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

import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor

from scipy import stats
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error
```

EDA and Data Processing

In [ ]:
```
df = pd.read_csv("dataset/Jamboree_Admission.csv")
df.head()
```

Out[ ]:

	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 [ ]:
```
df.shape
```

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

In [ ]:
```
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 [ ]:
```
df.describe()
```

Out[ ]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
count	500.000000	500.000000	500.000000	500.000000	500.000000	500.00000	500.000000	500.000000	500.00000
mean	250.500000	316.472000	107.192000	3.114000	3.374000	3.48400	8.576440	0.560000	0.72174
std	144.481833	11.295148	6.081868	1.143512	0.991004	0.92545	0.604813	0.496884	0.14114
min	1.000000	290.000000	92.000000	1.000000	1.000000	1.00000	6.800000	0.000000	0.34000
25%	125.750000	308.000000	103.000000	2.000000	2.500000	3.00000	8.127500	0.000000	0.63000
50%	250.500000	317.000000	107.000000	3.000000	3.500000	3.50000	8.560000	1.000000	0.72000
75%	375.250000	325.000000	112.000000	4.000000	4.000000	4.00000	9.040000	1.000000	0.82000
max	500.000000	340.000000	120.000000	5.000000	5.000000	5.00000	9.920000	1.000000	0.97000

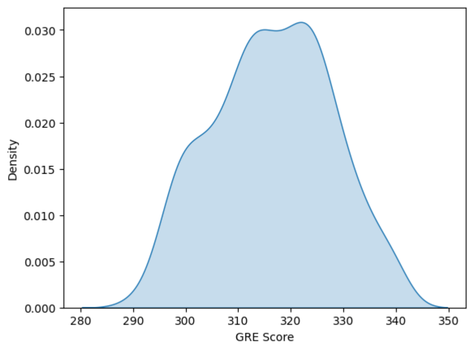
In [ ]:
```
df.drop(["Serial No."], axis=1, inplace=True)
```

In [ ]:
```
df.rename(columns=lambda x: x.strip(), inplace=True)
df.columns
```

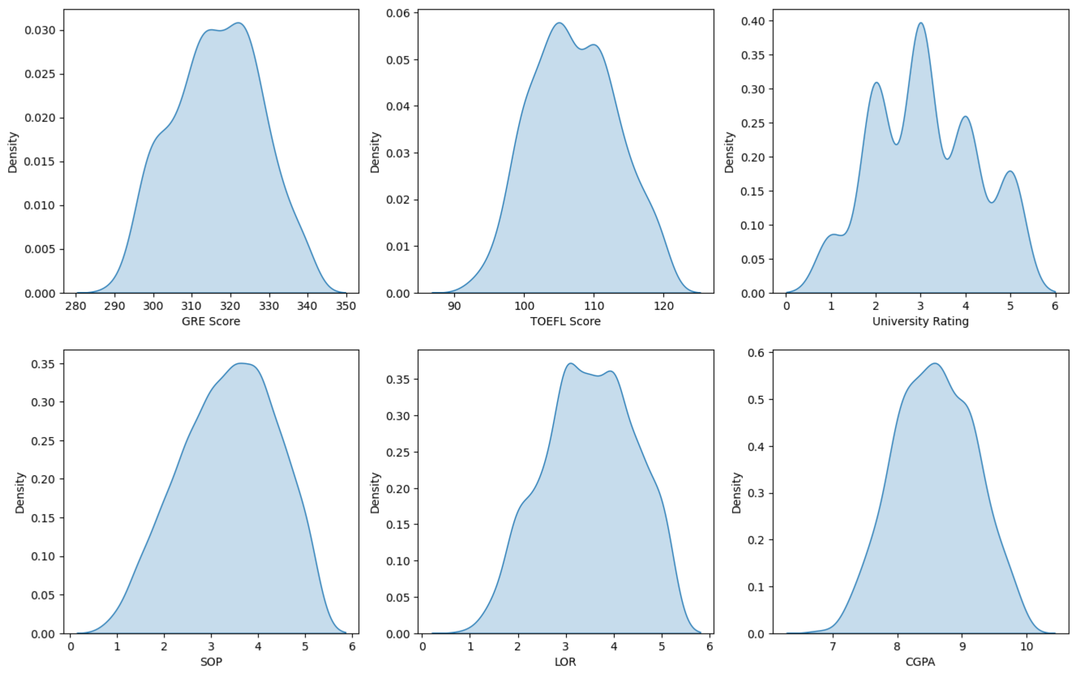
Out[ ]:
```
Index(['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR', 'CGPA',
       'Research', 'Chance of Admit'],
      dtype='object')
```

