In [83]:
```python
# Basic Exploratory Data Analysis

# iMPORTING Libraries

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.api as sm
```

In [84]:
```python
df = pd.read_csv('Jamboree_Admission.csv')
```

In [85]:
```python
df.head()
```

Out[85]:

	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 [86]:
```python
# cheking for null values

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

Out[86]:
```
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 [87]:
```python
# There are no null values in the dataset
```

In [88]:
```python
df.shape
```

Out[88]:
```
(500, 9)
```

In [89]:
```python
# The above dataset contains 500 rows and 9 columns
```

In [90]:
```python
# Cheking for unique values in each datset
df.nunique()
```

Out[90]:
```
Serial No.           500
GRE Score             49
TOEFL Score           29
University Rating      5
SOP                    9
LOR                    9
CGPA                 184
Research               2
Chance of Admit       61
dtype: int64
```

In [91]:
```python
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 [92]:
```
# As, University Rating, SOP, LOR and Research values are very small, we can consider them as categorical variables.
```

In [95]:
```
## datatypes of the dataset
df.dtypes
```

Out[95]:
```
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 [96]:
```
#dropping unwanted column "Serial No" and deciding the target variable
```

In [97]:
```
df.drop(columns = 'Serial No.', inplace = True)
```

In [98]:
```
df.head()
```

Out[98]:

	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 [99]:
```
# Weights/Independent Variable - GRE Score, TOEFL Score, University Raing, SOP, LOR, CGPA, Research
# Target Variable - Chance of Admit

df.columns = ['GRE_Score', 'TOEFL_Score', 'University_Rating', 'SOP', 'LOR', 'CGPA', 'Research', 'Chance_of_Admit' ]
```

In [100]:
```
df.head()
```

Out[100]:

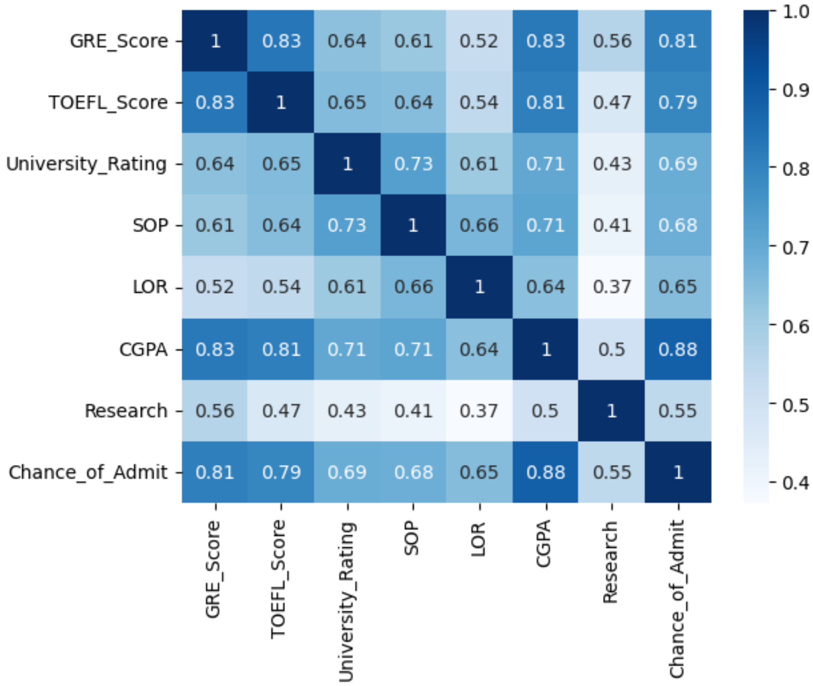
	GRE_Score	TOEFL_Score	University_Rating	SOP	LOR	CGPA	Research	Chance_of_Admit
0	337	118	4	4.5	4.5	9.65	1	0.92
1	324	107	4	4.0	4.5	8.87	1	0.76
2	316	104	3	3.0	3.5	8.00	1	0.72
3	322	110	3	3.5	2.5	8.67	1	0.80
4	314	103	2	2.0	3.0	8.21	0	0.65

In [101]:
```
df.dtypes
```

Out[101]:
```
GRE_Score              int64
TOEFL_Score            int64
University_Rating      int64
SOP                  float64
LOR                  float64
CGPA                 float64
Research               int64
Chance_of_Admit      float64
dtype: object
```

