Importing Liabraries and Dataset

In [4]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [5]:
```
!gdown "https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/839/origi
df = pd.read_csv("Jamboree_Admission.csv")
```

```
Downloading...
From: https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/839/origin
al/Jamboree_Admission.csv (https://d2beiqkhq929f0.cloudfront.net/public_assets/asse
ts/000/001/839/original/Jamboree_Admission.csv)
To: C:\Users\Satyam\Jamboree_Admission.csv

  0%|          | 0.00/16.2k [00:00<?, ?B/s]
100%|##########| 16.2k/16.2k [00:00<?, ?B/s]
```

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

Out[6]:

	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

Exploratory Data Analysis

In [7]:
```
df.shape
```

Out[7]:
```
(500, 9)
```

In [8]:
```
df.info()
```

```
&lt;class 'pandas.core.frame.DataFrame'&gt;
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 [9]:
```
df.describe()
```

Out[9]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
count	500.000000	500.000000	500.000000	500.000000	500.000000	500.00000	500.000000	500.000000
mean	250.500000	316.472000	107.192000	3.114000	3.374000	3.48400	8.576440	0.560000
std	144.481833	11.295148	6.081868	1.143512	0.991004	0.92545	0.604813	0.496884
min	1.000000	290.000000	92.000000	1.000000	1.000000	1.00000	6.800000	0.000000
25%	125.750000	308.000000	103.000000	2.000000	2.500000	3.00000	8.127500	0.000000
50%	250.500000	317.000000	107.000000	3.000000	3.500000	3.50000	8.560000	1.000000
75%	375.250000	325.000000	112.000000	4.000000	4.000000	4.00000	9.040000	1.000000
max	500.000000	340.000000	120.000000	5.000000	5.000000	5.00000	9.920000	1.000000

In [10]:
```
df.drop(columns="Serial No.",inplace=True)
```

In [11]:
```
df.shape
```

Out[11]:
```
(500, 8)
```

In [12]:
```
df.isna().sum()
```

Out[12]:
```