In [ ]:
```
sns.kdeplot(df["GRE Score"], fill=True)
```

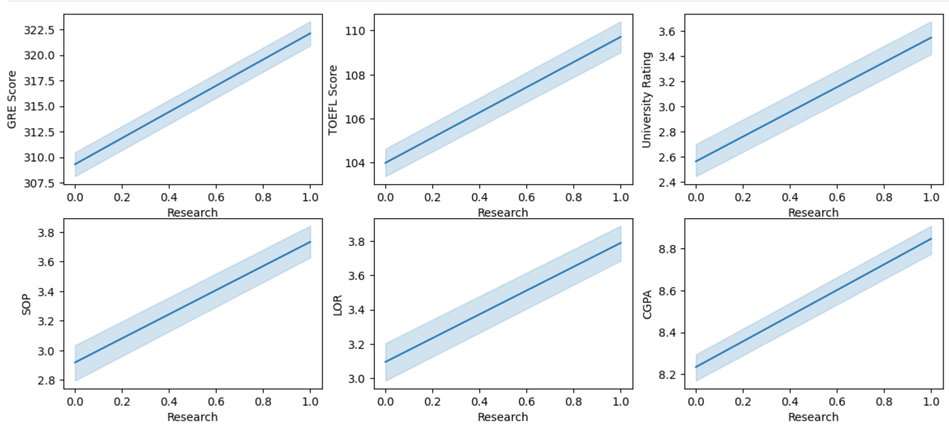
Out[ ]:
```
<Axes: xlabel='GRE Score', ylabel='Density'>
```



In [ ]:
```
fig=plt.figure(figsize=(16,10))

for ind_number, col_name in enumerate(df.columns):
    if ind_number<6:
        plt.subplot(2,3,ind_number+1)
        sns.kdeplot(df[col_name], fill=True)
```