In [102]:
```
# Checking the correlation among all variables
```

In [103]:
```
sns.heatmap(df.corr(), annot = True, cmap='Blues')
plt.show()
```

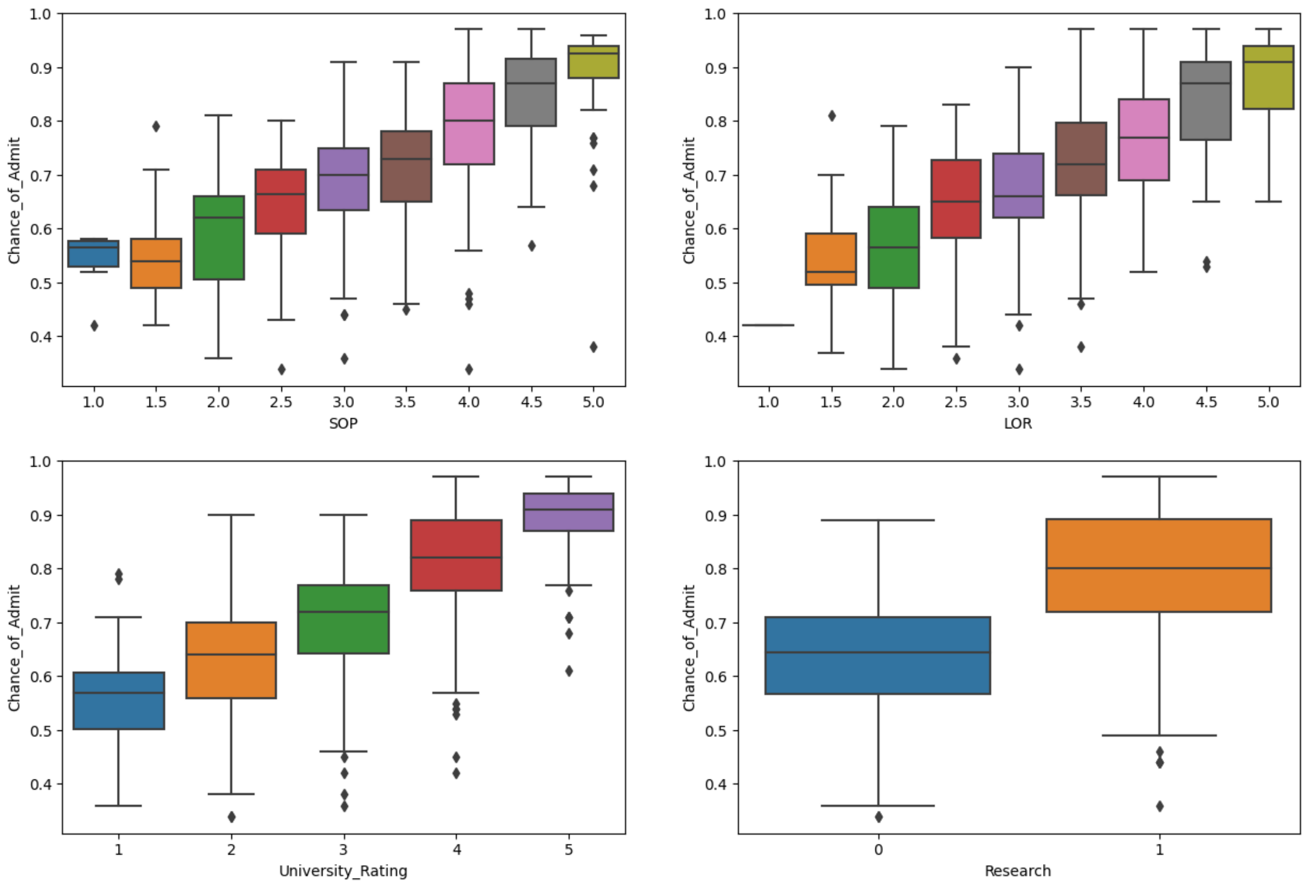


In [206]:
```
# Checking for outliers in the dataset

def detect_outliers(data):
    len_before = len(data)
    Q1 = np.percentile(data,25)
    Q3 = np.percentile(data,75)
    IQR = Q3 - Q1
    upperbound = Q3 + 1.5*IQR
    lowerbound = Q1 - 1.5*IQR
    if lowerbound < 0:
        lowerbound = 0

    len_after = len(data[data>lowerbound & data<upperbound])
    return np.round(((len_before - len_after)/len_before)*100,2)
```

In [207]:
```
for i in df.columns:
    print(f"Outliers in i is : {detect_outliers(df[i])}")
```