GRE Score            0
TOEFL Score          0
University Rating    0
SOP                  0
LOR                  0
CGPA                 0
Research             0
Chance of Admit      0
dtype: int64
```

In [13]:
```
df.nunique()
```

Out[13]:
```
GRE Score            49
TOEFL Score          29
University Rating     5
SOP                   9
LOR                   9
CGPA                184
Research              2
Chance of Admit      61
dtype: int64
```

In [14]:
```
df["GRE Score"].unique()
```

Out[14]:
```
array([337, 324, 316, 322, 314, 330, 321, 308, 302, 323, 325, 327, 328,
       307, 311, 317, 319, 318, 303, 312, 334, 336, 340, 298, 295, 310,
       300, 338, 331, 320, 299, 304, 313, 332, 326, 329, 339, 309, 315,
       301, 296, 294, 306, 305, 290, 335, 333, 297, 293], dtype=int64)
```

In [15]:
```
df.groupby(["GRE Score"])["Chance of Admit "].mean()
```

Out[15]:
```
GRE Score
290    0.460000
293    0.640000
294    0.475000
295    0.512000
296    0.522000
297    0.498333
298    0.507000
299    0.537000
300    0.595833
301    0.624545
302    0.558571
303    0.590000
304    0.570833
305    0.624545
306    0.642857
307    0.627000
308    0.655385
309    0.637778
310    0.667273
311    0.665000
312    0.685417
313    0.684167
314    0.696250
315    0.645385
316    0.661667
317    0.690000
318    0.702500
319    0.729167
320    0.790000
321    0.805294
322    0.784706
323    0.785385
324    0.813913
325    0.742667
326    0.822500
327    0.801176
328    0.848889
329    0.853000
330    0.906250
331    0.918889
332    0.893750
333    0.930000
334    0.916250
335    0.940000
336    0.948000
337    0.940000
338    0.920000
339    0.936667
340    0.947778
Name: Chance of Admit , dtype: float64
```

In [16]:
```python
df.groupby(["TOEFL Score"])["Chance of Admit "].mean()
```

Out[16]:
```
TOEFL Score
92     0.510000