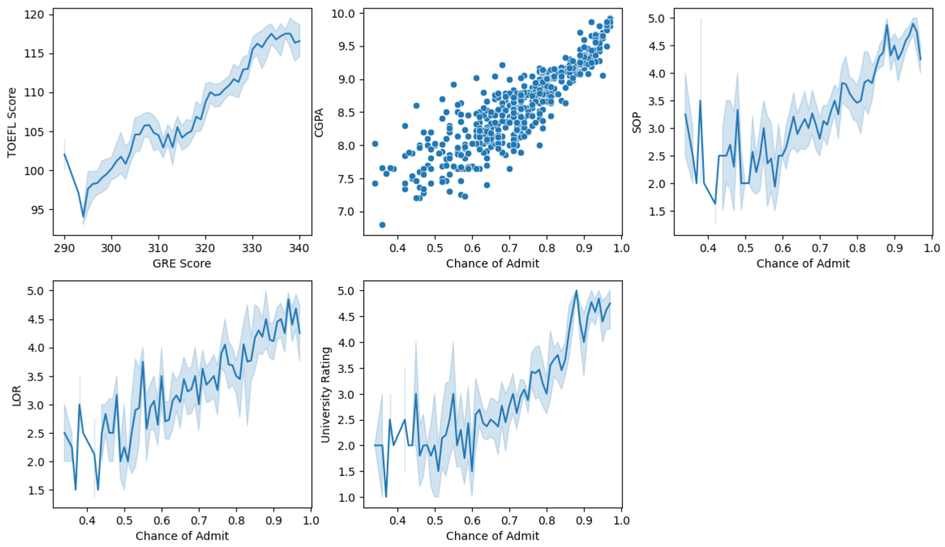


In [ ]:
```
# Bi Variate Analysis
fig=plt.figure(figsize=(14,6))

for ind_number, col_name in enumerate(df.columns):
    if ind_number<6:
        plt.subplot(2,3,ind_number+1)
        sns.lineplot(x='Research', y=col_name, data=df)

# plt.subplot(1,2,2)
# sns.scatterplot(x='Research', y='TOEFL Score', data=df)
```

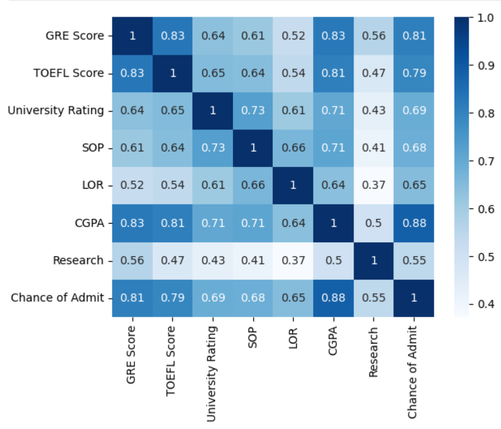


In [ ]:
```
fig=plt.figure(figsize=(14,8))

plt.subplot(2,3,1)
sns.lineplot(y='TOEFL Score', x='GRE Score', data=df)

plt.subplot(2,3,2)
sns.scatterplot(y='CGPA', x='Chance of Admit', data=df)

plt.subplot(2,3,3)
sns.lineplot(y='SOP', x='Chance of Admit', data=df)

plt.subplot(2,3,4)
sns.lineplot(y='LOR', x='Chance of Admit', data=df)

plt.subplot(2,3,5)
sns.lineplot(y='University Rating', x='Chance of Admit', data=df)
```

Out[ ]:
```
<Axes: xlabel='Chance of Admit', ylabel='University Rating'>
```



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



In [ ]:
```
fig=plt.figure(figsize=(16,10))

for ind_number, col_name in enumerate(df.columns):
    if ind_number<6:
        plt.subplot(2,3,ind_number+1)
        sns.boxplot(x=col_name, data=df)
```

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

Out[ ]:

	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

Multicollinearity and VIF

In [ ]:
```
df_test = df.copy()

# Standard Scaler
numeric_columns = ['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR', 'CGPA', 'Research']
scaler = StandardScaler()
scaler.fit(df_test[numeric_columns])
df_test[numeric_columns] = scaler.transform(df_test[numeric_columns])

# ## Min-Max Scaler Data Preparation
# scaler = MinMaxScaler()
# df_test = pd.DataFrame(scaler.fit_transform(df_test), columns=df_test.columns)
# df_test.head()
```

In [ ]:
```
y = df_test[["Chance of Admit"]]
x = df_test.drop(["Chance of Admit"], axis=1)

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.1, random_state=2)
x_train
```

Out[ ]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
407	-1.637030	-1.183716	-0.099793	-0.882817	0.558125	-1.036796	0.886405
291	-1.459786	-0.854540	-0.975168	-1.892906	-1.605151	-1.169201	-1.128152
259	1.287504	1.943453	0.775582	1.642404	1.098944	1.263738	0.886405
231	0.224037	-0.196189	-0.099793	0.127271	-1.064332	-0.407873	0.886405
478	0.135415	-0.689952	-0.099793	0.632315	1.098944	-0.143063	0.886405
...	...	...	...	...	...	...	...
22	1.021637	1.449690	1.650957	1.642404	1.639763	1.528547	0.886405
72	0.401282	0.626751	1.650957	1.642404	1.639763	1.445794	0.886405
493	-1.459786	-2.006655	-0.975168	-0.377773	-2.145970	-0.589930	0.886405
15	-0.219074	-0.360777	-0.099793	0.127271	-1.064332	-0.457525	-1.128152
168	-2.080142	-1.677479	-0.975168	-1.387862	0.558125	-1.285055	0.886405

