Business Case

Jamboree Education - Linear Regression

Suman Debnath

Introduction

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.

Business Problem

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.

Dataset

Dataset link: delhivery_data.csv

Column Profiling:

- 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)

Summary

- The first column was observed to be a unique row identifier and was dropped as it wasn't necessary for model building.

- University Rating, SOP, LOR strength, and research appear to be discrete random variables but also qualify as ordinal numeric data.
- All the other features are numeric, ordinal, and continuous in nature.
- The data contains no null values.
- No significant outliers were detected in the data.
- The Chance of Admission (target variable) and GRE score (an independent feature) show near-normal distributions.
- Independent Variables (Input Data): GRE Score, TOEFL Score, University Rating, SOP, LOR, CGPA, Research
- Target/Dependent Variable: Chance of Admit (the value we aim to predict)
- The correlation heatmap reveals that the GRE score, TOEFL score, and CGPA have a very strong correlation with the Chance of Admission.
- University rating, SOP, LOR, and Research are somewhat less correlated with the Chance of Admission compared to other features.
- The "Chance of Admit" lies within the range of 0 to 1, which is optimal, indicating no outliers or misleading data in the column.
- The GRE score typically ranges from 290 to 340.
- TOEFL scores are generally between 92 to 120.
- University ratings, SOP, and LOR values are spread across a range of 1 to 5.
- The CGPA range is from 6.8 to 9.92.
- Boxplots depicting the distribution of the Chance of Admission (probability of securing admission) based on GRE scores suggest that a higher GRE score increases the likelihood of obtaining admission.
- Similarly, students with higher TOEFL scores are more likely to gain admission.
- From the count plots, it's evident that a stronger Statement of Purpose (SOP) correlates positively with the Chance of Admission.
- Similar patterns can be observed with Letter of Recommendation strength and University rating—both positively influence the Chance of Admission.
- Notably, students involved in research have a higher probability of admission, though some outliers exist within this category.

Recommendation

- Educational institutions can assist students in not only enhancing their CGPA but also in crafting compelling LORs and SOPs, thereby improving their chances of admission to top-tier universities.
- It's essential to hold seminars to raise awareness regarding the significance of maintaining a high CGPA and the value of research capabilities, both of which can greatly improve admission prospects.
- Recognizing that students cannot retroactively change their attributes, it's pivotal to target them early—perhaps at the undergraduate level—with awareness and marketing campaigns. This not only boosts the institution's reputation but also prepares students for their future endeavors.

- Introducing a dashboard for students on the institution's website can foster healthy competition and provide them with a clear progress report, aiding their self-assessment.
- Incorporating additional features like study hours, lecture attendance, assignment completion rates, and mock test scores could provide a more comprehensive performance overview, enabling students to self-reflect and improve continuously.

Detailed Analysis

Importing all the `libs`

In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib import figure
import warnings
import statsmodels.api as sm
from scipy.stats import norm
from scipy.stats import t
import plotly.express as px

import scipy.stats as stats

warnings.filterwarnings('ignore')
%matplotlib inline
```

Loading the `data`

In [2]:

```
data_set = 'https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/
# data_set = 'jamboree_admission.csv'
df = pd.read_csv(data_set)
```

Exploratory Data Exploration (EDA)

In [3]:

```
df.shape
```

Out[3]:

```
(500, 9)
```

In [4]:

```
df.head()
```

Out[4]:

	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

In [5]:

```
df.dtypes
```

Out[5]:

```
Serial No.             int64
GRE Score              int64
TOEFL Score            int64
University Rating      int64
SOP                  float64
LOR                  float64
CGPA                 float64
Research               int64
Chance of Admit      float64
dtype: object
```

In [6]:

```
df.columns
```

Out[6]:

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

Check for `null` values

In [7]:

```
np.any(df.isna())
```

Out[7]:

```
False
```

In [8]:

```
df.isna().sum()
```

Out[8]:

```
Serial No.           0
GRE Score            0
TOEFL Score          0
University Rating    0
SOP                  0
LOR                  0
CGPA                 0
Research             0
Chance of Admit      0
dtype: int64
```

In [9]:

```
df['Research'].value_counts()
```

Out[9]:

```
1    280
0    220
Name: Research, dtype: int64
```

In [10]:

```
cols_value_count = {col: len(df[col].value_counts()) for col in df.columns}
```

In [11]:
```
cols_value_count = dict(sorted(cols_value_count.items(), key=lambda item: item
cols_value_count
```

Out[11]:
```
{'Research': 2,
 'University Rating': 5,
 'SOP': 9,
 'LOR ': 9,
 'TOEFL Score': 29,
 'GRE Score': 49,
 'Chance of Admit ': 61,
 'CGPA': 184,
 'Serial No.': 500}
```

Observation

- Features: There are no missing values, and all features are numeric.
- The `Research` field is binary, while all others represent continuous real numbers.
- `University Rating`, `SOP`, `LOR`, and `Research` appear to be categorical variables, given their limited unique values.
- While most features are numeric and ordinal (with `University Rating`, `SOP`, `LOR`, and `Research` being discrete), the others are continuous.
- Additionally, `SOP`, `University Rating`, `LOR`, and `Research` can be viewed as numeric ordinal data.

Analyzing Linear Relationships Between Dataset Features

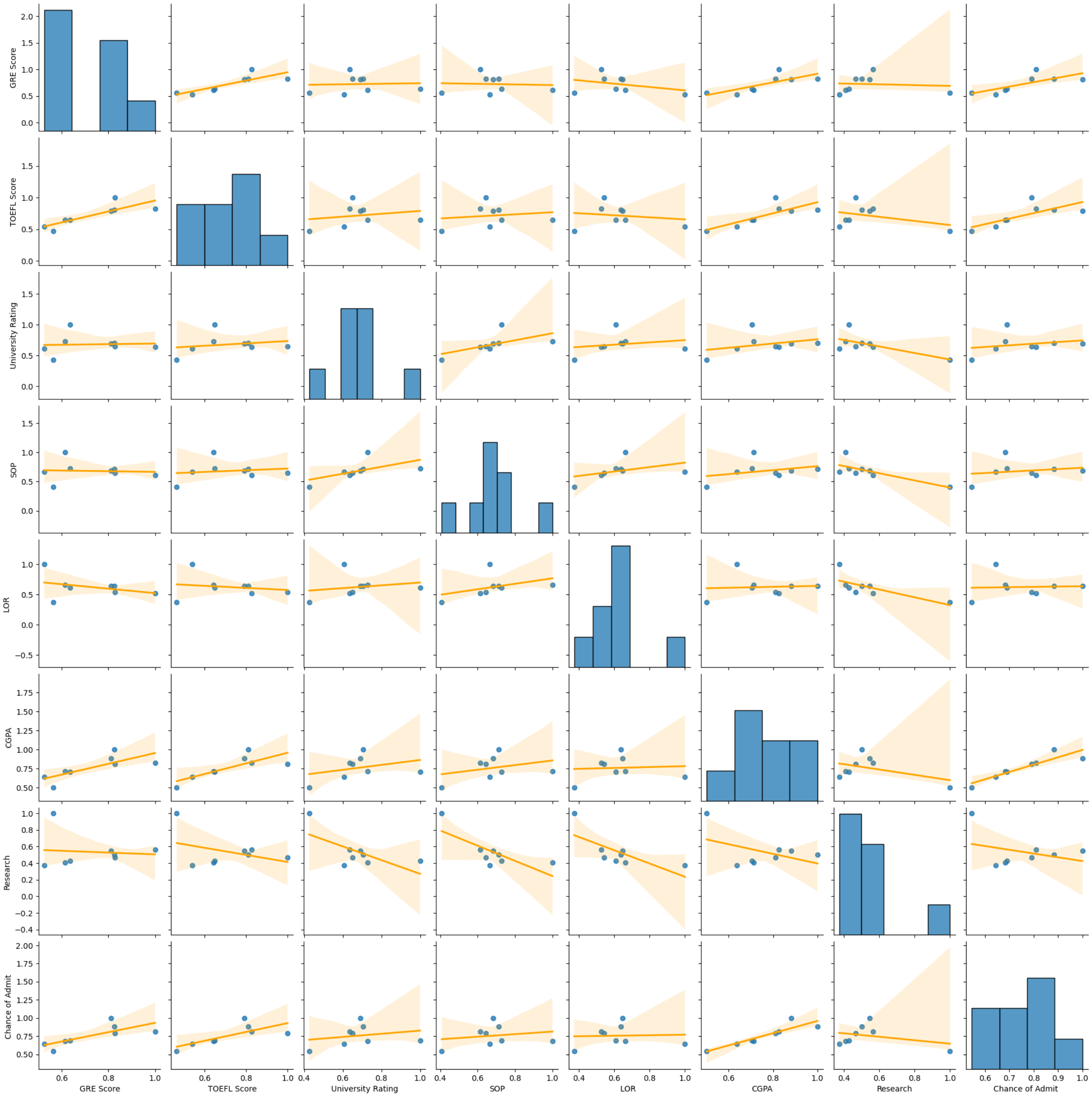
In [12]:
```
# Dorping Serial No. as its not useful for this problem and it might bluf the
df.drop(["Serial No."],axis=1,inplace=True)
```

In [13]:
```
df.sample(5)
```

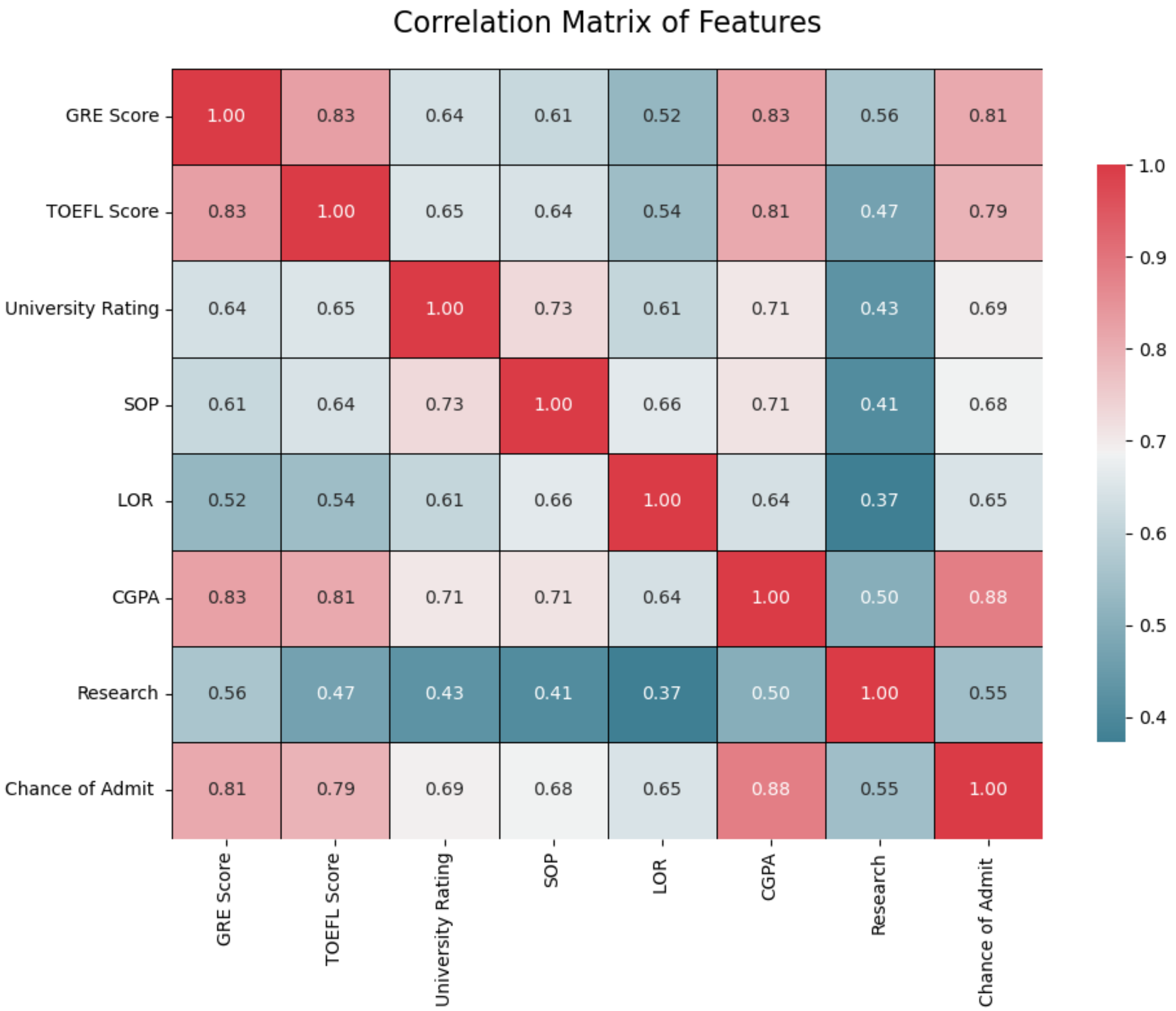
Out[13]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
171	334	117	5	4.0	4.5	9.07	1	0.89
263	324	111	3	2.5	1.5	8.79	1	0.70
476	304	104	3	2.5	2.0	8.12	0	0.62
470	320	110	5	4.0	4.0	9.27	1	0.87
431	320	112	2	3.5	3.5	8.78	1	0.73

In [14]:
```
sns.pairplot(df.corr(), kind='reg', plot_kws={'line_kws':{'color':'orange'}});
```



In [15]:
```
plt.figure(figsize=(10,8))