93     0.460000
94     0.490000
95     0.516667
96     0.476667
97     0.502857
98     0.567000
99     0.566957
100    0.597083
101    0.610500
102    0.651667
103    0.678000
104    0.678966
105    0.648108
106    0.676429
107    0.708214
108    0.724211
109    0.752632
110    0.767045
111    0.804000
112    0.800000
113    0.860526
114    0.840556
115    0.879091
116    0.901875
117    0.927500
118    0.925000
119    0.930000
120    0.934444
Name: Chance of Admit , dtype: float64
```

In [17]:
```python
df.groupby("Research")["Chance of Admit "].mean()
```

Out[17]:
```
Research
0    0.634909
1    0.789964
Name: Chance of Admit , dtype: float64
```

In [18]:
```python
df.groupby("University Rating")["Chance of Admit "].mean()
```

Out[18]:
```
University Rating
1    0.562059
2    0.626111
3    0.702901
4    0.801619
5    0.888082
Name: Chance of Admit , dtype: float64
```

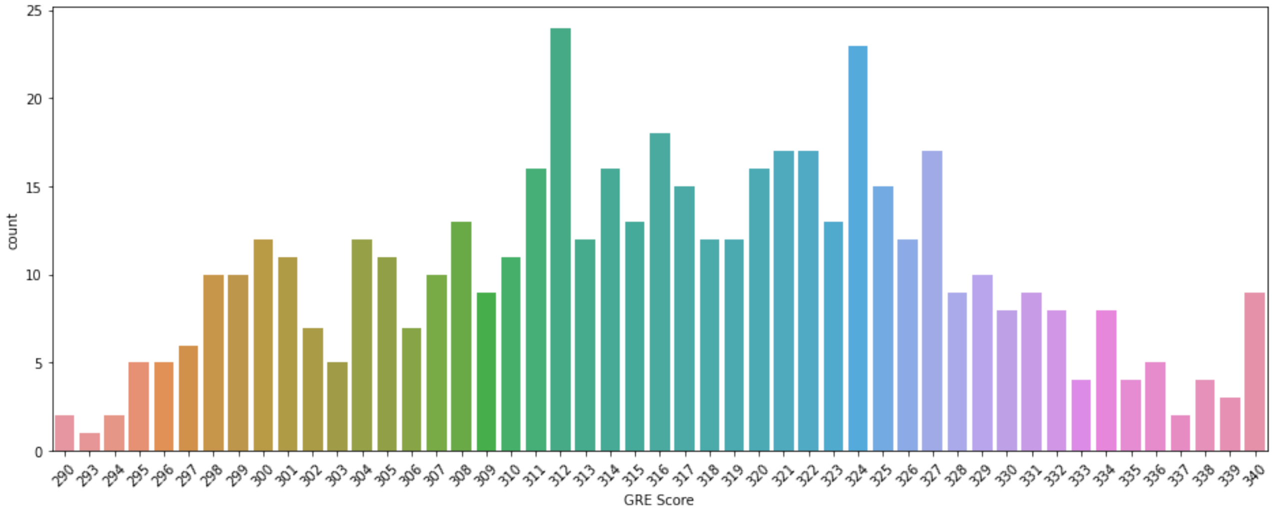
In [19]:
```
df.groupby("SOP")["Chance of Admit "].mean()
```

Out[19]:
```
SOP
1.0    0.538333
1.5    0.546400
2.0    0.589535
2.5    0.645312
3.0    0.678500
3.5    0.712045
4.0    0.782809
4.5    0.850000
5.0    0.885000
Name: Chance of Admit , dtype: float64
```

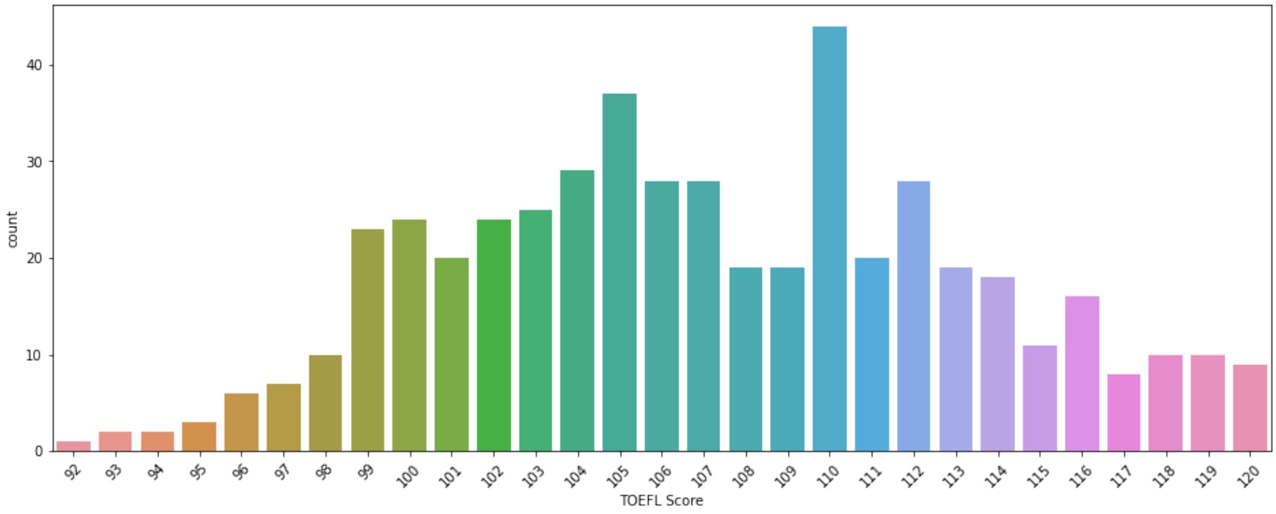
In [20]:
```
df.groupby("LOR ")["Chance of Admit "].mean()
```

Out[20]:
```
LOR
1.0    0.420000
1.5    0.550000
2.0    0.568261
2.5    0.640600
3.0    0.668485
3.5    0.723023
4.0    0.764149
4.5    0.831905
5.0    0.872600
Name: Chance of Admit , dtype: float64
```

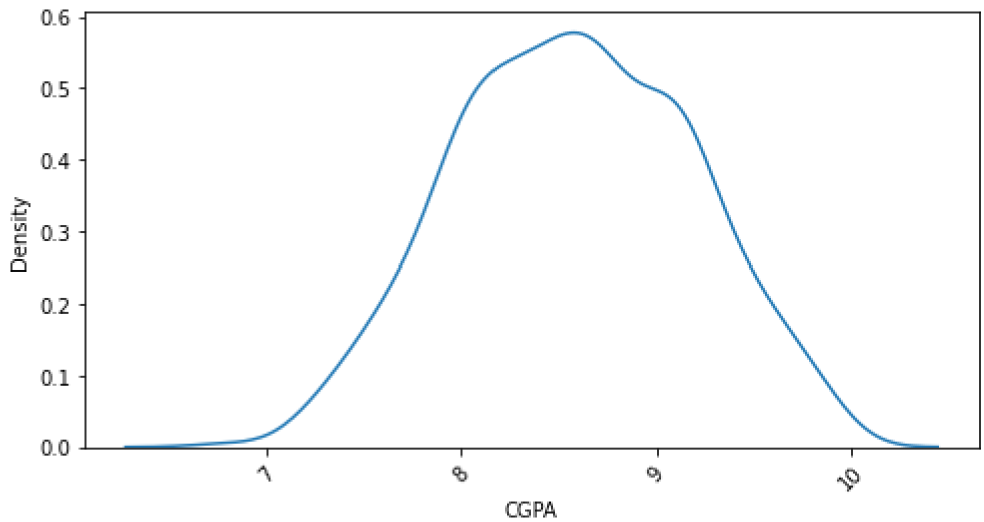
In [21]:
```
plt.figure(figsize=(16,6))
sns.countplot(x=df["GRE Score"].sort_values(ascending=False))
plt.xticks(rotation=45)
plt.show()
```



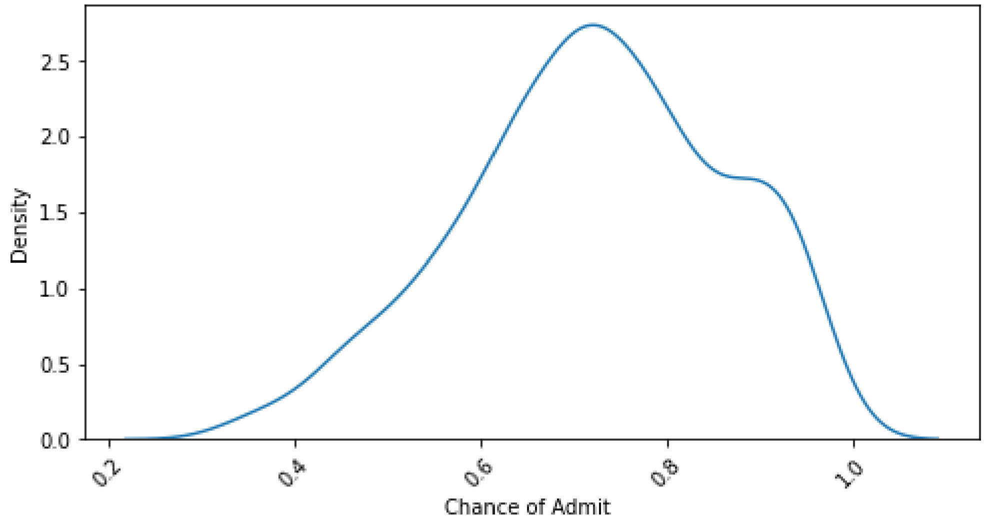
In [22]:
```
plt.figure(figsize=(16,6))
sns.countplot(x=df["TOEFL Score"].sort_values(ascending=False))