450 rows × 7 columns

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

Out[ ]:

	Features	VIF
5	CGPA	4.71
0	GRE Score	4.35
1	TOEFL Score	3.83
3	SOP	2.76
2	University Rating	2.60
4	LOR	2.01
6	Research	1.46

In [ ]:
```
vif_thr = 5
r2_thr = 0.85
i=1
features_removed = ['CGPA']

def adj_r2_func(r2, x):
    return 1 - (1-r2) * (x.shape[0]-1)/(x.shape[0]-x.shape[1]-1)

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

    x_train = x_train[vif["Features"][1:].values]
    x_test = x_test[vif["Features"][1:].values]

    model = LinearRegression()
    model.fit(x_train, y_train)

    adj_r2 = adj_r2_func(r2_score(y_test, model.predict(x_test)), x_test)

    if (vif.iloc[0]['VIF'] < vif_thr) or (adj_r2 < r2_thr):
        print('Reached threshold')
        print('Highest vif:',vif.iloc[0]["Features"])
        print('Current adj.R2',adj_r2)
        print('Features removed:', i)
        print('List of features removed:', features_removed)
        break

    features_removed.append(vif.iloc[0]['Features'])
    i+=1
```

```
Reached threshold
Highest vif: CGPA
Current adj.R2 0.7610020551043735
Features removed: 1
List of features removed: ['CGPA']
```

In [ ]:
```

```

Model Implementation

Stats Model

In [ ]:
```
df_stats = df.copy()
df_stats.head()
```

Out[ ]:

	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 [ ]:
```
# Standard Scaler
numeric_columns = ['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR', 'CGPA', 'Research']
scaler = StandardScaler()
scaler.fit(df_stats[numeric_columns])
df_stats[numeric_columns] = scaler.transform(df_stats[numeric_columns])

# ## Min-Max Scaler Data Preparation
# scaler = MinMaxScaler()
# df_stats = pd.DataFrame(scaler.fit_transform(df_stats), columns=df_stats.columns)
# df_stats.head()
```

In [ ]:
```
y = df_stats[["Chance of Admit"]]
x = df_stats.drop(["Chance of Admit"], axis=1)

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.05, random_state=2)
x_train
```

Out[ ]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
171	1.553371	1.614278	1.650957	0.632315	1.098944	0.816871	0.886405
18	0.135415	0.462163	-0.099793	0.632315	-0.523513	0.370005	-1.128152
350	0.135415	-0.031601	-0.099793	-0.377773	0.017306	-0.507177	0.886405
476	-1.105297	-0.525364	-0.099793	-0.882817	-1.605151	-0.755436	-1.128152
20	-0.396319	-0.031601	-0.099793	-0.377773	-1.605151	-1.119549	0.886405
...	...	...	...	...	...	...	...
22	1.021637	1.449690	1.650957	1.642404	1.639763	1.528547	0.886405
72	0.401282	0.626751	1.650957	1.642404	1.639763	1.445794	0.886405
493	-1.459786	-2.006655	-0.975168	-0.377773	-2.145970	-0.589930	0.886405
15	-0.219074	-0.360777	-0.099793	0.127271	-1.064332	-0.457525	-1.128152
168	-2.080142	-1.677479	-0.975168	-1.387862	0.558125	-1.285055	0.886405

475 rows × 7 columns

In [ ]:
```
y_train = np.array(y_train)
x_train = sm.add_constant(x_train)

x_train
```