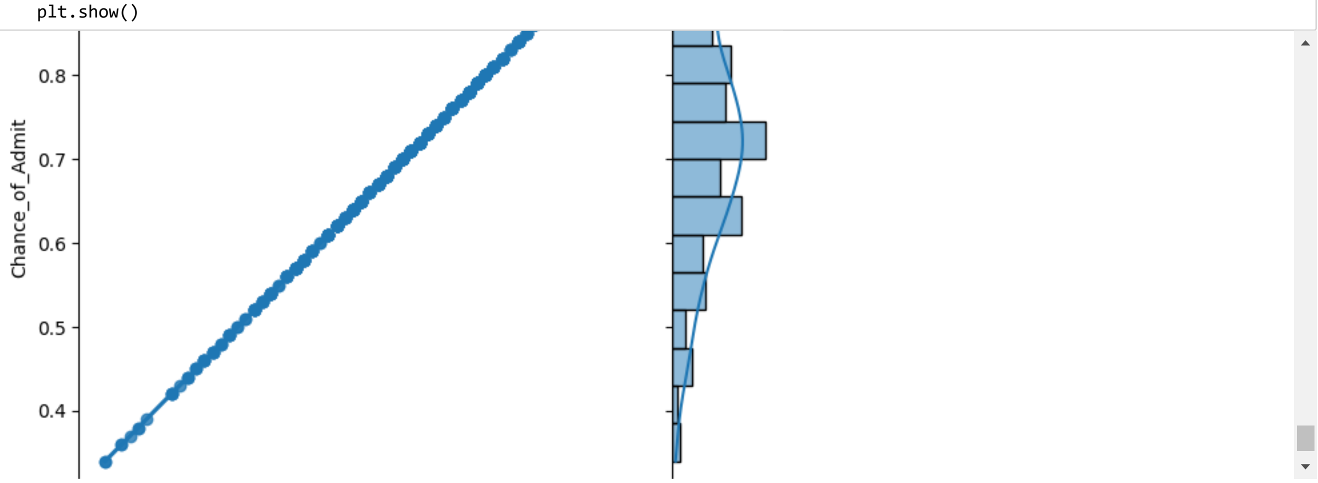
```
--> 401 res_values = na_logical_op(lvalues, rvalues, op)
    402 # error: Cannot call function of unknown type
    403 res_values = filler(res_values)  # type: ignore[operator]

File ~\AppData\Local\Programs\Python\Python311\Lib\site-packages\pandas\core\ops\array_ops.py:334, in na_logical_op(x, y, op)
    326         except (
    327             TypeError,
    328             ValueError,
   (...)
    331             NotImplementedError,
    332         ) as err:
    333             typ = type(y).__name__
--> 334             raise TypeError(
    335                 f"Cannot perform '{op.__name__}' with a dtyped [{x.dtype}] array "
    336                 f"and scalar of type [{typ}]"
    337             ) from err
    339 return result.reshape(x.shape)

TypeError: Cannot perform 'rand_' with a dtyped [int64] array and scalar of type [bool]
```

In [105]:
```
# Outliers in the categorical data
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()
```



In [ ]:
```
# Outliers exist every categorical data
# The above box plots also shows that all features ar positively correlated with the target variable
```

In [106]:
```
# Checking for correlation of features with the target variable

for col in df.columns:
    print(col)
    plt.figure(figsize=(3,3))
    sns.jointplot(data = df,x = df[col], y = df["Chance_of_Admit"],kind="reg")
    plt.show()
```



In [107]:
```
# Above correlation plot shows a linear relation between every feature in the dataset and the target variable - Chance of Admit
```

In [187]:
```
## Descriptive analysis of numerical variables

df.describe()
```

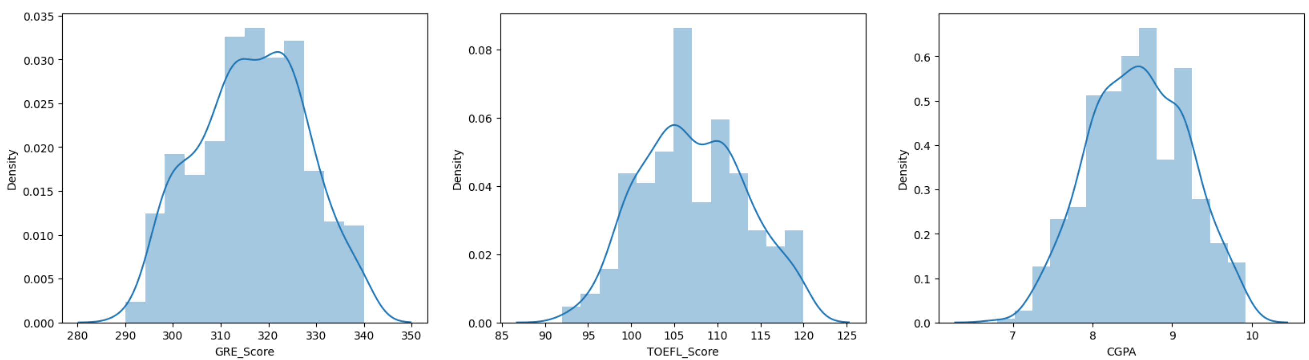
Out[187]:

	GRE_Score	TOEFL_Score	University_Rating	SOP	LOR	CGPA	Research	Chance_of_Admit
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 [ ]:
```
# chances of admit is within 0 to 1 (no outliers observed).
# Range of GRE score is between 290 to 340.
# Range of TOEFL score is between 92 to 120.
# University rating , SOP and LOR are distributed between range of 1 to 5.
# CGPA range is between 6.8 to 9.92
```