plt.xticks(rotation=45)
plt.show()
```



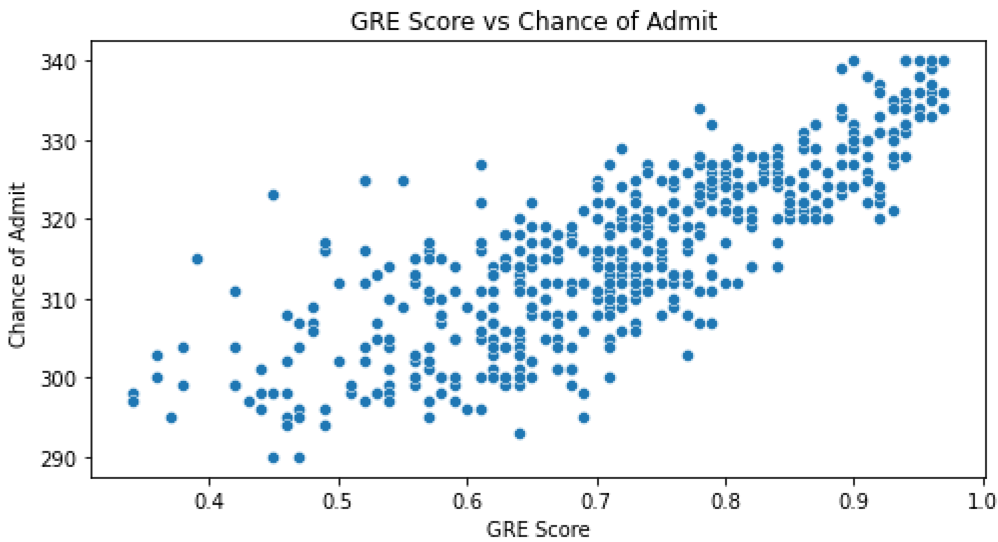
In [23]:
```
plt.figure(figsize=(8,4))
sns.kdeplot(x=df["CGPA"])
plt.xticks(rotation=45)
plt.show()
```



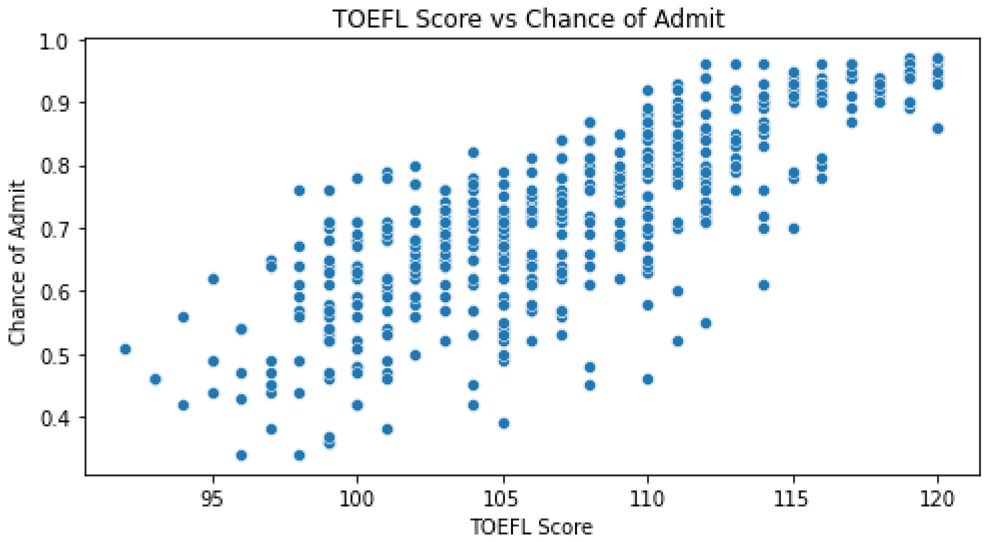
In [24]:
```
plt.figure(figsize=(8,4))
sns.kdeplot(x=df["Chance of Admit "])
plt.xticks(rotation=45)
plt.show()
```



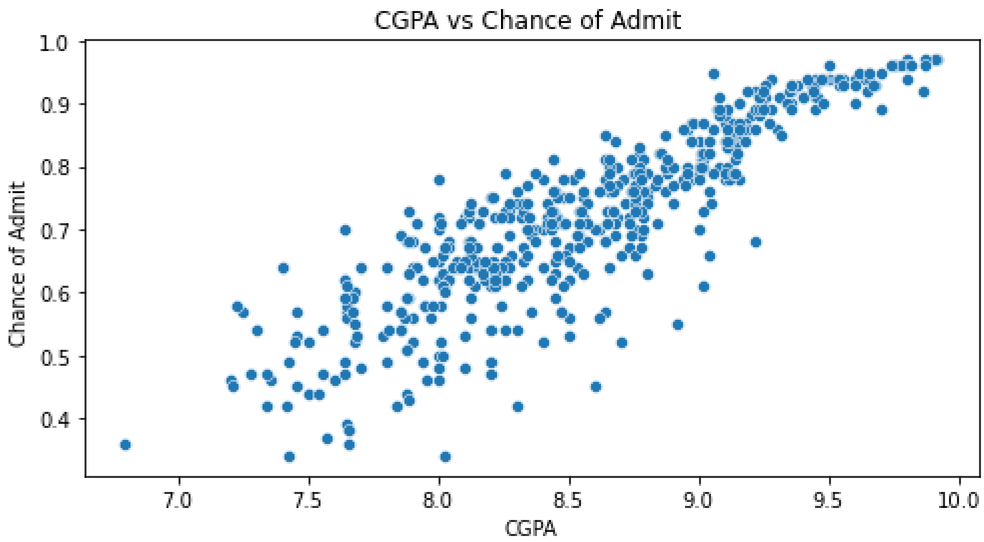
In [25]:
```
plt.figure(figsize=(8,4))
sns.scatterplot(x=df["Chance of Admit "],y=df["GRE Score"])
plt.xlabel("GRE Score")
plt.ylabel("Chance of Admit ")
plt.title("GRE Score vs Chance of Admit ")
plt.show()
```



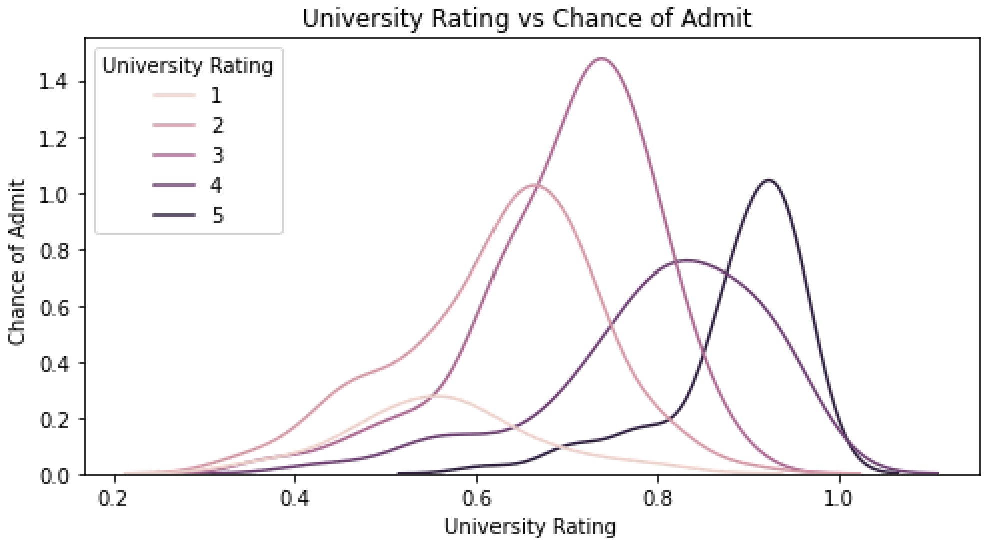
In [26]:
```
plt.figure(figsize=(8,4))
sns.scatterplot(y=df["Chance of Admit "],x=df["TOEFL Score"])
plt.xlabel("TOEFL Score")
plt.ylabel("Chance of Admit ")
plt.title("TOEFL Score vs Chance of Admit ")
plt.show()
```



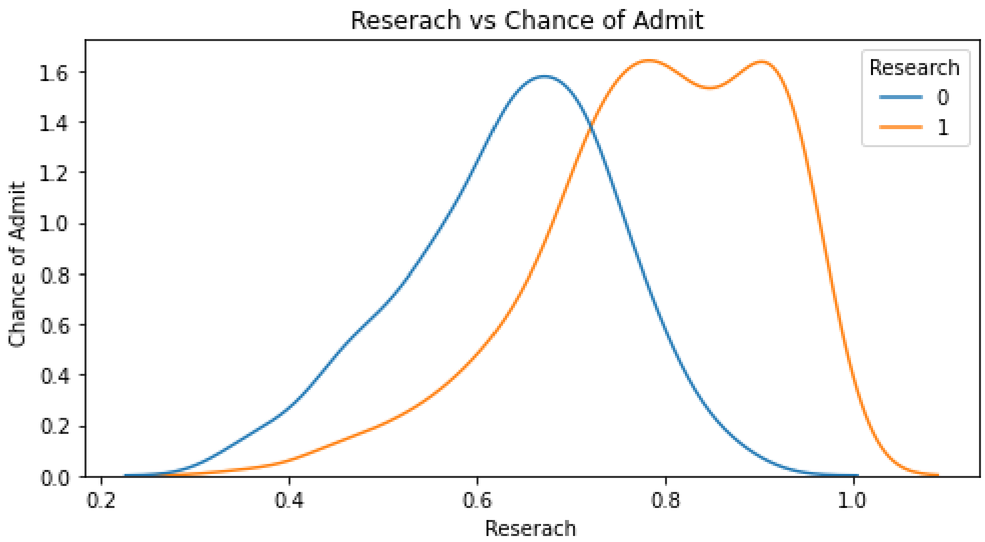
In [27]:
```
plt.figure(figsize=(8,4))
sns.scatterplot(y=df["Chance of Admit "],x=df["CGPA"])
plt.xlabel("CGPA")
plt.ylabel("Chance of Admit ")
plt.title("CGPA vs Chance of Admit ")
plt.show()
```



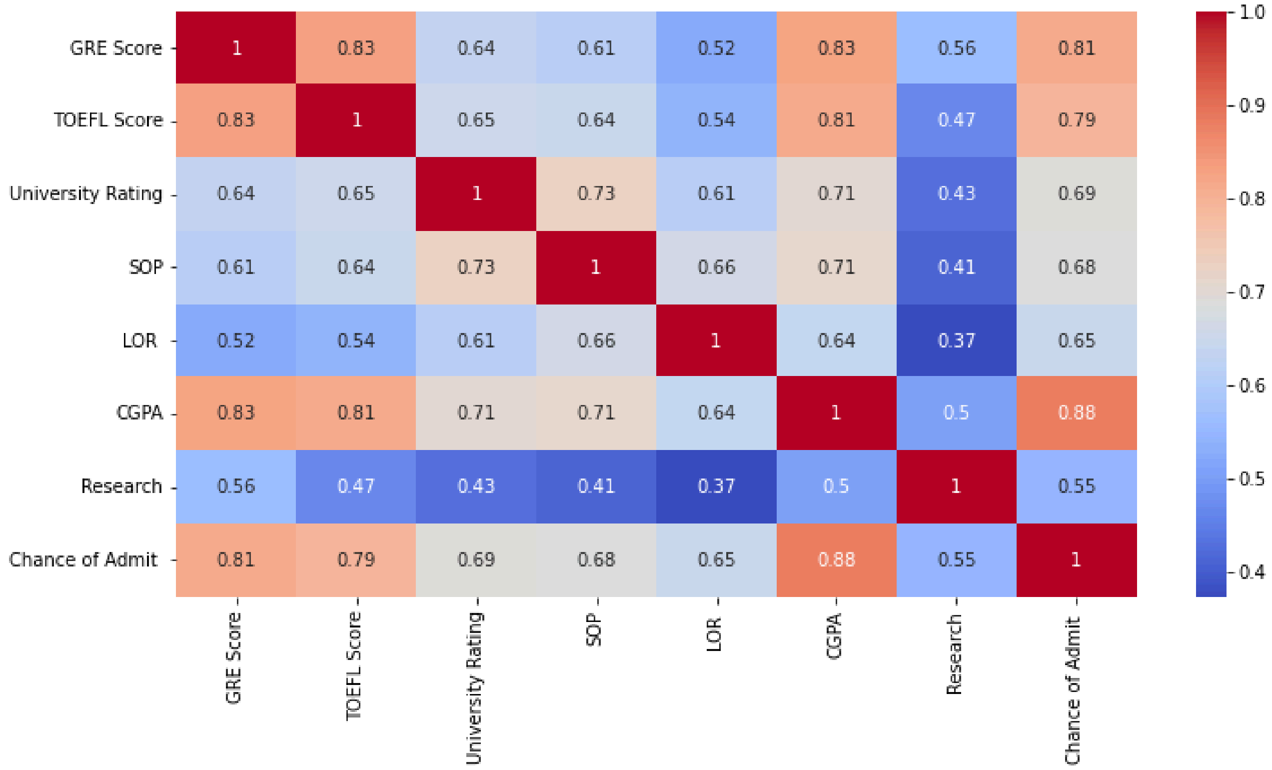
In [28]:
```
plt.figure(figsize=(8,4))
sns.kdeplot(x=df["Chance of Admit "],hue=df["University Rating"])
plt.xlabel("University Rating")
plt.ylabel("Chance of Admit ")
plt.title("University Rating vs Chance of Admit ")
plt.show()
```



In [29]:
```
plt.figure(figsize=(8,4))
sns.kdeplot(x=df["Chance of Admit "],hue=df["Research"])
plt.xlabel("Reserach")
plt.ylabel("Chance of Admit ")
plt.title("Reserach vs Chance of Admit ")
plt.show()
```



In [30]:
```
plt.figure(figsize=(12,6))
sns.heatmap(df.corr(),annot=True,cmap="coolwarm")
plt.show()
```



Outlier Detection and Treatment [...]

Model Building

In [36]:
```
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