Out[ ]:

	const	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
171	1.0	1.553371	1.614278	1.650957	0.632315	1.098944	0.816871	0.886405
18	1.0	0.135415	0.462163	-0.099793	0.632315	-0.523513	0.370005	-1.128152
350	1.0	0.135415	-0.031601	-0.099793	-0.377773	0.017306	-0.507177	0.886405
476	1.0	-1.105297	-0.525364	-0.099793	-0.882817	-1.605151	-0.755436	-1.128152
20	1.0	-0.396319	-0.031601	-0.099793	-0.377773	-1.605151	-1.119549	0.886405
...	...	...	...	...	...	...	...	...
22	1.0	1.021637	1.449690	1.650957	1.642404	1.639763	1.528547	0.886405
72	1.0	0.401282	0.626751	1.650957	1.642404	1.639763	1.445794	0.886405
493	1.0	-1.459786	-2.006655	-0.975168	-0.377773	-2.145970	-0.589930	0.886405
15	1.0	-0.219074	-0.360777	-0.099793	0.127271	-1.064332	-0.457525	-1.128152
168	1.0	-2.080142	-1.677479	-0.975168	-1.387862	0.558125	-1.285055	0.886405

475 rows × 8 columns

In [ ]:
```
x_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 475 entries, 171 to 168
Data columns (total 8 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   const               475 non-null    float64
 1   GRE Score           475 non-null    float64
 2   TOEFL Score         475 non-null    float64
 3   University Rating   475 non-null    float64
 4   SOP                 475 non-null    float64
 5   LOR                 475 non-null    float64
 6   CGPA                475 non-null    float64
 7   Research            475 non-null    float64
dtypes: float64(8)
memory usage: 33.4 KB
```

In [ ]:
```
model = sm.OLS(y_train, x_train)
results = model.fit()

# Summary statistics of the model
print(results.summary())
```

```
                            OLS Regression Results                            
==============================================================================
Dep. Variable:                      y   R-squared:                       0.825
Model:                            OLS   Adj. R-squared:                  0.822
Method:                 Least Squares   F-statistic:                     313.5
Date:                Thu, 07 Dec 2023   Prob (F-statistic):          5.08e-172
Time:                        18:26:03   Log-Likelihood:                673.17
No. Observations:                 475   AIC:                            -1330.
Df Residuals:                     467   BIC:                            -1297.
Df Model:                           7                                         
Covariance Type:            nonrobust                                         
=====================================================================================
                        coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------------
const                 0.7221      0.003    265.958      0.000       0.717       0.727
GRE Score             0.0243      0.006      4.245      0.000       0.013       0.036
TOEFL Score           0.0152      0.005      2.823      0.005       0.005       0.026
University Rating     0.0078      0.004      1.797      0.073      -0.001       0.016
SOP                   0.0003      0.005      0.076      0.939      -0.009       0.009
LOR                   0.0168      0.004      4.336      0.000       0.009       0.024
CGPA                  0.0703      0.006     11.915      0.000       0.059       0.082
Research              0.0108      0.003      3.270      0.001       0.004       0.017
==============================================================================
Omnibus:                      104.689   Durbin-Watson:                   1.970
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              237.982
Skew:                          -1.138   Prob(JB):                     2.10e-52
Kurtosis:                       5.616   Cond. No.                         5.57
==============================================================================

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

Sklearn Model Implementation

In [ ]:
```
df_model=df.copy()

numeric_columns = ['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR', 'CGPA', 'Research']
scaler = StandardScaler()
scaler.fit(df_model[numeric_columns])
df_model[numeric_columns] = scaler.transform(df_model[numeric_columns])

# ## Min-Max Scaler Data Preparation
# scaler = MinMaxScaler()
# df_model = pd.DataFrame(scaler.fit_transform(df_model), columns=df_model.columns)
# df_model.head()
```

In [ ]:
```
y = df_model[["Chance of Admit"]]
x = df_model.drop(["Chance of Admit"], axis=1)