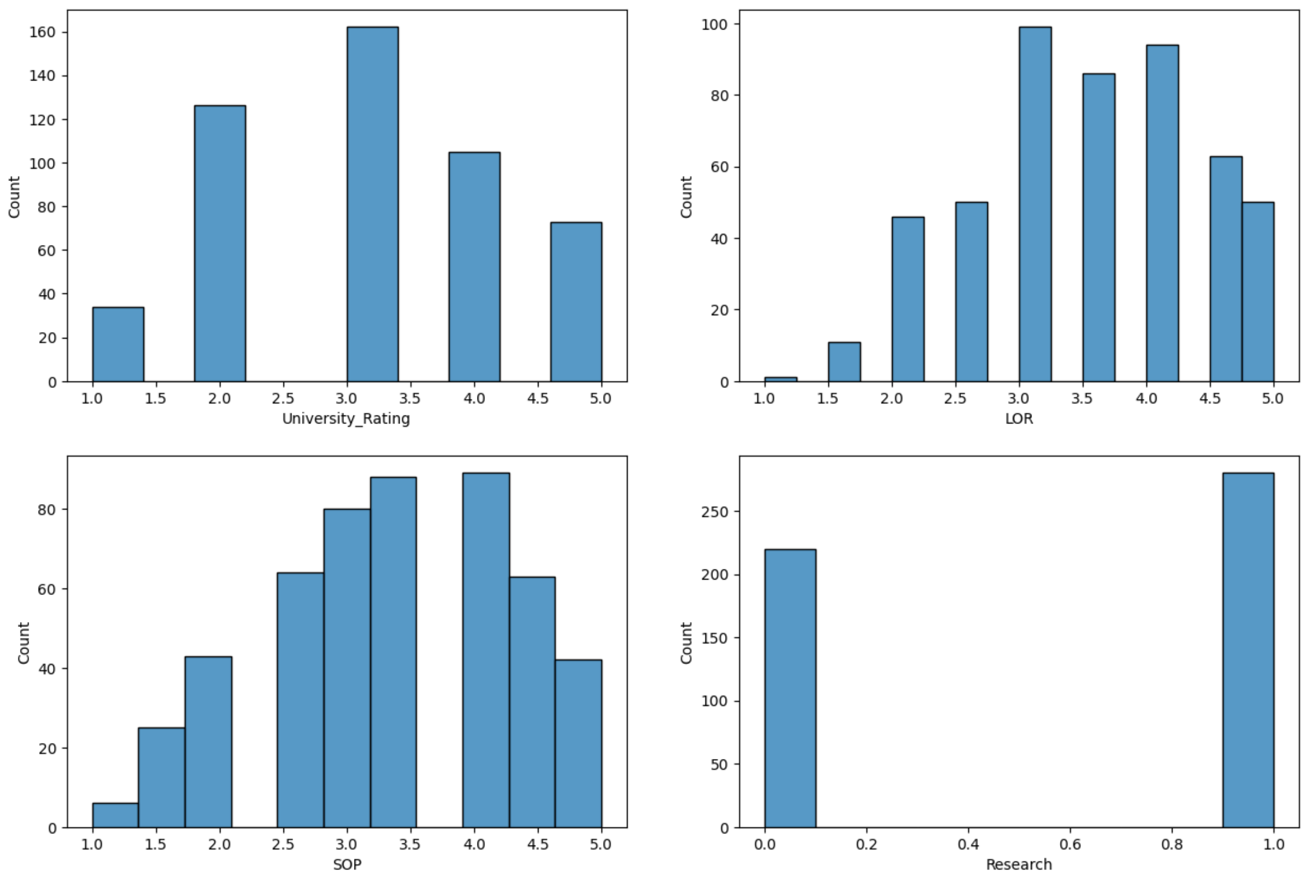
In [202]:
```
## Graphical analysis

# Distribution of numerical variables

# sns.distplot(df["TOEFL_Score"])
# sm.qqplot(df["TOEFL_Score"],fit=True, line="45")
# plt.show()
# plt.figure(figsize=(14,5))
# sns.boxplot(y = df["Chance_of_Admit"], x = df["TOEFL_Score"])

plt.figure(figsize=(20,5))
plt.subplot(1,3,1)
sns.distplot(df["GRE_Score"])
plt.subplot(1,3,2)
sns.distplot(df["TOEFL_Score"])
plt.subplot(1,3,3)
sns.distplot(df["CGPA"])

plt.show()
```

```
C:\Users\Admin\AppData\Local\Temp\ipykernel_2764\867199610.py:13: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751 (https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5
751)

  sns.distplot(df["GRE_Score"])
C:\Users\Admin\AppData\Local\Temp\ipykernel_2764\867199610.py:15: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751 (https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5
751)

  sns.distplot(df["TOEFL_Score"])
C:\Users\Admin\AppData\Local\Temp\ipykernel_2764\867199610.py:17: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751 (https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5
751)

  sns.distplot(df["CGPA"])
```



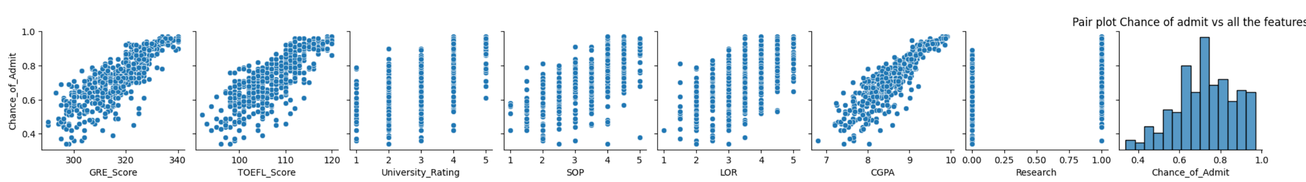
In [204]:
```
# Distribution of categorical variables

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

Out[204]: <Axes: xlabel='Research', ylabel='Count'>



In [205]:
```
sns.pairplot(df,y_vars = ["Chance_of_Admit"])
plt.title("Pair plot Chance of admit vs all the features")
plt.show()
```