In [37]:
```
scale = StandardScaler()
df_scale = pd.DataFrame(scale.fit_transform(df),columns=df.columns)
```

In [38]:
```
df_scale.head()
```

Out[38]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	1.819238	1.778865	0.775582	1.137360	1.098944	1.776806	0.886405	1.406107
1	0.667148	-0.031601	0.775582	0.632315	1.098944	0.485859	0.886405	0.271349
2	-0.041830	-0.525364	-0.099793	-0.377773	0.017306	-0.954043	0.886405	-0.012340
3	0.489904	0.462163	-0.099793	0.127271	-1.064332	0.154847	0.886405	0.555039
4	-0.219074	-0.689952	-0.975168	-1.387862	-0.523513	-0.606480	-1.128152	-0.508797

In [39]:
```
x= df_scale.drop(columns="Chance of Admit ")
y = df_scale["Chance of Admit "]
```

In [40]:
```
x.shape,y.shape
```

Out[40]: ((500, 7), (500,))

In [41]:
```
x
```

Out[41]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
0	1.819238	1.778865	0.775582	1.137360	1.098944	1.776806	0.886405
1	0.667148	-0.031601	0.775582	0.632315	1.098944	0.485859	0.886405
2	-0.041830	-0.525364	-0.099793	-0.377773	0.017306	-0.954043	0.886405
3	0.489904	0.462163	-0.099793	0.127271	-1.064332	0.154847	0.886405
4	-0.219074	-0.689952	-0.975168	-1.387862	-0.523513	-0.606480	-1.128152
...	...	...	...	...	...	...	...
495	1.376126	0.132987	1.650957	1.137360	0.558125	0.734118	0.886405
496	1.819238	1.614278	1.650957	1.642404	1.639763	2.140919	0.886405
497	1.198882	2.108041	1.650957	1.137360	1.639763	1.627851	0.886405
498	-0.396319	-0.689952	0.775582	0.632315	1.639763	-0.242367	-1.128152
499	0.933015	0.955926	0.775582	1.137360	1.098944	0.767220	-1.128152

500 rows × 7 columns

In [42]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_state=2)
```

In [43]:
```
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

Out[43]: ((400, 7), (100, 7), (400,), (100,))

In [44]:
```
x_sm= sm.add_constant(x_train)
sm_model= sm.OLS(y_train,x_sm).fit()
print(sm_model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:         Chance of Admit   R-squared:                       0.829
Model:                             OLS   Adj. R-squared:                  0.826
Method:                  Least Squares   F-statistic:                     272.1
Date:                 Fri, 28 Jun 2024   Prob (F-statistic):          3.33e-146
Time:                         13:01:32   Log-Likelihood:                -210.19
No. Observations:                  400   AIC:                             436.4
Df Residuals:                      392   BIC:                             468.3
Df Model:                            7
Covariance Type:             nonrobust
================================================================================
==
                        coef    std err          t      P>|t|      [0.025      0.97
5]
--------------------------------------------------------------------------------
--
const                 0.0087      0.021      0.419      0.676      -0.032       0.0
49
GRE Score             0.1708      0.044      3.893      0.000       0.085       0.2
57
TOEFL Score           0.1272      0.042      3.024      0.003       0.044       0.2
10
University Rating     0.0392      0.033      1.185      0.237      -0.026       0.1
04
SOP                   0.0147      0.034      0.428      0.669      -0.053       0.0
82
LOR                   0.1220      0.030      4.131      0.000       0.064       0.1
80
CGPA                  0.4858      0.046     10.633      0.000       0.396       0.5
76
Research              0.0870      0.025      3.476      0.001       0.038       0.1
36
==============================================================================
Omnibus:                       94.166   Durbin-Watson:                   1.943
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              231.309
Skew:                          -1.158   Prob(JB):                     5.92e-51
Kurtosis:                       5.918   Cond. No.                         5.53
==============================================================================

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

In [45]:
```
from sklearn.linear_model import LinearRegression
```

In [46]:
```
lr = LinearRegression()
model = lr.fit(x_train,y_train)
weight = model.coef_
bias = model.intercept_
print(weight)
print(bias)
```

```
[0.17078857 0.12715241 0.03923295 0.01471374 0.12196046 0.48577536
 0.08700309]
0.008653833200591151
```

In [47]:
```
dw = np.zeros_like(weight)
dw
```

Out[47]:
```
array([0., 0., 0., 0., 0., 0., 0.])
```

In [48]:
```
def optimization(x_train, y_train, weight, bias, iteration=10000, learning_rate=0.05
    for i in range(iteration):
        dw = np.zeros_like(weight)
        dw0 = 0
        x_train_array = np.array(x_train)

        y_pred = np.dot(x_train_array, weight) + bias

        for j in range(x_train.shape[1]):
            dw[j] = (-2/y_train.shape[0]) * np.sum((y_train - y_pred) * x_train_arra
            dw0 = (-2/y_train.shape[0]) * np.sum(y_train - y_pred)

        weight = weight - learning_rate * dw
        bias = bias - learning_rate * dw0

        print("Iteration:", i+1)
        print("Weight:", weight)
        print("Bias:", bias)
        print("dw:", dw)
        print("dw0:", dw0)
        print("Mean Squared Error:", np.mean((y_train - y_pred)**2))
        print(f"r2score : {1-(np.sum((y_train - y_pred)**2)/np.sum((y_train - y.mean
        print("-" * 50)

    return weight, bias
```

In [49]:
```python
optimization(x_train, y_train, weight, bias)
```