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.1, random_state=2)
x_train
```

Out[ ]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
407	-1.637030	-1.183716	-0.099793	-0.882817	0.558125	-1.036796	0.886405
291	-1.459786	-0.854540	-0.975168	-1.892906	-1.605151	-1.169201	-1.128152
259	1.287504	1.943453	0.775582	1.642404	1.098944	1.263738	0.886405
231	0.224037	-0.196189	-0.099793	0.127271	-1.064332	-0.407873	0.886405
478	0.135415	-0.689952	-0.099793	0.632315	1.098944	-0.143063	0.886405
...	...	...	...	...	...	...	...
22	1.021637	1.449690	1.650957	1.642404	1.639763	1.528547	0.886405
72	0.401282	0.626751	1.650957	1.642404	1.639763	1.445794	0.886405
493	-1.459786	-2.006655	-0.975168	-0.377773	-2.145970	-0.589930	0.886405
15	-0.219074	-0.360777	-0.099793	0.127271	-1.064332	-0.457525	-1.128152
168	-2.080142	-1.677479	-0.975168	-1.387862	0.558125	-1.285055	0.886405

450 rows × 7 columns

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

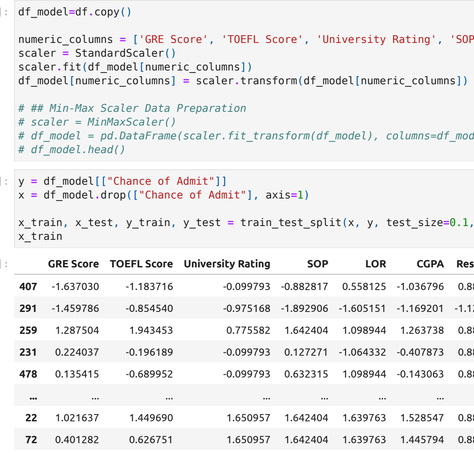
Out[ ]:
```
LinearRegression()
```

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

Out[ ]:
```
array([[0.02396356, 0.0150632 , 0.00747459, 0.00072564, 0.01713408,
        0.07113889, 0.01010487]])
```

In [ ]:
```
imp = pd.DataFrame(list(zip(x_test.columns, np.abs(model.coef_)[0])),
                   columns=['feature', 'coeff'])
sns.barplot(x='feature', y='coeff', data=imp)
plt.xticks(rotation=90)
```

Out[ ]:
```
([0, 1, 2, 3, 4, 5, 6],
 [Text(0, 0, 'GRE Score'),
  Text(1, 0, 'TOEFL Score'),
  Text(2, 0, 'University Rating'),
  Text(3, 0, 'SOP'),
  Text(4, 0, 'LOR'),
  Text(5, 0, 'CGPA'),
  Text(6, 0, 'Research')])
```

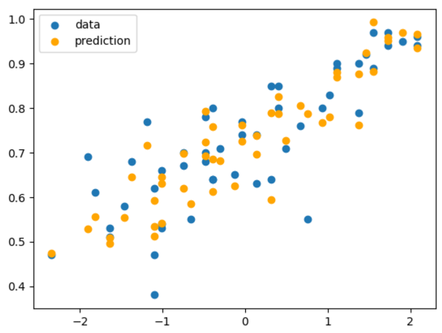


In [ ]:
```
model.intercept_
```

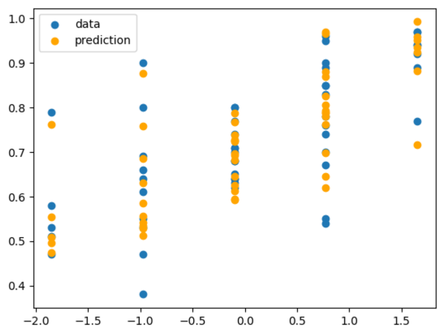
Out[ ]:
```
array([0.72157922])
```

In [ ]:
```
y_hat = model.predict(x_test)
```

In [ ]:
```
fig = plt.figure()
x_feature = x_test["GRE Score"]
plt.scatter(x_feature, y_test,label='data')
plt.scatter(x_feature, y_hat,color='orange',label='prediction')
plt.legend()
plt.show()
```



In [ ]:
```
fig = plt.figure()
x_feature = x_test["University Rating"]
plt.scatter(x_feature, y_test,label='data')
plt.scatter(x_feature, y_hat,color='orange',label='prediction')
plt.legend()
plt.show()
```



In [ ]:
```
## MAE
abs(y_hat - y_test).mean()
```

Out[ ]:
```
Chance of Admit    0.039666
dtype: float64
```

In [ ]:
```
## RMSE
mean_squared_error(y_test, y_hat, squared=False)
```

Out[ ]:
```
0.05877180888433025
```

In [ ]:
```
## R2 Value
r2_score(y_test, y_hat)
```

Out[ ]:
```
0.8439659610816171
```

In [ ]:
```
## Adjusted r2 value
adj_r2_func(r2_score(y_test, y_hat), x_test)
```

Out[ ]:
```
0.8179602879285534
```

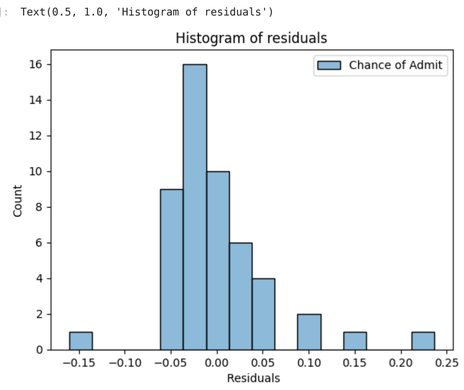
In [ ]:
```
errors = y_hat - y_test
```

In [ ]:
```
## Means of Residuals
np.array(errors).mean()
```

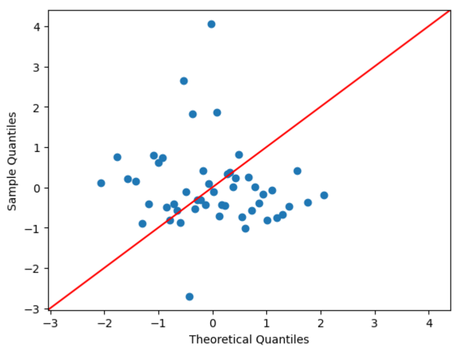
Out[ ]:
```
-0.0016077872077638522
```

In [ ]:
```
## Normality of Residuals

sns.histplot(errors)
plt.xlabel(" Residuals")
plt.title("Histogram of residuals")
```

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



In [ ]:
```
# Create a QQ plot
sm.qqplot(errors, line ='45', fit=True, dist=stats.norm)
plt.show()
```

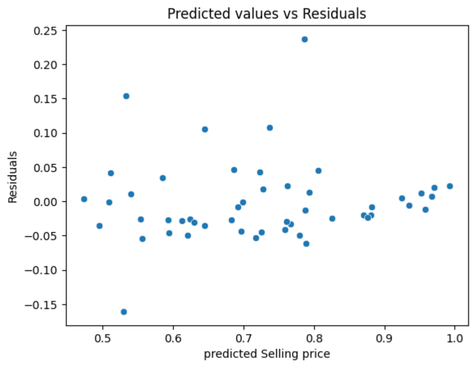


In [ ]:
```
## Normality of Residuals
res = stats.shapiro(errors)
print(res.statistic) ## Closer the value to 1, more is the normality.
```

```
0.8413031697273254
```

In [ ]:
```
## Linearity of variable / Homoscedasticity
sns.scatterplot(x=y_hat.flatten(),y=np.array(errors).flatten())
plt.xlabel("predicted Selling price")
plt.ylabel("Residuals")
plt.title("Predicted values vs Residuals")
```

Out[ ]:
```
Text(0.5, 1.0, 'Predicted values vs Residuals')
```



In [ ]: