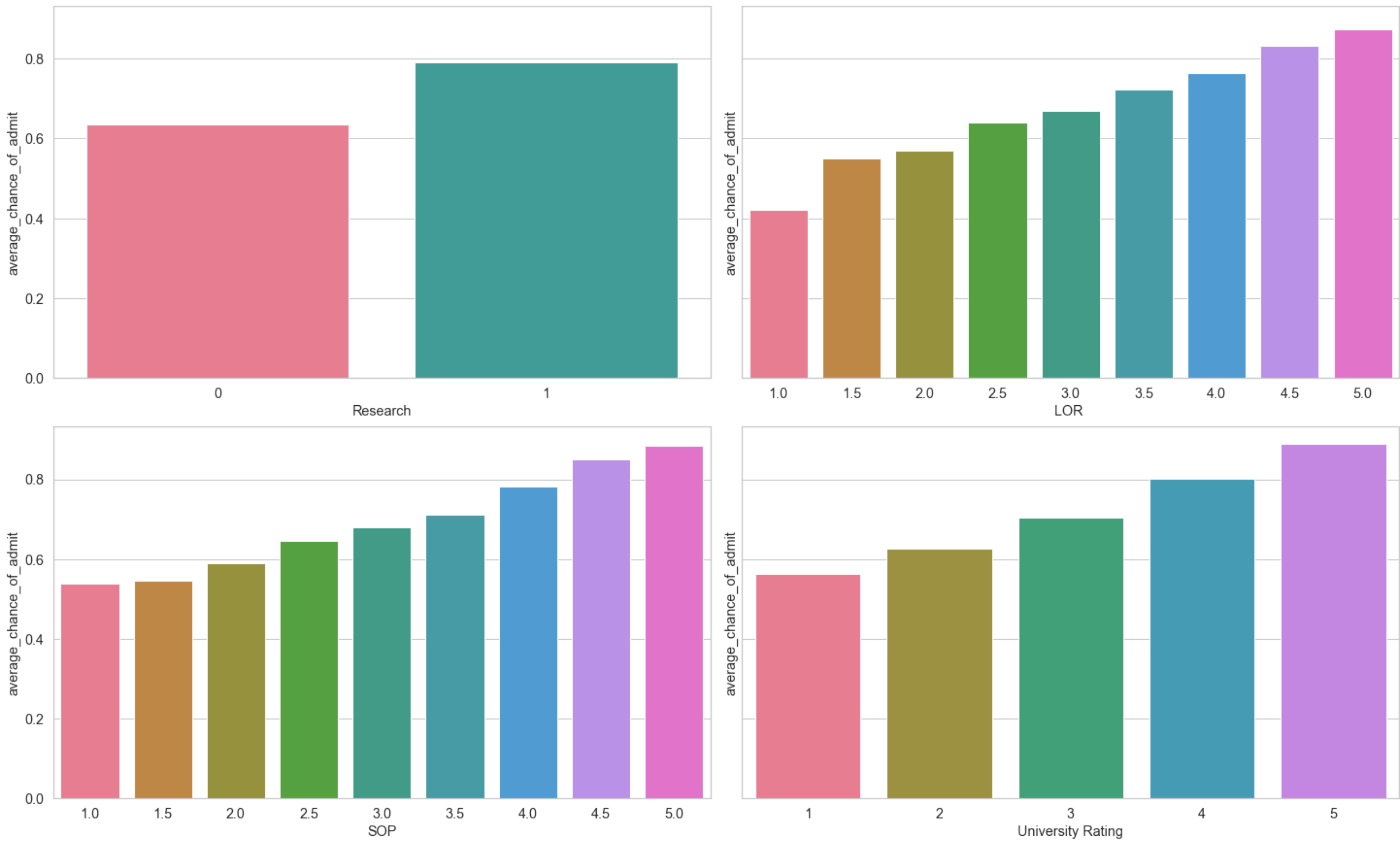
In [16]:
```
column = 'Chance of Admit '

draw_plot_numerical(column, 15, color='purple')
humidity_left_outliers, humidity_right_outliers = describe_and_find_outliers(column)
```



```
--------------------------------------------------------------------------------------------------
count    500.00000
mean       0.72174
std        0.14114
min        0.34000
25%        0.63000
50%        0.72000
75%        0.82000
max        0.97000
--------------------------------------------------------------------------------------------------

Outliers on left extreme:-

Total 2 outliers which are lesser than 0.3450000000000001
--------------------------------------------------------------------------------------------------
No outliers on right extreme
--------------------------------------------------------------------------------------------------
```

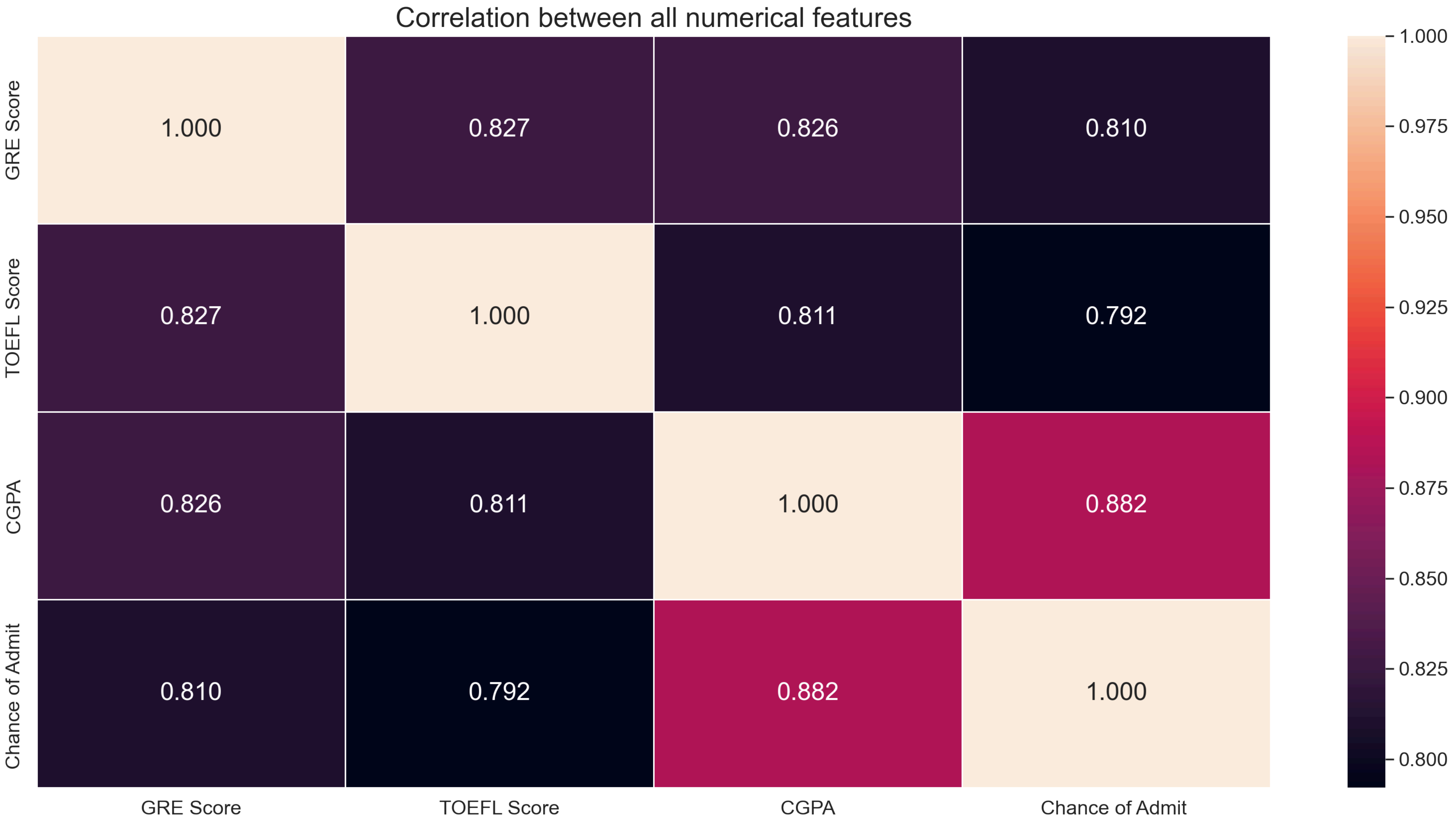
In [ ]:

In [17]:
```
fig, ax = plt.subplots(nrows=2, ncols=2, figsize=(20, 12), constrained_layout=True, sharey = True)

cols = ['University Rating', 'SOP', 'LOR ', 'Research']
for i in range(len(ax)):
    for j in range(len(ax[i])):
        column = cols.pop()

        data = df.groupby(column).aggregate(average_chance_of_admit = ('Chance of Admit ', 'mean')).reset_inc
        sns.barplot(data = data, x = column, y = 'average_chance_of_admit', ax=ax[i][j], palette='husl')
```


average_chance_of_admit
0.0
0.2
0.4
0.6
0.8
0
1
Research
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
LOR
SOP
1
2
3
4
5
University Rating