```
------------------------------------------------
Iteration: 5557
Weight: [0.17078857 0.12715241 0.03923295 0.01471374 0.12196046 0.48577536
 0.08700309]
Bias: 0.008653833200591113
dw: [-7.88258347e-17  1.99840144e-17  6.88338275e-17 -1.66533454e-17
  4.44089210e-18  1.15463195e-16 -2.22044605e-17]
dw0: 1.1102230246251566e-17
Mean Squared Error: 0.167470436798737
r2score : 0.8293240202443011
------------------------------------------------
Iteration: 5558
Weight: [0.17078857 0.12715241 0.03923295 0.01471374 0.12196046 0.48577536
 0.08700309]
Bias: 0.008653833200591113
dw: [-7.88258347e-17  1.99840144e-17  6.88338275e-17 -1.66533454e-17
  4.44089210e-18  1.15463195e-16 -2.22044605e-17]
dw0: 1.1102230246251566e-17
Mean Squared Error: 0.167470436798737
```

In [50]:
```python
dw = np.zeros_like(weight)
dw
```

Out[50]:
```
array([0., 0., 0., 0., 0., 0., 0.])
```

In [51]:
```python
np.array(x_train)
```

Out[51]:
```
array([[-0.0418297 , -0.68995225, -0.97516761, ...,  1.09894429,
         0.27070162, -1.12815215],
       [-0.83942999, -0.36077656, -0.97516761, ...,  1.09894429,
        -0.75543561,  0.88640526],
       [ 0.66714832,  0.79133837,  0.77558214, ..., -1.06433187,
        -0.78853681,  0.88640526],
       ...,
       [-1.45978576, -2.00665503, -0.97516761, ..., -2.14596996,
        -0.5899296 ,  0.88640526],
       [-0.21907421, -0.36077656, -0.09979274, ..., -1.06433187,
        -0.4575248 , -1.12815215],
       [-2.08014153, -1.67747933, -0.97516761, ...,  0.55812525,
        -1.28505482,  0.88640526]])
```

In [52]:
```python
x_train.shape
```

Out[52]:
```
(400, 7)
```

In [53]:
```python
y_train.shape
```

Out[53]:
```
(400,)
```

In [54]:
```python
# y_train-model.predict(y_train)
```

In [55]:
```
(y_train-model.predict(x_train))
```

Out[55]:
```
428   -0.247584
490    0.020705
53     0.160450
336   -0.052289
154    0.238680
         ...
22     0.071908
72     0.251773
493    0.289241
15    -0.762052
168    0.517925
Name: Chance of Admit , Length: 400, dtype: float64
```

In [56]:
```
(-2/y_train.shape[0])*np.sum((y_train-model.predict(x_train))*x_train.T)
```

Out[56]:
```
428   -0.003532
490    0.000189
53    -0.001524
336   -0.000426
154    0.000110
         ...
22    -0.003530
72    -0.010440
493    0.009645
15    -0.012202
168    0.015437
Length: 400, dtype: float64
```

In [57]:
```
model.predict(x_train)
```

```
Out[57]: array([ 2.24765719e-02, -3.87657330e-01, -1.72790335e-01,  3.99486947e-02,
         3.16358711e-01, -1.08393712e-01,  8.46490731e-02, -1.57891549e-01,
        -8.66910018e-01, -4.56715198e-01, -9.24188187e-01,  8.00334090e-01,
        -1.32059490e+00,  3.19104236e-01,  1.18265839e+00, -2.57295025e-01,
         7.10003961e-01, -9.06092019e-01, -2.10468719e-01,  3.69177321e-01,
         1.32779864e+00,  5.18929413e-01, -2.10692670e+00, -1.43763936e-01,
        -8.06317868e-01,  9.66446001e-01,  1.31388556e-01,  7.63809604e-01,
         7.30179294e-01, -1.44587421e+00,  1.58269941e+00, -1.14488817e+00,
         8.15398592e-02,  5.23490153e-01, -1.90467279e+00, -1.39910698e+00,
         1.69100047e+00, -1.07173534e+00,  1.32422365e+00, -4.35664183e-01,
         1.16528270e+00, -8.67553219e-01,  1.85048036e-01,  1.14076091e-01,
         1.25040064e+00, -4.22818864e-01,  6.97225718e-01, -8.01371078e-01,
        -4.93863496e-01, -1.50695295e+00,  4.51543844e-01,  8.97279017e-01,
         7.53793473e-01, -6.33407568e-01,  4.13386551e-01, -2.89795250e-01,
        -1.64701404e+00,  1.32223861e+00,  1.23954629e+00,  1.55432261e+00,
        -1.27328978e-01, -2.45759257e-01,  1.21703136e+00, -1.35579508e+00,
         1.16969290e-01,  1.00534165e+00, -1.84250179e-03,  1.18171484e+00,
         1.71581021e-01, -2.79265339e-01, -2.06488814e-01, -1.20359815e-01,
        -5.91926028e-01, -1.49877518e+00, -4.04699981e-01,  2.50730800e-01,
        -5.04142512e-01, -1.37480109e+00,  9.12982560e-01,  1.32966439e+00,
         1.36581396e+00, -8.15590460e-01, -1.02936464e-02, -5.41663658e-01,
        -1.04013512e-01,  4.70695904e-01,  1.00082895e+00, -1.10609706e+00,
        -1.71817656e-01, -7.12293475e-01,  3.71953055e-01,  1.46021301e-01,
         1.43628520e+00, -5.19119266e-02,  1.42497259e+00,  1.53502751e+00,
        -4.17877252e-01,  9.88196066e-01,  6.58219569e-01,  8.91642375e-01,
         6.23643981e-01,  9.84709263e-01, -5.45409668e-01, -3.42449989e-01,
        -1.52283294e-02,  6.85652294e-01, -9.82874417e-01, -6.07408569e-01,
        -6.87211295e-01, -3.11770943e-01, -1.52868786e-01,  7.28116559e-01,
        -2.05967732e+00,  1.72448024e+00,  1.29003117e+00,  9.94115736e-01,
        -1.65482424e+00, -1.52240844e+00, -5.48959904e-01, -4.93201126e-01,
         5.82142300e-01, -8.43821697e-01,  6.71779440e-01, -2.59207528e-01,
         9.32982253e-01,  1.35970952e+00,  1.07117872e-01, -9.38387680e-01,
         8.39111741e-01,  3.97591968e-01,  2.47572257e-01, -2.39215419e-01,
        -5.73983850e-01,  9.95772726e-01,  8.45481347e-01, -3.78229174e-01,
        -1.83521120e+00,  1.19368269e-01, -1.78718040e+00,  3.63681668e-01,
        -1.55029038e+00, -1.45344978e+00,  1.72889694e+00,  4.09607080e-01,
         1.19265708e+00, -5.74996442e-01, -7.51102646e-01,  2.98790709e-01,
        -4.17389993e-01,  1.02153190e+00,  6.64132865e-01, -1.28168960e+00,
        -9.36974652e-01, -8.72043621e-01, -1.46637847e+00, -7.26734204e-01,
        -4.26633803e-01, -5.01889192e-01,  1.88442718e-01, -2.25099542e-02,
         8.98158810e-01, -9.47445331e-01,  3.83207744e-01, -1.09098403e+00,
         4.97798296e-01,  1.15316291e-01, -6.31783703e-01, -5.86489141e-01,
         6.45452925e-01, -6.49476273e-01,  5.48878039e-02,  8.18895501e-01,
         4.40470883e-01, -7.59101933e-01,  1.36547866e+00,  1.59805057e+00,
        -2.87378714e-01, -6.94550365e-01, -5.31744914e-01,  9.38317273e-01,
         1.57030030e+00,  1.72245518e+00, -4.24093341e-01,  2.11248664e-01,
        -5.20269884e-01, -1.02929478e+00,  1.59372062e-01, -9.88770063e-01,
        -3.65865481e-01, -1.02356539e+00,  1.12412728e-01,  1.04215747e+00,
         1.76704458e-01, -6.85127330e-01,  1.04286165e-01,  1.08548443e+00,
         1.16324260e+00, -1.73970106e-02, -1.67811909e-02,  7.02490735e-01,
         7.35768974e-01,  4.81690410e-01, -5.49051884e-01, -4.16660519e-01,
        -2.00085548e+00, -1.16017610e+00,  5.16757262e-01,  4.36399642e-01,
        -5.29993798e-01, -9.35341688e-01, -3.10942355e-01,  1.48620343e-02,
        -1.42045087e+00, -3.60485320e-01,  4.44808471e-01,  6.79820378e-01,
        -2.07320850e-02, -7.30662907e-01,  8.06151581e-01, -7.83678076e-01,
        -1.38519765e+00,  8.43902698e-01, -4.81636448e-01,  6.30955790e-01,
         1.11334649e+00, -1.13867974e+00, -9.71371652e-01,  1.23336470e-01,
        -5.13647594e-01,  8.83480195e-01,  1.38196533e+00, -1.49998422e+00,
        -8.06797113e-01, -5.76729932e-01,  3.05679122e-01,  3.39862214e-01,
        -8.28663855e-01, -1.09003257e+00,  1.49357364e+00, -1.23539598e+00,
         4.63959752e-01,  8.97695916e-01, -1.31883487e+00,  8.41863662e-01,
         1.15885750e+00, -1.30696411e-01,  1.63083453e+00, -1.79984207e+00,
         1.65436971e+00,  7.86718591e-01,  8.93535292e-01, -1.55210139e+00,
```

```
       -1.48330475e+00,  1.27366460e+00, -1.10290595e+00,  1.13685915e+00,
        6.32445242e-01,  2.35052133e-01, -3.49694588e-01,  1.13093828e+00,
        4.75134124e-01,  5.03886701e-01,  3.42185839e-01, -7.15910074e-02,
        3.35677319e-01, -3.28396607e-01, -4.72579030e-01, -9.47836442e-02,
       -1.23685893e+00, -1.09358646e+00,  5.85565249e-01, -5.59189876e-01,
        1.66698550e+00, -1.44754106e+00, -1.80350823e+00, -7.20865939e-02,
        5.04605984e-01,  3.80293716e-01,  1.37030328e+00,  1.63665129e+00,
       -1.35256963e+00,  1.60509820e+00,  9.81493571e-01,  5.29175631e-01,
       -8.12323483e-01,  1.90536604e+00, -9.78372117e-01, -7.15821185e-01,
        1.99762825e+00, -7.98553473e-01, -7.30558678e-01,  1.34175991e+00,
       -3.76642123e-01, -4.87323773e-01,  8.77654465e-01, -4.46440050e-02,
       -8.27124386e-01,  6.70443126e-01,  5.89006802e-01, -2.43503552e-01,
       -6.34171116e-01, -7.58934074e-01,  9.55542507e-01, -3.35055701e-01,
       -7.23560828e-01,  1.70703952e+00,  7.45285095e-01, -5.15994624e-01,
        1.86019746e+00, -1.23932635e+00, -5.29410636e-01, -2.47569917e-01,
       -1.17193309e+00, -2.64191767e-02, -5.36653336e-01, -1.33655364e+00,
        1.66407415e+00, -1.16185854e-01,  7.33802154e-01,  7.98245267e-01,
        3.87841664e-01, -7.74369787e-01,  2.95365274e-01,  4.17481622e-01,
        8.86955103e-01, -9.69627440e-01,  1.17092337e+00, -1.20046068e-01,
        1.42615657e-02,  1.35193623e+00,  9.76836928e-01,  5.53196876e-01,
        8.95267012e-02,  5.60525459e-01, -4.70592519e-01, -4.55412009e-01,
       -3.65796739e-01,  8.09465217e-01, -8.85092210e-01, -1.42605668e+00,
       -6.96232231e-01, -2.58250891e-01, -4.15282790e-01, -4.10820822e-01,
        2.75898584e-01, -5.02185104e-01, -8.68679550e-01, -1.91168797e-03,
       -1.92928534e-01, -2.64276239e-01, -4.42458570e-01,  4.78371150e-02,
       -5.63340902e-01,  6.26669387e-01,  8.59655661e-03,  1.54017193e+00,
        8.94781544e-01,  4.49207192e-01, -7.30447645e-01, -8.57510027e-02,
       -7.21910555e-01, -4.45966571e-01,  9.33439838e-01, -8.79010622e-01,
       -1.66314133e-01,  3.80568053e-01, -2.03714196e+00,  1.54943608e+00,
       -1.52947293e+00, -4.55081587e-01, -7.40274187e-01, -6.24839849e-01,
        3.20353506e-03,  7.98135519e-01, -4.41158075e-01, -1.48692884e-01,
       -5.85264328e-01, -6.27590918e-01, -4.20085189e-01, -4.80609270e-01,
       -1.03366128e+00,  6.01878106e-01,  9.70012677e-01,  4.26424417e-01,
        1.55149624e+00,  1.77366357e+00, -8.84616801e-01, -1.14810149e+00,
        1.93471019e-01, -1.63312515e-01,  1.72479318e-01,  4.84685199e-01,
        4.14041859e-01,  1.66707472e-01, -1.10736328e-01,  1.47604408e+00,
        1.22525632e+00, -1.01080584e+00, -5.26891080e-01, -1.09764421e+00])
```

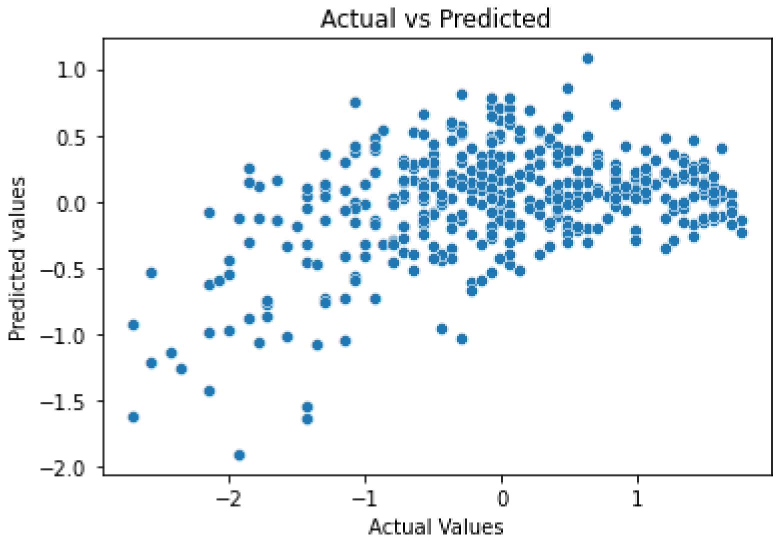
Checking Linearity Between Input and Output [...]

Checking Multicollinearity [...]

Normality of Residuals [...]

Checking for Non Heteroskedasticity

In [83]:
```
sns.scatterplot(x=y_train,y=error)
plt.xlabel("Actual Values")
plt.ylabel("Predicted values")
plt.title("Actual vs Predicted")
plt.show()
```



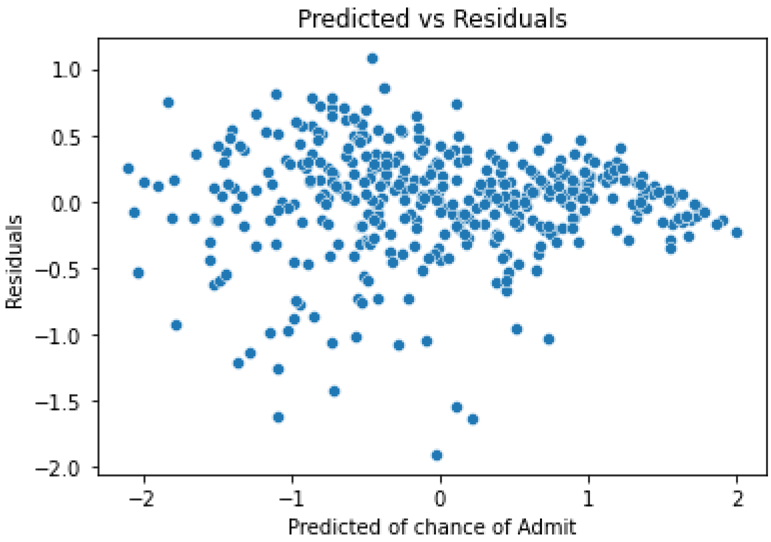
In [84]:
```
from statsmodels.compat import lzip
import statsmodels.stats.api as sms