```
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure layou
t has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



In [108]:
```
# Linear Regression
```

In [109]:
```
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
```

In [110]:
```
# Independent variables - X
# Dependent/Target variable - y

X = df.drop(columns = 'Chance_of_Admit', axis = 1)
y = df['Chance_of_Admit'].values
```

In [113]:
```
# Standardizing the dataset using StandardScaler
scaler = StandardScaler()
```

In [114]:
```
x = scaler.fit_transform(X)
```

In [115]:
```
# Spliting the data into train and test data

X_train, X_test, y_train, y_test = train_test_split(x,y,test_size = 0.2, random_state = 1)
```

In [182]:
```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[182]:
```
((400, 7), (100, 7), (400,), (100,))
```

In [117]:
```
# Fitting the model
linear = LinearRegression()
```

In [118]:
```
linear.fit(X_train, y_train)
```

Out[118]:
```
▾ LinearRegression
LinearRegression()
```

In [119]:
```
# Checking the r2_score on train and test data
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
```

In [208]:
```
# r2_score on training data
y_pred = linear.predict(X_train)
r2score_train = r2_score(y_train,y_pred)
r2score_train
```

Out[208]:
```
0.8215099192361265
```

In [210]:
```
# r2_score on test data
r2score_test = r2_score(y_test,linear.predict(X_test))
r2score_test
```

Out[210]:
```
0.8208741703103732
```

In [213]:
```
# Calculating feature coefficients and intercepts
ws = pd.DataFrame(linear.coef_.reshape(1,-1),columns=df.columns[:-1])
ws["Intercept"] = linear.intercept_
ws
```

Out[213]:

	GRE_Score	TOEFL_Score	University_Rating	SOP	LOR	CGPA	Research	Intercept
0	0.020675	0.019284	0.007001	0.002975	0.013338	0.070514	0.009873	0.722881

In [214]:
```
def AdjustedR2score(R2,n,d):
    return 1-(((1-R2)*(n-1))/(n-d-1))
```

In [221]:
```
# Checking the model metrics

# Training data performance checked

y_pred = linear.predict(X_train)
print("MSE:",mean_squared_error(y_train,y_pred)) # MSE
print("RMSE:",np.sqrt(mean_squared_error(y_train,y_pred))) #RMSE
print("MAE :",mean_absolute_error(y_train,y_pred) ) # MAE
print("r2_score:",r2_score(y_train,y_pred)) # r2score
print("Adjusted R2 score :", AdjustedR2score(r2_score(y_train,y_pred),len(X),X.shape[1]))
```

```
MSE: 0.0035733525638779674
RMSE: 0.05977752557506849
MAE : 0.04294488315548092
r2_score: 0.8215099192361265
Adjusted R2 score : 0.8189704262171282
```

In [222]:
```
# Test Performance checked

y_pred = linear.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.shape[1]))
```

```
MSE: 0.003459098897136383
RMSE: 0.05881410457650769
MAE : 0.040200193804157944
r2_score: 0.8208741703103732
Adjusted R2 score : 0.8183256320830818
```

In [137]:
```
# Checking the assumptions of linear regression

# 1. multicollinearity check

vif = []
for i in range(X_train.shape[1]):
    vif.append(variance_inflation_factor(exog = X_train, exog_idx = i))
vif
```

Out[137]:
```
[4.873264779539272,
 4.243883338617031,
 2.798251888543383,
 2.9200455031169206,
 2.0793343045164447,
 4.751389166380193,
 1.5081475402055684]
```

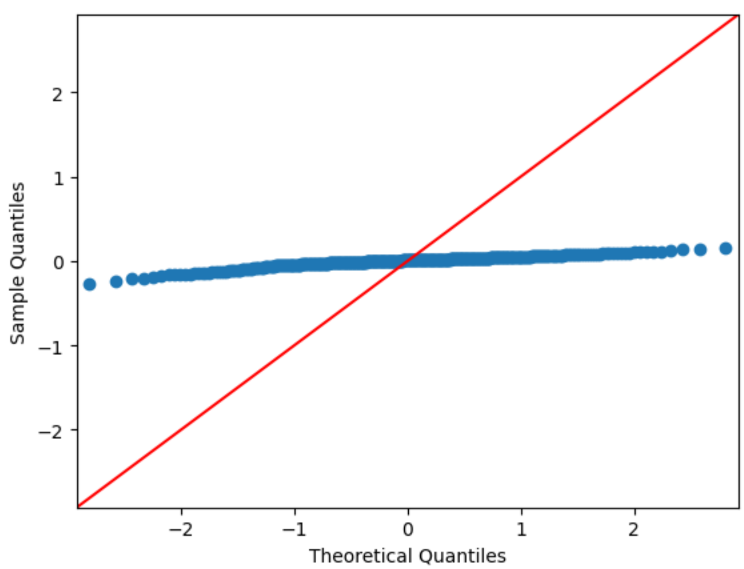
In [138]:
```
pd.DataFrame({"coeff_name":X.columns, "vif":np.round(vif,2)})
```

Out[138]:

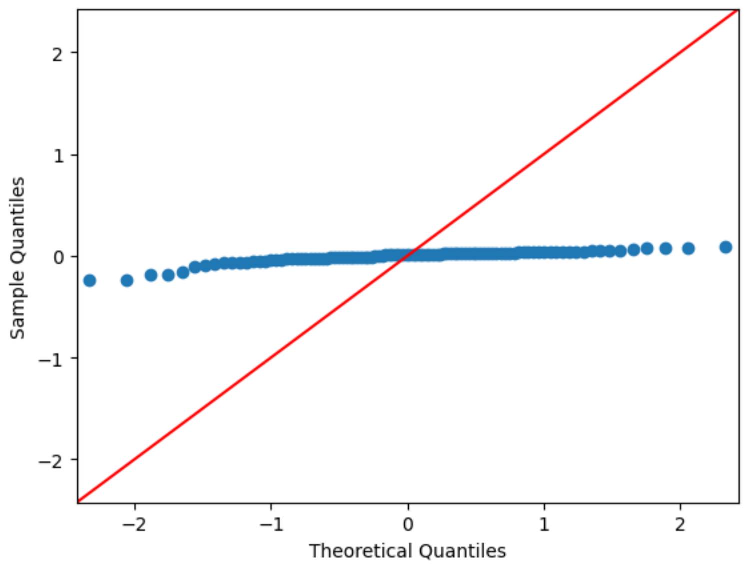
	coeff_name	vif
0	GRE_Score	4.87
1	TOEFL_Score	4.24
2	University_Rating	2.80
3	SOP	2.92
4	LOR	2.08
5	CGPA	4.75
6	Research	1.51

In [139]:
```
# From above analysis, we can say that since the vif_score of every feature is < 5, multicollinearity does not exist.
```

In [223]:
```
# 2. Normality check of residuals

y_pred = linear.predict(X_train)
residual = y_train - y_pred
sm.qqplot(residual, line = '45')
plt.show()
```

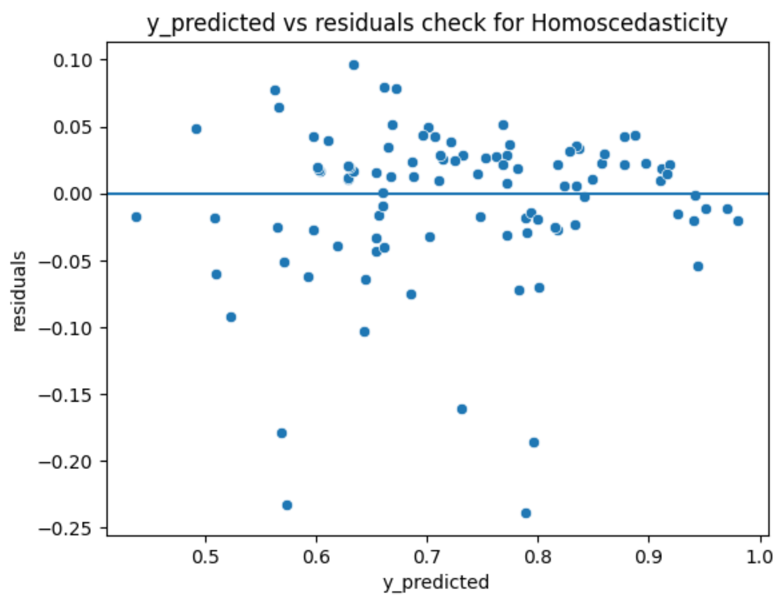


In [228]:
```
y_pred = linear.predict(X_test)
residual = y_test - y_pred
sm.qqplot(residual, line = '45')
plt.show()
```



In [229]:
```
# the above plot shows that residuals are not normally distributed
```

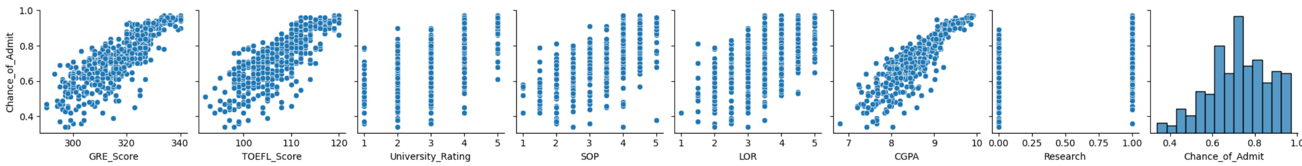
In [230]:
```
# 3. Test for Homoscedasticity

y_train_pred = linear.predict(X_train)
sns.scatterplot(x = y_pred,y = residual)
plt.xlabel('y_predicted')
plt.ylabel('residuals')
plt.axhline(y=0)
plt.title('y_predicted vs residuals check for Homoscedasticity')
plt.show()
```



In [231]:
```
# the above analysis for homoscadesticity show that the spread for residuals is evenly distributed
```

In [233]:
```python
# 4. Linearity check betweeen dependent and independent variables
sns.pairplot(df,y_vars = ["Chance_of_Admit"])
plt.show()
```

```
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure layou
t has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



In [146]:
```python
# The above plots show a linear relationship between the dependent and the target variable.
```

In [147]:
```python
# Model Regularization

from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV
```

In [148]:
```python
# 1. Lasso Regression - L1 Regularization

lasso_regressor = Lasso()
parameters = {'alpha': [1,2,3,5,10,20,30,40,50,60,70,80,90,100]}
```

In [149]:
```python
lassocv =  GridSearchCV(lasso_regressor, param_grid = parameters, scoring = "neg_mean_squared_error", cv = 5)
lassocv.fit(X_train, y_train)
```

Out[149]:
```
▸ GridSearchCV
▸ estimator: Lasso
    ▸ Lasso
```

In [150]:
```python
print(lassocv.best_params_)
```

```
{'alpha': 1}
```

In [151]:
```python
## gives the best/optimal alpha value as 1
```

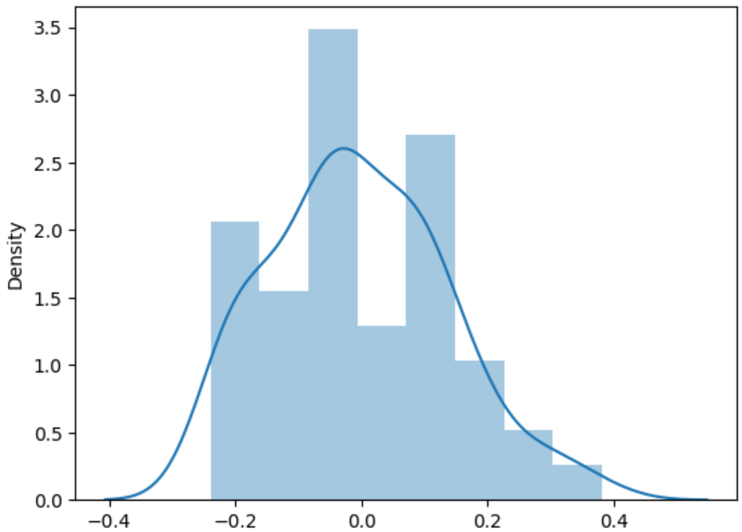
In [152]:
```python
lassocv.best_score_
```

Out[152]:
```
-0.020239457812499997
```

In [153]:
```python
lasso_pred = lassocv.predict(X_test)
```

In [154]:
```
sns.distplot((lasso_pred - y_test), kde = True)
plt.show()
```

```
C:\Users\Admin\AppData\Local\Temp\ipykernel_2764\1166656843.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751 (https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5
751)

  sns.distplot((lasso_pred - y_test), kde = True)
```



In [155]:
```
# The variance is too less within -0.4 to +0.4
```

In [170]:
```
lasso_score = r2_score(lasso_pred,y_test)
```

In [234]:
```
lasso_score
```

Out[234]: -3.9201041394409206e+29

In [158]:
```
# 2. Ridge regression - L1 regularization
from sklearn.linear_model import Ridge
```

In [159]:
```
ridge_regressor = Ridge()
```

In [160]:
```
parameters = {'alpha': [1,2,3,5,10,20,30,40,50,60,70,80,90,100]}
```

In [162]:
```
ridgecv = GridSearchCV(ridge_regressor, param_grid = parameters, scoring = "neg_mean_squared_error", cv = 5)
ridgecv.fit(X_train, y_train)
```

Out[162]:
```
▸ GridSearchCV
▸ estimator: Ridge
    ▸ Ridge
```

In [163]:
```
print(ridgecv.best_params_)
```

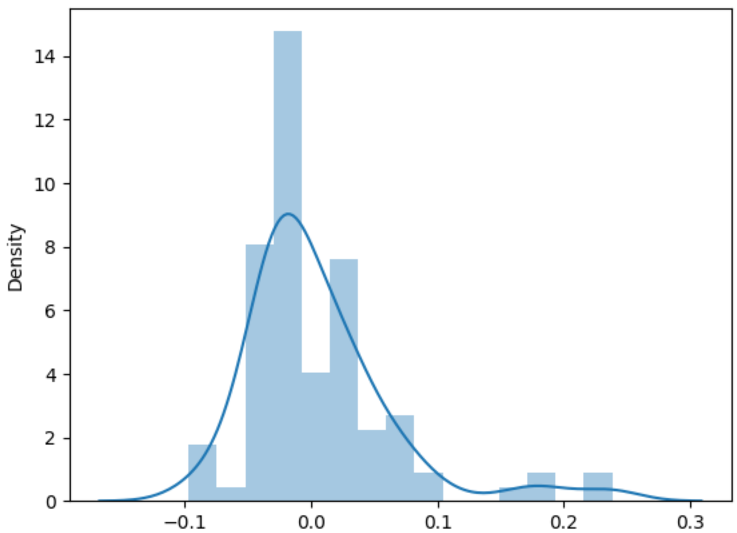
```
{'alpha': 10}
```

In [164]:
```
ridgecv.best_score_
```

Out[164]: -0.0038659877350002087

In [166]:
```
ridge_pred = ridgecv.predict(X_test)
sns.distplot((ridge_pred - y_test), kde = True)
plt.show()
```

```
C:\Users\Admin\AppData\Local\Temp\ipykernel_2764\1307189591.py:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751 (https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5
751)

  sns.distplot((ridge_pred - y_test), kde = True)
```



In [167]:
```
# The variance is too less within -0.2 to +0.3
```

In [169]:
```
ridge_score = r2_score(ridge_pred,y_test)
ridge_score
```

Out[169]: 0.758631384490402

In [239]:
```
# Checking for the errors after regularization

# Lasso Regression
LassoModel = Lasso(alpha=1)
LassoModel.fit(X_train , y_train)
trainR2 = LassoModel.score(X_train,y_train)
testR2 = LassoModel.score(X_test,y_test)
```

In [240]:
```
(trainR2,testR2)
```

Out[240]: (0.0, -0.000859904976438175)

In [248]:
```
Lasso_Model_coefs = pd.DataFrame(LassoModel.coef_.reshape(1,-1),columns=df.columns[:-1])
Lasso_Model_coefs["Intercept"] = LassoModel.intercept_
Lasso_Model_coefs
```

Out[248]:

	GRE_Score	TOEFL_Score	University_Rating	SOP	LOR	CGPA	Research	Intercept
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.720925

In [252]:
```
y_pred = LassoModel.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.shape[1]))
```

```
MSE: 0.019327605625
RMSE: 0.13902375921043136
MAE : 0.11310749999999999
r2_score: -0.000859904976438175
Adjusted R2 score : -0.015099781673257429
```

In [249]:
```
RidgeModel = Ridge(alpha = 10)
RidgeModel.fit(X_train,y_train)
trainR2 = RidgeModel.score(X_train,y_train)
testR2 = RidgeModel.score(X_test,y_test)
(trainR2,testR2)
```

Out[249]:
```
(0.8211455313249441, 0.8195916906322652)
```

In [250]:
```
RidgeModel_coefs = pd.DataFrame(RidgeModel.coef_.reshape(1,-1),columns=df.columns[:-1])
RidgeModel_coefs["Intercept"] = RidgeModel.intercept_
RidgeModel_coefs
```

Out[250]:

	GRE_Score	TOEFL_Score	University_Rating	SOP	LOR	CGPA	Research	Intercept
0	0.022297	0.020349	0.007755	0.004297	0.013707	0.064838	0.010049	0.7229

In [251]:
```
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.shape[1]))
```

```
MSE: 0.003483864862200326
RMSE: 0.059024273499979026
MAE : 0.04048175325171785
r2_score: 0.8195916906322652
Adjusted R2 score : 0.8170249057428869
```

In [253]:
```
# SUMMARY
#Linear Regression
y_pred = linear.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))) #RMSE
LinearRegression_model_metrics.append(mean_absolute_error(y_test,y_pred) ) # MAE
LinearRegression_model_metrics.append(r2_score(y_test,y_pred)) # r2score
LinearRegression_model_metrics.append(AdjustedR2score(r2_score(y_test,y_pred),len(X),X.shape[1]))  # adjusted R2 score

#Ridge Regression
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))) #RMSE
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),X.shape[1]))  # adjusted R2 score

#Lasso Regression
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))) #RMSE
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),X.shape[1]))  # adjusted R2 score
```

In [254]:
```
A = pd.DataFrame([LinearRegression_model_metrics,LassoModel_model_metrics,RidgeModel_model_metrics],columns=["MSE","RMSE","MAE","
A
```

Out[254]:

	MSE	RMSE	MAE	R2_SCORE	ADJUSTED_R2
Linear Regression Model	0.003459	0.058814	0.040200	0.820874	0.818326
Lasso Regression Model	0.019328	0.139024	0.113107	-0.000860	-0.015100
Ridge Regression Model	0.003484	0.059024	0.040482	0.819592	0.817025

In [ ]:
```
# Insights , Feature Importance and Interpretations and Recommendations :

# - University Rating , SOP and LOR strength and research seem to be discrete random Variables , but also ordinal numeric data.

# - all the other features are numeric, ordinal and continuous.

# - No null values were present in data.

# - No Significant amount of outliers were found in data.

# - correlation heatmap shows a strong correlation between the GRE score, TOEFL score and CGPA with Change of admission.

# - University rating, SOP ,LOR and Research have comparatively slightly less correlated than other features.

# - Students having high GRE score , has higher probability of getting admission .

# - Students having high TOEFL score , has higher probability of getting admission .

# - the performance metrics are very similar for dataset fitted using linear model and after ridge regression

# Actionable Insights and Recommendations :

# - Awareness of CGPA and Research Capabilities : Seminars can be organised to increase the awareness regarding CGPA and Research
# - GRE and TOEFL scores also play an important role as they are linearly distributed with the probability of getting admission.
# - Proper awareness and coaching for the above mentioned exams as well as maintaing good SOP and LOR can help the student get ad
```