Jamboree_Admission_Linear_Regression_ML_Model

October 8, 2024

1 Jamboree Admission - Linear Regression

Company Profile:

- Jamboree is a renowned educational institution that has successfully assisted numerous students in gaining admission to top colleges abroad. With their proven problem-solving methods, they have helped students achieve exceptional scores on exams like GMAT, GRE, and SAT with minimal effort.
- To further support students, Jamboree has recently introduced a new feature on their website. This feature enables students to assess their probability of admission to Ivy League colleges, considering the unique perspective of Indian applicants.
- By conducting a thorough analysis, we can assist Jamboree in understanding the crucial factors impacting graduate admissions and their interrelationships. Additionally, we can provide predictive insights to determine an individual's admission chances based on various variables.

Column Profiling:

- Serial No.: This column represents the unique row identifier for each applicant in the dataset.
- GRE Scores: This column contains the GRE (Graduate Record Examination) scores of the applicants, which are measured on a scale of 0 to 340.
- TOEFL Scores: This column includes the TOEFL (Test of English as a Foreign Language) scores of the applicants, which are measured on a scale of 0 to 120.
- University Rating: This column indicates the rating or reputation of the university that the applicants are associated with. The rating is based on a scale of 0 to 5, with 5 representing the highest rating.
- SOP: This column represents the strength of the applicant's statement of purpose, rated on a scale of 0 to 5, with 5 indicating a strong and compelling SOP.
- LOR: This column represents the strength of the applicant's letter of recommendation, rated on a scale of 0 to 5, with 5 indicating a strong and compelling LOR.
- CGPA: This column contains the undergraduate Grade Point Average (GPA) of the applicants, which is measured on a scale of 0 to 10.
- Research: This column indicates whether the applicant has research experience (1) or not (0).
- Chance of Admit: This column represents the estimated probability or chance of admission for each applicant, ranging from 0 to 1.

[42]:
```
import pandas as pd
import matplotlib.pyplot as plt
```

```
import seaborn as sns
import numpy as np
```

[43]:
```
data = pd.read_csv(r"C:\Users\n.rahman\OneDrive - BALADNA\Desktop\BALADNA\Ex␣
↪Docs\SCALER-DSML\Module 9 - Intro to ML Maths\Jamboree_Admission.csv")
data.head()
```

[43]:
```
   Serial No.  GRE Score  TOEFL Score  University Rating  SOP  LOR   CGPA  \
0           1        337          118                  4  4.5   4.5  9.65
1           2        324          107                  4  4.0   4.5  8.87
2           3        316          104                  3  3.0   3.5  8.00
3           4        322          110                  3  3.5   2.5  8.67
4           5        314          103                  2  2.0   3.0  8.21

   Research  Chance of Admit
0         1             0.92
1         1             0.76
2         1             0.72
3         1             0.80
4         0             0.65
```

1.1 EDA

[44]:
```
data.shape
```

[44]:
```
(500, 9)
```

[45]:
```
data.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
```

[46]:
```
data.isnull().sum() #no missing values
```

[46]:
```
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
```

[144]:
```
data.duplicated().sum() #no duplicates
```

[144]:
```
0
```

[98]:
```
#rename the columns names
data.rename(columns={"Chance of Admit ":"Admit_Chance","LOR ":"LOR"},␣
 ↪inplace=True)
```

[99]:
```
data.head()
```

[99]:
```
   GRE Score  TOEFL Score  University Rating  SOP  LOR  CGPA  Research  \
0        337          118                  4  4.5  4.5  9.65         1
1        324          107                  4  4.0  4.5  8.87         1
2        316          104                  3  3.0  3.5  8.00         1
3        322          110                  3  3.5  2.5  8.67         1
4        314          103                  2  2.0  3.0  8.21         0

   Admit_Chance
0          0.92
1          0.76
2          0.72
3          0.80
4          0.65
```

[100]:
```
data.columns
```

[100]:
```
Index(['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR', 'CGPA',
       'Research', 'Admit_Chance'],
      dtype='object')
```

[53]:
```
# drop unwanted features -serial no definetely not required.
data.drop(columns="Serial No.",axis=1,inplace=True)
```

[101]:
```
data.head()
```

[101]:
```
   GRE Score  TOEFL Score  University Rating  SOP  LOR  CGPA  Research  \
0        337          118                  4  4.5  4.5  9.65         1
```

```
1        324        107                4  4.0  4.5  8.87        1
2        316        104                3  3.0  3.5  8.00        1
3        322        110                3  3.5  2.5  8.67        1
4        314        103                2  2.0  3.0  8.21        0

   Admit_Chance
0          0.92
1          0.76
2          0.72
3          0.80
4          0.65
```

[103]:
```python
data["University Rating"].value_counts()
```

[103]:
```
University Rating
3    162
2    126
4    105
5     73
1     34
Name: count, dtype: int64
```

[104]:
```python
data["Research"].value_counts()
```

[104]:
```
Research
1    280
0    220
Name: count, dtype: int64
```

converting the columns university rating and research columns as categorical variables

[105]:
```python
data["University Rating"] = data["University Rating"].astype(dtype="category")
data["Research"] = data["Research"].astype(dtype="category")
```

[106]:
```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
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    category
 3   SOP                500 non-null    float64
 4   LOR                500 non-null    float64
 5   CGPA               500 non-null    float64
 6   Research           500 non-null    category
 7   Admit_Chance       500 non-null    float64
```

```
dtypes: category(2), float64(4), int64(2)
memory usage: 24.9 KB
```

[107]:
```python
data.describe()
```

[107]:

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

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

1.1.1 Graphical Analysis (Univariate & Bivariate Analysis)

[69]:
```python
data.sample(5)
```

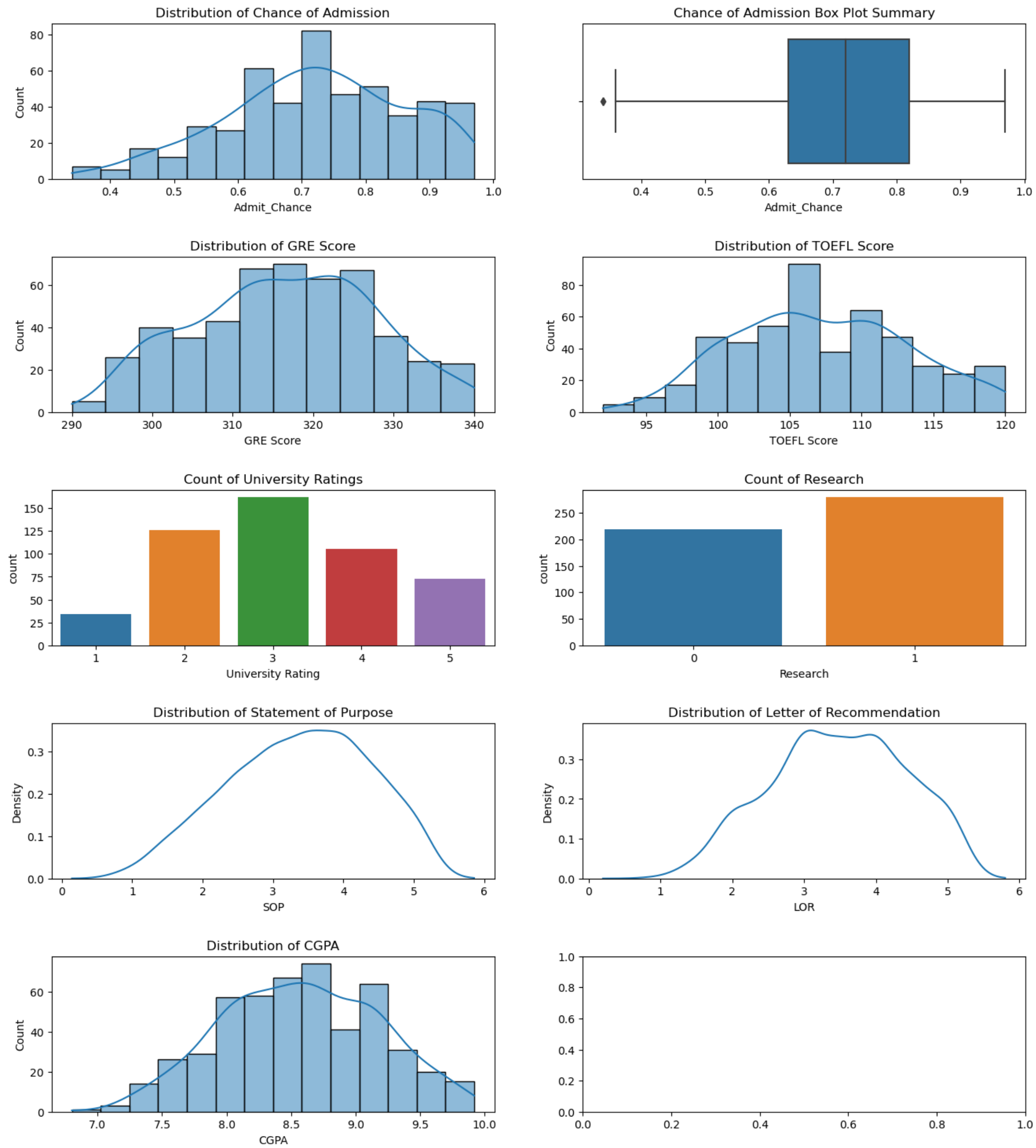
[69]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
260	327	108	5	5.0	3.5	9.13	1
291	300	102	2	1.5	2.0	7.87	0
450	320	112	4	3.0	4.5	8.86	1
36	299	106	2	4.0	4.0	8.40	0
280	311	102	3	4.5	4.0	8.64	1

	Admit_Chance
260	0.87
291	0.56
450	0.82
36	0.64
280	0.68

[329]:
```python
fig, axs = plt.subplots(nrows=5, ncols=2, figsize=(16,18))

#first row
sns.histplot(data=data,x="Admit_Chance",kde=True,ax =axs[0,0]).
 ↪set_title("Distribution of Chance of Admission")
```

```
sns.boxplot(data=data,x="Admit_Chance",ax=axs[0,1]).set_title("Chance of␣
↪Admission Box Plot Summary")

#second row
sns.histplot(data=data,x="GRE Score", kde=True,ax =axs[1,0]).
↪set_title("Distribution of GRE Score")
sns.histplot(data=data,x="TOEFL Score",kde=True, ax = axs[1,1]).
↪set_title("Distribution of TOEFL Score")

#third row
sns.countplot(data=data,x=data["University Rating"],ax = axs[2,0]).
↪set_title("Count of University Ratings")
sns.countplot(data=data,x=data["Research"],ax = axs[2,1]).set_title("Count of␣
↪Research")

#fourth row
sns.kdeplot(data=data,x="SOP",ax =axs[3,0]).set_title("Distribution of␣
↪Statement of Purpose")
sns.kdeplot(data=data,x="LOR", ax = axs[3,1]).set_title("Distribution of Letter␣
↪of Recommendation")


sns.histplot(data=data,x="CGPA",kde=True,ax=axs[4,0]).set_title("Distribution␣
↪of CGPA")

fig.subplots_adjust(hspace=0.5)
plt.show()
```



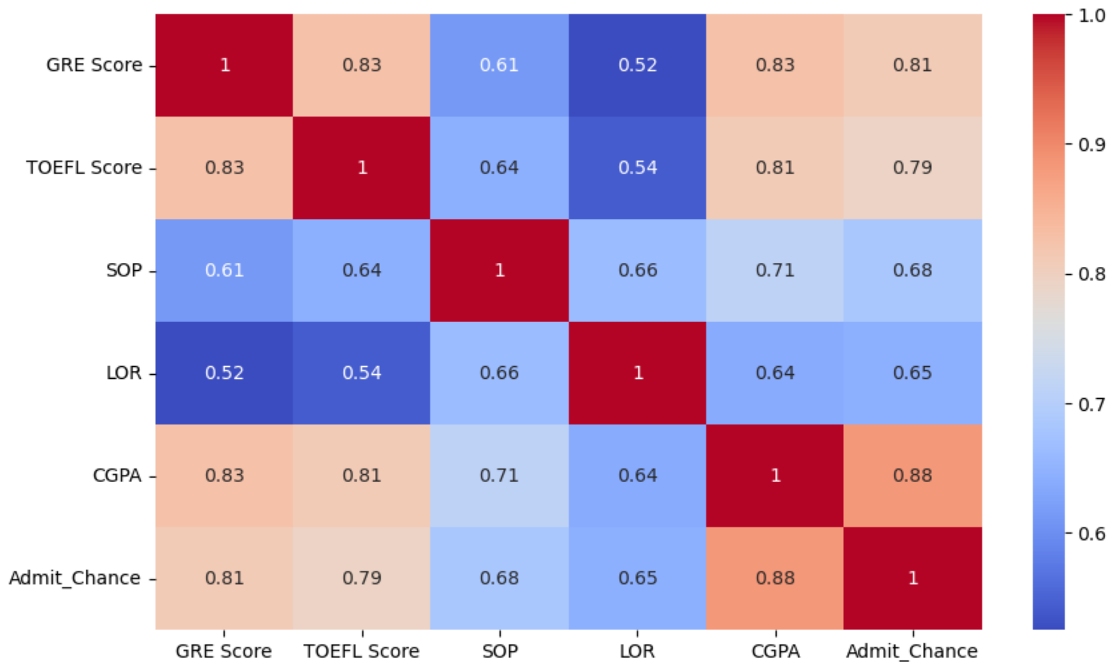
Insights:

- The distribution of the target variable (chance of admission) is slightly left-skewed and not normally distributed.
- Box plot analysis indicates that most of the data is concentrated between 0.7 and 0.8.
- GRE and TOEFL scores show a roughly normal distribution in the histogram, though not a perfect bell curve, with some outliers present.
- For the categorical variables, university rating and research, the majority of applicants have a university rating of 3, and most have research experience.
- The variables CGPA, SOP, and LOR also exhibit a near normal distribution in the his-

togram/KDE plot, though not perfectly symmetrical.

correlation matrix for numerical columns.

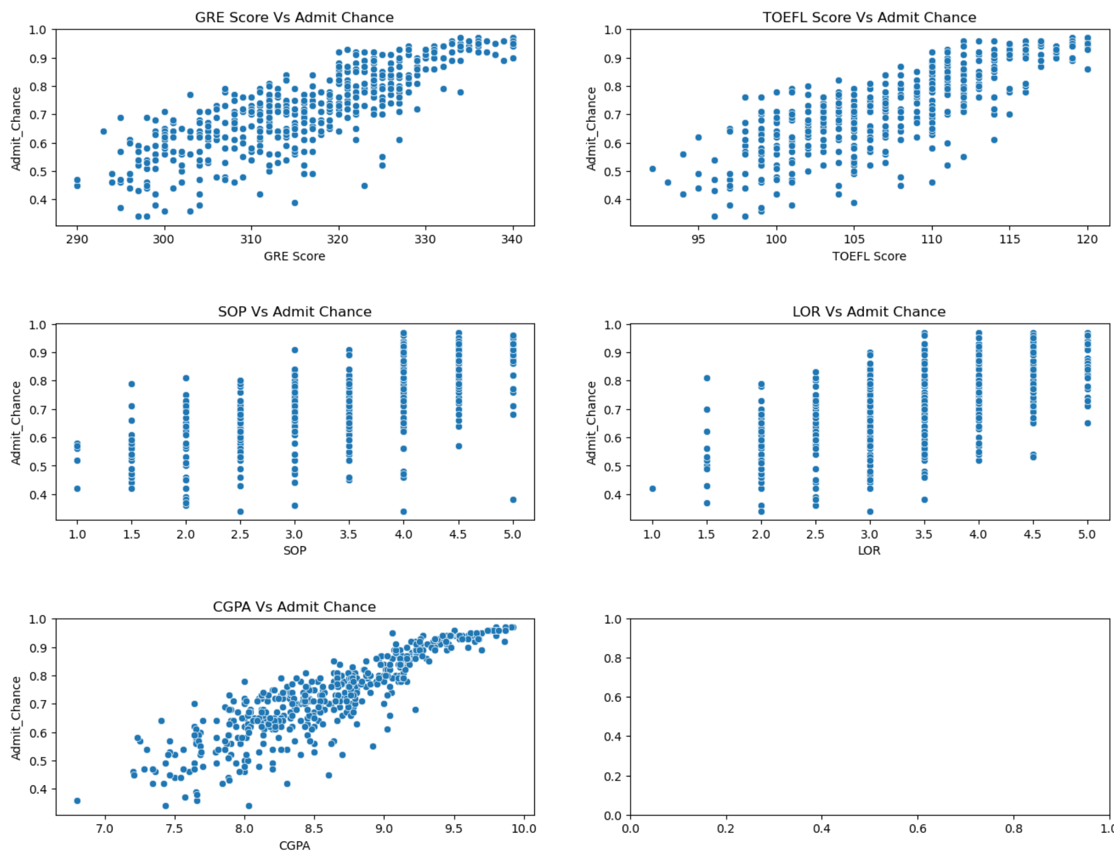
[152]:
```
# Remove categorical columns (University Rating and Research)
df_numeric = data.drop(["University Rating","Research"],axis=1)
corr_matrix = df_numeric.corr()

#plot

plt.figure(figsize=(10,6))
sns.heatmap(corr_matrix,cmap="coolwarm",annot=True)
plt.show()
```



checking the correlation for each independent numerical variables vs target variable

[130]:
```
fig, axs = plt.subplots(nrows=3, ncols=2, figsize=(16,12))

#first row
sns.scatterplot(data=data,x=data["GRE␣
↪Score"],y=data["Admit_Chance"],ax=axs[0,0]).set_title("GRE Score Vs Admit␣
↪Chance")
sns.scatterplot(data=data,x=data["TOEFL␣
↪Score"],y=data["Admit_Chance"],ax=axs[0,1]).set_title("TOEFL Score Vs Admit␣
↪Chance")

#second row
```

```
sns.scatterplot(data=data,x=data["SOP"],y=data["Admit_Chance"],ax=axs[1,0]).
↪set_title("SOP Vs Admit Chance")
sns.scatterplot(data=data,x=data["LOR"],y=data["Admit_Chance"],ax=axs[1,1]).
↪set_title("LOR Vs Admit Chance")

#third
sns.scatterplot(data=data,x=data["CGPA"],y=data["Admit_Chance"],ax=axs[2,0]).
↪set_title("CGPA Vs Admit Chance")

fig.subplots_adjust(hspace=0.5)
plt.show()
```



[135]:
```
data.head()
```

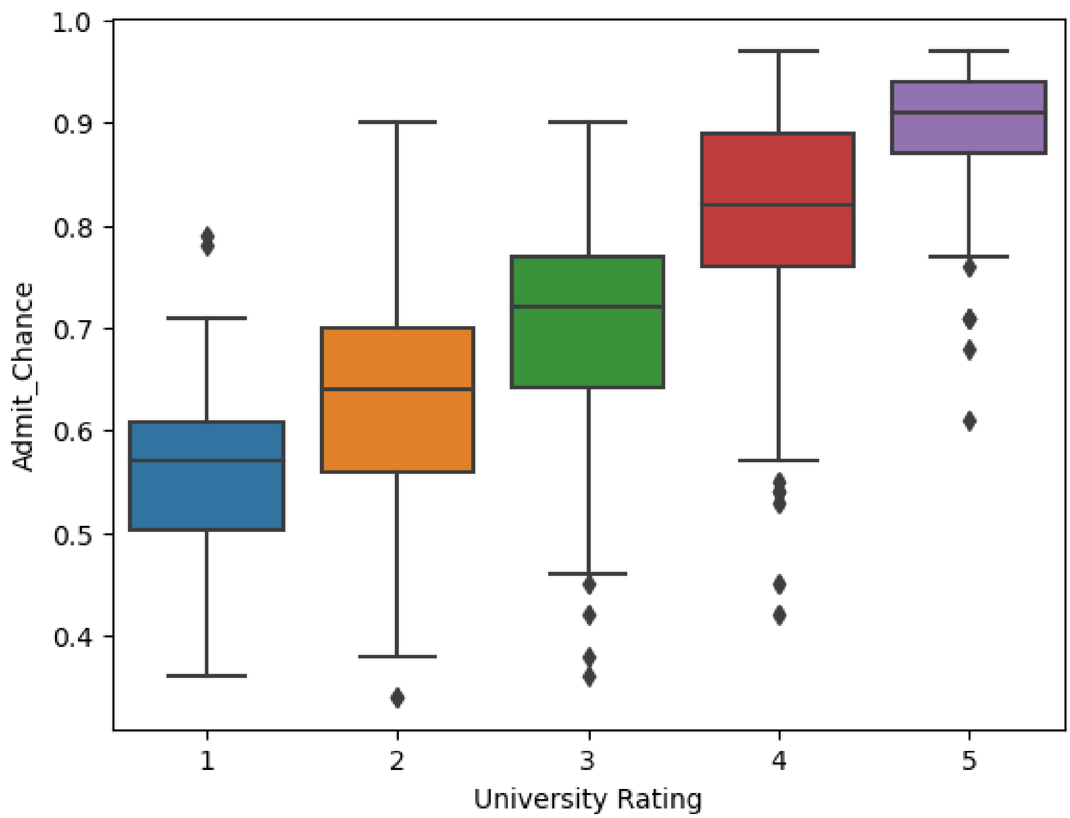
[135]:
```
   GRE Score  TOEFL Score  University Rating  SOP  LOR   CGPA  Research  \
0        337          118                  4  4.5  4.5   9.65         1
1        324          107                  4  4.0  4.5   8.87         1
2        316          104                  3  3.0  3.5   8.00         1
3        322          110                  3  3.5  2.5   8.67         1
4        314          103                  2  2.0  3.0   8.21         0
```

```
   Admit_Chance
0          0.92
1          0.76
2          0.72
3          0.80
4          0.65
```

box plot for categorical variable vs target (Admit_Chance)

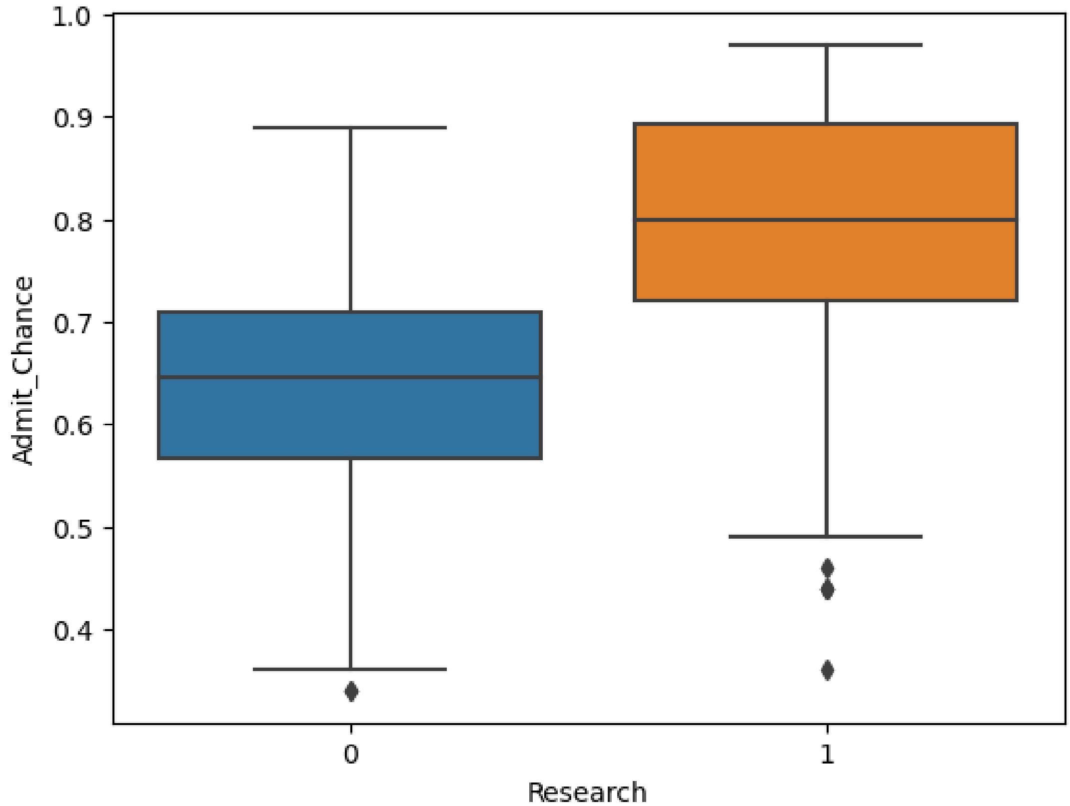
[138]:
```
sns.boxplot(data=data,x="University Rating",y="Admit_Chance")
```

[138]: <Axes: xlabel='University Rating', ylabel='Admit_Chance'>



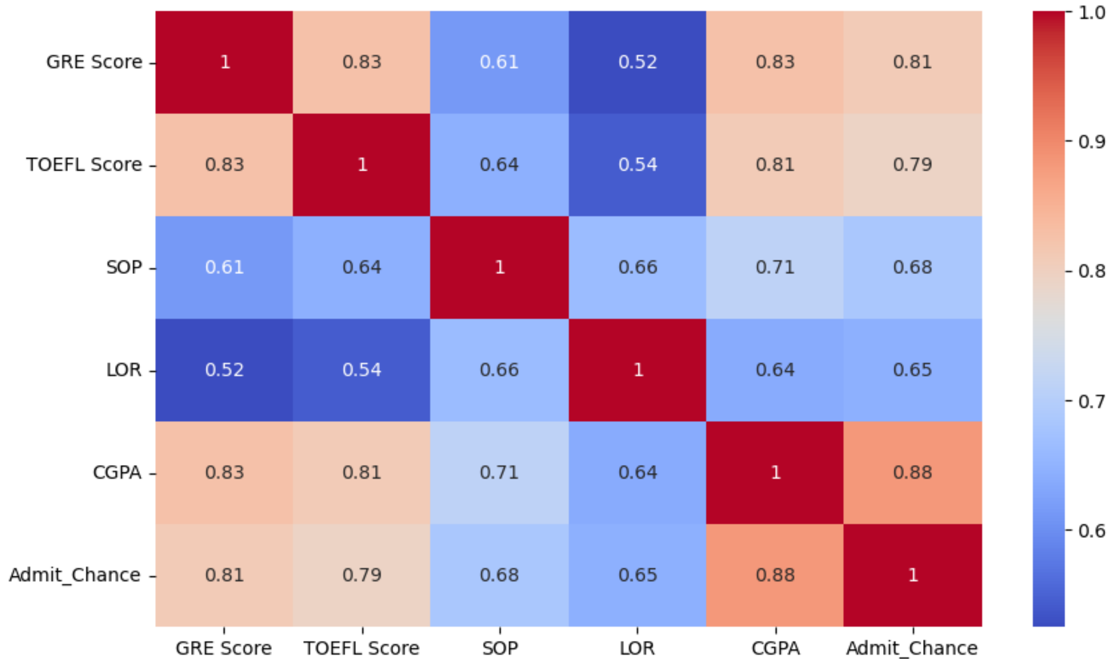
[139]:
```
sns.boxplot(data=data,x="Research",y="Admit_Chance")
```

[139]: <Axes: xlabel='Research', ylabel='Admit_Chance'>



1.1.2 Check the correlation among independent variables and how they interact with each other.

[154]:
```
plt.figure(figsize=(10,6))
sns.heatmap(corr_matrix,cmap="coolwarm",annot=True)
plt.show()
```



[155]:
```
corr_matrix
```

[155]:
```
              GRE Score  TOEFL Score       SOP       LOR      CGPA  \
GRE Score      1.000000     0.827200  0.613498  0.524679  0.825878
TOEFL Score    0.827200     1.000000  0.644410  0.541563  0.810574
SOP            0.613498     0.644410  1.000000  0.663707  0.712154
LOR            0.524679     0.541563  0.663707  1.000000  0.637469
CGPA           0.825878     0.810574  0.712154  0.637469  1.000000
Admit_Chance   0.810351     0.792228  0.684137  0.645365  0.882413

              Admit_Chance
GRE Score         0.810351
TOEFL Score       0.792228
SOP               0.684137
LOR               0.645365
CGPA              0.882413
Admit_Chance      1.000000
```

Insights:

- We checked the correlation between the target and independent variables, as saw there is linearity relationship with target variable. This will be further analysed on the linearity assumption of linear regression.
- We alos see some correlation between the independent features but not greater than 0.90. So lets not drop any feature now. Will see later on during assumption check of multicollinearity and based on VIF threshold will drop the features if required.

1.1.3 Preparing the data for modeling:

[156]:
```
data.head()
```

[156]:
```
   GRE Score  TOEFL Score  University Rating  SOP  LOR  CGPA  Research  \
0        337          118                  4  4.5  4.5  9.65         1
1        324          107                  4  4.0  4.5  8.87         1
2        316          104                  3  3.0  3.5  8.00         1
3        322          110                  3  3.5  2.5  8.67         1
4        314          103                  2  2.0  3.0  8.21         0

   Admit_Chance
0          0.92
1          0.76
2          0.72
3          0.80
4          0.65
```

1.1.4 Encoding & Transformation of Categorical Variables

[157]:
```
data["University Rating"].value_counts()
```

[157]:
```
University Rating
3    162
2    126
4    105
5     73
1     34
Name: count, dtype: int64
```

[159]:
```
data["Research"].value_counts()
```

[159]:
```
Research
1    280
0    220
Name: count, dtype: int64
```

1.1.5 Insights

- Univerisity Rating - No need of any encoding. Since university rating is an ordinal variable, algorithm will interpret the ordinal relationships. So no need of encoding.
- Research - This is a binary categorical variable, where 0 = No Research and 1 = Has Research experience. No need for one-hot encoding or label encoding here either since it's binary and already numeric.

1.1.6 TRAIN-TEST-SPLIT

[174]:
```python
from sklearn.model_selection import train_test_split

X = data.drop(["Admit_Chance"],axis=1)
y = data["Admit_Chance"]

#splitting the data for training and testing
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.
 ↪2,random_state=10)
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

[174]:
```
((400, 7), (400,), (100, 7), (100,))
```

[211]:
```python
X_train
```

[211]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
305	321	109	3	3.5	3.5	8.80	1
107	338	117	4	3.5	4.5	9.46	1
350	318	107	3	3.0	3.5	8.27	1
334	312	107	4	4.5	4.0	8.65	1
142	331	115	5	4.0	3.5	9.44	1
..	…	…	…	…	…	…	…
320	317	106	3	4.0	3.5	8.50	1
15	314	105	3	3.5	2.5	8.30	0
484	317	106	3	3.5	3.0	7.89	1
125	300	100	3	2.0	3.0	8.66	1
265	313	102	3	2.5	2.5	8.68	0

[400 rows x 7 columns]

[212]:
```python
X_test
```

[212]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
151	332	116	5	5.0	5.0	9.28	1
424	325	114	5	4.0	5.0	9.46	1
154	326	108	3	3.0	3.5	8.89	0
190	324	111	5	4.5	4.0	9.16	1
131	303	105	5	5.0	4.5	8.65	0
..	…	…	…	…	…	…	…
50	313	98	3	2.5	4.5	8.30	1
264	325	110	2	3.0	2.5	8.76	1
34	331	112	5	4.0	5.0	9.80	1
78	296	95	2	3.0	2.0	7.54	1
223	308	109	2	3.0	4.0	8.45	0

[100 rows x 7 columns]

1.1.7 NORMALIZING THE DATA

[192]:
```python
from sklearn.preprocessing import MinMaxScaler

#initiate the minmaxscaler
scaler = MinMaxScaler()

#Fit and tranform the training data
X_train_scaled = pd.DataFrame(scaler.fit_transform(X_train),columns = X_train.
 ↪columns)

#tranform the testing data - tranform the test data using the same scaling␣
 ↪parameters from the training data, ensuring consistency
X_test_scaled = pd.DataFrame(scaler.transform(X_test),columns=X_test.columns)


#This process ensures that the test data is treated in the same way as the␣
 ↪training data, without introducing bias or data leakage.
#By following this process, your model will generalize better to unseen data.
```

[193]:
```python
X_train_scaled
```

[193]:
```
     GRE Score  TOEFL Score  University Rating    SOP    LOR      CGPA  \
0         0.62     0.592593               0.50  0.625  0.625  0.641026
1         0.96     0.888889               0.75  0.625  0.875  0.852564
2         0.56     0.518519               0.50  0.500  0.625  0.471154
3         0.44     0.518519               0.75  0.875  0.750  0.592949
4         0.82     0.814815               1.00  0.750  0.625  0.846154
..         …          …                …     …      …      …
395       0.54     0.481481               0.50  0.750  0.625  0.544872
396       0.48     0.444444               0.50  0.625  0.375  0.480769
397       0.54     0.481481               0.50  0.625  0.500  0.349359
398       0.20     0.259259               0.50  0.250  0.500  0.596154
399       0.46     0.333333               0.50  0.375  0.375  0.602564

     Research
0         1.0
1         1.0
2         1.0
3         1.0
4         1.0
..        …
395       1.0
396       0.0
397       1.0
398       1.0
399       0.0
```

[400 rows x 7 columns]

[194]:
```
X_train_scaled.describe()
```

[194]:
```
        GRE Score  TOEFL Score  University Rating         SOP         LOR  \
count  400.000000   400.000000         400.000000  400.000000  400.000000
mean     0.524250     0.523426           0.516875    0.583438    0.608750
std      0.229924     0.225096           0.282969    0.248490    0.232572
min      0.000000     0.000000           0.000000    0.000000    0.000000
25%      0.340000     0.370370           0.250000    0.375000    0.500000
50%      0.520000     0.518519           0.500000    0.625000    0.625000
75%      0.700000     0.703704           0.750000    0.750000    0.750000
max      1.000000     1.000000           1.000000    1.000000    1.000000

             CGPA    Research
count  400.000000  400.000000
mean     0.564647    0.567500
std      0.198243    0.496043
min      0.000000    0.000000
25%      0.416667    0.000000
50%      0.562500    1.000000
75%      0.717949    1.000000
max      1.000000    1.000000
```

[196]:
```
X_test_scaled
```

[196]:
```
    GRE Score  TOEFL Score  University Rating    SOP    LOR      CGPA  \
0        0.84     0.851852               1.00  1.000  1.000  0.794872
1        0.70     0.777778               1.00  0.750  1.000  0.852564
2        0.72     0.555556               0.50  0.500  0.625  0.669872
3        0.68     0.666667               1.00  0.875  0.750  0.756410
4        0.26     0.444444               1.00  1.000  0.875  0.592949
..        …          …                 …     …      …      …
95       0.46     0.185185               0.50  0.375  0.875  0.480769
96       0.70     0.629630               0.25  0.500  0.375  0.628205
97       0.82     0.703704               1.00  0.750  1.000  0.961538
98       0.12     0.074074               0.25  0.500  0.250  0.237179
99       0.36     0.592593               0.25  0.500  0.750  0.528846

    Research
0        1.0
1        1.0
2        0.0
3        1.0
4        0.0
..       …
```

```
95      1.0
96      1.0
97      1.0
98      1.0
99      0.0

[100 rows x 7 columns]
```

[197]:
```
X_test_scaled.describe()
```

[197]:
```
        GRE Score  TOEFL Score  University Rating       SOP        LOR  \
count  100.000000   100.000000         100.000000  100.00000  100.000000
mean     0.550200     0.534444           0.575000    0.63375    0.670000
std      0.208845     0.226805           0.294092    0.24182    0.220851
min      0.100000    -0.037037           0.000000    0.12500    0.125000
25%      0.420000     0.370370           0.250000    0.50000    0.500000
50%      0.580000     0.518519           0.500000    0.62500    0.750000
75%      0.700000     0.675926           0.750000    0.78125    0.875000
max      1.000000     1.000000           1.000000    1.00000    1.000000

             CGPA    Research
count  100.000000  100.000000
mean     0.588269    0.530000
std      0.174828    0.501614
min      0.201923    0.000000
25%      0.470353    0.000000
50%      0.592949    1.000000
75%      0.714744    1.000000
max      0.961538    1.000000
```

1.1.8 FITTING THE MODEL FOR TRAINING USING TRAINING DATA AND PREDICTING USING TEST DATA

Linear Regression Model Using sklearn library

[198]:
```
from sklearn.linear_model import LinearRegression

model = LinearRegression()

model.fit(X_train_scaled,y_train)
```

[198]:
```
LinearRegression()
```

[200]:
```
np.round(model.coef_,2) #7 features with 7 weights
```

[200]:
```
array([0.12, 0.05, 0.02, 0.02, 0.06, 0.36, 0.02])
```

[202]:
```
np.round(model.intercept_,2) #y-intercept or the baseline prediction
```

[202]: 0.35

[227]:
```
#predicting the model for both training and testing data

y_predict_train = model.predict(X_train_scaled)
y_predict_test = model.predict(X_test_scaled)
```

[228]:
```
len(y_predict_train), len(y_predict_test)
```

[228]: (400, 100)

[230]:
```
y_predict_train[:10]
```

[230]:
```
array([0.7721161 , 0.92432841, 0.69650938, 0.74818464, 0.89475943,
       0.52198513, 0.78677509, 0.47048339, 0.7811877 , 0.82146918])
```

[231]:
```
y_predict_test[:10]
```

[231]:
```
array([0.90887315, 0.90353254, 0.76777555, 0.84866602, 0.71681359,
       0.75063014, 0.65146102, 0.84594493, 0.62033211, 0.74704486])
```

1.1.9 Model Evaluation on sklearn linear regression (Calculation of R-square, Adjusted R-Square & MSE)

[232]:
```
from sklearn.metrics import r2_score
```

[238]:
```
r2_train = r2_score(y_train,y_predict_train)
r2_test = r2_score(y_test,y_predict_test)

print(f'R² for training data is {np.round(r2_train,2)}')
print(f'R² for testing data is {np.round(r2_test,2)}')
```

```
R² for training data is 0.83
R² for testing data is 0.8
```

insights on evaluation of R^2:

- R^2 for the training set gives you an idea of how well the model fits the training data.
- R^2 for the test set tells you how well the model generalizes to unseen data.

[240]:
```
#calculation of Adjusted R-Square

n = X_train_scaled.shape[0]  # number of data points
p = X_train_scaled.shape[1]  # number of features (predictors)

adjusted_r2_train = 1 - (1 - r2_train) * (n - 1) / (n - p - 1)
adjusted_r2_test = 1 - (1 - r2_test) * (n - 1) / (n - p - 1)

print(f'Adjusted R² for Training Data: {np.round(adjusted_r2_train,2)}')
```

```
print(f'Adjusted R² for Testing Data: {np.round(adjusted_r2_test,2)}')
```

```
Adjusted R² for Training Data: 0.82
Adjusted R² for Testing Data: 0.79
```

[340]:
```
from sklearn.metrics import mean_squared_error

#calculate MSE for training data
train_mse = mean_squared_error(y_train,y_predict_train)
print("MSE for training data",np.round(train_mse,3))

#calculate MSE for testing data
test_mse = mean_squared_error(y_test,y_predict_test)
print("MSE for test data",np.round(test_mse,3))
```

```
MSE for training data 0.004
MSE for test data 0.004
```

Insights:

- R^2 for both training and testing are nearly the same with no much difference indicating the model is being trained with good data and predicated or generalized with unseen data in good as well.
- Adjusted R^2 for both training and testing data also not much deviating showing the model patterns are captured with relevant features.

1.1.10 Lets build model using OLS statsmodel.

[213]:
```
import statsmodels.api as sm

X_train_scaled_const = sm.add_constant(X_train_scaled)
X_train_scaled_const
```

[213]:
```
     const  GRE Score  TOEFL Score  University Rating    SOP    LOR      CGPA  \
0      1.0       0.62     0.592593               0.50  0.625  0.625  0.641026
1      1.0       0.96     0.888889               0.75  0.625  0.875  0.852564
2      1.0       0.56     0.518519               0.50  0.500  0.625  0.471154
3      1.0       0.44     0.518519               0.75  0.875  0.750  0.592949
4      1.0       0.82     0.814815               1.00  0.750  0.625  0.846154
..      …          …           …                 …      …      …       …
395    1.0       0.54     0.481481               0.50  0.750  0.625  0.544872
396    1.0       0.48     0.444444               0.50  0.625  0.375  0.480769
397    1.0       0.54     0.481481               0.50  0.625  0.500  0.349359
398    1.0       0.20     0.259259               0.50  0.250  0.500  0.596154
399    1.0       0.46     0.333333               0.50  0.375  0.375  0.602564

     Research
0         1.0
```

```
1         1.0
2         1.0
3         1.0
4         1.0
..        ...
395       1.0
396       0.0
397       1.0
398       1.0
399       0.0

[400 rows x 8 columns]
```

[219]:
```python
y_train = y_train.reset_index(drop=True)
y_train
```

[219]:
```
0      0.74
1      0.91
2      0.74
3      0.73
4      0.92
       ...
395    0.75
396    0.54
397    0.73
398    0.64
399    0.71
Name: Admit_Chance, Length: 400, dtype: float64
```

[222]:
```python
model2 = sm.OLS(y_train,X_train_scaled_const)
results = model2.fit()
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:           Admit_Chance   R-squared:                       0.826
Model:                            OLS   Adj. R-squared:                  0.822
Method:                 Least Squares   F-statistic:                     265.1
Date:                Wed, 02 Oct 2024   Prob (F-statistic):         2.29e-144
Time:                        16:11:51   Log-Likelihood:                 559.41
No. Observations:                 400   AIC:                            -1103.
Df Residuals:                     392   BIC:                            -1071.
Df Model:                           7
Covariance Type:            nonrobust
================================================================================
=====
                        coef    std err          t      P>|t|      [0.025
0.975]
```

```
------------------------------------------------------------------------------------
-----
const                   0.3511      0.010     35.437      0.000       0.332
0.371
GRE Score               0.1192      0.028      4.295      0.000       0.065
0.174
TOEFL Score             0.0491      0.027      1.826      0.069      -0.004
0.102
University Rating       0.0205      0.017      1.205      0.229      -0.013
0.054
SOP                     0.0240      0.021      1.172      0.242      -0.016
0.064
LOR                     0.0602      0.018      3.272      0.001       0.024
0.096
CGPA                    0.3639      0.034     10.828      0.000       0.298
0.430
Research                0.0219      0.007      2.927      0.004       0.007
0.037
==============================================================================
Omnibus:                       87.655   Durbin-Watson:                   1.963
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              194.225
Skew:                          -1.122   Prob(JB):                     6.68e-43
Kurtosis:                       5.572   Cond. No.                         22.7
==============================================================================

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

insights:

- Summary results are self explanaitory on the co-efficients of every features.
- The R^2 and Adjusted R^2 in the summary table are based on the training data.
- Not much varation between the R^2 and adjusted R^2 indicating the model are trained with relevant features and not with noisy data.
- The R^2 and Adjusted R^2 for test data we have already calculated in the sklearn regression model.

1.2 Testing the Assumptions of Linear Regression

Multicollinearity Check using VIF score

[243]:
```python
from statsmodels.stats.outliers_influence import variance_inflation_factor
X = X_train_scaled_const
vif = pd.DataFrame()
vif["Features"] = X.columns



vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
```

```
#for i in range(X.shape[1]) - This for loop will run for all the features. X.
 ↪shape[1] is the total no of columns we have in X dataframe
#[variance_inflation_factor(X.values, i) - this will calculate the VIF for all␣
 ↪values, in the range of i (say here from 0 to 7)
#x.values returns an array form with all feature values without showing the␣
 ↪feature names.


vif["VIF"] = round(vif["VIF"],2) #rounding of the VIF values to 2 decimal places
vif = vif.sort_values("VIF",ascending=False) #sorting the VIF values in␣
 ↪descending order
vif
```

[243]:

	Features	VIF
0	const	10.77
6	CGPA	4.86
1	GRE Score	4.46
2	TOEFL Score	4.02
4	SOP	2.85
3	University Rating	2.53
5	LOR	2.00
7	Research	1.51

insights:

- No variables are above VIF threshold 5 except constant which is added to build the OLS regression model. So no other variables need to be dropped and retrained. Will retain the constant variable as it is for now.
- If we had calculated the VIF from sklearn regression model then constant variable will not be part of this VIF table.

Linear relationship between independent & dependent variables.

[245]:

```
corr_matrix
```

[245]:

	GRE Score	TOEFL Score	SOP	LOR	CGPA	\
GRE Score	1.000000	0.827200	0.613498	0.524679	0.825878	
TOEFL Score	0.827200	1.000000	0.644410	0.541563	0.810574	
SOP	0.613498	0.644410	1.000000	0.663707	0.712154	
LOR	0.524679	0.541563	0.663707	1.000000	0.637469	
CGPA	0.825878	0.810574	0.712154	0.637469	1.000000	
Admit_Chance	0.810351	0.792228	0.684137	0.645365	0.882413	

	Admit_Chance
GRE Score	0.810351
TOEFL Score	0.792228
SOP	0.684137
LOR	0.645365

```
CGPA                 0.882413
Admit_Chance         1.000000
```

insights:

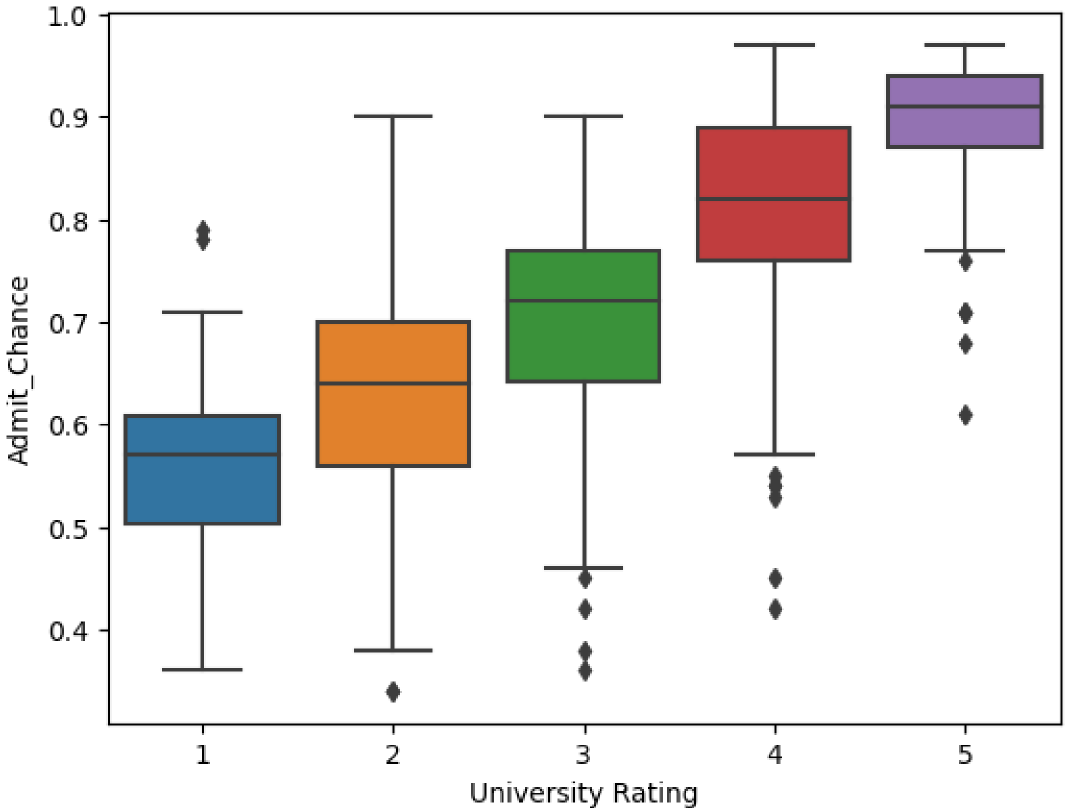
- As we saw earlier in our visuals of scatterplots and heatmap that all our numerical features have postive correlation with target variable.
- GRE Score have positive linear relationship with target with correlation score of 0.81
- TOEFL Score have positive linear relationship with target with correlation score of 0.79
- CGPA Score have positive linear relationship with target with correlation score of 0.88
- SOP & LOP have positive linear relationship with target with correlation scoreS of 0.68 and 0.64. Although its not strong correlation with target variable.
- For two categorical variables, University Rating and Research have plotted using boxplot showing the distribution of target variable for each category assesing how each category of the feature is associated with the target variable.
  - Boxplots shows clear increasing pattern in the median of target variable when university rating is high, as you see as rating increases, tha chance of admission also increases in a consistent manner, this suggest a linear relationship.Same applies for the Research. If research done, then chance of admission is high as well.
  - Lets conduct statistical test to prove the significance.

Hypothesis Testing for Categorical Variables to check the significance(University Rating and Research)

- University Rating with 5 catergories - One Way ANOVA test. - to check the relationship between the university rating and chance of admission

[247]:
```python
sns.boxplot(data=data,x="University Rating",y="Admit_Chance")
```

[247]:
```
<Axes: xlabel='University Rating', ylabel='Admit_Chance'>
```



[257]:
```
from scipy.stats import f_oneway

#Null Hypothesis H0 - There is no difference in the means of target variable␣
↪across categories
#Alternative Hypothesis H1 - There is significance difference in the means of␣
↪target variable atlease on one of the categories

alpha = 0.05
rating_1 = data.loc[data["University Rating"]==1]["Admit_Chance"]
rating_2 = data.loc[data["University Rating"]==2]["Admit_Chance"]
rating_3 = data.loc[data["University Rating"]==3]["Admit_Chance"]
rating_4 = data.loc[data["University Rating"]==4]["Admit_Chance"]
rating_5 = data.loc[data["University Rating"]==5]["Admit_Chance"]
```

[264]:
```
np.mean(rating_1),np.mean(rating_2),np.mean(rating_3),np.mean(rating_4),np.
↪mean(rating_5)
```

[264]:
```
(0.5620588235294117,
 0.6261111111111112,
 0.7029012345679012,
```

```
 0.8016190476190477,
 0.8880821917808219)
```

[258]:
```python
f_stat,p_val = f_oneway(rating_1,rating_2,rating_3,rating_4,rating_5)
f_stat,p_val
```

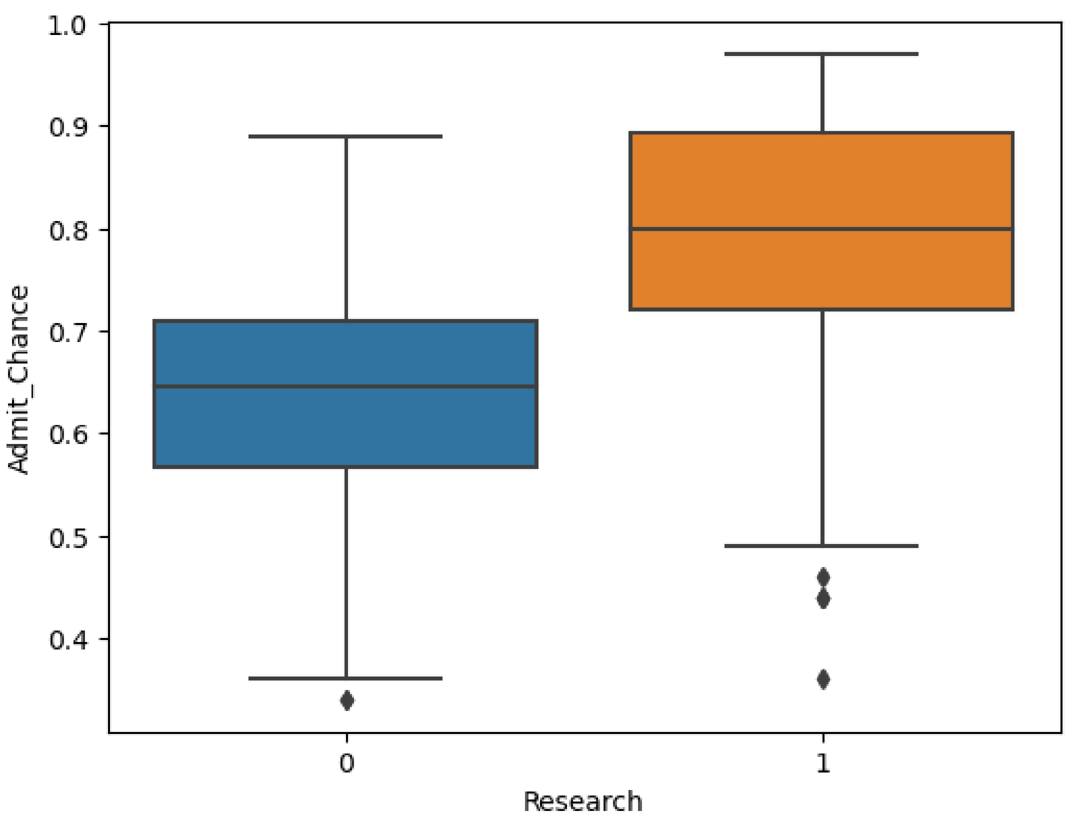
[258]: (114.00804341400004, 7.753395328023128e-69)

[265]:
```python
if p_val<alpha:
    print("Reject the null hypothesis: There is significant difference in␣
 ↪means")
    print("Concludes that University Rating have significant effect on chance␣
 ↪of admission")
else:
    print("Fail to reject the null hypothesis: There is no difference")
```

```
Reject the null hypothesis: There is significant difference in means
Concludes that University Rating have significant effect on chance of admission
```

- Research with 2 catergories - T-test. - to check the relationship between the Research and chance of admission

[267]:
```python
sns.boxplot(data=data,x="Research",y="Admit_Chance")
plt.show()
```



[268]:
```
data.head()
```

[268]:
```
   GRE Score  TOEFL Score  University Rating  SOP  LOR  CGPA  Research  \
0        337          118                  4  4.5  4.5  9.65         1
1        324          107                  4  4.0  4.5  8.87         1
2        316          104                  3  3.0  3.5  8.00         1
3        322          110                  3  3.5  2.5  8.67         1
4        314          103                  2  2.0  3.0  8.21         0

   Admit_Chance
0          0.92
1          0.76
2          0.72
3          0.80
4          0.65
```

[272]:
```
data.groupby("Research")["Admit_Chance"].mean()
#mean shows that those who research paper have higher mean or chance of␣
 ↪admission.
```

[272]:
```
Research
0    0.634909
1    0.789964
Name: Admit_Chance, dtype: float64
```

[271]:
```
res_0 = data.loc[data["Research"]==0]["Admit_Chance"]
res_1 = data.loc[data["Research"]==1]["Admit_Chance"]
```

[273]:
```
from scipy.stats import ttest_ind

#Null hypothesis H0: Both research papers means is same res_0 = res_1
#Alternative Hypothesis H1: Research paper of 1 mean is greater than res_0 i.e␣
 ↪res_1>res_0

alpha = 0.05

t_stat,p_val = ttest_ind(res_1,res_0,alternative="greater")
t_stat,p_val
```

[273]:
```
(14.538797385517404, 1.7977467729204891e-40)
```

[278]:
```
if p_val< alpha:
    print("Reject H0:Applicants those who have research paper i.e 1 have␣
 ↪greater chance of admmission")
```

```
    print("Concludes that Research have significant effect on chance of␣
↪admission")
else:
    print ('Fail to Reject H0')
```

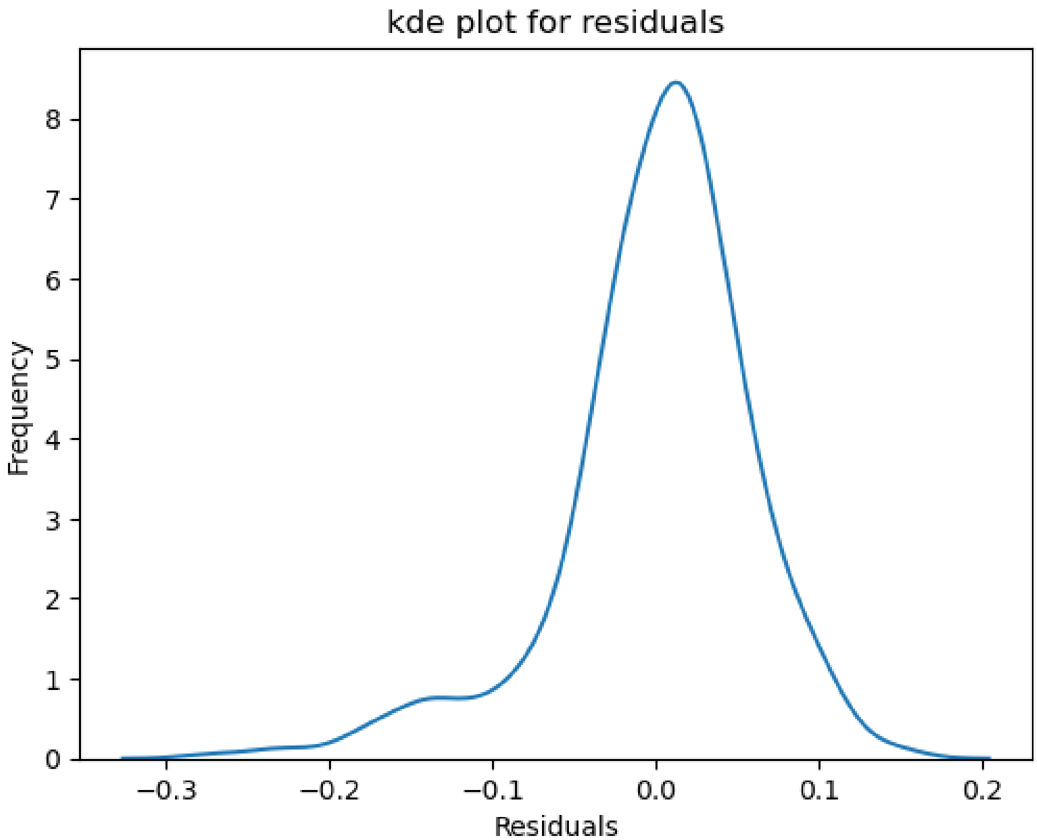
```
Reject H0:Applicants those who have research paper i.e 1 have greater chance of
admmission
Concludes that Research have significant effect on chance of admission
```

Normality of Residuals

[282]:
```
#lets predict the model fitted using OLS
#predict the model using the training data
y_predict = results.predict(X_train_scaled_const)
y_predict.head()
```

[282]:
```
0    0.772116
1    0.924328
2    0.696509
3    0.748185
4    0.894759
dtype: float64
```

[318]:
```
#calculate the errors using the target of training data
errors = y_train - y_predict

# Plot a histogram/kde of residuals to visually inspect normality
sns.kdeplot(errors)
plt.xlabel("Residuals")
plt.ylabel("Frequency")
plt.title("kde plot for residuals")
plt.show()
```

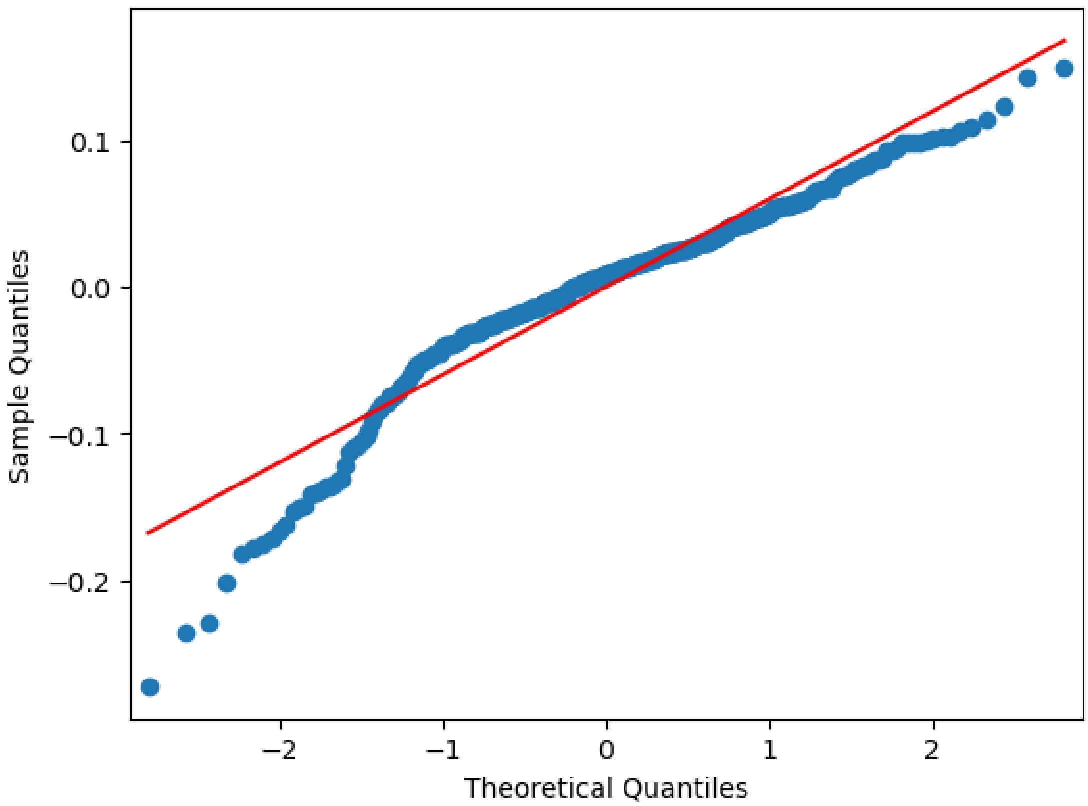


[319]:
```
from scipy.stats import shapiro

alpha = 0.05
test_stat, p_value = shapiro(errors)
test_stat,p_value

#test_stat is closer to 1 denotes a high level of normality of error␣
↪distribution
```

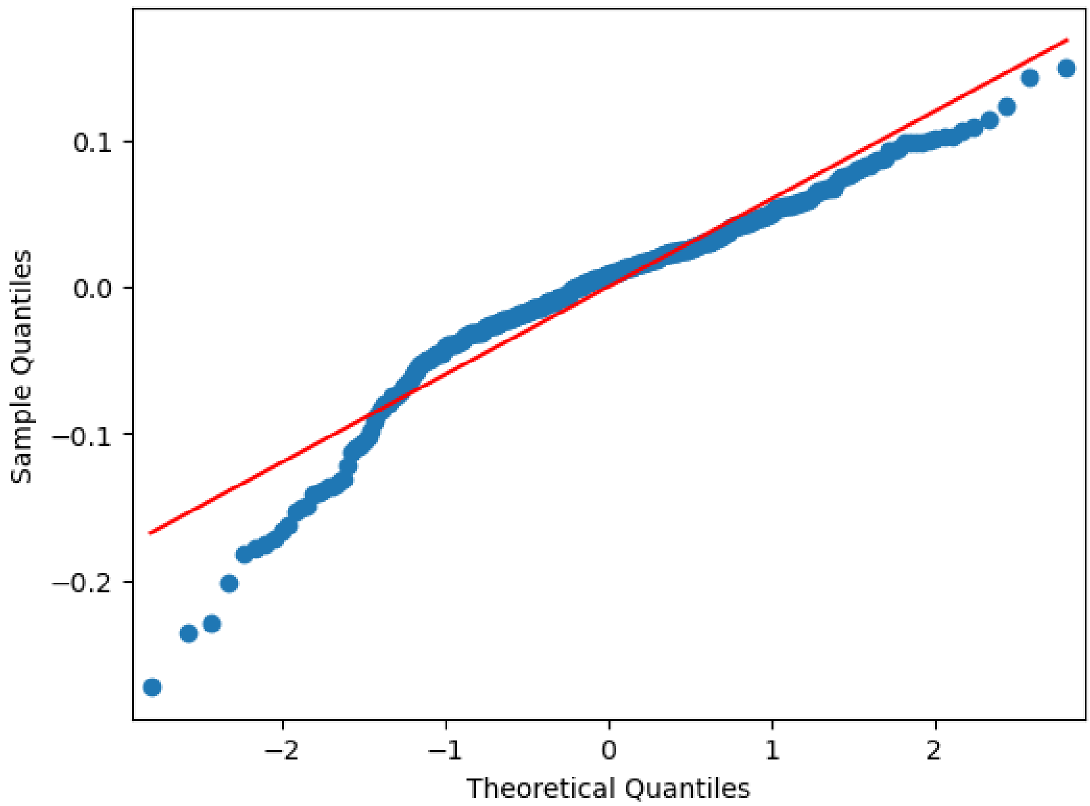
[319]: (0.9310511350631714, 1.2451858120987591e-12)

[320]:
```
import statsmodels.api as sm

sm.qqplot(errors,line='s')
```

[320]:

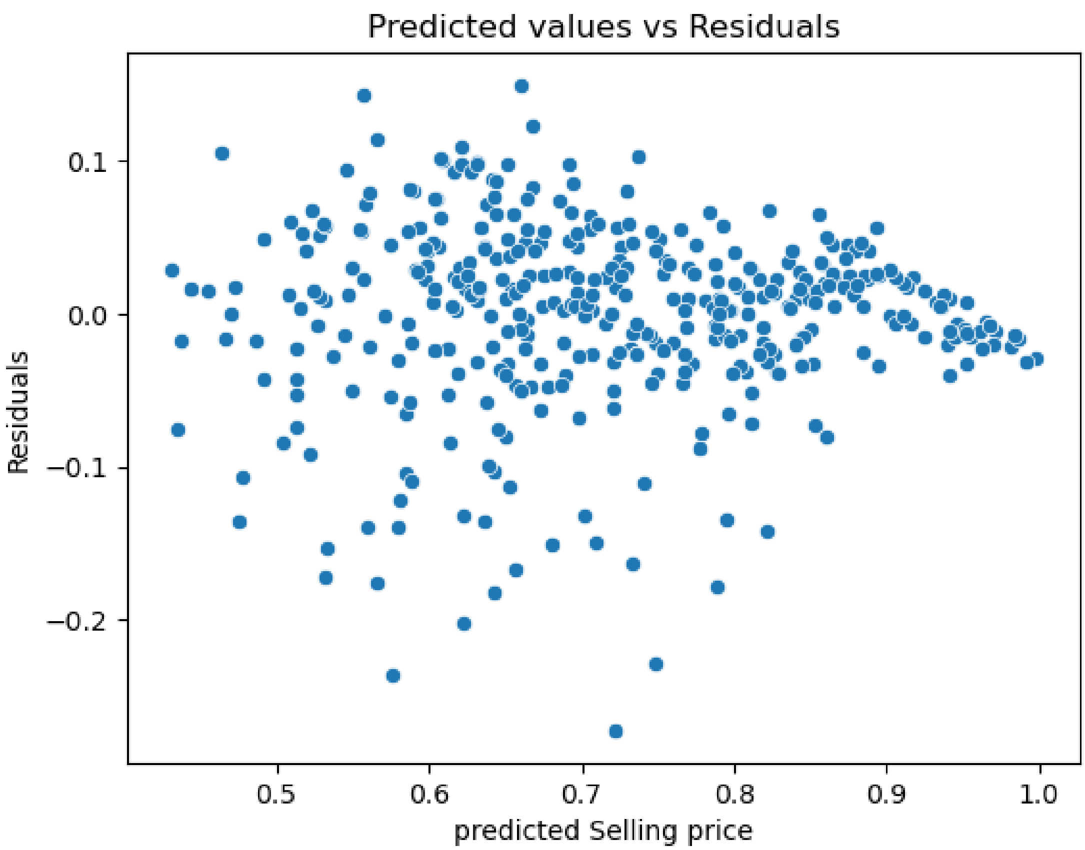


0.1
0.0
−0.1
−0.2
Sample Quantiles
−2
−1
0
1
2
Theoretical Quantiles



Test for Homoscedasticity

[321]:
```
sns.scatterplot(x=y_predict,y=errors)
plt.xlabel("predicted Selling price")
plt.ylabel("Residuals")
plt.title("Predicted values vs Residuals")
plt.show()
```



- Notice that as we go from left to right,the spread of errors is almost constant,no much variance or diverting. So we can assume that heteroskedasticity does not exist in our data. There are outliers present in the dataset
- We can also use Goldfeld-Quandt statistical Test to check homoskedacity.

[322]:
```
# Performing the Goldfeld-Quandt test to check for Homoscedasticity -
import statsmodels.api as sm
from statsmodels.stats.diagnostic import het_goldfeldquandt

alpha = 0.05
#Null Hypothesis (H ): The data is homoscedastic (i.e., the variance of the␣
↪residuals is constant).
#Alternative Hypothesis (H ): The data is heteroscedastic (i.e., the variance␣
↪of the residuals is not constant).

gq_test = het_goldfeldquandt(y_train, X_train_scaled)
test_stat = gq_test[0]
p_val = gq_test[1]
test_stat,p_val
```

[322]: (0.974982315321618, 0.5697583370711886)

[323]:
```
if p_val<alpha:
    print("Reject the null hypothesis")
    print("The variance of residuals is not constant (heteroscedasticity)")
else:
    print("Fail to reject the null hypothesis")
    print("The variance of residuals is constant (Homoscedasticity)")
```

```
Fail to reject the null hypothesis
The variance of residuals is constant (Homoscedasticity)
```

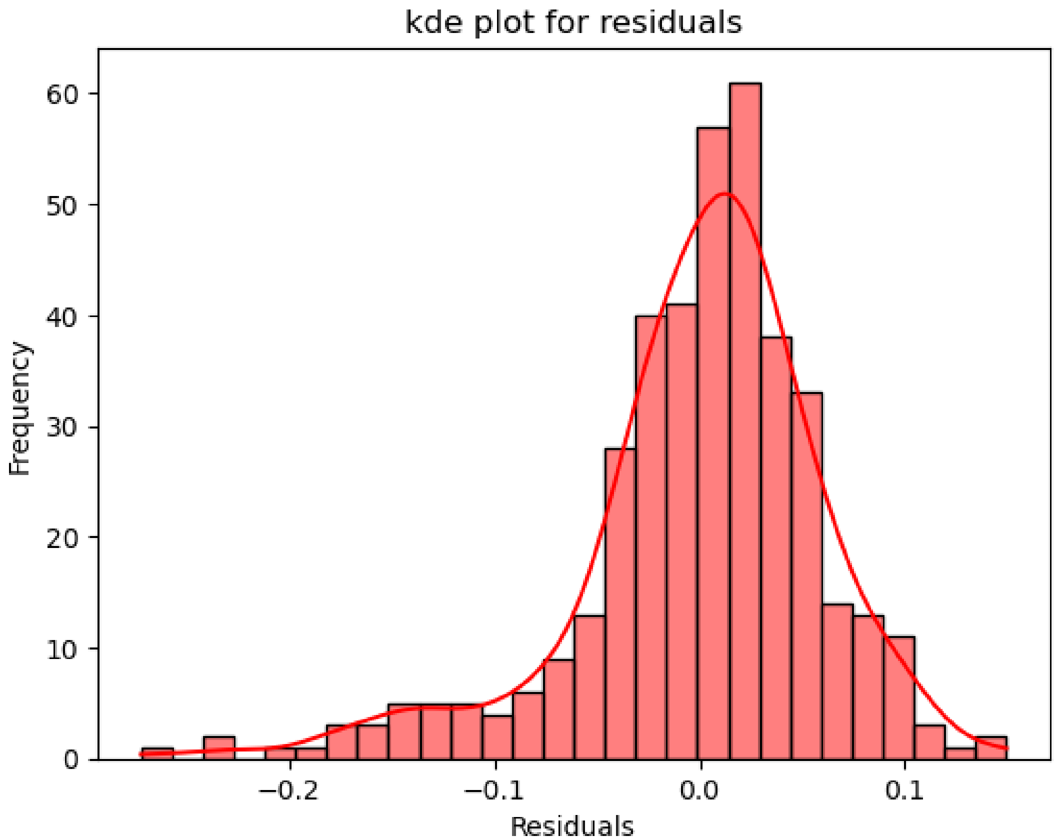
insight: test_stats is also close to 1, which typically suggests that the variance generally supports the assumption of homoscedasticity

Mean of residuals should be close to zero

[331]:
```
residuals = y_train - y_predict
print(f"Mean of residuals: {np.mean(residuals)}")
```

```
Mean of residuals: 6.605826996519682e-16
```

[335]:
```
# Plot a histogram/kde of residuals to visually inspect normality
sns.histplot(residuals,kde=True, color="r")
plt.xlabel("Residuals")
plt.ylabel("Frequency")
plt.title("kde plot for residuals")
plt.show()
```



insights :

- The mean of residuals we obtained, 6.605826996519682e-16, is essentially zero (close to 0) which meets the assumption.
- If mean of residuals is significantly non-zero, then the model is overestimating or underestimating the observed values.
- If the mean of residuals is close to zero then on average predections made by linear regression model are accurate, within the equal balance of overestimating and underestimating. This is the desired charecteristics for well-fitted regression model.

1.3 Lets see how L1 and L2 regularisation work

1.3.1 L1 - Lasso Regularisation

[428]:
```
from sklearn.linear_model import Lasso, Ridge

lasso_model = Lasso(alpha=0.01)  # Alpha is the regularization strength

# Fit the models to the training data
lasso_model.fit(X_train_scaled, y_train)
```

[428]: Lasso(alpha=0.01)

[429]:
```
lasso_model.coef_
```

[429]:
```
array([0.08962753, 0.01038033, 0.0473052 , 0.0017367 , 0.        ,
       0.1814805 , 0.04494164])
```

[430]:
```
#prediction
lasso_predict_train = lasso_model.predict(X_train_scaled)
lasso_predict_test = lasso_model.predict(X_test_scaled)
```

[431]:
```
lasso_predict_train[:10]
```

[431]:
```
array([0.76027213, 0.84403755, 0.72308006, 0.74690571, 0.84160083,
       0.60211905, 0.76278605, 0.55965349, 0.76729267, 0.77932455])
```

[432]:
```
lasso_predict_test[:10]
```

[432]:
```
array([0.83490532, 0.83162441, 0.72892671, 0.81144553, 0.69710559,
       0.7528502 , 0.66482375, 0.81648921, 0.64800995, 0.74262123])
```

Model Evaluation for L1 Lasso Regularization

[433]:
```
from sklearn.metrics import r2_score
```

[434]:
```
r2_lasso_train = r2_score(y_train,lasso_predict_train)
r2_lasso_test = r2_score(y_test,lasso_predict_test)

print(f'R² for training data is {np.round(r2_lasso_train,2)}')
print(f'R² for testing data is {np.round(r2_lasso_test,2)}')
```

```
R² for training data is 0.69
R² for testing data is 0.66
```

[435]:
```
#calculation of Adjusted R-Square

n = X_train_scaled.shape[0]  # number of data points
p = X_train_scaled.shape[1]  # number of features (predictors)

adjusted_r2_train = 1 - (1 - r2_lasso_train) * (n - 1) / (n - p - 1)
adjusted_r2_test = 1 - (1 - r2_lasso_test) * (n - 1) / (n - p - 1)

print(f'Adjusted R² for Training Data: {np.round(adjusted_r2_train,2)}')
print(f'Adjusted R² for Testing Data: {np.round(adjusted_r2_test,2)}')
```

```
Adjusted R² for Training Data: 0.68
Adjusted R² for Testing Data: 0.65
```

1.3.2 L2 - Ridge Regularisation

[436]:
```
from sklearn.linear_model import Ridge

ridge_model = Ridge(alpha=0.0001)  # Alpha is the regularization strength

# Fit the models to the training data
ridge_model.fit(X_train_scaled, y_train)
```

[436]:
```
Ridge(alpha=0.0001)
```

[437]:
```
lasso_model.coef_
```

[437]:
```
array([0.08962753, 0.01038033, 0.0473052 , 0.0017367 , 0.        ,
       0.1814805 , 0.04494164])
```

[438]:
```
#prediction
ridge_predict_train = ridge_model.predict(X_train_scaled)
ridge_predict_test = ridge_model.predict(X_test_scaled)
```

[439]:
```
ridge_predict_train[:10]
```

[439]:
```
array([0.77211589, 0.92432799, 0.69651033, 0.74818507, 0.89475884,
       0.52198435, 0.78677511, 0.47048338, 0.78118786, 0.82146773])
```

[440]:
```
ridge_predict_test[:10]
```

[440]:
```
array([0.9088738 , 0.90353192, 0.76777465, 0.84866591, 0.71681374,
       0.75062941, 0.65145998, 0.84594516, 0.62033145, 0.74704427])
```

Model Evaluation for L2 Ridge Regularization

[441]:
```
from sklearn.metrics import r2_score

r2_ridge_train = r2_score(y_train,ridge_predict_train)
r2_ridge_test = r2_score(y_test,ridge_predict_test)

print(f'R² for training data is {np.round(r2_ridge_train,2)}')
print(f'R² for testing data is {np.round(r2_ridge_test,2)}')
```

```
R² for training data is 0.83
R² for testing data is 0.8
```

[442]:
```
#calculation of Adjusted R-Square

n = X_train_scaled.shape[0]   # number of data points
p = X_train_scaled.shape[1]   # number of features (predictors)

adjusted_r2_train = 1 - (1 - r2_ridge_train) * (n - 1) / (n - p - 1)
```

```
adjusted_r2_test = 1 - (1 - r2_ridge_test) * (n - 1) / (n - p - 1)

print(f'Adjusted R² for Training Data: {np.round(adjusted_r2_train,2)}')
print(f'Adjusted R² for Testing Data: {np.round(adjusted_r2_test,2)}')
```

```
Adjusted R² for Training Data: 0.82
Adjusted R² for Testing Data: 0.79
```

1.4 Actionable Insights and Recommendations:

Significance of Predictor Variables:

- GRE Score: The GRE score is one of the most significant predictors of admission chances. Higher GRE scores consistently correspond to a higher probability of admission. Hence, applicants should be encouraged to improve their GRE scores as much as possible.
- TOEFL Score: TOEFL is another important factor, especially for international applicants.Jamboree could offer specialized TOEFL workshops or practice materials to boost students' language proficiency.
- CGPA: The model shows that CGPA is a critical factor in predicting admission chances. Encouraging students to maintain or improve their academic performance is crucial.
- SOP and LOR: While GRE and CGPA are important, subjective factors like SOP (Statement of Purpose) and LOR (Letters of Recommendation) also play a vital role in differentiating candidates with similar scores.
- University Rating and Research Experience: Research experience has been found to be significant for applicants aiming for top tier schools. Those with prior research experience, or those from highly rated universities, tend to have a stronger profile.

Additional Data Sources for Model Improvement:

- Extracurricular Activities: Including a variable for students such as leadership roles or relevant work experience, could enhance the model's ability to predict success in admissions.
- Personal Interviews: Some Ivy League schools also conduct interviews as part of the admissions process. Including data on interview performance could improve the accuracy of the model in predicting admission chances.

Model Implementation in the Real World:

- Jamboree could implement this model on their website, where students can input their scores, and other relevant details to assess their chances of admission. This would provide real time feedback and personalized insights to help students identify areas of improvement.
- A real-time dashboard could be developed where students can see how changes in their scores (GRE, TOEFL, etc.) impact their predicted chances of admission. This would encourage students to work on areas that maximize their chances.

Potential Business Benefits from Improving the Model:

- Personalized Student Experience: By using this model, Jamboree can offer a personalized admission roadmap for each student which could increase student satisfaction and retention based on the output the students get from the model.
- Attracting New Students: A predictive tool like this could be a valuable marketing tool to attract more students to Jamboree's platform offering the tailored services for students.

- Revenue Growth: As the model helps students improve their scores and profile, Jamboree can see an increase in demand for their various services leading to higher revenue.

```
[ ]:
```

```
[ ]:
```