# Using a cooler colormap for a more striking visual.
cmap = sns.diverging_palette(220, 10, as_cmap=True)

# Drawing the heatmap.
sns.heatmap(df.corr(), annot=True, cmap=cmap, fmt=".2f", linewidths=.5, lineco

# Adding a title to the heatmap.
plt.title("Correlation Matrix of Features", size=16, pad=20)

# Display the heatmap.
plt.tight_layout()
plt.show()
```



	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
GRE Score	1.00	0.83	0.64	0.61	0.52	0.83	0.56	0.81
TOEFL Score	0.83	1.00	0.65	0.64	0.54	0.81	0.47	0.79
University Rating	0.64	0.65	1.00	0.73	0.61	0.71	0.43	0.69
SOP	0.61	0.64	0.73	1.00	0.66	0.71	0.41	0.68
LOR	0.52	0.54	0.61	0.66	1.00	0.64	0.37	0.65
CGPA	0.83	0.81	0.71	0.71	0.64	1.00	0.50	0.88
Research	0.56	0.47	0.43	0.41	0.37	0.50	1.00	0.55
Chance of Admit	0.81	0.79	0.69	0.68	0.65	0.88	0.55	1.00

Observation

- **Independent Variables (Input Data):** GRE Score, TOEFL Score, University Rating, SOP, LOR, CGPA, Research
- **Target/Dependent Variable:** Chance of Admit (the value we aim to predict)
- We can observe that GRE Score, TOEFL Score, and CGPA have a very high correlation with the Chance of Admission.
- University Rating, SOP, LOR, and Research are comparatively less correlated than the other features.

In [16]:
```
data = df.copy()
```

In [17]:
```
# changing / removing space between column names.
df.columns  = ['GRE_Score', 'TOEFL_Score', 'University_Rating', 'SOP', 'LOR',
        'Research', 'Chance_of_Admit']
```

In [18]:
```
df.sample(5)
```

Out[18]:

	GRE_Score	TOEFL_Score	University_Rating	SOP	LOR	CGPA	Research	Chance_of_Adm
183	314	110	3	4.0	4.0	8.80	0	0.7
315	308	104	2	2.5	3.0	8.07	0	0.6
100	322	107	3	3.5	3.5	8.46	1	0.7
63	315	107	2	4.0	3.0	8.50	1	0.5
241	317	103	2	2.5	2.0	8.15	0	0.6

Check for outliers

In [19]:

```python
def detect_outliers_percentage(data):
    """Detect the percentage of outliers in the given data using IQR method."""

    # Calculate Q1, Q3 and IQR
    Q1 = np.percentile(data, 25)
    Q3 = np.percentile(data, 75)
    IQR = Q3 - Q1

    # Determine the upper and lower bounds to detect outliers
    upper_bound = Q3 + 1.5 * IQR
    lower_bound = Q1 - 1.5 * IQR

    # Ensure that the lower bound does not go below zero, if needed
    lower_bound = max(0, lower_bound)

    # Calculate the number of outliers
    outliers = data[~((data >= lower_bound) & (data <= upper_bound))]

    # Calculate the percentage of outliers in the data
    outlier_percentage = len(outliers) / len(data) * 100

    return f"{outlier_percentage:.2f}% Outliers"

# Detect and display outliers for each column in the dataframe
for col in df.columns:
    print(f"{col} : {detect_outliers_percentage(df[col])}")
```

```
GRE_Score : 0.00% Outliers
TOEFL_Score : 0.00% Outliers
University_Rating : 0.00% Outliers
SOP : 0.00% Outliers
LOR : 0.20% Outliers
CGPA : 0.00% Outliers
Research : 0.00% Outliers
Chance_of_Admit : 0.40% Outliers
```

In [20]:

```python
df.describe()
```

Out[20]:

	GRE_Score	TOEFL_Score	University_Rating	SOP	LOR	CGPA	Res
count	500.000000	500.000000	500.000000	500.000000	500.00000	500.000000	500.0
mean	316.472000	107.192000	3.114000	3.374000	3.48400	8.576440	0.5
std	11.295148	6.081868	1.143512	0.991004	0.92545	0.604813	0.4
min	290.000000	92.000000	1.000000	1.000000	1.00000	6.800000	0.0
25%	308.000000	103.000000	2.000000	2.500000	3.00000	8.127500	0.0
50%	317.000000	107.000000	3.000000	3.500000	3.50000	8.560000	1.0
75%	325.000000	112.000000	4.000000	4.000000	4.00000	9.040000	1.0
max	340.000000	120.000000	5.000000	5.000000	5.00000	9.920000	1.0

Observation

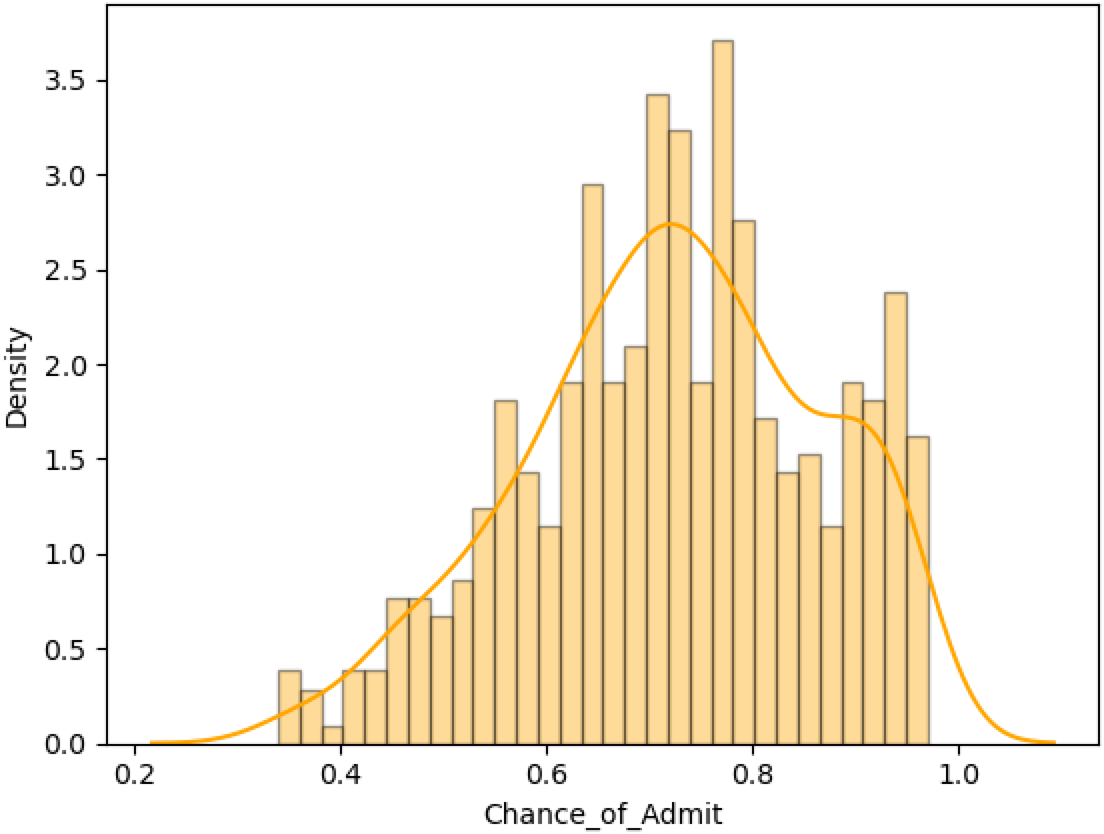
- The "chances of admit" is a probability measure that falls within the range of 0 to 1, which indicates no outliers or misleading data in the column.
- The GRE score typically ranges from 290 to 340.
- The TOEFL score ranges between 92 and 120.
- University rating, SOP, and LOR are all distributed within a range of 1 to 5.
- The CGPA spans a range from 6.8 to 9.92.

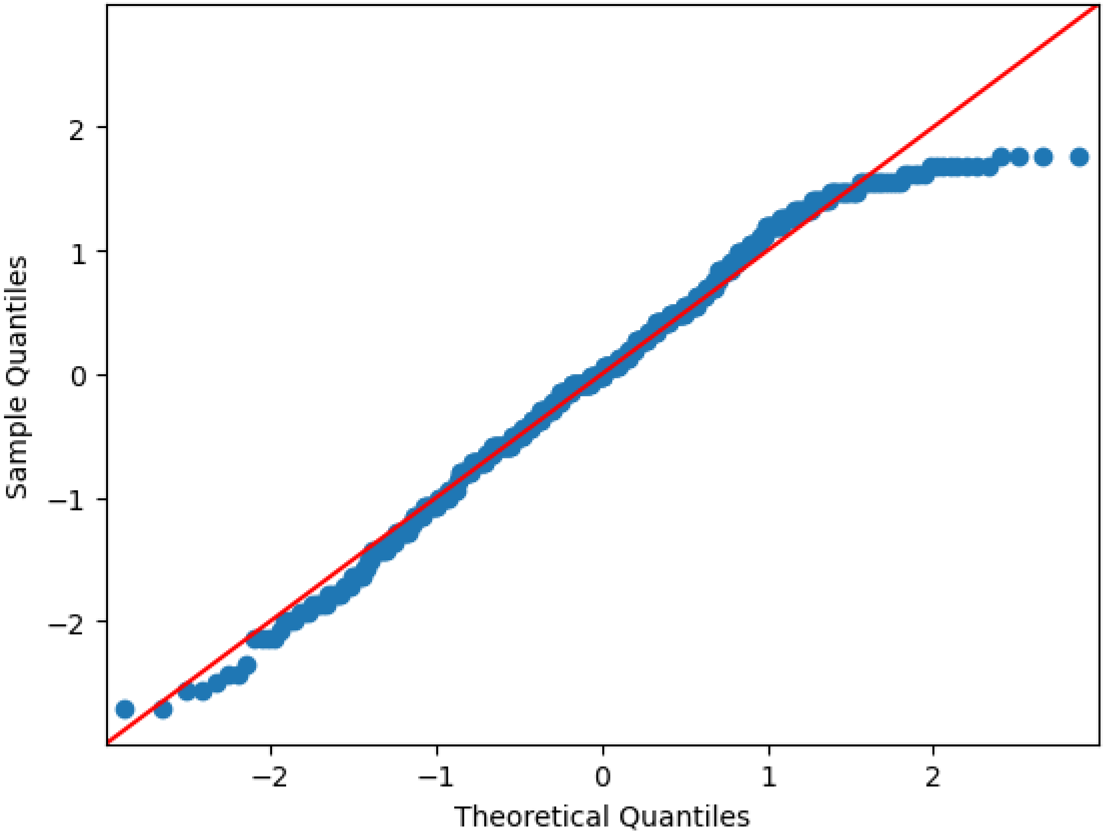
Distribution

`Chance_of_Admit`

In [21]:

```
sns.distplot(df["Chance_of_Admit"], bins=30, color='orange', hist_kws={'edgeco
sm.qqplot(df["Chance_of_Admit"],fit=True, line="45")
plt.show()
```



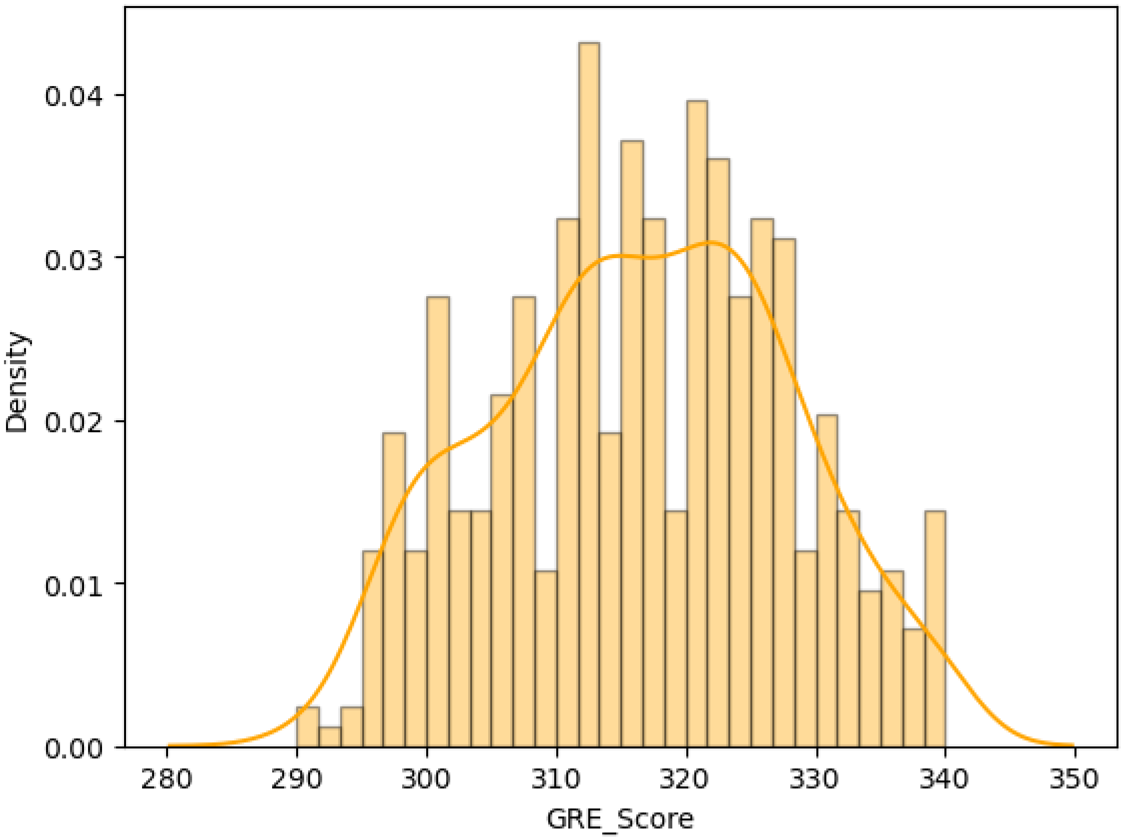


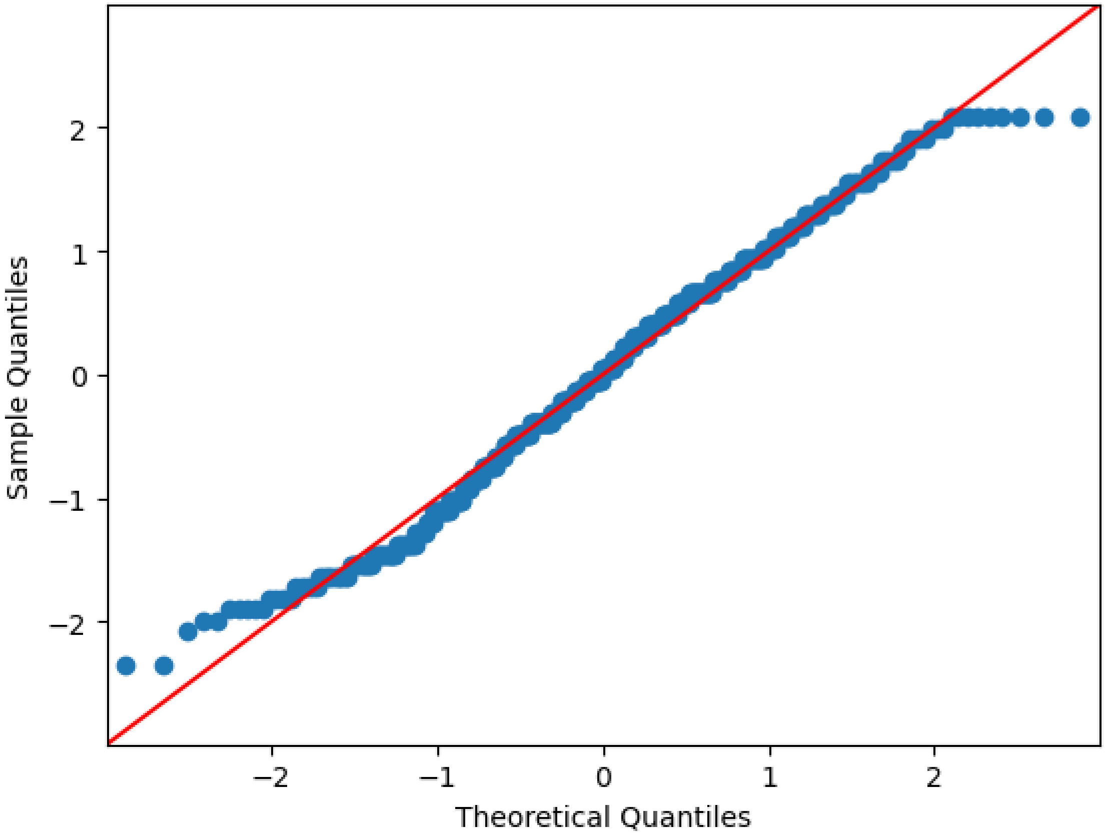
`GRE_Score`

In [22]:

```
sns.distplot(df["GRE_Score"], bins=30, color='orange', hist_kws={'edgecolor':'
sm.qqplot(df["GRE_Score"],fit=True, line="45")
```

```
plt.show()
```





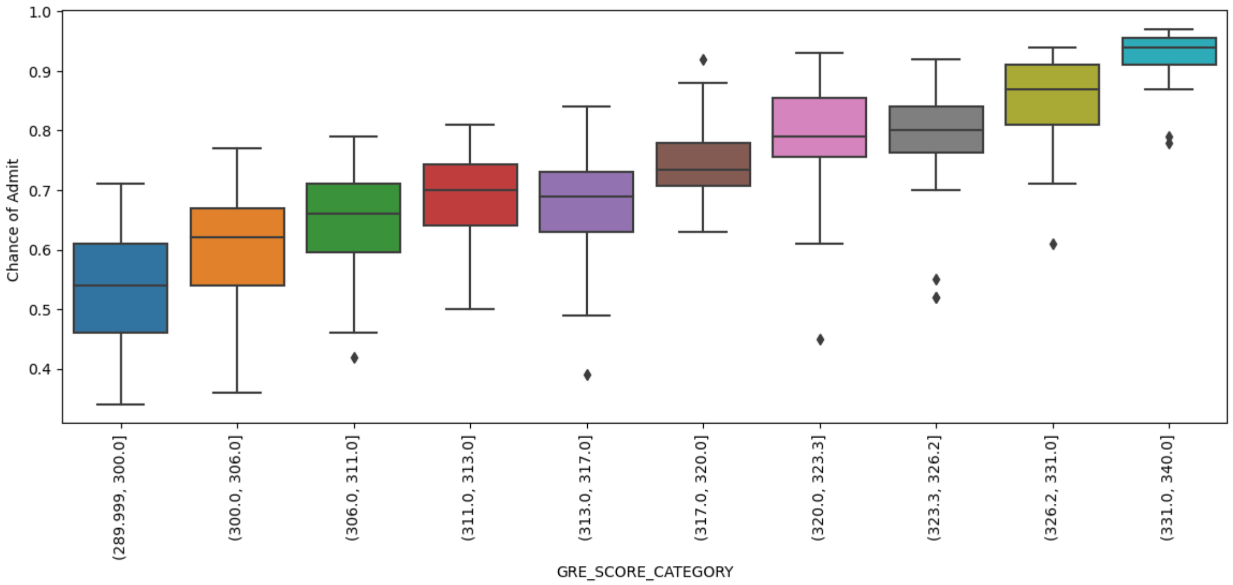
In [23]:
```
data.head()
```

Out[23]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	337	118	4	4.5	4.5	9.65	1	0.92
1	324	107	4	4.0	4.5	8.87	1	0.76
2	316	104	3	3.0	3.5	8.00	1	0.72
3	322	110	3	3.5	2.5	8.67	1	0.80
4	314	103	2	2.0	3.0	8.21	0	0.65

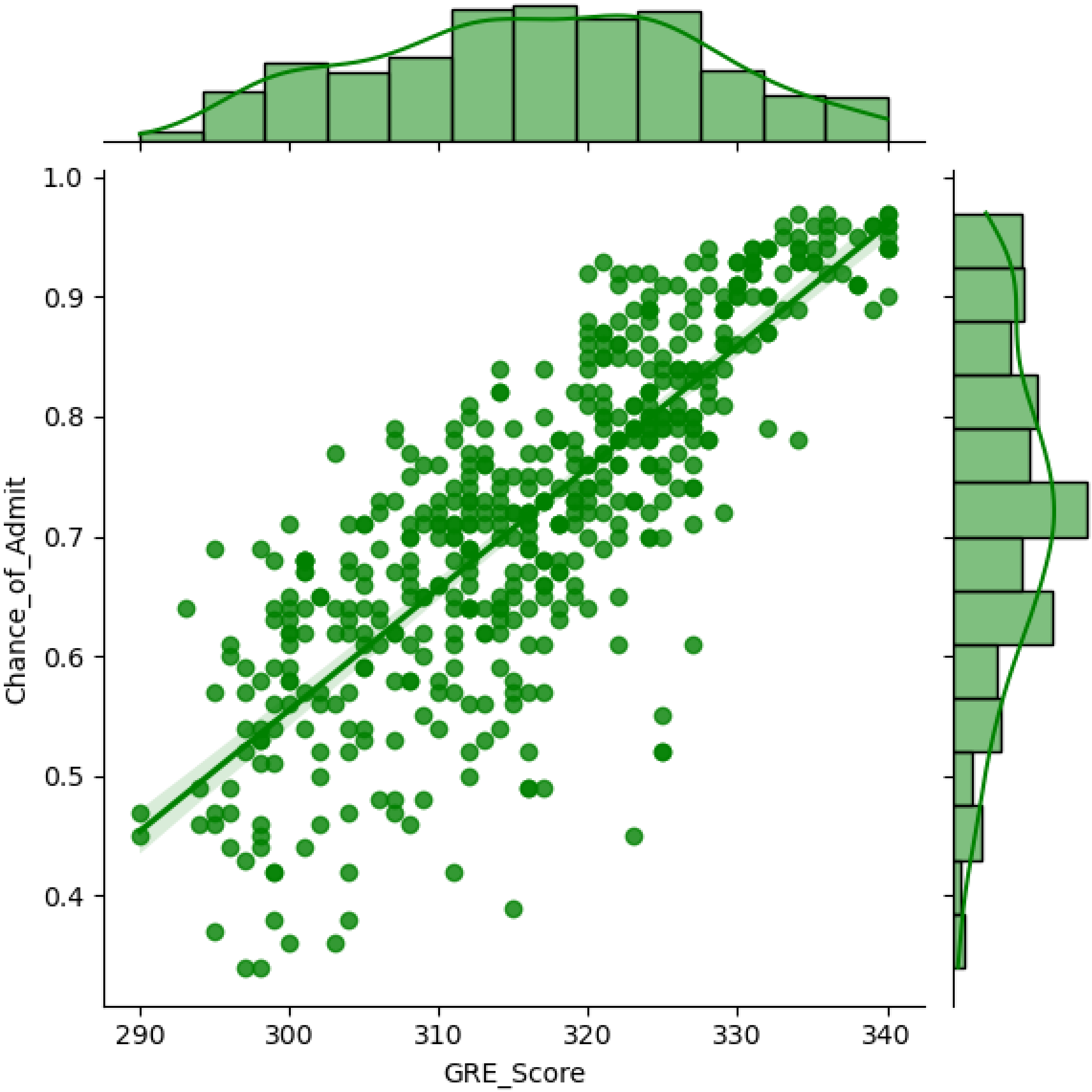
In [24]:

```
data["GRE_SCORE_CATEGORY"]=pd.qcut(data["GRE Score"],10)
plt.figure(figsize=(14,5))
sns.boxplot(y = data["Chance of Admit "], x = data["GRE_SCORE_CATEGORY"])
plt.xticks(rotation = 90)
plt.show()
```



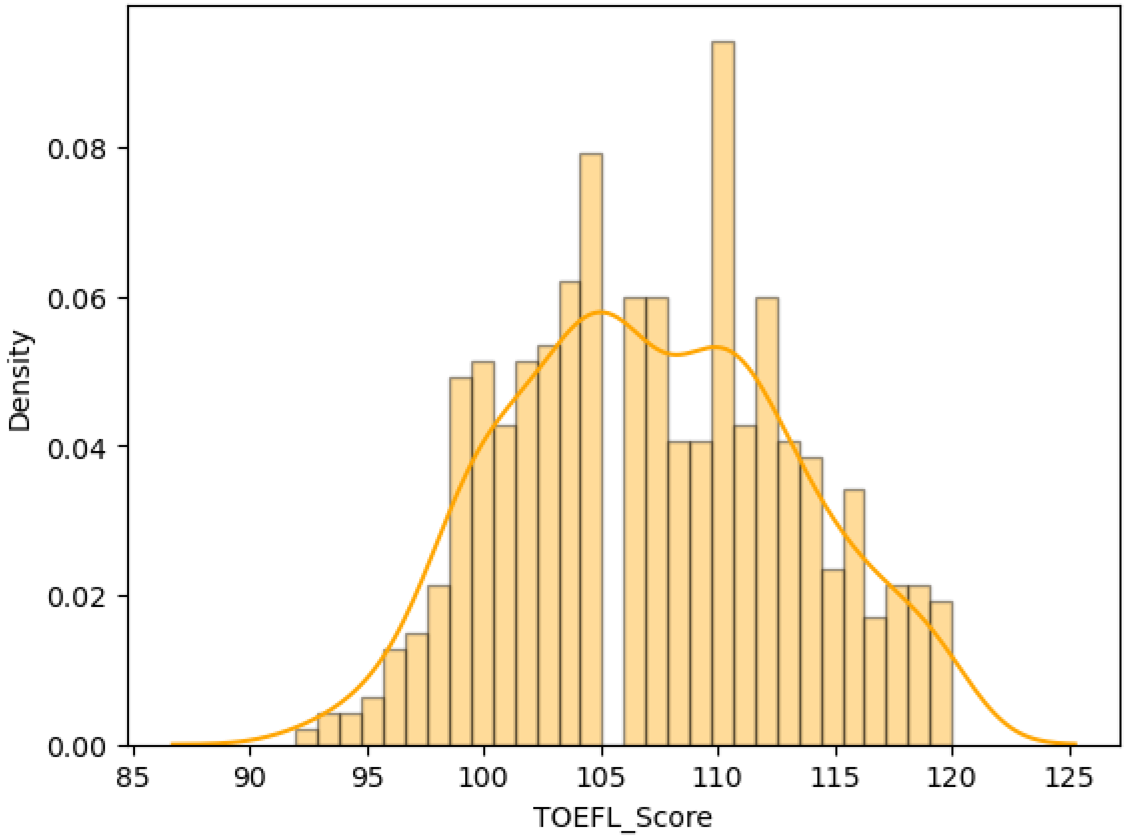
In [25]:

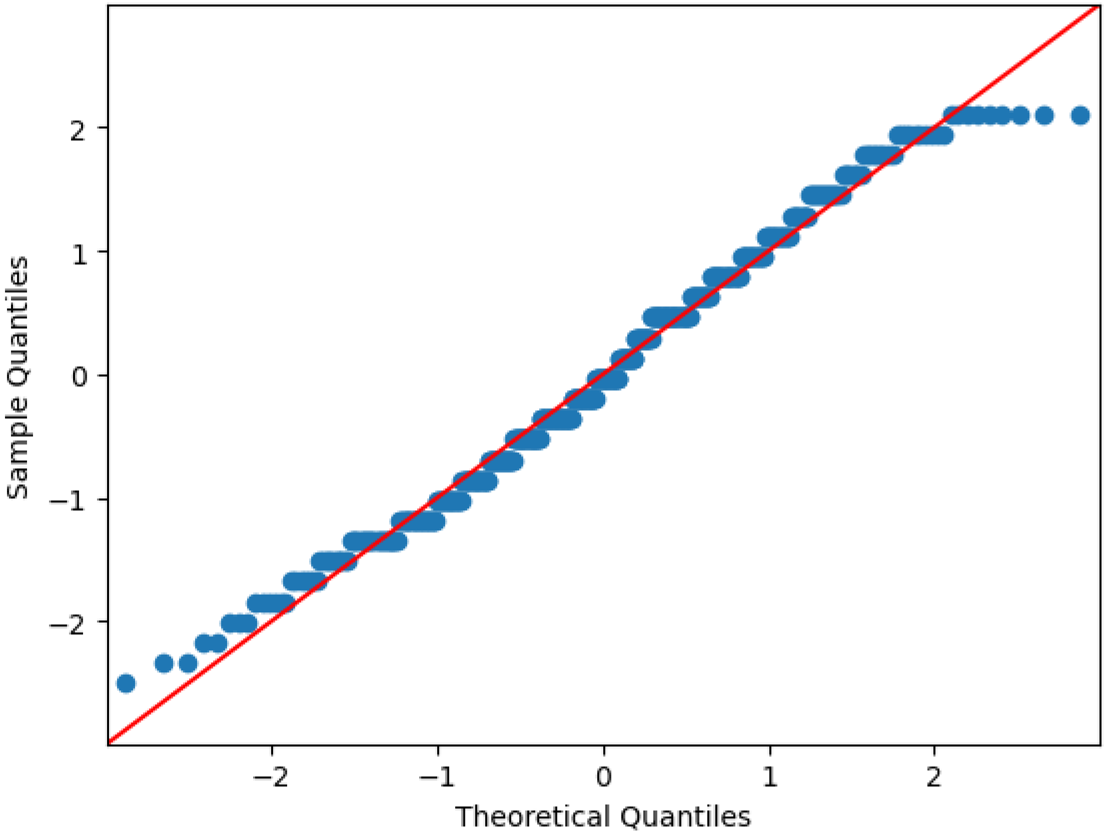
```
sns.jointplot(x=df["GRE_Score"], y=df["Chance_of_Admit"], kind="reg", color="g
plt.show()
```



TOEFL_Score

In [26]:
```
sns.distplot(df["TOEFL_Score"], bins=30, color='orange', hist_kws={'edgecolor'
sm.qqplot(df["TOEFL_Score"],fit=True, line="45")
plt.show()
```



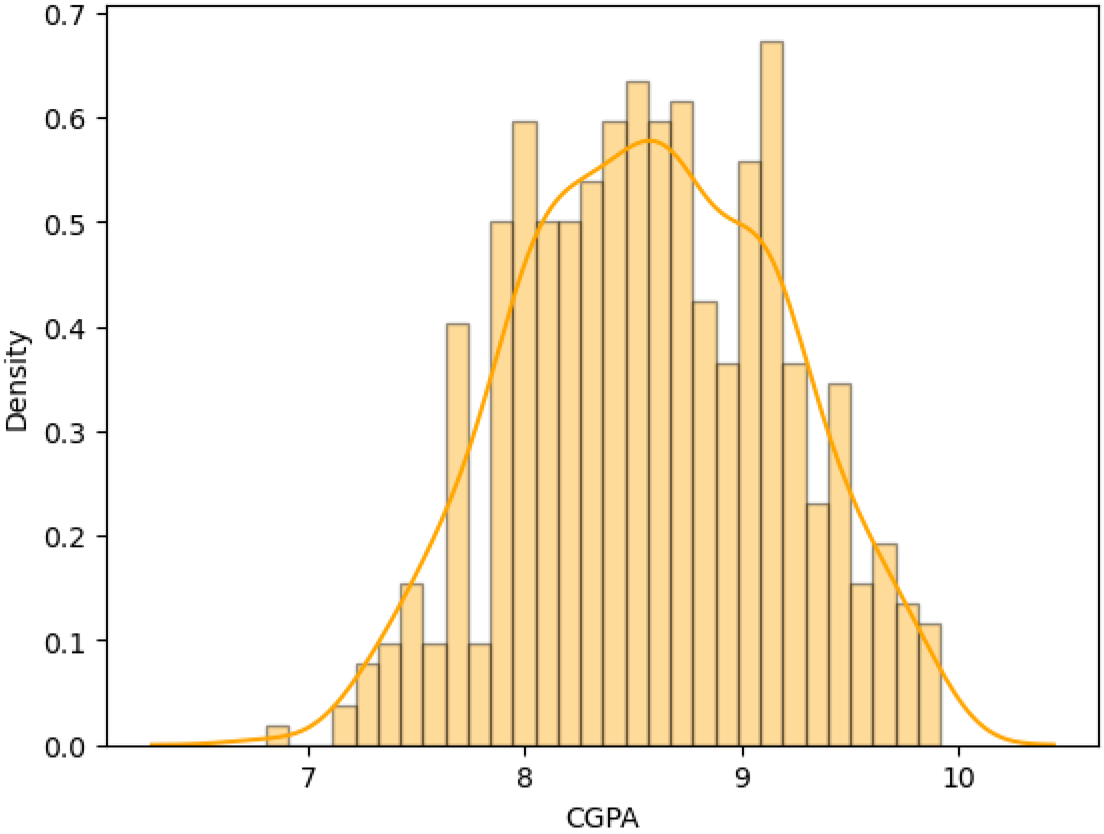


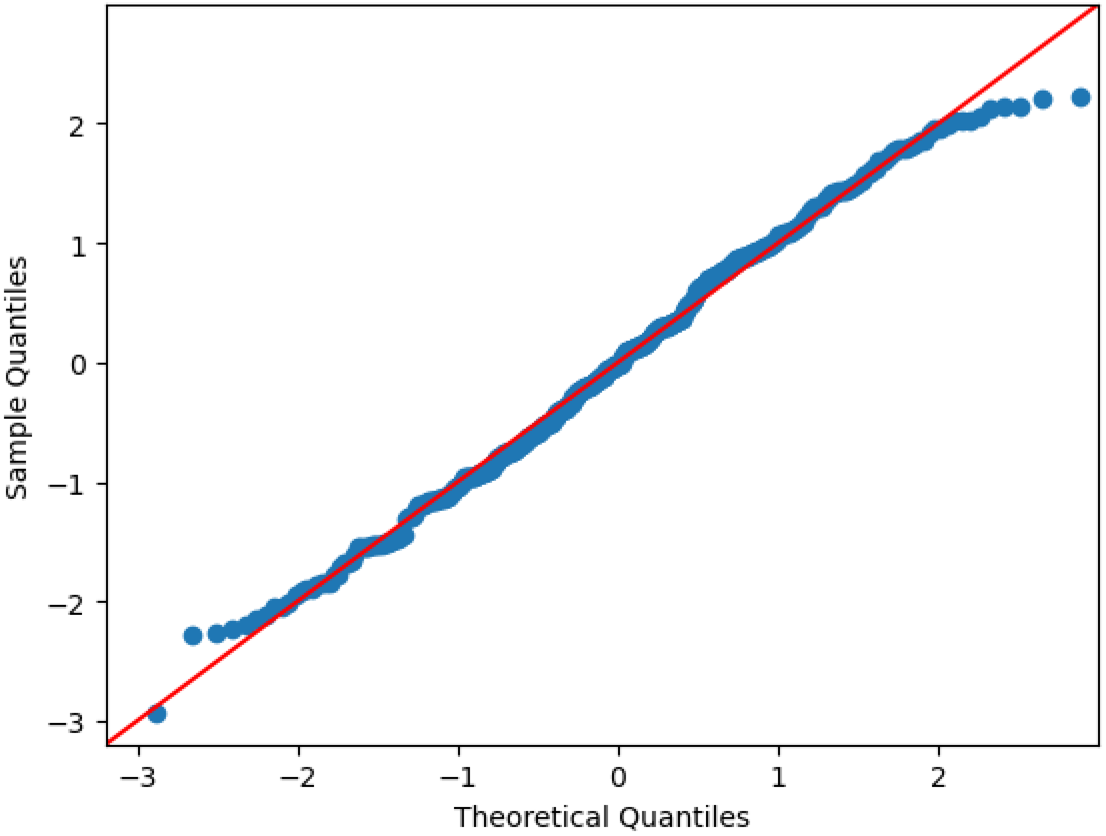
CGPA

In [27]:

```
sns.distplot(df["CGPA"], bins=30, color='orange', hist_kws={'edgecolor':'black
sm.qqplot(df["CGPA"],fit=True, line="45")
```

```
plt.show()
```





In [28]:
```
df
```

Out[28]:

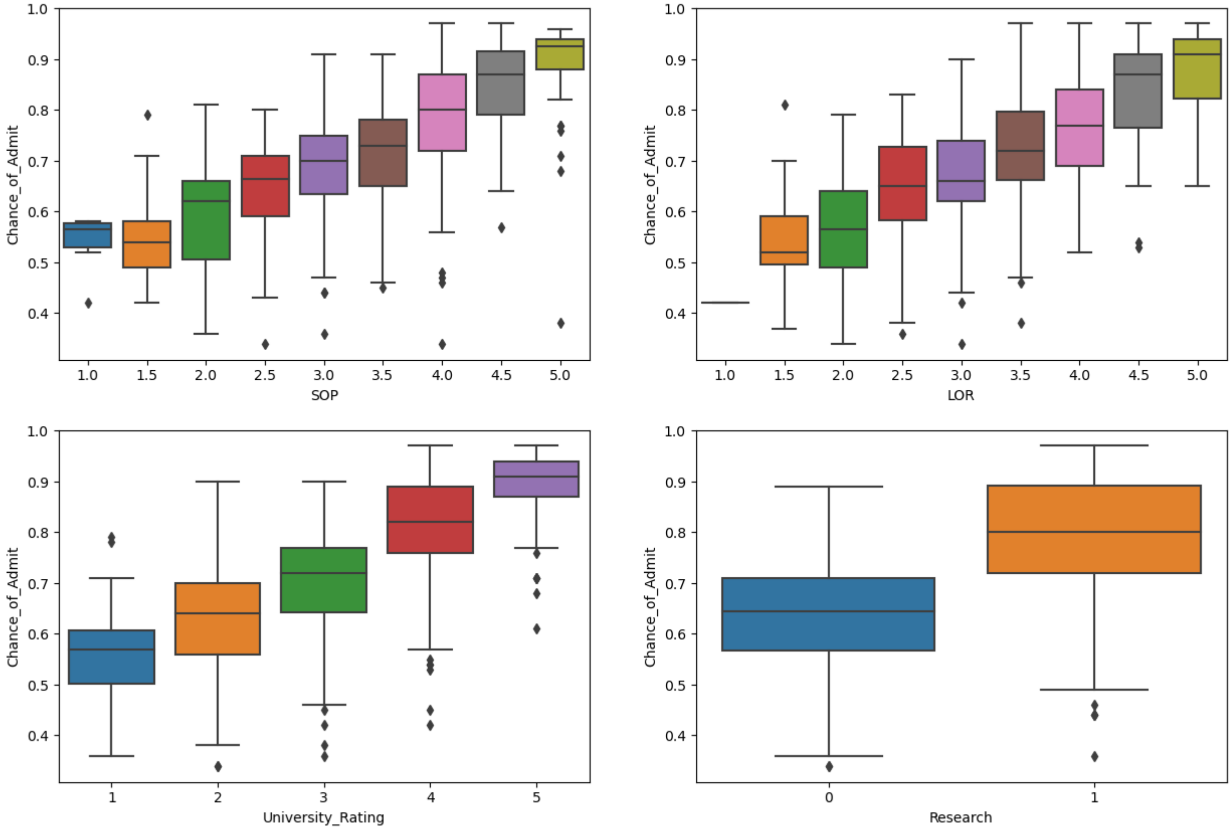
	GRE_Score	TOEFL_Score	University_Rating	SOP	LOR	CGPA	Research	Chance_of_Adm
0	337	118	4	4.5	4.5	9.65	1	0.9
1	324	107	4	4.0	4.5	8.87	1	0.7
2	316	104	3	3.0	3.5	8.00	1	0.7
3	322	110	3	3.5	2.5	8.67	1	0.8
4	314	103	2	2.0	3.0	8.21	0	0.6
...	...	...	...	...	...	...	...	
495	332	108	5	4.5	4.0	9.02	1	0.8
496	337	117	5	5.0	5.0	9.87	1	0.9
497	330	120	5	4.5	5.0	9.56	1	0.9
498	312	103	4	4.0	5.0	8.43	0	0.7
499	327	113	4	4.5	4.5	9.04	0	0.8

500 rows × 8 columns

Categorical features v/s chances of admission

In [29]:

```
plt.figure(figsize=(15,10))
plt.subplot(2,2,1)
sns.boxplot(y = df["Chance_of_Admit"], x = df["SOP"])
plt.subplot(2,2,2)
sns.boxplot(y = df["Chance_of_Admit"], x = df["LOR"])
plt.subplot(2,2,3)
sns.boxplot(y = df["Chance_of_Admit"], x = df["University_Rating"])
plt.subplot(2,2,4)
sns.boxplot(y = df["Chance_of_Admit"], x = df["Research"])
plt.show()
```



Observation

- From the plots above, we can observe that the strength of the Statement of Purpose (SOP) is positively correlated with the Chance of Admission.
- A similar pattern can be seen with the Letter of Recommendation Strength and University Rating, both showing a positive correlation with the Chance of Admission.
- Students with research experience have higher chances of admission, although there are some outliers within that category.

Building a Linear Regression Model

1. Implementing using `numpy`

In [30]:
```
class LinearRegression():
    def __init__(self, learning_rate=0.01, epochs=50):
        self.learning_rate = learning_rate
        self.epochs = epochs

    def predict(self, X):
        return np.dot(X, self.W) + self.w0

    def update_weights(self):
        Y_pred = self.predict(self.X)

        # Calculate the gradients
        # for w1, w2, .... wd
        dW = -2 * np.dot(self.X.T, (self.Y - Y_pred))/self.X.shape[0]

        # for w0
        dw0 = -2 * np.sum(self.Y - Y_pred)/self.X.shape[0]

        # Update the weights
        self.W = self.W - self.learning_rate * dW
        self.w0 = self.w0 - self.learning_rate * dw0

        return self.W, self.w0

    def fit(self, X, Y):

        self.X = X
        self.Y = Y
        self.error_list = []

        # no_of traning_examples, no_of_features
        self.m, self.d = self.X.shape

        # weight initialization
        self.W = np.zeros(self.d) * 0.01
        self.w0 = 0

        # Gradient Decent Learning
        for i in range(self.epochs):
            self.update_weights()
            Y_pred = self.predict(self.X)

            error = np.square(Y - Y_pred).mean()
            self.error_list.append(error)

        return self
```

In [31]:
```
df.head()
```

Out[31]:

	GRE_Score	TOEFL_Score	University_Rating	SOP	LOR	CGPA	Research	Chance_of_Admit
0	337	118	4	4.5	4.5	9.65	1	0.92
1	324	107	4	4.0	4.5	8.87	1	0.76
2	316	104	3	3.0	3.5	8.00	1	0.72
3	322	110	3	3.5	2.5	8.67	1	0.80
4	314	103	2	2.0	3.0	8.21	0	0.65

In [32]:
```
# define X and y
X = df.drop('Chance_of_Admit', axis=1)
y = df["Chance_of_Admit"]
```

In [33]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X = scaler.fit_transform(X)
```

In [34]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando
```

In [35]:
```
lr = LinearRegression(learning_rate=0.01, epochs=2000)
```

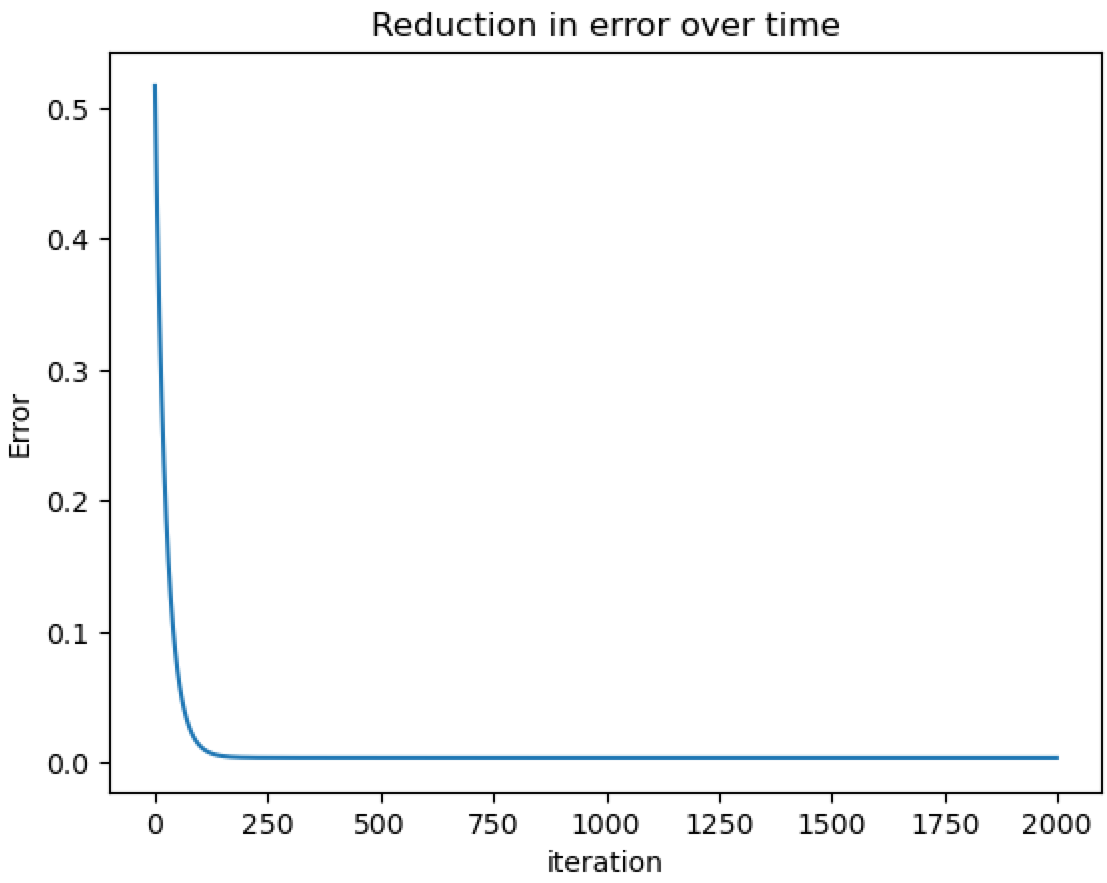
In [36]:
```
lr.fit(X_train, y_train)
```

Out[36]:
```
<__main__.LinearRegression at 0x14b1dbfa0>
```

In [37]:
```
lr.predict(X_test)
```

Out[37]:
```
array([0.65363193, 0.70043048, 0.94400606, 0.73135168, 0.81719492,
       0.66512168, 0.74544354, 0.71401327, 0.78873315, 0.65659002,
       0.66884678, 0.56213707, 0.78220221, 0.79559866, 0.77166447,
       0.85777859, 0.62918086, 0.7619383 , 0.89747136, 0.67179447,
       0.62843001, 0.7938741 , 0.8420017 , 0.59216461, 0.78876654,
       0.56886065, 0.95157775, 0.64437149, 0.86025851, 0.71083796,
       0.63332995, 0.81500395, 0.59772989, 0.91045698, 0.50794677,
       0.81813632, 0.68717741, 0.63320355, 0.65956449, 0.91168276,
       0.56570431, 0.66080637, 0.77232555, 0.97094476, 0.77182889,
       0.52232782, 0.66695742, 0.63032126, 0.65365658, 0.66063868,
       0.83353815, 0.9185174 , 0.87826396, 0.61930393, 0.76828126,
       0.64295331, 0.74767045, 0.60336178, 0.65945121, 0.69648931,
       0.43763098, 0.72186144, 0.75306944, 0.84913026, 0.9801405 ,
       0.61055524, 0.73188306, 0.7739596 , 0.9414187 , 0.70256316,
       0.60280788, 0.65413478, 0.82415015, 0.49108857, 0.92579076,
       0.5973991 , 0.83680529, 0.94072902, 0.71118345, 0.76867536,
       0.83475554, 0.50993976, 0.91584225, 0.78937354, 0.79910127,
       0.68669724, 0.87777091, 0.88687074, 0.56515801, 0.60074878,
       0.62923345, 0.78113665, 0.57127085, 0.70739607, 0.80038413,
       0.83441485, 0.82819747, 0.57301002, 0.72500885, 0.68547453])
```

In [38]:
```
%matplotlib inline
fig = plt.figure()
plt.plot(lr.error_list)
plt.title("Reduction in error over time")
plt.xlabel("iteration")
plt.ylabel("Error")
plt.show()
```



In [39]:
```
from sklearn.metrics import r2_score,mean_squared_error,mean_absolute_error, ac
```

R2 Score on train data

In [40]:
```
r2_score(y_train,lr.predict(X_train))
```

Out[40]:
```
0.8215098977894868
```

R2 Score on test data

In [41]:
```
r2_score(y_test,lr.predict(X_test))
```

Out[41]:
```
0.8208665894121212
```

All the feature's coefficients and Intercept

In [42]:
```
pd.DataFrame(lr.W.reshape(1,-1),columns=df.columns[:-1])
```

Out[42]:

	GRE_Score	TOEFL_Score	University_Rating	SOP	LOR	CGPA	Research
0	0.020692	0.019301	0.007002	0.002982	0.013343	0.07047	0.009872

In [43]:
```
lr.w0
```

Out[43]:
```
0.7228811807361624
```

In [84]:
```
y_pred = lr.predict(X_test)

print("MSE:",mean_squared_error(y_test,y_pred)) # MSE
print("RMSE:",np.sqrt(mean_squared_error(y_test,y_pred))) #RMSE
print("MAE :",mean_absolute_error(y_test,y_pred) ) # MAE
print("r2_score:",r2_score(y_test,y_pred)) # r2score
print("Adjusted R2 score :", AdjustedR2score(r2_score(y_test,y_pred),len(X),X.s
```

```
MSE: 0.0034592452918625264
RMSE: 0.05881534911791757
MAE : 0.04020210840364795
r2_score: 0.8208665894121212
Adjusted R2 score : 0.8183179433265213
```

2. Implementing using `sklearn`

In [44]:
```
from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

from statsmodels.stats.outliers_influence import variance_inflation_factor

from sklearn.metrics import r2_score,mean_squared_error,mean_absolute_error, ad
from sklearn.feature_selection import f_regression
```

In [45]:
```
X = df.drop(["Chance_of_Admit"],axis = 1)  # independent variables
y = df["Chance_of_Admit"].values.reshape(-1,1) # target / dependent variables
```

In [46]:
```
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

In [47]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, randor
```

In [48]:
```
LinearRegression = LinearRegression()
LinearRegression.fit(X_train,y_train)
```

Out[48]:
```
▾ LinearRegression
LinearRegression()
```

`R2 Score` on train data

In [49]:
```
r2_score(y_train,LinearRegression.predict(X_train))
```

Out[49]:
```
0.8215099192361265
```

`R2 Score` on test data

In [50]:
```
r2_score(y_test,LinearRegression.predict(X_test))
```

Out[50]:
```
0.8208741703103732
```

All the feature's coefficients and Intercept

In [51]:
```
LinearRegression_Model_coefs = pd.DataFrame(LinearRegression.coef_.reshape(1,-
LinearRegression_Model_coefs
```

Out[51]:

	GRE_Score	TOEFL_Score	University_Rating	SOP	LOR	CGPA	Research
0	0.020675	0.019284	0.007001	0.002975	0.013338	0.070514	0.009873

In [52]:
```
LinearRegression.intercept_[0]
```

Out[52]:
```
0.7228813180778462
```

In [53]:
```
def AdjustedR2score(R2,n,d):
    return 1-(((1-R2)*(n-1))/(n-d-1))
```

In [54]:
```
y_pred = LinearRegression.predict(X_test)

print("MSE:",mean_squared_error(y_test,y_pred)) # MSE
print("RMSE:",np.sqrt(mean_squared_error(y_test,y_pred))) #RMSE
print("MAE :",mean_absolute_error(y_test,y_pred) ) # MAE
print("r2_score:",r2_score(y_test,y_pred)) # r2score
print("Adjusted R2 score :", AdjustedR2score(r2_score(y_test,y_pred),len(X),X.
```

```
MSE: 0.003459098897136383
RMSE: 0.05881410457650769
MAE : 0.04020019380415795
r2_score: 0.8208741703103732
Adjusted R2 score : 0.8183256320830818
```

Assumptions of Linear Regression

- No multicollinearity.
- The mean of residuals is close to zero.
- Linearity of variables.
- Homoscedasticity (constant variance of the errors).
- Normality of residuals.

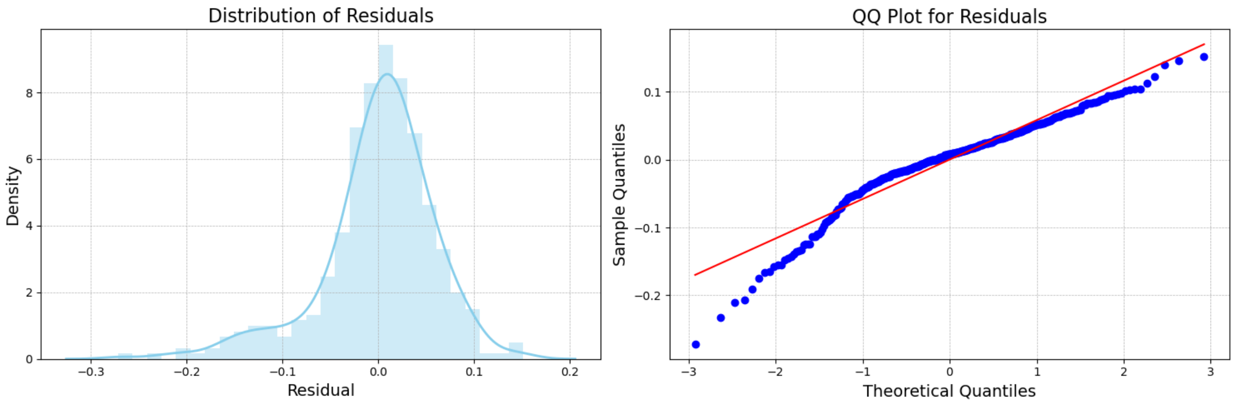
Residual analysis

In [55]:
```
y_predicted = LinearRegression.predict(X_train)
y_predicted.shape
```

Out[55]:
```
(400, 1)
```

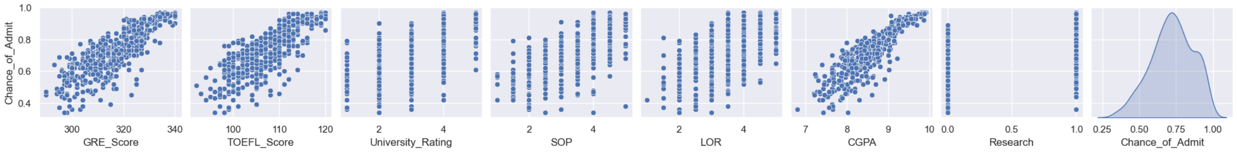
In [56]:
```
residuals = (y_train - y_predicted)

plt.figure(figsize=(15, 5))

# Residual distribution plot
plt.subplot(1, 2, 1)
sns.distplot(residuals, color='skyblue', kde_kws={"lw": 2})
```

```
plt.title('Distribution of Residuals', fontsize=16)
plt.xlabel('Residual', fontsize=14)
plt.ylabel('Density', fontsize=14)
plt.grid(True, which="both", linestyle="--", linewidth=0.5)

# QQ-plot for residuals
plt.subplot(1, 2, 2)
stats.probplot(residuals.reshape(-1,), plot=plt)
plt.title('QQ Plot for Residuals', fontsize=16)
plt.xlabel('Theoretical Quantiles', fontsize=14)
plt.ylabel('Sample Quantiles', fontsize=14)
plt.grid(True, which="both", linestyle="--", linewidth=0.5)

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



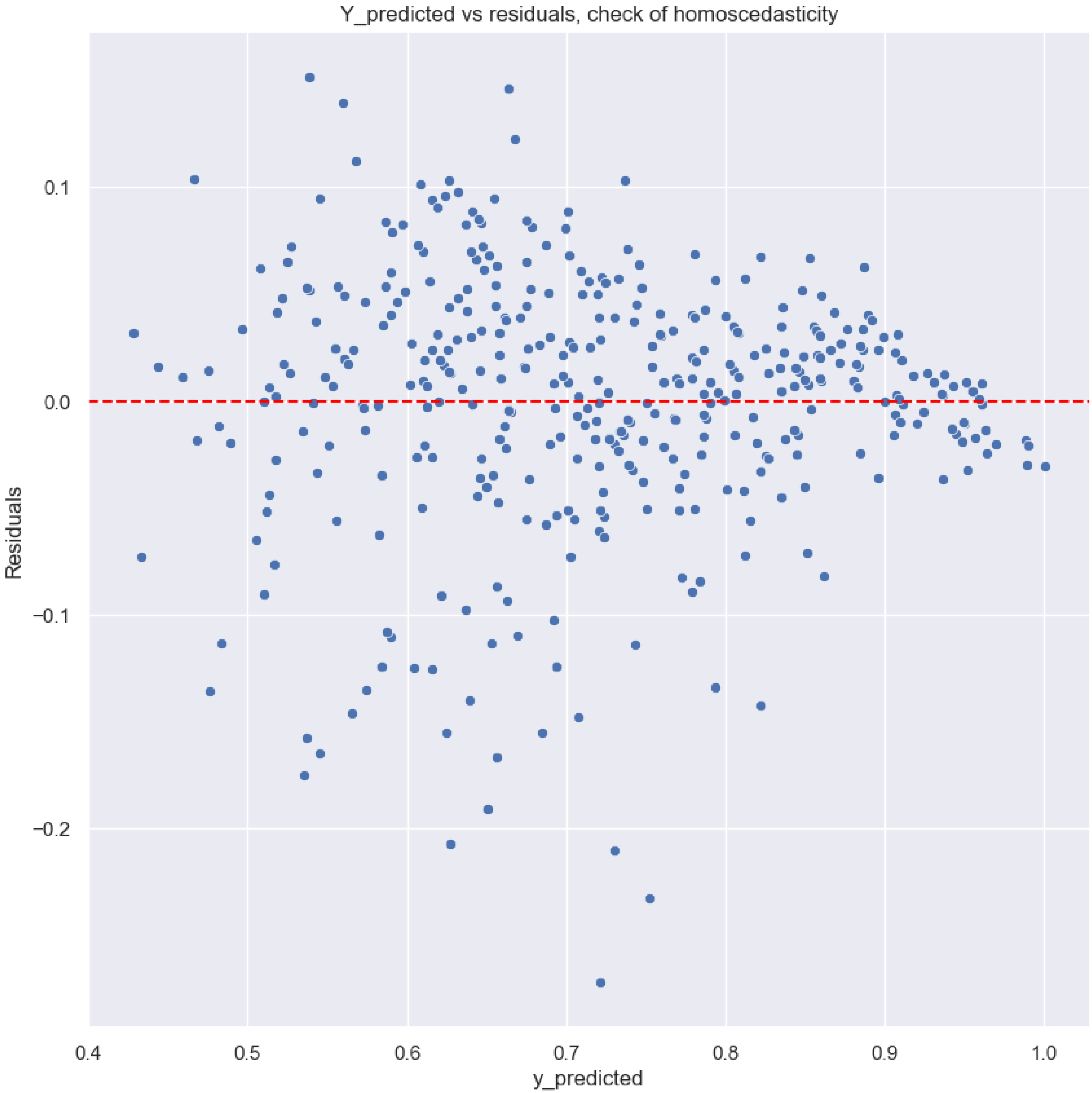
In [57]:

```
palette = sns.color_palette("viridis")

# Define size of the plots
sns.set(rc={'figure.figsize':(10,10)})

# Create pairplots
sns.pairplot(df, y_vars=["Chance_of_Admit"], palette=palette, diag_kind='kde')

# Show the plots
plt.show()
```



In [58]:

```
# Test of homoscedasticity
sns.scatterplot(x=y_predicted.reshape(-1), y=residuals.reshape(-1))
plt.xlabel('y_predicted')
plt.ylabel('Residuals')
plt.axhline(y=0, color='red', linestyle='--')
plt.title("Y_predicted vs residuals, check of homoscedasticity")
plt.show()
```



Model Regularisation

```
In [59]: from sklearn.linear_model import Ridge  # L2 regualrization
         from sklearn.linear_model import Lasso  # L1 regualrization
         from sklearn.linear_model import ElasticNet
```

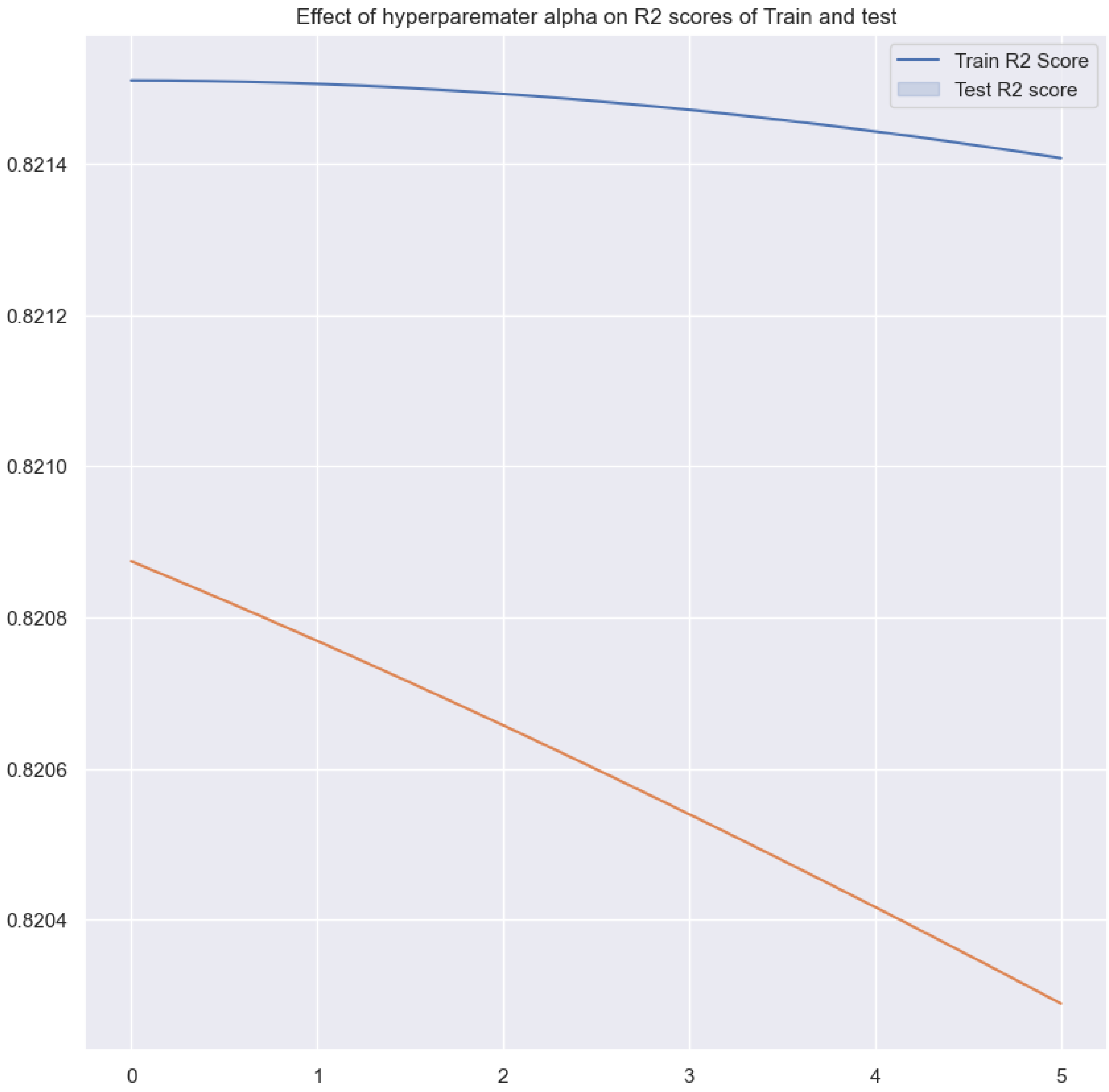
Ridge (`L2 regualrization`)

```
In [60]: ## Hyperparameter Tuning : for appropriate lambda value :

         train_R2_score = []
         test_R2_score = []
         lambdas = []
         train_test_difference_Of_R2 =  []
         lambda_ = 0
         while lambda_ <= 5:
             lambdas.append(lambda_)
```

```
    RidgeModel = Ridge(lambda_)
    RidgeModel.fit(X_train,y_train)
    trainR2 = RidgeModel.score(X_train,y_train)
    testR2 = RidgeModel.score(X_test,y_test)
    train_R2_score.append(trainR2)
    test_R2_score.append(testR2)

    lambda_ += 0.01
```

In [61]:

```
plt.figure(figsize = (10,10))
sns.lineplot(x=lambdas,y=train_R2_score,)
sns.lineplot(x=lambdas, y=test_R2_score)
plt.legend(['Train R2 Score','Test R2 score'])
plt.title("Effect of hyperparemater alpha on R2 scores of Train and test")


plt.show()
```



In [62]:

```
RidgeModel = Ridge(alpha = 0.1)
RidgeModel.fit(X_train,y_train)
```

```
trainR2 = RidgeModel.score(X_train,y_train)
testR2 = RidgeModel.score(X_test,y_test)
```

In [63]:

```
trainR2,testR2
```

Out[63]:

```
(0.8215098726041209, 0.8208639536156422)
```

In [64]:

```
RidgeModel.coef_
```

Out[64]:

```
array([[0.02069489, 0.01929637, 0.00700953, 0.00298992, 0.01334235,
        0.07044884, 0.00987467]])
```

In [65]:

```
RidgeModel_coefs = pd.DataFrame(RidgeModel.coef_.reshape(1,-1),columns=df.colu
RidgeModel_coefs["Intercept"] = RidgeModel.intercept_
RidgeModel_coefs
```

Out[65]:

	GRE_Score	TOEFL_Score	University_Rating	SOP	LOR	CGPA	Research	Interce
0	0.020695	0.019296	0.00701	0.00299	0.013342	0.070449	0.009875	0.7228

In [66]:

```
LinearRegression_Model_coefs
```

Out[66]:

	GRE_Score	TOEFL_Score	University_Rating	SOP	LOR	CGPA	Research
0	0.020675	0.019284	0.007001	0.002975	0.013338	0.070514	0.009873

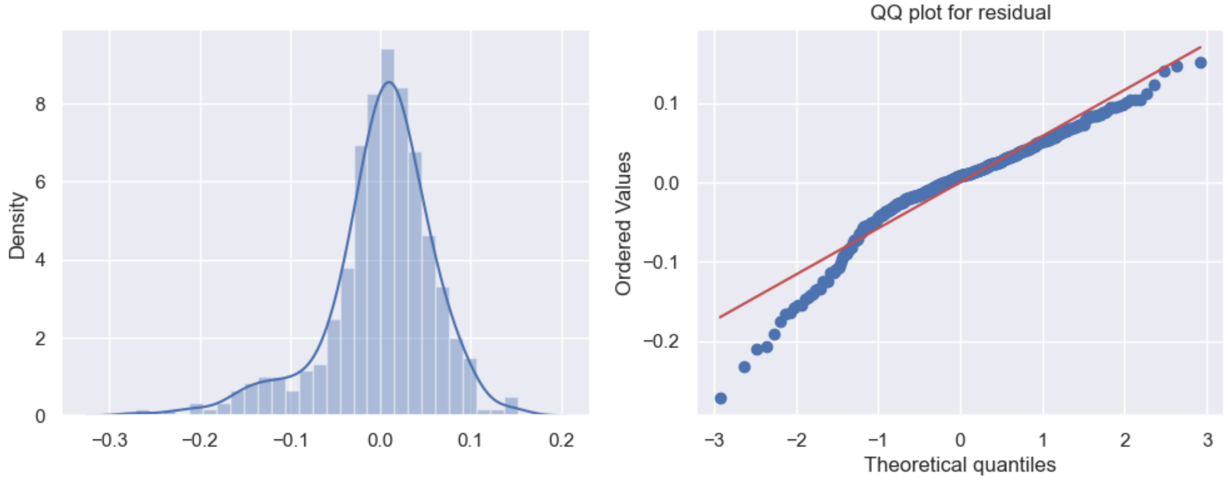
In [67]:

```
y_pred = RidgeModel.predict(X_test)

print("MSE:",mean_squared_error(y_test,y_pred)) # MSE
print("RMSE:",np.sqrt(mean_squared_error(y_test,y_pred))) #RMSE
print("MAE :",mean_absolute_error(y_test,y_pred) ) # MAE
print("r2_score:",r2_score(y_test,y_pred)) # r2score
print("Adjusted R2 score :", AdjustedR2score(r2_score(y_test,y_pred),len(X),X.s
```

```
MSE: 0.0034592961917283335
RMSE: 0.05881578182535988
MAE : 0.04020305511705697
r2_score: 0.8208639536156422
Adjusted R2 score : 0.8183152700288729
```

In [68]:

```
y_predicted = RidgeModel.predict(X_train)

residuals = (y_train - y_predicted)
plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
sns.distplot(residuals)
plt.subplot(1,2,2)
stats.probplot(residuals.reshape(-1,), plot = plt)
plt.title('QQ plot for residual')
plt.show()
```



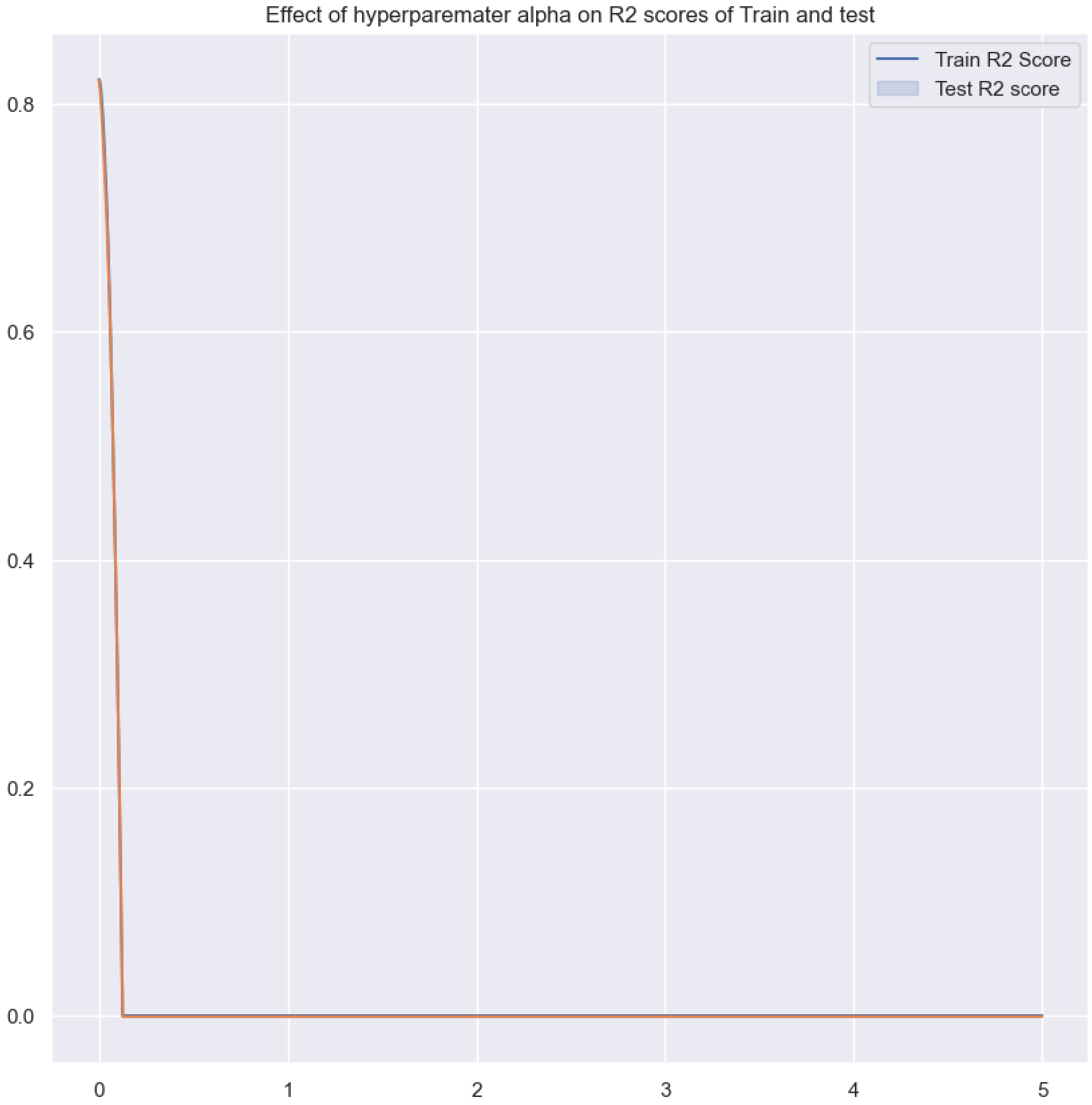
Lasso (`L1 regularization`)

In [69]:

```
## Hyperparameter Tuning : for appropriate lambda value :

train_R2_score = []
test_R2_score = []
lambdas = []
train_test_difference_Of_R2 =  []
lambda_ = 0
while lambda_ <= 5:
    lambdas.append(lambda_)
    LassoModel = Lasso(alpha=lambda_)
    LassoModel.fit(X_train , y_train)
    trainR2 = LassoModel.score(X_train,y_train)
    testR2 = LassoModel.score(X_test,y_test)
    train_R2_score.append(trainR2)
    test_R2_score.append(testR2)

    lambda_ += 0.001
```

In [70]:

```
plt.figure(figsize = (10,10))
sns.lineplot(x=lambdas,y=train_R2_score,)
sns.lineplot(x=lambdas, y=test_R2_score)
plt.legend(['Train R2 Score','Test R2 score'])
plt.title("Effect of hyperparemater alpha on R2 scores of Train and test")


plt.show()
```



In [71]:
```
LassoModel = Lasso(alpha=0.001)
LassoModel.fit(X_train , y_train)
trainR2 = LassoModel.score(X_train,y_train)
testR2 = LassoModel.score(X_test,y_test)
```

In [72]:
```
trainR2,testR2
```

Out[72]:
```
(0.82142983289567, 0.8198472607571161)
```

In [73]:
```
Lasso_Model_coefs = pd.DataFrame(LassoModel.coef_.reshape(1,-1),columns=df.colu
Lasso_Model_coefs["Intercept"] = LassoModel.intercept_
Lasso_Model_coefs
```

Out[73]:

	GRE_Score	TOEFL_Score	University_Rating	SOP	LOR	CGPA	Research	Interc
0	0.020616	0.019069	0.006782	0.002808	0.012903	0.070605	0.009278	0.7228

In [74]:

```
RidgeModel_coefs
```

Out[74]:

	GRE_Score	TOEFL_Score	University_Rating	SOP	LOR	CGPA	Research	Interce
0	0.020695	0.019296	0.00701	0.00299	0.013342	0.070449	0.009875	0.7228

In [75]:

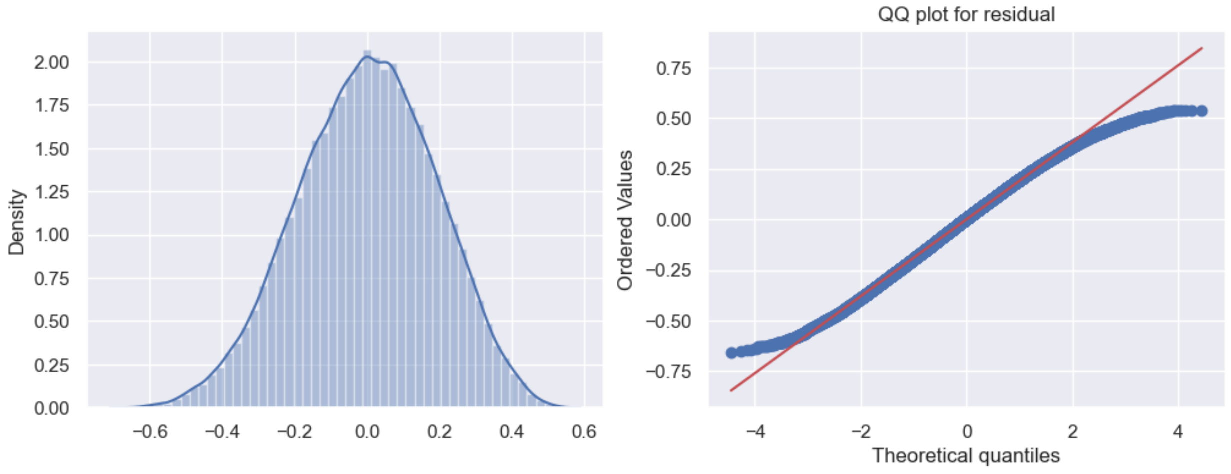
```
LinearRegression_Model_coefs
```

Out[75]:

	GRE_Score	TOEFL_Score	University_Rating	SOP	LOR	CGPA	Research
0	0.020675	0.019284	0.007001	0.002975	0.013338	0.070514	0.009873

In [76]:

```
y_predicted = LassoModel.predict(X_train)

residuals = (y_train - y_predicted)
plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
sns.distplot(residuals)
plt.subplot(1,2,2)
stats.probplot(residuals.reshape(-1,), plot = plt)
plt.title('QQ plot for residual')
plt.show()
```



Report

In [77]:

```
y_pred = LinearRegression.predict(X_test)
LinearRegression_model_metrics = []
LinearRegression_model_metrics.append(mean_squared_error(y_test,y_pred)) # MSE
LinearRegression_model_metrics.append(np.sqrt(mean_squared_error(y_test,y_pred
LinearRegression_model_metrics.append(mean_absolute_error(y_test,y_pred) ) # M
LinearRegression_model_metrics.append(r2_score(y_test,y_pred)) # r2score
LinearRegression_model_metrics.append(AdjustedR2score(r2_score(y_test,y_pred),
```

In [78]:

```
y_pred = RidgeModel.predict(X_test)
RidgeModel_model_metrics = []
RidgeModel_model_metrics.append(mean_squared_error(y_test,y_pred)) # MSE
RidgeModel_model_metrics.append(np.sqrt(mean_squared_error(y_test,y_pred))) #R
RidgeModel_model_metrics.append(mean_absolute_error(y_test,y_pred) ) # MAE
RidgeModel_model_metrics.append(r2_score(y_test,y_pred)) # r2score
RidgeModel_model_metrics.append(AdjustedR2score(r2_score(y_test,y_pred),len(X)
```

In [79]:

```
y_pred = LassoModel.predict(X_test)
LassoModel_model_metrics = []
LassoModel_model_metrics.append(mean_squared_error(y_test,y_pred)) # MSE
LassoModel_model_metrics.append(np.sqrt(mean_squared_error(y_test,y_pred))) #RM
LassoModel_model_metrics.append(mean_absolute_error(y_test,y_pred) ) # MAE
LassoModel_model_metrics.append(r2_score(y_test,y_pred)) # r2score
LassoModel_model_metrics.append(AdjustedR2score(r2_score(y_test,y_pred),len(X)
```

In [80]:

```
perf = pd.DataFrame([LinearRegression_model_metrics,LassoModel_model_metrics,Ri
```

In [81]:

```
perf
```

Out[81]:

	MSE	RMSE	MAE	R2_SCORE	ADJUSTED_R2
Linear Regression Model	0.003459	0.058814	0.040200	0.820874	0.818326
Lasso Regression Model	0.003479	0.058982	0.040229	0.819847	0.817284
Ridge Regression Model	0.003459	0.058816	0.040203	0.820864	0.818315

In [82]:

```
coff = pd.DataFrame(LinearRegression_Model_coefs.append(Lasso_Model_coefs).appe
coff.index = ["Linear Regression Model","Lasso Regression Model","Ridge Regress
```

In [83]:

```
final_report = coff.reset_index().merge(perf.reset_index())
final_report
```

Out[83]:

	index	GRE_Score	TOEFL_Score	University_Rating	SOP	LOR	CGPA	Rese
0	Linear Regression Model	0.020675	0.019284	0.007001	0.002975	0.013338	0.070514	0.00
1	Lasso Regression Model	0.020616	0.019069	0.006782	0.002808	0.012903	0.070605	0.00
2	Ridge Regression Model	0.020695	0.019296	0.007010	0.002990	0.013342	0.070449	0.00

End of the Report