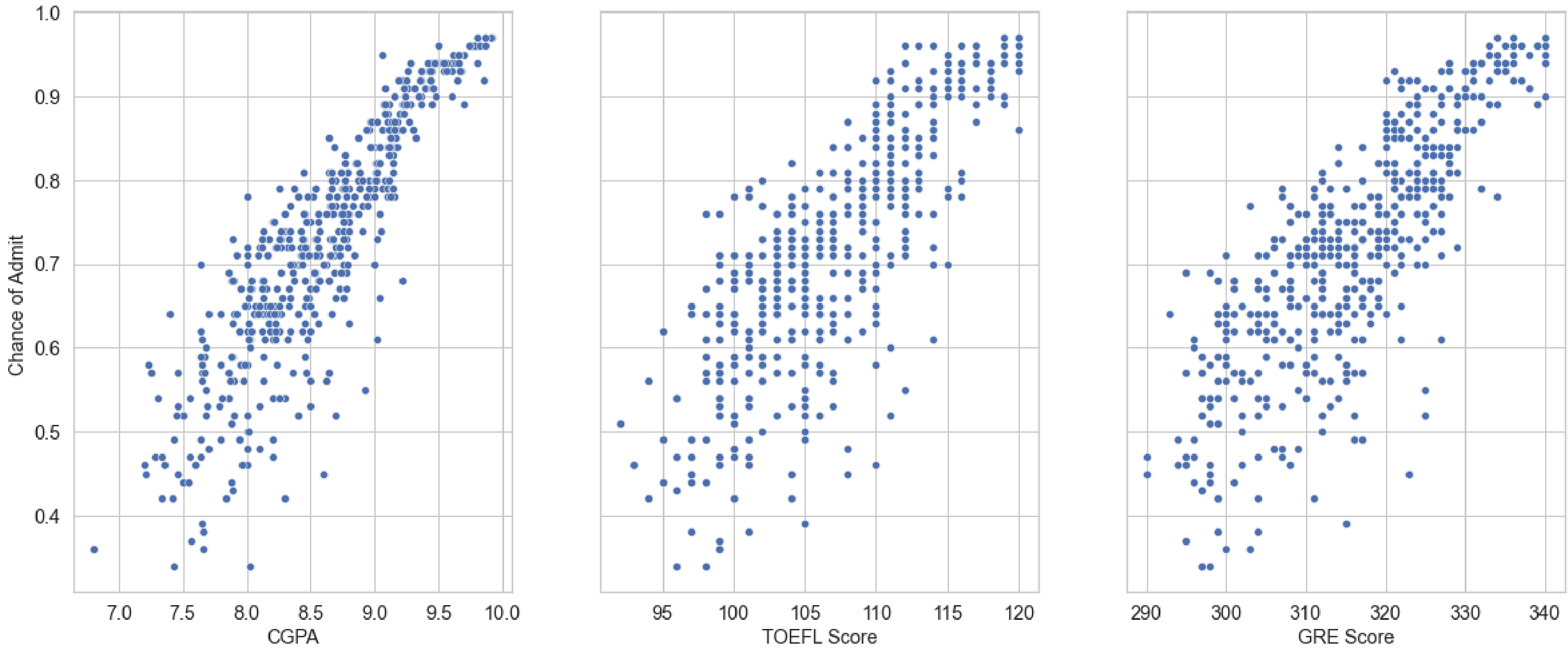
In [ ]:

In [18]:
```
plt.figure(figsize=(20, 10), dpi=200)
sns.heatmap(data=df[numerical_columns].corr(), annot=True,
            linewidths=1, annot_kws={"fontsize":18}, fmt='.3f')

plt.title('Correlation between all numerical features', fontsize=20)
plt.xlabel('')
plt.ylabel('')
plt.xticks(rotation=0)
plt.show()
```



In [ ]:

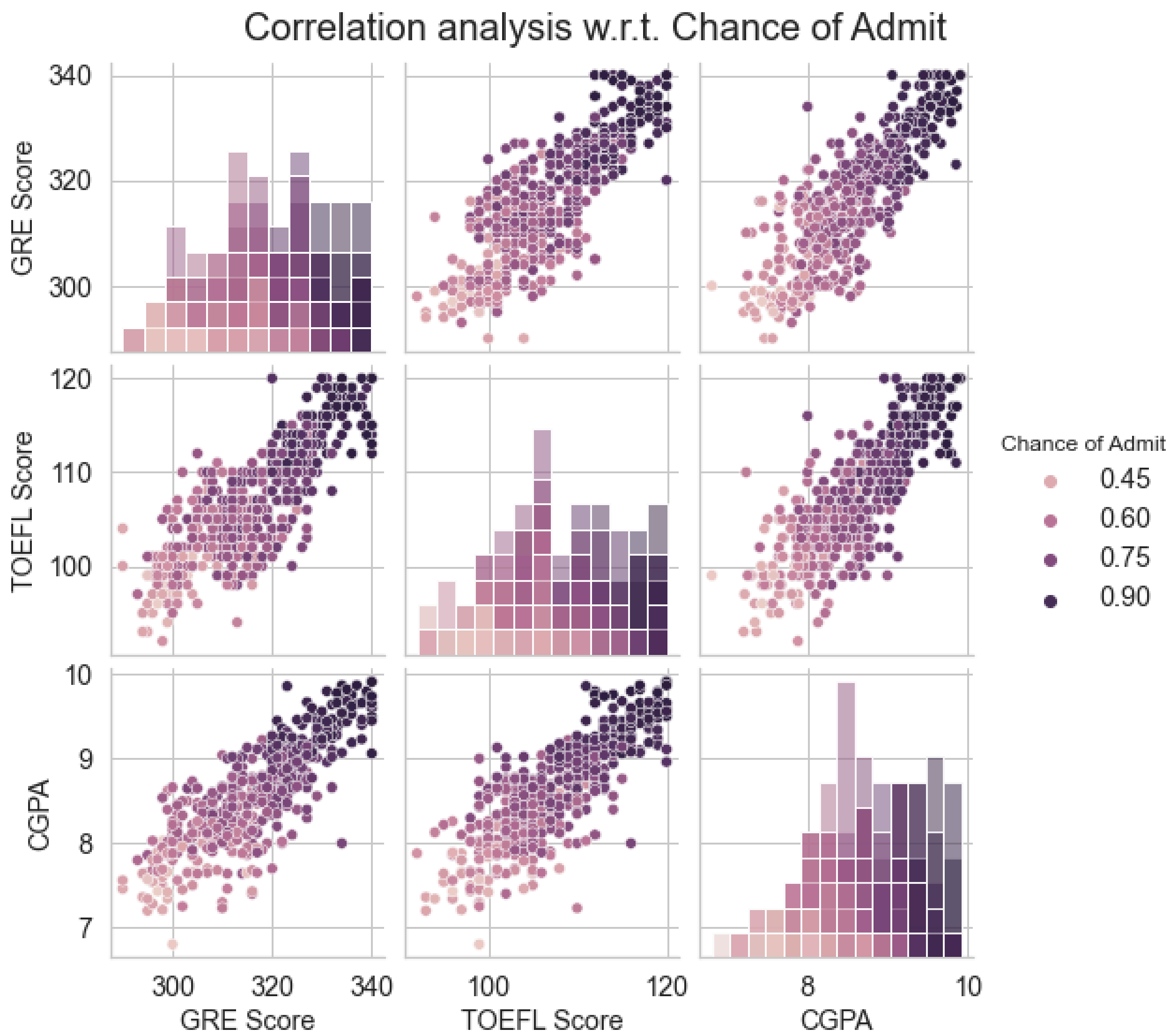
In [19]:
```
fig, ax = plt.subplots(nrows=1, ncols=3, figsize=(20, 8), constrained_layout=False, sharey = True)

cols = ['GRE Score', 'TOEFL Score', 'CGPA']
for i in range(len(ax)):
        column = cols.pop()
        sns.scatterplot(data = df, x = column, y = 'Chance of Admit ', ax=ax[i])
```



In [ ]:

In [20]:
```
df_numerical_pairplot = df[['GRE Score', 'TOEFL Score', 'CGPA', 'Chance of Admit ']]

plt.figure(figsize=(20, 20), dpi=400)
sns.pairplot(data=df_numerical_pairplot, hue='Chance of Admit ', diag_kind="hist")
plt.suptitle('Correlation analysis w.r.t. Chance of Admit', fontsize=20, position=(0.5, 1.0+0.03))
plt.show()
```

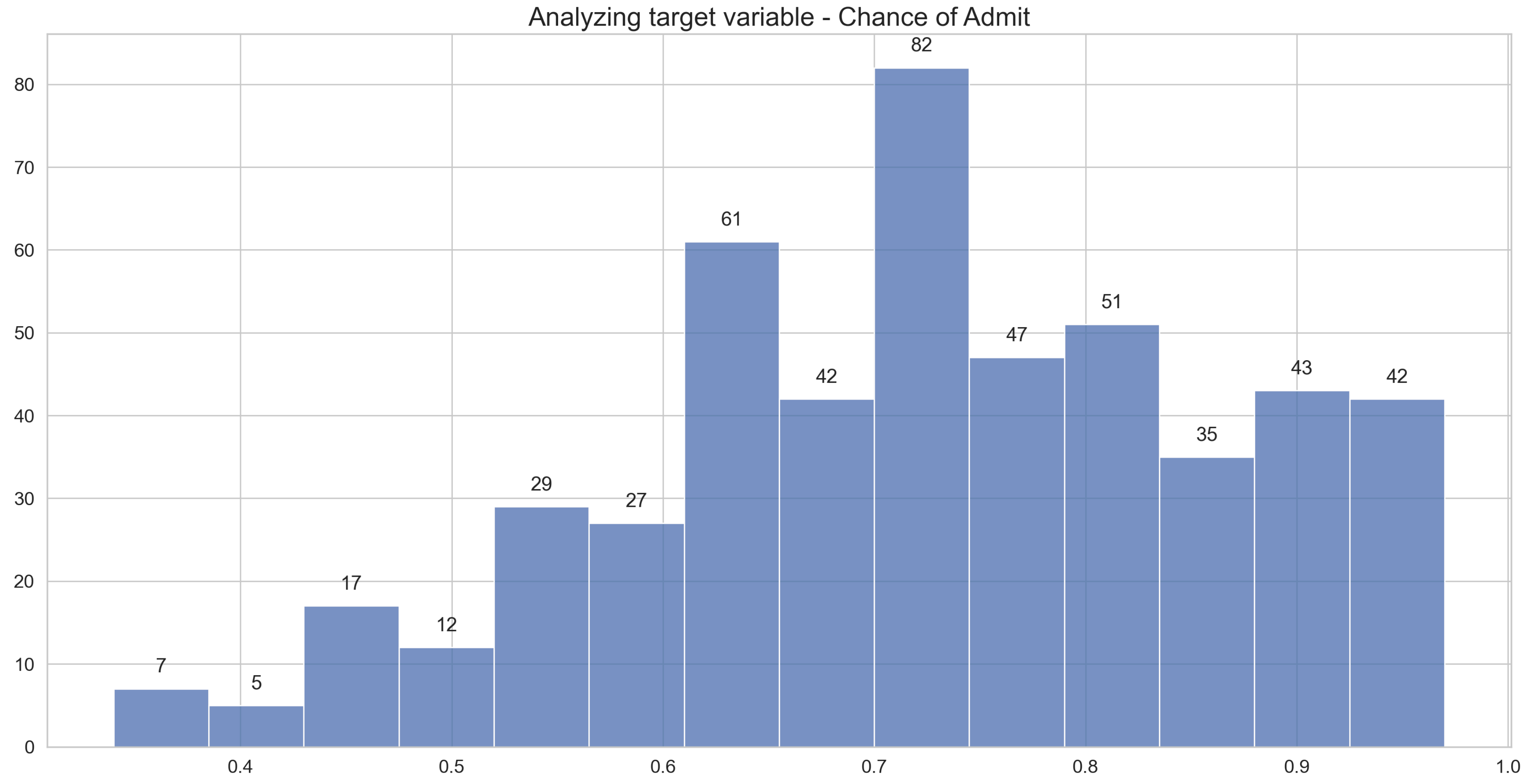
```
<Figure size 8000x8000 with 0 Axes>
```



Correlation analysis w.r.t. Chance of Admit
GRE Score
TOEFL Score
CGPA
340
320
300
120
110
100
10
9
8
7
300
320
340
100
120
8
10
GRE Score
TOEFL Score
CGPA
Chance of Admit
0.45
0.60
0.75
0.90

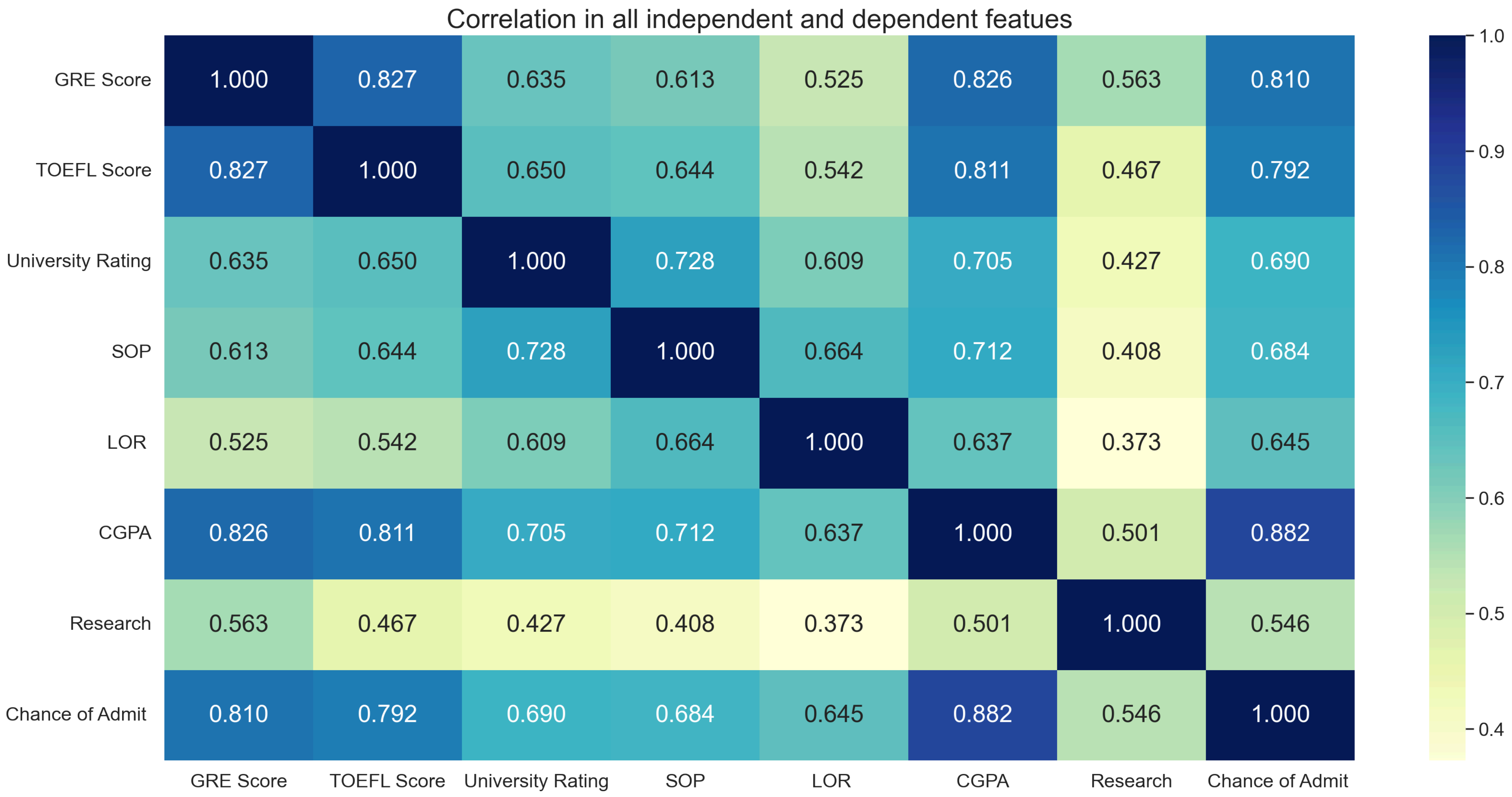

```
In [ ]:
```

In [21]:
```
plt.figure(figsize=(20, 10), dpi=200)
ax = sns.histplot(data=df, x='Chance of Admit ', palette='husl')
for bar in ax.patches:
    ax.text(bar.get_x() + bar.get_width() / 2, bar.get_height() + 2, bar.get_height(),
            horizontalalignment='center', fontsize = 15)

plt.title('Analyzing target variable - Chance of Admit', fontsize=20)
plt.xlabel('')
plt.ylabel('')
plt.show()
```



In [ ]:

In [22]:
```
plt.figure(figsize=(20, 10), dpi=200)
ax = sns.heatmap(df.corr(), cmap="YlGnBu", annot=True, annot_kws={"fontsize":18}, fmt='.3f')
plt.title('Correlation in all independent and dependent featues', fontsize=20)
plt.xlabel('')
plt.ylabel('')
plt.show()
```



	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
GRE Score	1.000	0.827	0.635	0.613	0.525	0.826	0.563	0.810
TOEFL Score	0.827	1.000	0.650	0.644	0.542	0.811	0.467	0.792
University Rating	0.635	0.650	1.000	0.728	0.609	0.705	0.427	0.690
SOP	0.613	0.644	0.728	1.000	0.664	0.712	0.408	0.684
LOR	0.525	0.542	0.609	0.664	1.000	0.637	0.373	0.645
CGPA	0.826	0.811	0.705	0.712	0.637	1.000	0.501	0.882
Research	0.563	0.467	0.427	0.408	0.373	0.501	1.000	0.546
Chance of Admit	0.810	0.792	0.690	0.684	0.645	0.882	0.546	1.000

In [ ]:

Data Preprocessing

In [23]:
```
df.columns
```

Out[23]:
```
Index(['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR ', 'CGPA',
       'Research', 'Chance of Admit '],
      dtype='object')
```

In [24]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 500 entries, 1 to 500
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    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 of Admit    500 non-null    float64
dtypes: float64(4), int64(4)
memory usage: 35.2 KB
```

In [25]:
```
# duplicate rows ?
df.duplicated().any()
```

Out[25]:
```
False
```

In [26]:
```
# Check missing values in any column
df.isnull().sum()
```

Out[26]:
```
GRE Score            0
TOEFL Score          0
University Rating    0
SOP                  0
LOR                  0
CGPA                 0
Research             0
Chance of Admit      0
dtype: int64
```

In [27]:
```
# No missing values
```

In [28]:
```
# Outlier treatment
# We saw outliers in Chance of Admit (target) column, let's check

column = 'Chance of Admit '

Q1 = df[column].quantile(.25)
Q3 = df[column].quantile(.75)
IQR = Q3 - Q1

right = df[df[column] > (Q3 + 1.5 * IQR)] # No right outliers
left = df[df[column] < (Q1 - 1.5 * IQR)]

print('Outliers on left extreme:-')
print(f'Total {len(left)} outliers which are lesser than {round(Q1 - 1.5 * IQR, 3)}')

df[df['Chance of Admit '] < (Q1 - 1.5 * IQR)]
```

```
Outliers on left extreme:-
Total 2 outliers which are lesser than 0.345
```

Out[28]:

Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
93	298	98	2	4.0	3.0	8.03	0	0.34
377	297	96	2	2.5	2.0	7.43	0	0.34

In [29]:
```
# We can see there are 2 outliers on left extreme which are very close to Q1 - 1.5 * IQR value
# Thus not removing as data points are already less
```

In [30]:
```
# Data preparation for modeling
```

In [31]:
```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

df = pd.DataFrame(scaler.fit_transform(df), columns=df.columns)
df.head()
```

Out[31]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	0.94	0.928571	0.75	0.875	0.875	0.913462	1.0	0.920635
1	0.68	0.535714	0.75	0.750	0.875	0.663462	1.0	0.666667
2	0.52	0.428571	0.50	0.500	0.625	0.384615	1.0	0.603175
3	0.64	0.642857	0.50	0.625	0.375	0.599359	1.0	0.730159
4	0.48	0.392857	0.25	0.250	0.500	0.451923	0.0	0.492063

In [ ]:
```

```

Model building

In [32]:
```
def adj_r(r_squared, X, y):
    n = X.shape[0]
    k = X.shape[1]

    adj_r_squared = 1 - (1 - r_squared) * (n - 1) / (n - k - 1)
    return adj_r_squared
```

In [33]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

In [34]:
```
y = df['Chance of Admit ']
X = df.drop('Chance of Admit ', axis=1)
y.shape, X.shape
```

Out[34]:
```
((500,), (500, 7))
```

In [35]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

Out[35]:
```
((400, 7), (400,), (100, 7), (100,))
```

Linear Regression

In [36]:
```
model = LinearRegression()
model.fit(X_train, y_train)
```

Out[36]:
```
▾ LinearRegression
LinearRegression()
```

In [37]:
```
model.coef_
```

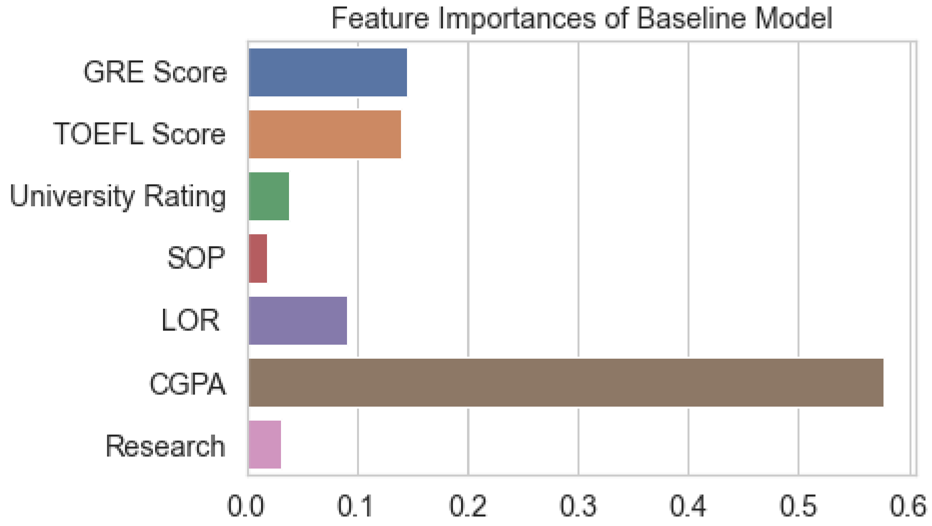
Out[37]:
```
array([0.14541621, 0.14106352, 0.03891338, 0.01907985, 0.09159877,
       0.57796411, 0.03157108])
```

In [38]:
```
model.intercept_
```

Out[38]:
```
0.018693219780458 57
```

In [39]:
```
# Feature importance
```

In [40]:
```
sns.barplot(x = model.coef_,y = X_train.columns)
plt.ylabel('')
plt.title("Feature Importances of Baseline Model")
plt.show()
```



In [41]:
```
print('\n','-'*30,'R2 Score','-'*30,sep = '')
r2_train = model.score(X_train, y_train)
r2_test = model.score(X_test, y_test)
print("Training R2 Score for Model:",r2_train)
print("Testing R2 Score for Model:",r2_test)

print('\n','-'*30,'Adj R2','-'*30,sep = '')
print("Training adj R2 Score for Model:",adj_r(r2_train,X_train,y_train))
print("Testing adj R2 Score for Model:",adj_r(r2_test,X_test,y_test))
```

```
------------------------------R2 Score------------------------------
Training R2 Score for Model: 0.8215099192361265
Testing R2 Score for Model: 0.820874170310373

------------------------------Adj R2------------------------------
Training adj R2 Score for Model: 0.818322596365343
Testing adj R2 Score for Model: 0.8072450310948579
```

In [ ]:

StatsModel statistics

In [42]:
```
# Model statistics
import statsmodels.api as sm

X_sm = sm.add_constant(X_train)
sm_model = sm.OLS(y_train, X_sm).fit()

print(sm_model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:      Chance of Admit   R-squared:                       0.822
Model:                          OLS   Adj. R-squared:                  0.818
Method:               Least Squares   F-statistic:                     257.7
Date:              Sat, 11 May 2024   Prob (F-statistic):          2.10e-142
Time:                      23:05:56   Log-Likelihood:                 374.46
No. Observations:               400   AIC:                            -732.9
Df Residuals:                   392   BIC:                            -701.0
Df Model:                         7
Covariance Type:          nonrobust
=====================================================================================
                        coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------------
const                 0.0187      0.016      1.155      0.249      -0.013       0.051
GRE Score             0.1454      0.046      3.135      0.002       0.054       0.237
TOEFL Score           0.1411      0.045      3.156      0.002       0.053       0.229
University Rating     0.0389      0.028      1.387      0.166      -0.016       0.094
SOP                   0.0191      0.032      0.591      0.555      -0.044       0.083
LOR                   0.0916      0.029      3.105      0.002       0.034       0.150
CGPA                  0.5780      0.054     10.743      0.000       0.472       0.684
Research              0.0316      0.012      2.668      0.008       0.008       0.055
==============================================================================
Omnibus:                     80.594   Durbin-Watson:                   1.932
Prob(Omnibus):                0.000   Jarque-Bera (JB):              167.116
Skew:                        -1.064   Prob(JB):                     5.14e-37
Kurtosis:                     5.346   Cond. No.                         23.4
==============================================================================

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

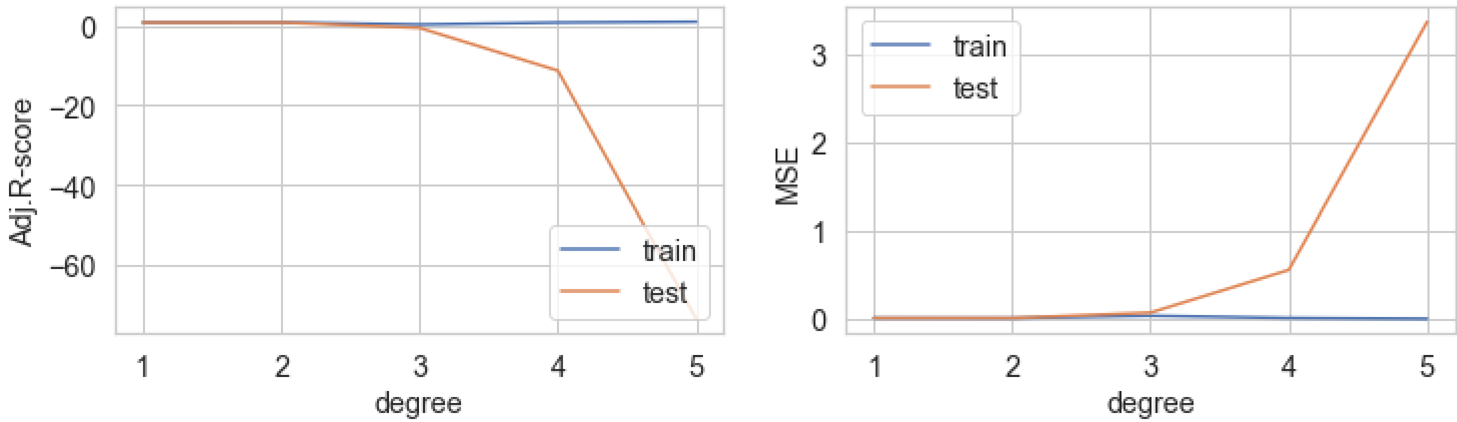
In [ ]:

Polynomial regression

In [43]:
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from sklearn.metrics import mean_absolute_error as mae, mean_squared_error as mse

degrees = 6
train_scores = []
test_scores = []

train_loss = []
test_loss = []

for degree in range(1, degrees):

    polyreg_scaled = make_pipeline(PolynomialFeatures(degree), LinearRegression())
    polyreg_scaled.fit(X_train, y_train)

    train_score = polyreg_scaled.score(X_train, y_train)
    test_score = polyreg_scaled.score(X_test, y_test)

    train_scores.append(adj_r(train_score,X_train,y_train))
    test_scores.append(adj_r(test_score,X_test,y_test))

    output1 = polyreg_scaled.predict(X_train)
    output2 = polyreg_scaled.predict(X_test)

    train_loss.append(mse(y_train,output1))
    test_loss.append(mse(y_test,output2))

list(zip(train_scores, test_scores))
```

Out[43]:
```
[(0.818322596365343, 0.8072450310948581),
 (0.8343814516805089, 0.8103896615317002),
 (0.3102578041958476, -0.5425975146644288),
 (0.8116824491657619, -11.2873982976964),
 (1.0, -73.58353630304119)]
```

In [44]:
```
fig, axes = plt.subplots(1, 2, figsize=(12, 3))
axes[0].plot(list(range(1, 6)), train_scores, label="train")
axes[0].plot(list(range(1, 6)), test_scores, label="test")
axes[0].legend(loc='lower right')
axes[0].set_xlabel("degree")
axes[0].set_ylabel("Adj.R-score")

axes[1].plot(list(range(1, 6)), train_loss, label="train")
axes[1].plot(list(range(1, 6)), test_loss, label="test")
axes[1].legend(loc='upper left')
axes[1].set_xlabel("degree")
axes[1].set_ylabel("MSE")

plt.show()
```



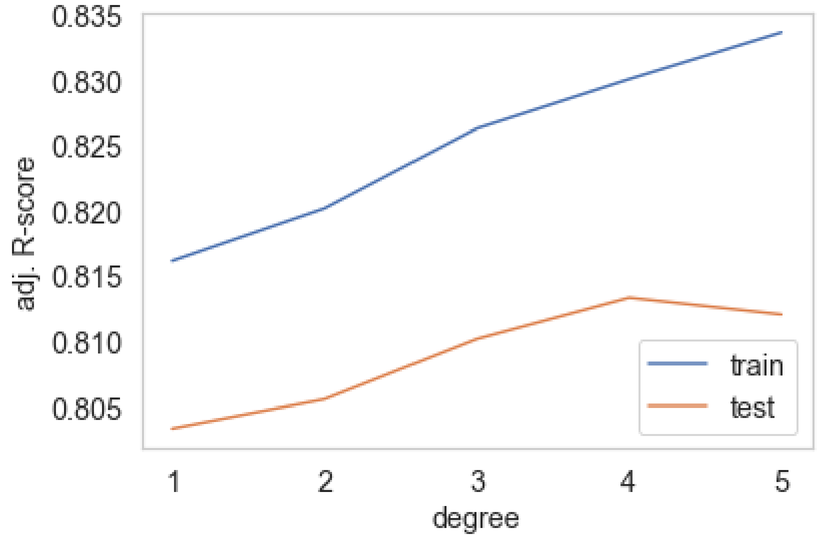
In [45]:
```
# Best degree is 2 for polynomial regression
```

In [ ]:
```
```

Ridge regression

In [46]:
```
# Ridge regression
from sklearn.linear_model import Ridge
```

In [47]:
```
max_degree = 6 # max polynomial degree
train_scores = []
test_scores = []

for degree in range(1, max_degree):
    polyreg_scaled = make_pipeline(PolynomialFeatures(degree), Ridge())
    polyreg_scaled.fit(X_train, y_train)
    train_score = adj_r(polyreg_scaled.score(X_train, y_train), X_train, y_train)
    test_score= adj_r(polyreg_scaled.score(X_test, y_test), X_test, y_test)
    train_scores.append(train_score)
    test_scores.append(test_score)

plt.figure()
plt.plot(list(range(1, 6)), train_scores, label="train")
plt.plot(list(range(1, 6)), test_scores, label="test")
plt.legend(loc='lower right')
plt.xlabel("degree")
plt.ylabel("adj. R-score")
plt.grid()
plt.show()
```



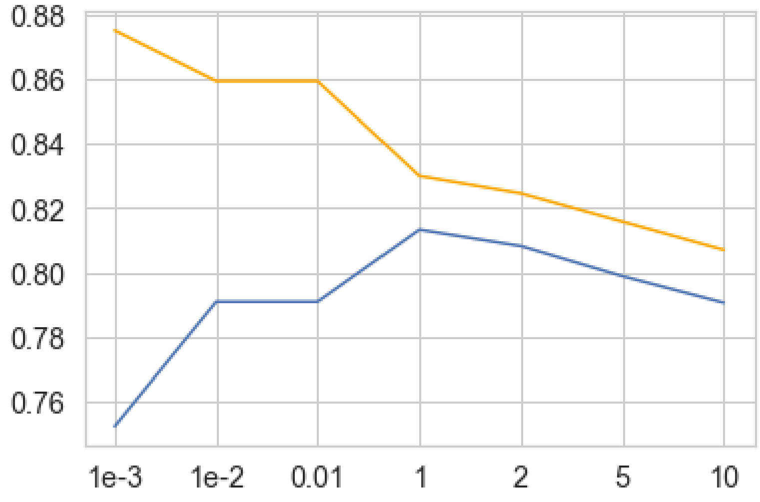
In [48]:
```
list(zip(train_scores, test_scores))
```

Out[48]:
```
[(0.8161639848952402, 0.8033038729045326),
 (0.8201824597832146, 0.8055616138667799),
 (0.8263128737192985, 0.8101710534304669),
 (0.830075653239801, 0.8133295202005358),
 (0.8336543468838196, 0.8120487565940218)]
```

In [49]:
```
# Best degree 4 for ridge regression
```

In [50]:

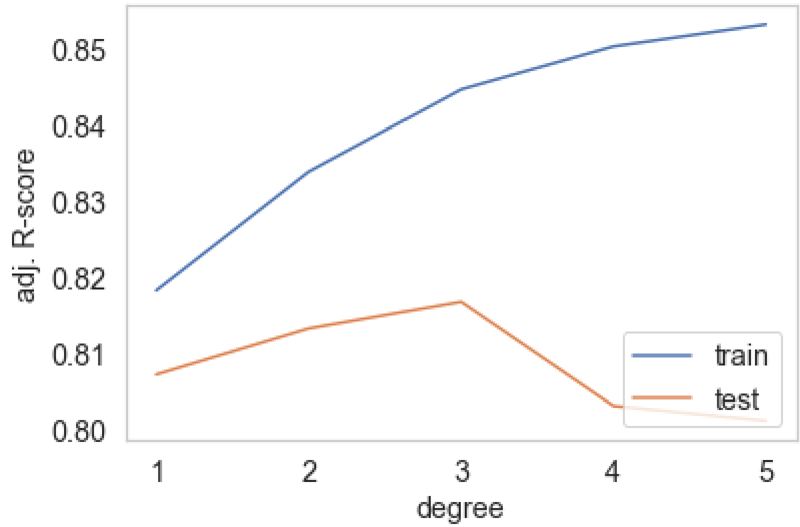
```
# alpha

train_scores = []
test_scores = []
rate_list = [1e-3, 1e-2, 0.01, 1, 2, 5,10]

for rate in rate_list:
    polyreg_scaled = make_pipeline(PolynomialFeatures(4), Ridge(alpha=rate))
    polyreg_scaled.fit(X_train, y_train)
    train_score = adj_r(polyreg_scaled.score(X_train, y_train), X_train, y_train)
    test_score= adj_r(polyreg_scaled.score(X_test, y_test), X_test, y_test)
    train_scores.append(train_score)
    test_scores.append(test_score)

alpha_values = ['1e-3', '1e-2', '0.01', '1', '2', '5','10']
sns.lineplot(x = alpha_values,y = test_scores)
sns.lineplot(x = alpha_values,y = train_scores, color='orange')
plt.show()
```



In [51]:
```
list(zip(train_scores, test_scores))
```
Out[51]:
```
[(0.8752210089273907, 0.7522079038261402),
 (0.8594724385938847, 0.7909020581802956),
 (0.8594724385938847, 0.7909020581802956),
 (0.830075653239801, 0.8133295202005358),
 (0.8246589241497648, 0.8082402577252896),
 (0.815857636358381, 0.7988852161462322),
 (0.8070742980727972, 0.7905989755828433)]
```
In [52]:
```
# Best alpha value as 1 for ridge
```
In [ ]:
```

```
Lasso Regression

In [53]:
```
from sklearn.linear_model import Lasso
```

In [54]:
```
max_degree = 6 # max polynomial degree
train_scores = []
test_scores = []

for degree in range(1, max_degree):
    polyreg_scaled = make_pipeline(PolynomialFeatures(degree), Lasso(alpha=0.00001))
    polyreg_scaled.fit(X_train, y_train)
    train_score = adj_r(polyreg_scaled.score(X_train, y_train), X_train, y_train)
    test_score= adj_r(polyreg_scaled.score(X_test, y_test), X_test, y_test)
    train_scores.append(train_score)
    test_scores.append(test_score)

plt.figure()
plt.plot(list(range(1, 6)), train_scores, label="train")
plt.plot(list(range(1, 6)), test_scores, label="test")
plt.legend(loc='lower right')
plt.xlabel("degree")
plt.ylabel("adj. R-score")
plt.grid()
plt.show()
```

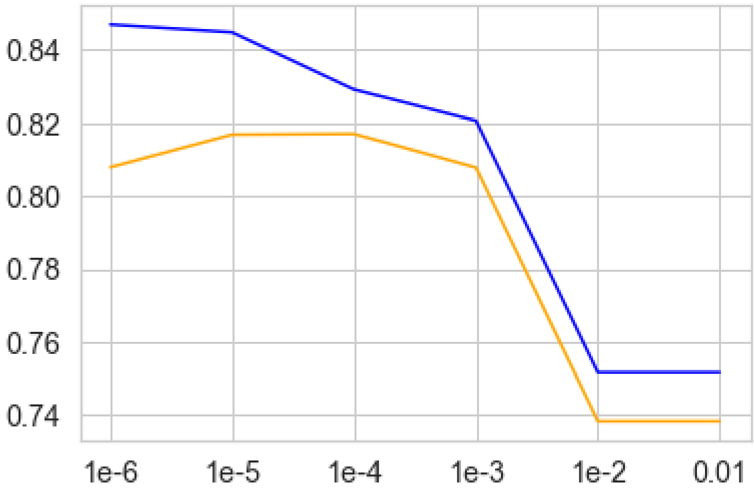


In [55]:
```
list(zip(train_scores, test_scores))
```

Out[55]:
```
[(0.8183225354081434, 0.8072331416608731),
 (0.8339824855416468, 0.8132971623647036),
 (0.8448311997644529, 0.8168152886406825),
 (0.850499097994552, 0.8030888287517679),
 (0.8533938353644899, 0.8011607224349304)]
```

In [56]:
```
# Best degree 1 for lasso regression
```

In [57]:
```
# alpha

train_scores = []
test_scores = []
rate_list = [1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 0.01]

for rate in rate_list:
    polyreg_scaled = make_pipeline(PolynomialFeatures(3), Lasso(alpha=rate))
    polyreg_scaled.fit(X_train, y_train)
    train_score = adj_r(polyreg_scaled.score(X_train, y_train), X_train, y_train)
    test_score= adj_r(polyreg_scaled.score(X_test, y_test), X_test, y_test)
    train_scores.append(train_score)
    test_scores.append(test_score)

alpha_values = ['1e-6', '1e-5', '1e-4', '1e-3', '1e-2', '0.01']
sns.lineplot(x = alpha_values,y = test_scores, color='orange')
sns.lineplot(x = alpha_values,y = train_scores, color='blue')
plt.show()
```



In [58]:
```
list(zip(train_scores, test_scores))
```

Out[58]:
```
[(0.8469702557945598, 0.8079054847277304),
 (0.8448311997644529, 0.8168152886406825),
 (0.8291728839705977, 0.8170183453359505),
 (0.8206210286127736, 0.8078154077097811),
 (0.7516224790060904, 0.7382661660668085),
 (0.7516224790060904, 0.7382661660668085)]
```

In [59]:
```
# Best alpha value as 1e-4 for lasso
```

In [ ]:
```

```

ElasticNet

In [60]:
```
from sklearn.linear_model import ElasticNet, ElasticNetCV

alpha = np.arange(1,10,1) * (10**-5)
l1_ratio = np.arange(1,10,1) * (10**-4)

elastic_net_cv_model = ElasticNetCV(alphas = alpha, l1_ratio = l1_ratio, cv = 10, random_state = 33)
elastic_net_cv_model.fit(X_train,y_train)

print("Training Score: ", adj_r(elastic_net_cv_model.score(X_train,y_train), X_train, y_train))
print("Testing Score: ", adj_r(elastic_net_cv_model.score(X_test,y_test), X_test, y_test))
print("Alpha: ", elastic_net_cv_model.alpha_)
print("LT Ratio: ", elastic_net_cv_model.l1_ratio_)
```

```
Training Score:  0.8183177290081365
Testing Score:  0.8071595953252747
Alpha:  9e-05
LT Ratio:  0.0009000000000000001
```

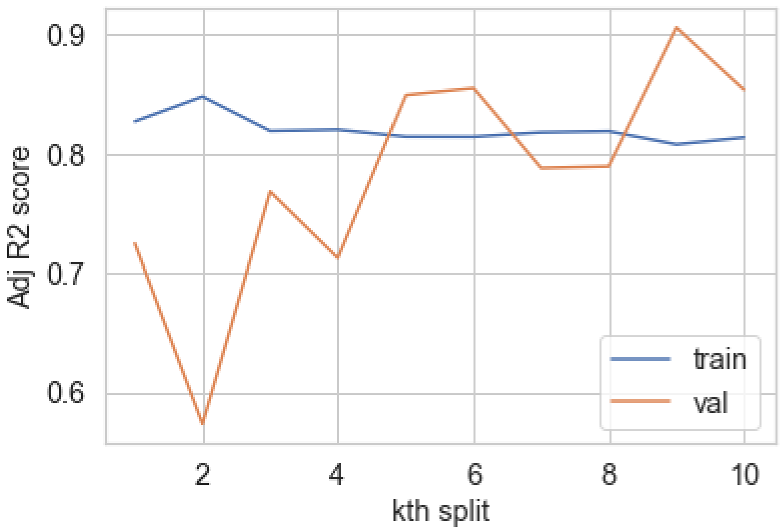
In [ ]:
```

```

K-Fold Cross Validation

In [61]:
```
# Performing k-fold cross validation
from sklearn.model_selection import KFold
```

In [62]:
```python
kf = KFold(n_splits=10)
train_scores = []
val_scores = []

for train_index, val_index in kf.split(X): #iterating through the K-folds

    X_train, X_val = X.iloc[train_index], X.iloc[val_index]
    y_train, y_val = y.iloc[train_index], y.iloc[val_index]

    polyreg_scaled = make_pipeline(LinearRegression())
    polyreg_scaled.fit(X_train, y_train) #training model

    train_score = adj_r(polyreg_scaled.score(X_train, y_train), X_train, y_train)
    val_score= adj_r(polyreg_scaled.score(X_val, y_val), X_val, y_val)

    train_scores.append(train_score)
    val_scores.append(val_score)

print(f"Training Score using KFold cross validation for k = 10 is {np.mean(train_scores).round(2)}")
print(f"Validation Score using KFold cross validation for k = 10 is {np.mean(val_scores).round(2)}")

plt.figure()
plt.plot(list(range(1, 11)), train_scores, label="train")
plt.plot(list(range(1, 11)), val_scores, label="val")
plt.legend(loc='lower right')
plt.xlabel("kth split")
plt.ylabel("Adj R2 score")
plt.show()
```

```
Training Score using KFold cross validation for k = 10 is 0.82
Validation Score using KFold cross validation for k = 10 is 0.78
```

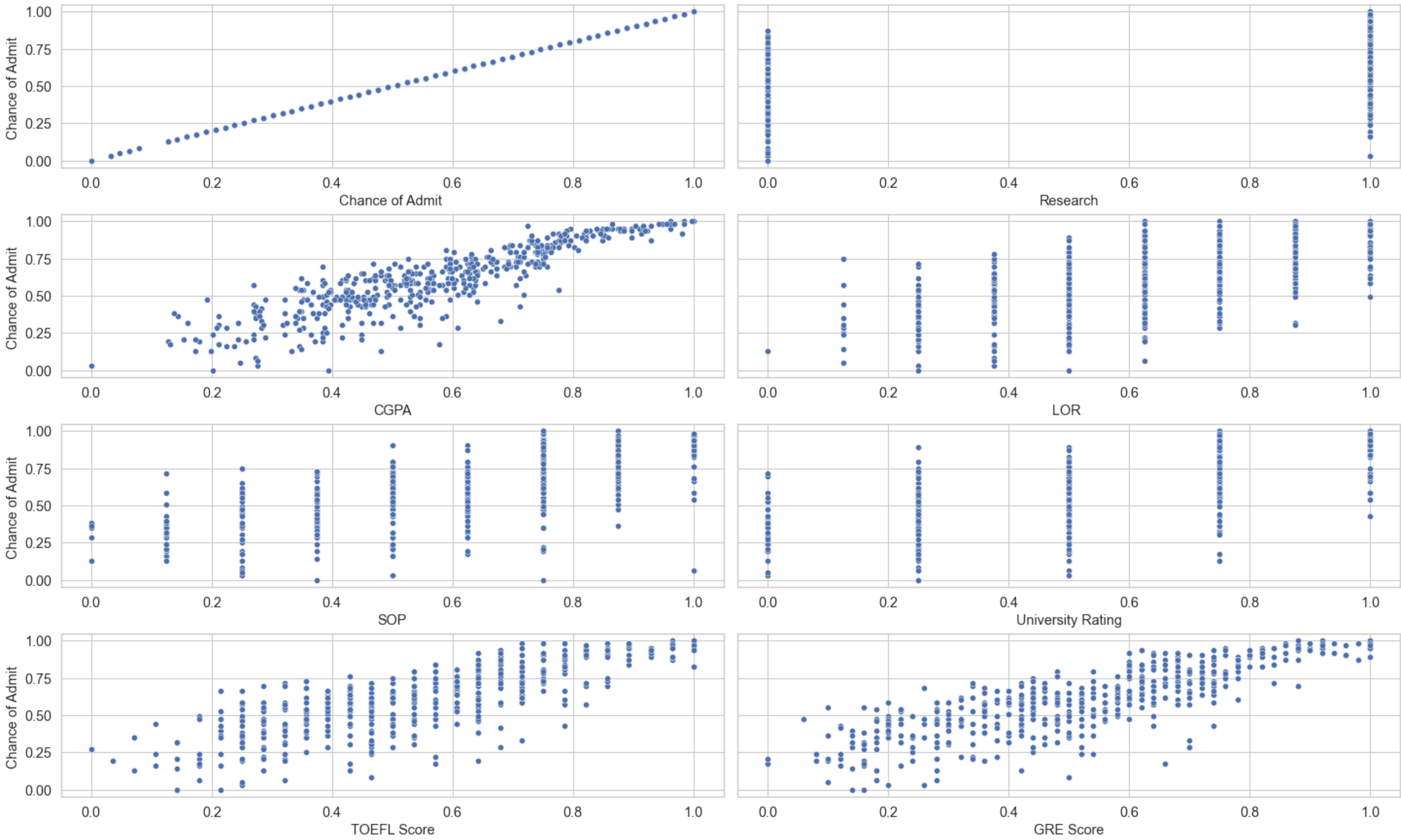


In [ ]:

Testing the assumptions of the linear regression model

Assumption of Linearity

In [63]:
```
fig, ax = plt.subplots(nrows=4, ncols=2, figsize=(20, 12), constrained_layout=True, sharey = True)

cols = df.columns.to_list()
for i in range(len(ax)):
    for j in range(len(ax[i])):
        column = cols.pop()
        sns.scatterplot(data = df, x = column, y = 'Chance of Admit ', ax=ax[i][j])
```



In [64]:
```
# Linearity of variables refers to the assumption that there is a linear relationship
# between the independent variables and the dependent variable in a regression model.
# It means that the effect of the independent variables on the dependent variable
# is constant across different levels of the independent variables.
```

In [ ]:

```

```

Non multi-collinear features

In [65]:

```
# VIF
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [66]:

```
vif = pd.DataFrame()
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X_train.values, i) for i in range(X_train.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[66]:

	Features	VIF
5	CGPA	41.97
1	TOEFL Score	29.84
0	GRE Score	29.37
3	SOP	19.13
4	LOR	16.05
2	University Rating	11.20
6	Research	3.36

In [67]:

```
# 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 sc
# TOEFL score and CGPA.
```

In [ ]:

```

```

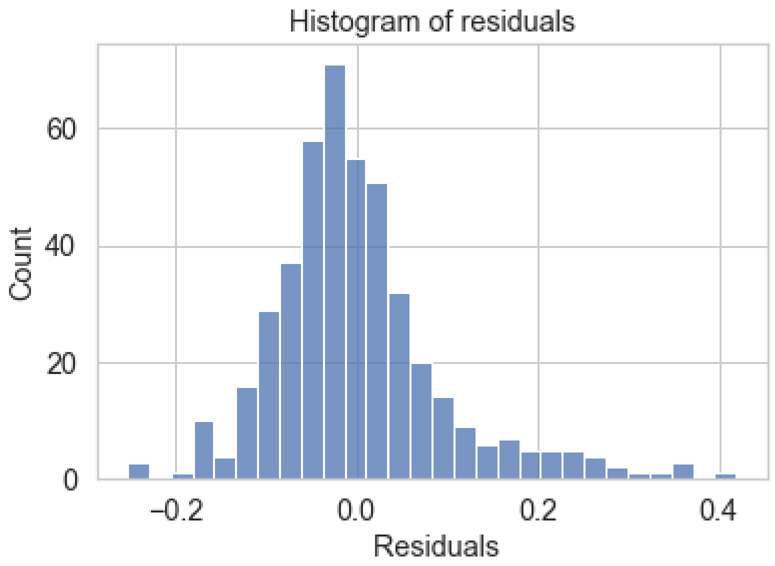
Errors are normally distributed

In [68]:
```
X_sm = sm.add_constant(X_train)
sm_model = sm.OLS(y_train, X_sm).fit()
```

In [69]:
```
Y_hat = sm_model.predict(X_sm)
errors = Y_hat - y_train
```

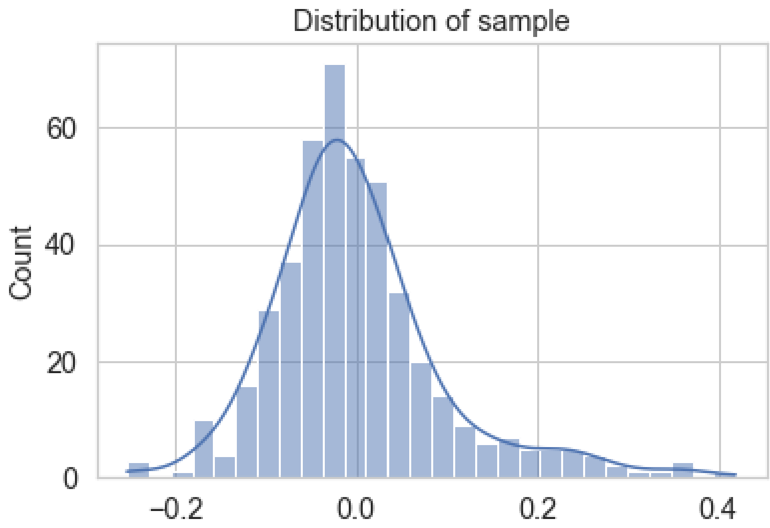
In [70]:
```
sns.histplot(errors)
plt.xlabel(" Residuals")
plt.title("Histogram of residuals")
```

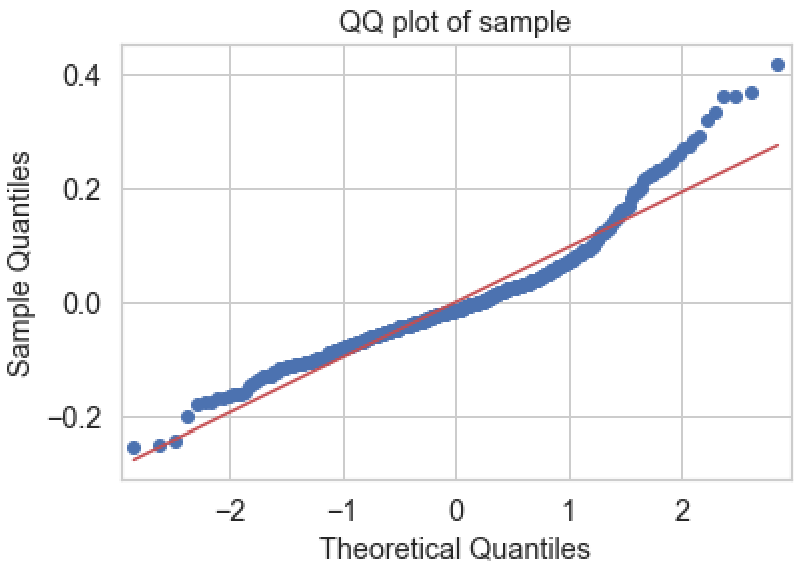
Out[70]:
```
Text(0.5, 1.0, 'Histogram of residuals')
```



In [71]:
```python
# Goodness of fit plots - qq plots, visual check for normality
from statsmodels.graphics.gofplots import qqplot

# Test for Gaussian (Statistical Test for Normality)
from scipy.stats import shapiro

# Setting significance value as 0.05 for experiment
ALPHA = 0.05

def test_normality(sample):

    # Visual Analysis
    sns.histplot(sample, kde = True)
    plt.title('Distribution of sample')

    # QQplot
    qqplot(sample, line = 's')
    plt.title('QQ plot of sample')
    plt.show()

    # Shapiro-Wilk test
    print("\nPerforming Shapiro-Wilk test for normality:-\n")
    print("Null Hypothesis: Given sample is normally distributed")
    print("Alternate Hypothesis: Given sample is not normally distributed")
    print(f'ALPHA (Significance value): {ALPHA}\n')

    statistic, p_value = shapiro(sample)

    print(f'Statistic: {statistic}, p-value: {p_value}')
    if p_value < ALPHA:
        print("Reject null hypothesis, given sample is not normal distributed.")
    else:
        print("Fail to reject null hypothesis, given sample is normal distributed.")
```

In [72]:
```
test_normality(errors)
```





```
Performing Shapiro-Wilk test for normality:-

Null Hypothesis: Given sample is normally distributed
Alternate Hypothesis: Given sample is not normally distributed
ALPHA (Significance value): 0.05

Statistic: 0.9276833534240723, p-value: 6.376348152823189e-14
Reject null hypothesis, given sample is not normal distributed.
```

In [ ]:

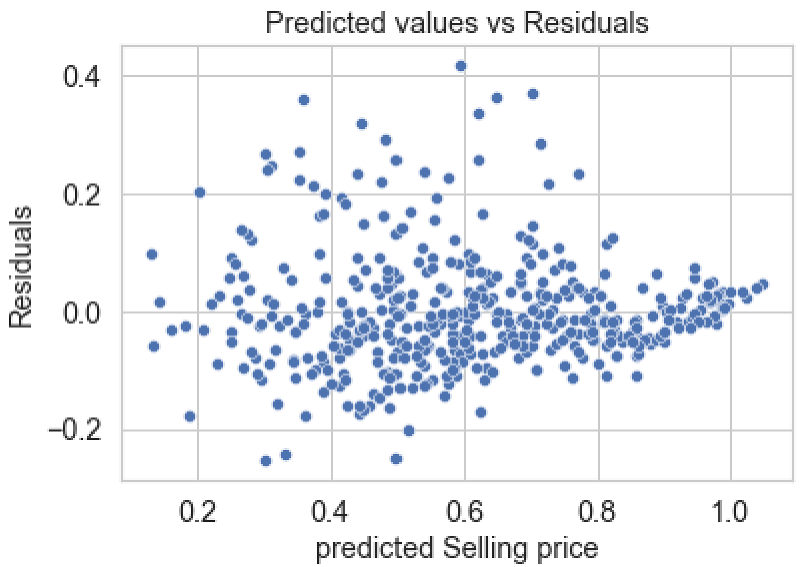
Heteroskedasticity should not exist

In [73]:

```
sns.scatterplot(x=Y_hat,y=errors)
plt.xlabel("predicted Selling price")
plt.ylabel("Residuals")
plt.title("Predicted values vs Residuals")
```

Out[73]:

```
Text(0.5, 1.0, 'Predicted values vs Residuals')
```



In [74]:
```
# 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_sm)
lzip(name, test)
```

Out[74]:
```
[('F statistic', 0.4403978262218863), ('p-value', 0.9999999986748125)]
```

In [75]:
```
# From the goldfeld-quandt test:

# F Statistic comes out to be 0.44 => Implying minimal difference in variance between groups
# p-value of 0.999 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.
```

In [ ]:

Mean of Residuals

In [76]:
```
residuals = y_test.values - model.predict(X_test)
residuals.reshape((-1,))
print('Mean of Residuals: ', residuals.mean())
```

```
Mean of Residuals:  -0.009058079982908366
```

In [77]:
```
# Since the mean of residuals is very close to 0, we can say that the model is unbiased
```

In [ ]:

Identify best model and performance evaluation

In [78]:

```
# Best testing adj R score in Lasso
# Best degree 1 for lasso regression
# Best alpha value as 1e-4 for lasso
```

In [79]:

```
final_model = make_pipeline(PolynomialFeatures(1), Lasso(alpha=1e-4))

final_model.fit(X_train, y_train)

y_pred_train = final_model.predict(X_train)
y_pred_test = final_model.predict(X_test)
```

In [80]:
```python
from sklearn.metrics import mean_absolute_error as mae, mean_squared_error as mse

r2_train = final_model.score(X_train, y_train)
r2_test = final_model.score(X_test, y_test)

print('\n','-'*30,'Adj R2','-'*30,sep = '')
print("Training adj R2 Score for Model:",adj_r(r2_train,X_train,y_train))
print("Testing adj R2 Score for Model:",adj_r(r2_test,X_test,y_test))

print('\n','-'*30,'MAE','-'*30,sep = '')
print("Training MAE Score for Model:",mae(y_pred_train,y_train))
print("Testing MAE Score for Model:",mae(y_pred_test,y_test))

print('\n','-'*30,'MSE','-'*30,sep = '')
print("Training MSE Score for Model:",mse(y_pred_train,y_train))
print("Testing MSE Score for Model:",mse(y_pred_test,y_test))

print('\n','-'*30,'RMSE','-'*30,sep = '')
print("Training MSE Score for Model:",np.sqrt(mse(y_pred_train,y_train)) )
print("Testing MSE Score for Model:",np.sqrt(mse(y_pred_test,y_test)))
```

```
------------------------------Adj R2------------------------------
Training adj R2 Score for Model: 0.8135512650540911
Testing adj R2 Score for Model: 0.813223280800496

------------------------------MAE------------------------------
Training MAE Score for Model: 0.06934097114541843
Testing MAE Score for Model: 0.06371909060364878

------------------------------MSE------------------------------
Training MSE Score for Model: 0.009361401990856l6
Testing MSE Score for Model: 0.008444988067891545

------------------------------RMSE------------------------------
Training MSE Score for Model: 0.0967543383567691
Testing MSE Score for Model: 0.09189661619391405
```

In [ ]:

Actionable Insights & Recommendations

Final Model:

- The Lasso Regression Model, with polynomial degree as 3 and alpha value of 1e-4, emerges as the best fit. Choosing simplicity over complexity aligns with Occam's Razor principle. Thus we can use simple linear regression as well at it gave a score of 0.8072 whereas aforementioned Lasso gave 0.8170.

Feature Importance:

- CGPA stands out as the most crucial feature upon coefficient comparison.

Additional Data Sources for Model Improvements:

- Incorporating a larger dataset could enhance model effectiveness by capturing a more diverse range of patterns and relationships.
- Consider integrating insights from candidates' professional, extracurricular experiences, certifications, and the reputation of their undergraduate institutions.

Model Implementation in the Real World:

- Implement the model in the admissions process to automate initial screening, improve efficiency, and reduce manual workload.
- Utilize the model as a decision support tool in admission committee meetings to provide insights into each candidate's predicted success.

Potential Business Benefits:

- Institutions can allocate resources more effectively by focusing on candidates with higher predicted success.
- Enhance decision-making by providing a data-driven approach, reducing biases, and gaining a competitive edge.
- Automation can reduce costs associated with manual application reviews and decision-making.

Insights:

- The target variable distribution (chances of admit) is left-skewed.
- Exam scores (CGPA/GRE/TOEFL) show strong positive correlations with chance of admit, along with categorical variables like university ranking, research, SOP, and LOR quality.
- CGPA is the most significant predictor variable, while SOP/University Rating are the least significant.

- Linear and Ridge Regression models capture up to 82% of the variance in the target variable, facing challenges due to multicollinearity.

Recommendations:

- Consider adding more independent features beyond highly correlated exam scores, such as work experience, internships, mock interview performance, extracurricular activities, or diversity variables.

Recommendations for Jamboree Education:

- Encourage students to prioritize improving GRE, TOEFL scores, and LOR quality.
- Promote the significance of research experience in enhancing admission chances.

Significance of Predictions:

- CGPA's emphasis highlights the importance of academic excellence.
- The model's consideration of holistic evaluation factors like SOP_LOR and Research Experience aids strategic planning for prospective applicants.
- Admissions committees benefit from insights into the relative importance of different factors when evaluating applicants.