# H0 : Constant Variance ( Non Hetroscadicicty)
# Ha : Not constant variance (Hetroscadicity)
alpha=0.05
F_stats,p_value,a=sms.het_goldfeldquandt(y_train,x_sm)
print(f"F_stats : {F_stats}")
print(f"p_value : {p_value}")
if p_value<alpha:
    print("Non constant Variance(Hetroscadicity)")
else:
    print("constant Variance(Non Hetroscadicity)")
```

```
F_stats : 1.0772994279987236
p_value : 0.3032327647981612
constant Variance(Non Hetroscadicity)
```

Auto Correlation

In [85]:
```
sns.scatterplot(x=y_hat,y=error)
plt.xlabel("Predicted of chance of Admit")
plt.ylabel("Residuals")
plt.title("Predicted vs Residuals")
plt.show()
```



Model performance evaluation

In [86]:
```
sm_model.rsquared
```

Out[86]: 0.829322723369172

In [87]:
```
sm_model.rsquared_adj
```

Out[87]: 0.8262749148579073

In [88]:
```
x_test_sm = sm.add_constant(x_test)
y_hat_test = sm_model.predict(x_test_sm)
MSE_train = 1/len(x_train)*np.sum((y_train-y_hat)**2)
MSE_test = 1/len(x_test)*np.sum((y_test-y_hat_test)**2)
```

In [89]:
```
MSE_test
```

Out[89]: 0.2227924252559508

In [90]:
```
MSE_train
```

Out[90]: 0.1674704367987369

In [91]:
```
MAE_train = 1/len(x_train)*np.sum(abs(y_train-y_hat))
```

In [92]:
```
MAE_test = 1/len(x_test)*np.sum(abs(y_test-y_hat_test))
```

In [93]:
```
MAE_train
```
Out[93]:
```
0.2932732079345014
```
In [94]:
```
MAE_test
```
Out[94]:
```
0.33546698609764797
```

Ridge and Lasso Linear_model

In [95]:
```
from sklearn.linear_model import Lasso,Ridge
from sklearn.metrics import mean_squared_error, mean_absolute_error
```
In [96]:
```
Lasso_model = Lasso()
Ridge_model = Ridge()
```
In [97]:
```
Lasso_model.fit(x_train,y_train)
```
Out[97]:
```
Lasso()
```
In [98]:
```
Ridge_model.fit(x_train,y_train)
```
Out[98]:
```
Ridge()
```

In [99]:
```python
Lasso_prediction=Lasso_model.predict(x_test)
Ridge_prediction = Ridge_model.predict(x_test)


Lasso_prediction_train=Lasso_model.predict(x_train)
Ridge_prediction_train = Ridge_model.predict(x_train)


print(f"train MSE for L1 : {mean_squared_error(y_train,Lasso_prediction_train)}")
print(f"test MSE for L1 : {mean_squared_error(y_test,Lasso_prediction)}")


print("-"*50)

print(f"train MSE for L2 : {mean_squared_error(y_train,Ridge_prediction_train)}")
print(f"test MSE for L2 : {mean_squared_error(y_test,Ridge_prediction)}")

print("-"*50)

print(f"train MAE for L1 : {mean_absolute_error(y_train,Lasso_prediction_train)}")
print(f"test MAE for L1 : {mean_absolute_error(y_test,Lasso_prediction)}")


print("-"*50)

print(f"train MAE for L2 : {mean_absolute_error(y_train,Ridge_prediction_train)}")
print(f"test MAE for L2 : {mean_absolute_error(y_test,Ridge_prediction)}")
```

```
train MSE for L1 : 0.9812110909232078
test MSE for L1 : 1.075192914788361
--------------------------------------------------
train MSE for L2 : 0.1674747155109731
test MSE for L2 : 0.22290602536422022
--------------------------------------------------
train MAE for L1 : 0.7958733499646117
test MAE for L1 : 0.8561396523802266
--------------------------------------------------
train MAE for L2 : 0.2932769009487325
test MAE for L2 : 0.33545910974132803
```

Insights and Recomondation

Insight Based On EDA

1- After Doing EDA I observed that the GRE Score, TOEFL Score, SOP, LOR and CGPA are positively corralated with Chance OF Admit.

2-After Doing EDA I observed that the Chance of getting admission in abroad is increasing with University Rating.

3-Student who has contribute to any Reserach has more chance of getting admission as compared to Non Researcher Student.

Insight Based on Stats Model.

1-Model which i have bulid using the given data has a R^2 =0.829 and adj_R^2 = 0.826.

2- Predictors and Target are linearly related with each other

3-No multicollinearity Present in predictors because every feature has VIF Score<5 so I will keep all the feature for the prediction.

4- Acual Targe values are not following normal distribution beacuse of that residuals are not following normal distribution.

5-After Performing goldfeldquandt statical test i observed that constant variance(NON Hetroscadicity) in errors.

6-There are no Pattern involves in residuals and Prediction.

7-Model has MSE(train) = 0.167 and MSE(test) = 0.222.

8-Model has MAE(train) = 0.293 and MAE(test) = 0.335.

Insight Based on Lasso and Ridge Linear Model.

1-MSE(train) and MSE(test) for Lasso linear Model is 0.981 and 1.075 respectively.

2-MAE(train) and MAE(test) for Lasso linear Model is 0.795 and 0.865 respectively

3-MSE(train) and MSE(test) for Ridge linear Model is 0.167 and 0.222 respectively.

4-MAE(train) and MAE(test) for Ridge linear Model is 0.293 and 0.335 respectively.

Recomondations-

1-There might be other features not explicitly mentioned, such as extracurricular activities, work experience, specific academic achievements, or demographic information about the applicants (like age, gender, nationality, etc.), which can also play a role in graduate admissions.

2-Instead of a binary "Research Experience" feature, consider quantifying the quality or impact of research through metrics like publication counts, journal/conference rankings, or citation indices.

3-Include any additional standardized tests or certifications relevant to the field of study (like subject-specific GREs)

In [ ]:

In [ ]:

In [ ]: