Jamboree Case Study

Gautam Naik (gautamnaik1994@gmail.com)

Github Link: https://github.com/gautamnaik1994/Jamboree-ML-Case-Study

Business Problem

Jamboree has helped thousands of students like you make it to top colleges abroad. Be it GMAT, GRE or SAT, their unique problem-solving methods ensure maximum scores with minimum effort.
They recently launched a feature where students/learners can come to their website and check their probability of getting into the IVY league college. This feature estimates the chances of graduate admission from an Indian perspective.

Your analysis will help Jamboree in understanding what factors are important in graduate admissions and how these factors are interrelated among themselves. It will also help predict one's chances of admission given the rest of the variables.

We will use Exploratory Data Analysis to find important factors and also Linear Regression to predict the chance to get admission and to rank the important factors by importance.

Metric

We will use R2 score, Root Mean Squared Error, Adjusted R2 score and plots to gauge the accuracy of the model.

Dataset:

- Serial No. (Unique row ID)
- GRE Scores (out of 340)
- TOEFL Scores (out of 120)
- University Rating (out of 5)
- Statement of Purpose and Letter of Recommendation Strength (out of 5)
- Undergraduate GPA (out of 10)
- Research Experience (either 0 or 1)
- Chance of Admit (ranging from 0 to 1)

In [ ]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
# from sklearnex import patch_sklearn
sns.set_style(style="whitegrid")
from scipy.stats import shapiro
from janitor import clean_names
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error, root_mean_squared_error
from sklearn.linear_model import Lasso, Ridge
from sklearn.preprocessing import PolynomialFeatures
# # patch_sklearn()
from statsmodels.stats.diagnostic import het_goldfeldquandt
```

In [ ]:
```python
df=pd.read_csv("./Admission_Predict_Ver1.1.csv")
df = clean_names(df, strip_underscores=True)
```

In [ ]:
```python
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 [ ]:
```python
df=df.drop_duplicates()
df=df.drop("serial_no",axis=1)
```

EDA

In [ ]:
```python
df.isnull().sum()
```

Out[ ]:
```
gre_score            0
toefl_score          0
university_rating    0
sop                  0
lor                  0
cgpa                 0
research             0
chance_of_admit      0
dtype: int64
```

In [ ]:
```python
df.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    int64
 3   sop                500 non-null    float64
 4   lor                500 non-null    float64
 5   cgpa               500 non-null    float64
 6   research           500 non-null    int64
 7   chance_of_admit    500 non-null    float64
dtypes: float64(4), int64(4)
memory usage: 31.4 KB
```

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

Out[ ]:

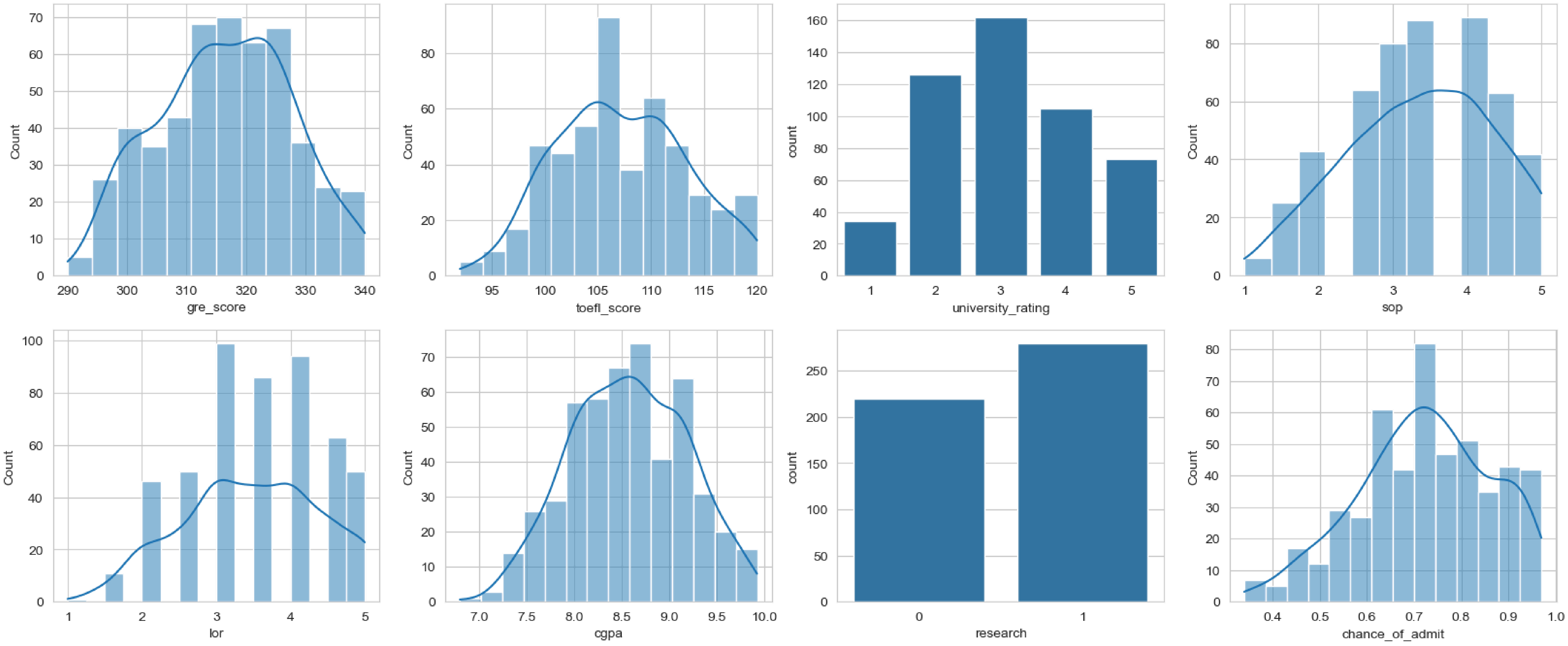
	gre_score	toefl_score	university_rating	sop	lor	cgpa	research	chance_of_admit
count	500.000000	500.000000	500.000000	500.000000	500.00000	500.000000	500.000000	500.00000
mean	316.472000	107.192000	3.114000	3.374000	3.48400	8.576440	0.560000	0.72174
std	11.295148	6.081868	1.143512	0.991004	0.92545	0.604813	0.496884	0.14114
min	290.000000	92.000000	1.000000	1.000000	1.00000	6.800000	0.000000	0.34000
25%	308.000000	103.000000	2.000000	2.500000	3.00000	8.127500	0.000000	0.63000
50%	317.000000	107.000000	3.000000	3.500000	3.50000	8.560000	1.000000	0.72000
75%	325.000000	112.000000	4.000000	4.000000	4.00000	9.040000	1.000000	0.82000
max	340.000000	120.000000	5.000000	5.000000	5.00000	9.920000	1.000000	0.97000

In [ ]:

In [ ]:
```
df.columns
```

Out[ ]:
```
Index(['gre_score', 'toefl_score', 'university_rating', 'sop', 'lor', 'cgpa',
       'research', 'chance_of_admit'],
      dtype='object')
```

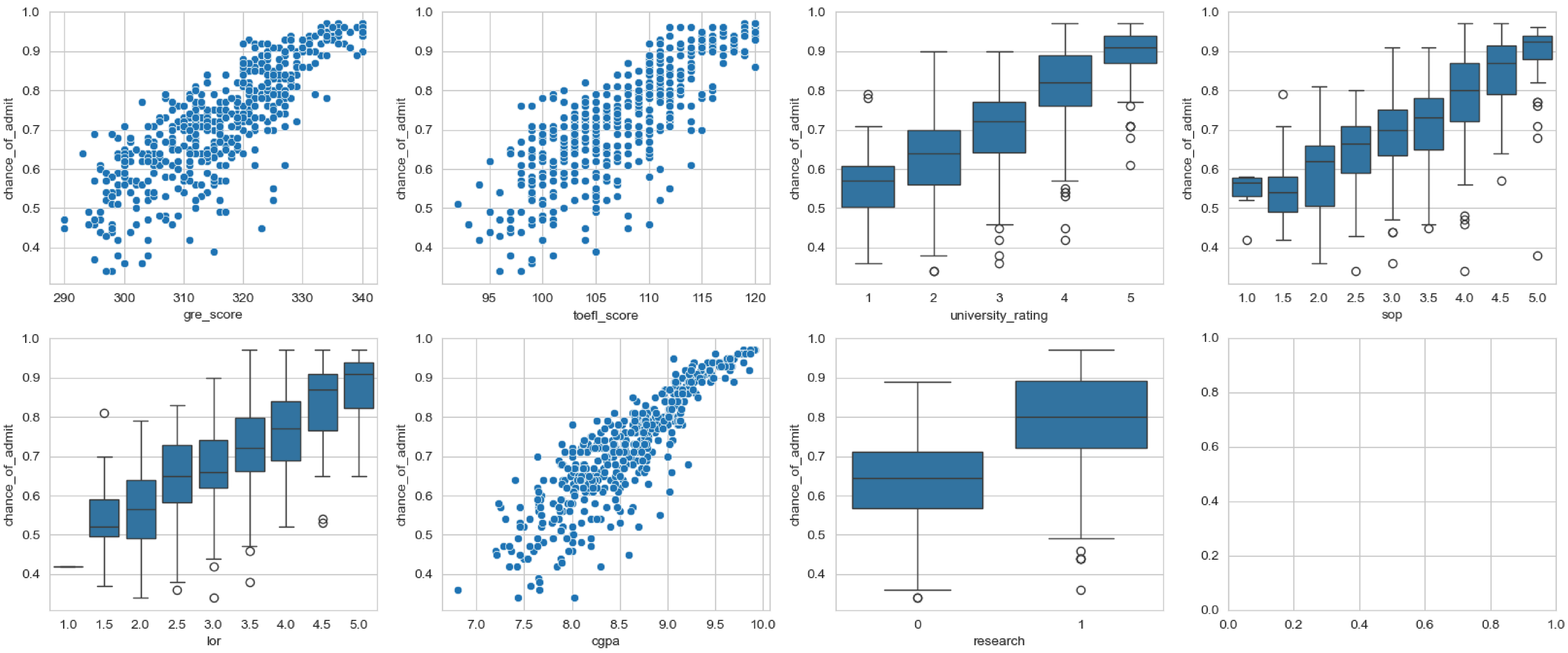
In [ ]:
```
fig, ax = plt.subplots(2, 4, figsize=(20, 8))
sns.histplot(df['gre_score'], kde=True, ax=ax[0][0])
sns.histplot(df['toefl_score'], kde=True, ax=ax[0][1])
sns.countplot(data=df, x='university_rating', ax=ax[0][2])
sns.histplot(df['sop'], kde=True, ax=ax[0][3])
sns.histplot(df['lor'], kde=True, ax=ax[1][0])
sns.histplot(df['cgpa'], kde=True, ax=ax[1][1])
sns.countplot(data=df, x='research', ax=ax[1][2])
sns.histplot(df['chance_of_admit'], kde=True, ax=ax[1][3]);
```



Observations

- From above plot we can see that all the features have almost normal distribution

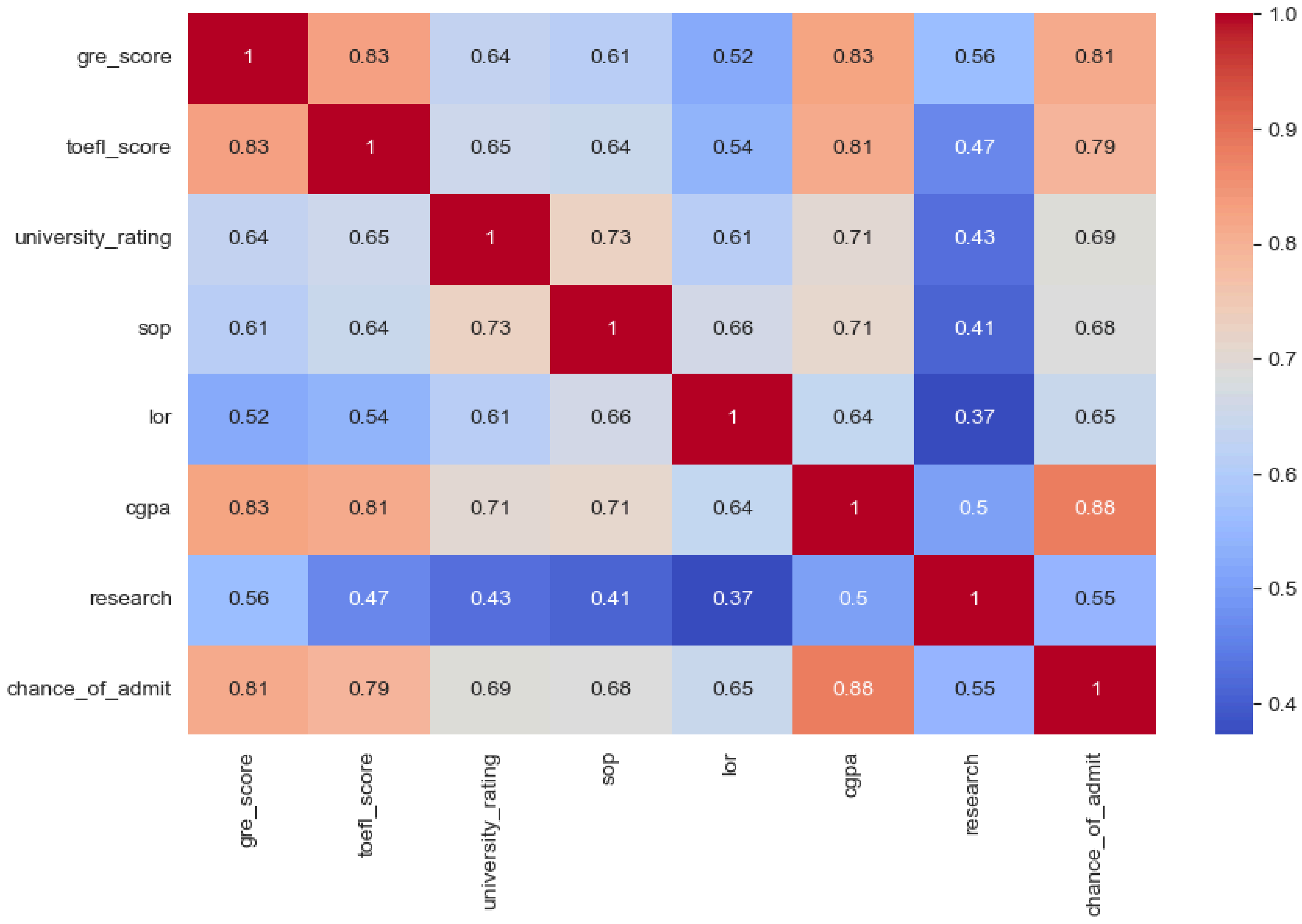
In [ ]:
```
fig, ax = plt.subplots(2, 4, figsize=(20, 8))
sns.scatterplot(data=df, x='gre_score', y='chance_of_admit' , ax=ax[0][0])
sns.scatterplot(data=df, x='toefl_score', y='chance_of_admit', ax=ax[0][1])
sns.boxplot(data=df, x='university_rating', y='chance_of_admit', ax=ax[0][2])
sns.boxplot(data=df, x='sop', y='chance_of_admit', ax=ax[0][3])
sns.boxplot(data=df, x='lor', y='chance_of_admit', ax=ax[1][0])
sns.scatterplot(data=df, x='cgpa', y='chance_of_admit', ax=ax[1][1])
sns.boxplot(data=df, x='research', y='chance_of_admit', ax=ax[1][2]);
# sns.scatterplot(data=df, x='chance_of_admit', y='chance_of_admit' ax=ax[1][3]);
```



Observations

- We can see that most of the features have linear relationship with chance_to_admit
- From above plot we can see that having a research increses the chance of admission

In [ ]:
```
plt.figure(figsize=(10, 6))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm');
```



Observations

- Above plot shows the correlation between the features
- We can see that most of the features are correlated with each other but all are under 0.9
- Hence we do not need to drop features

Model Building

In [ ]:
```
cols=['gre_score', 'toefl_score', 'university_rating', 'sop', 'lor', 'cgpa',
       'research']
```

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(df.drop(["chance_of_admit"], axis=1), df['chance_of_admit'], test_size=0.2, random_state=42)
```

In [ ]:
```
scaler = StandardScaler()
scaler.fit(X_train)
```

Out[ ]:
```
▾ StandardScaler
StandardScaler()
```

In [ ]:
```
X_train = pd.DataFrame(scaler.transform(X_train), columns = X_train.columns)
X_test = pd.DataFrame(scaler.transform(X_test), columns = X_test.columns)
```

VIF

In [ ]:
```
X_vif = pd.DataFrame(X_train, columns=df.drop(["chance_of_admit"], axis=1).columns)
vif = pd.DataFrame()

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

Out[ ]:

	Features	VIF
5	cgpa	4.65
0	gre_score	4.49
1	toefl_score	3.66
3	sop	2.79
2	university_rating	2.57
4	lor	1.98
6	research	1.52

Observations

- All the features have VIF < 5
- No need to remove features

OLS

In [ ]:
```
X_train_sm = sm.add_constant(X_train)
X_test_sm = sm.add_constant(X_test)
```

In [ ]:
```
ols_model_1 = sm.OLS(y_train.reset_index(drop=True), X_train_sm)
ols_model_1_result = ols_model_1.fit()
print(ols_model_1_result.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:        chance_of_admit   R-squared:                       0.821
Model:                            OLS   Adj. R-squared:                  0.818
Method:                 Least Squares   F-statistic:                     257.0
Date:                Tue, 16 Jul 2024   Prob (F-statistic):          3.41e-142
Time:                        14:39:13   Log-Likelihood:                 561.91
No. Observations:                 400   AIC:                            -1108.
Df Residuals:                     392   BIC:                            -1076.
Df Model:                           7
Covariance Type:            nonrobust
=====================================================================================
                        coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------------
const                 0.7242      0.003    241.441      0.000       0.718       0.730
gre_score             0.0267      0.006      4.196      0.000       0.014       0.039
toefl_score           0.0182      0.006      3.174      0.002       0.007       0.030
university_rating     0.0029      0.005      0.611      0.541      -0.007       0.012
sop                   0.0018      0.005      0.357      0.721      -0.008       0.012
lor                   0.0159      0.004      3.761      0.000       0.008       0.024
cgpa                  0.0676      0.006     10.444      0.000       0.055       0.080
research              0.0119      0.004      3.231      0.001       0.005       0.019
==============================================================================
Omnibus:                       86.232   Durbin-Watson:                   2.050
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              190.099
Skew:                          -1.107   Prob(JB):                     5.25e-42
Kurtosis:                       5.551   Cond. No.                         5.65
==============================================================================

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

In [ ]:
```python
y_test_pred = ols_model_1_result.predict(X_test_sm)
```

In [ ]:
```python
print("Mean Squared Error: ", mean_squared_error(y_test, y_test_pred))
print("Root Mean Squared Error: ", root_mean_squared_error(y_test, y_test_pred))
print("Mean Absolute Error: ", mean_absolute_error(y_test, y_test_pred))
print("R2 Score: ", r2_score(y_test, y_test_pred))
```

```
Mean Squared Error:  0.003704655398788415
Root Mean Squared Error:  0.06086588041578315
Mean Absolute Error:  0.04272265427705366
R2 Score:  0.8188432567829627
```

In [ ]:
```python
y_train_pred = ols_model_1_result.predict(X_train_sm)
```

In [ ]:
```python
print("Mean Squared Error: ", mean_squared_error(y_train, y_train_pred))
print("Root Mean Squared Error: ", root_mean_squared_error(y_train, y_train_pred))
print("Mean Absolute Error: ", mean_absolute_error(y_train, y_train_pred))
print("R2 Score: ", r2_score(y_train, y_train_pred))
```

```
Mean Squared Error:  0.0035265554784557583
Root Mean Squared Error:  0.05938480848210052
Mean Absolute Error:  0.042533340611643065
R2 Score:  0.8210671369321554
```

Dropping university_rating and sop as they donot contribute to model

In [ ]:
```python
X_train_sm = sm.add_constant(X_train.drop(["university_rating", "sop"], axis=1))
X_test_sm = sm.add_constant(X_test.drop(["university_rating", "sop"], axis=1))
```

In [ ]:
```python
ols_model_2 = sm.OLS(y_train.reset_index(drop=True), X_train_sm)
ols_model_2_result = ols_model_2.fit()
print(ols_model_2_result.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:        chance_of_admit   R-squared:                       0.821
Model:                            OLS   Adj. R-squared:                  0.818
Method:                 Least Squares   F-statistic:                     360.8
Date:                Tue, 16 Jul 2024   Prob (F-statistic):          1.36e-144
Time:                        14:39:14   Log-Likelihood:                 561.54
No. Observations:                 400   AIC:                            -1111.
Df Residuals:                     394   BIC:                            -1087.
Df Model:                           5
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
const           0.7242      0.003    241.830      0.000       0.718       0.730
gre_score       0.0269      0.006      4.245      0.000       0.014       0.039
toefl_score     0.0191      0.006      3.391      0.001       0.008       0.030
lor             0.0172      0.004      4.465      0.000       0.010       0.025
cgpa            0.0691      0.006     11.147      0.000       0.057       0.081
research        0.0122      0.004      3.328      0.001       0.005       0.019
==============================================================================
Omnibus:                       84.831   Durbin-Watson:                   2.053
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              185.096
Skew:                          -1.094   Prob(JB):                     6.41e-41
Kurtosis:                       5.514   Cond. No.                         4.76
==============================================================================

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

In [ ]:
```python
y_train_pred = ols_model_2_result.predict(X_train_sm)
```

In [ ]:
```python
print("Mean Squared Error: ", mean_squared_error(y_train, y_train_pred))
print("Root Mean Squared Error: ", root_mean_squared_error(y_train, y_train_pred))
print("Mean Absolute Error: ", mean_absolute_error(y_train, y_train_pred))
print("R2 Score: ", r2_score(y_train, y_train_pred))
```

```
Mean Squared Error:  0.0035331469389868714
Root Mean Squared Error:  0.05944028044169098
Mean Absolute Error:  0.04269126483606393
R2 Score:  0.8207326947514393
```

In [ ]:
```python
y_test_pred = ols_model_2_result.predict(X_test_sm)
```

In [ ]:
```python
print("Mean Squared Error: ", mean_squared_error(y_test, y_test_pred))
print("Root Mean Squared Error: ", root_mean_squared_error(y_test, y_test_pred))
print("Mean Absolute Error: ", mean_absolute_error(y_test, y_test_pred))
print("R2 Score: ", r2_score(y_test, y_test_pred))
```

```
Mean Squared Error:  0.003773020765116896
Root Mean Squared Error:  0.06142491974041884
Mean Absolute Error:  0.0429234557826578
R2 Score:  0.8155002070847484
```

Observations

- Train score and Test score is almost the same
- Hence no need to change anything in the model

Test for Homoscedasticity using Goldfeld Quant Test

- Null Hypothesis: The variances of the error terms are equal (homoscedasticity). In other words, there is no heteroscedasticity.
- Alternative Hypothesis: The variances of the error terms are not equal (heteroscedasticity). In other words, heteroscedasticity is present.

In [ ]:
```
het_goldfeldquandt(ols_model_2_result.resid, ols_model_2_result.model.exog)
```

Out[ ]: `(0.9592288620962857, 0.613902484588438, 'increasing')`

- Since p value os 0.61 we fail to reject the null hypothesis. This means there is no heteroscedasticity

In [ ]:
```
het_goldfeldquandt(ols_model_2_result.resid, ols_model_2_result.model.exog, alternative='decreasing')
```

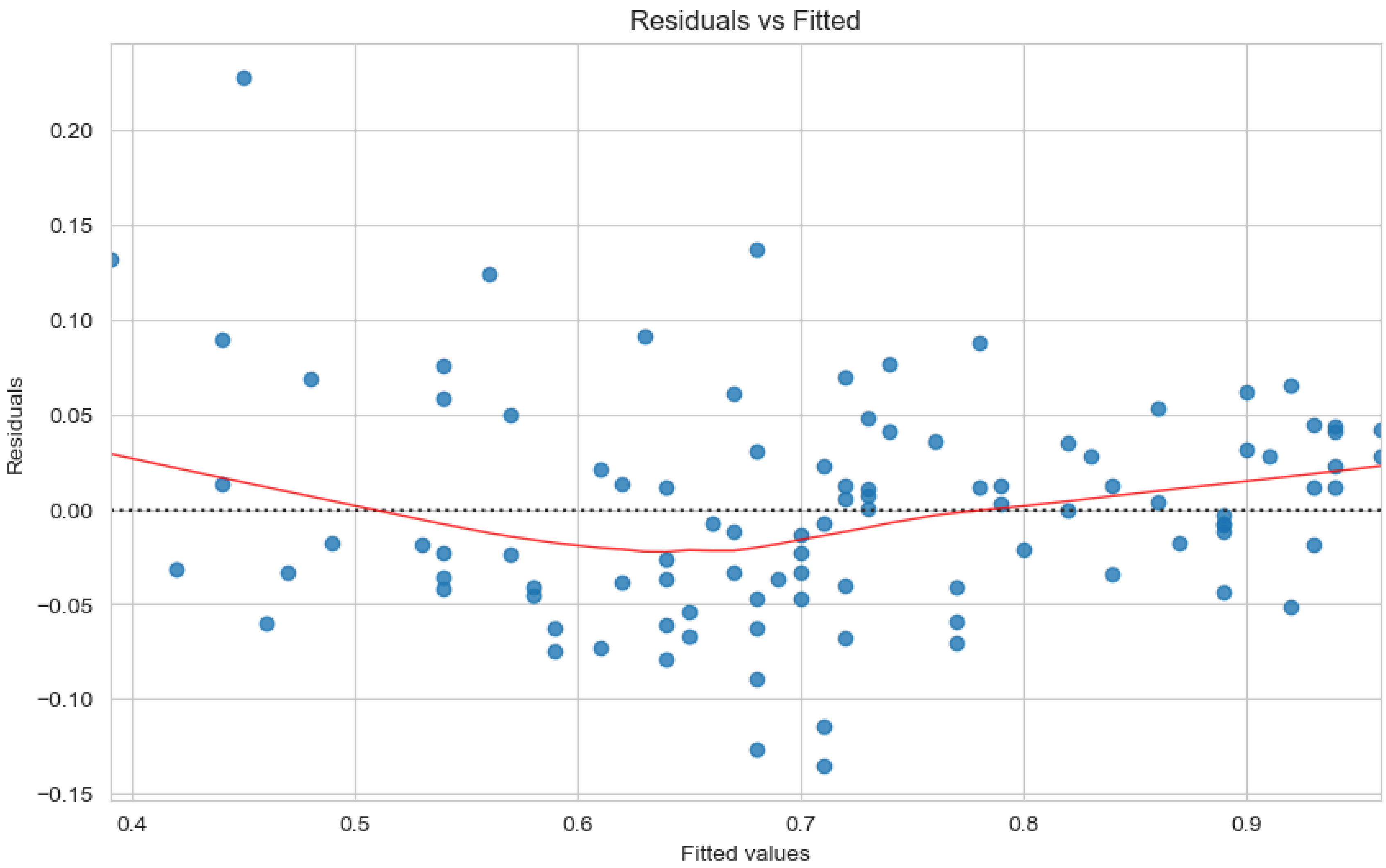
Out[ ]: `(0.9592288620962857, 0.3860975154115565, 'decreasing')`

- Since p value os 0.31 we fail to reject the null hypothesis. This means there is no heteroscedasticity

Observations

From both test above we can see that there is no heteroscedasticity

In [ ]:
```
# residual plot
plt.figure(figsize=(10, 6))
sns.residplot(x=y_test, y=y_test_pred, lowess=True, line_kws={'color': 'red', 'lw': 1, 'alpha': 0.8})
plt.xlabel('Fitted values')
plt.ylabel('Residuals')
plt.title('Residuals vs Fitted')
plt.show()
```

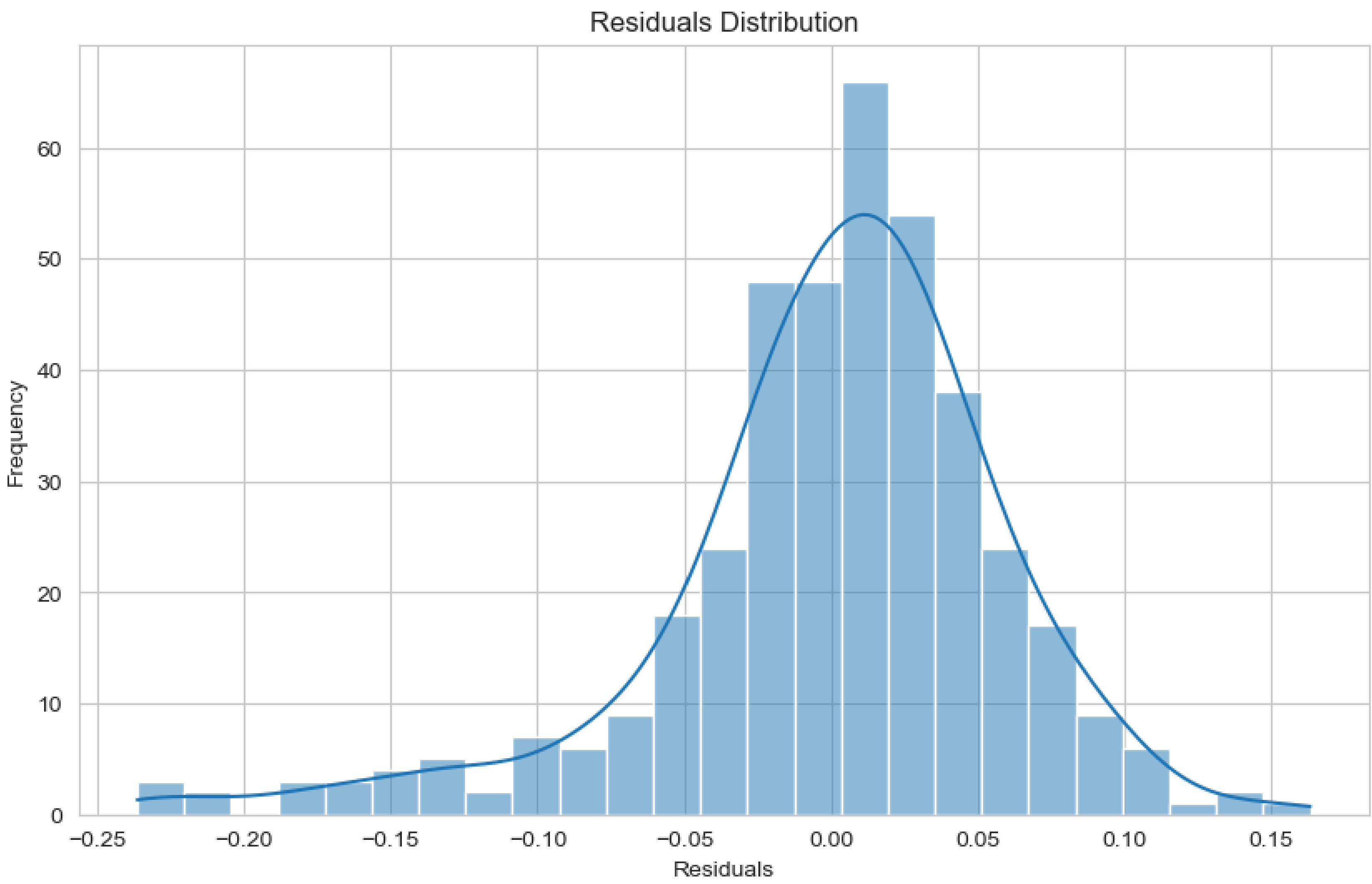


Observations

- From above plot we can see that residuals are equally distributted around 0

Residual Normality Check

In [ ]:
```
plt.figure(figsize=(10, 6))
sns.histplot(ols_model_2_result.resid, kde=True)
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.title('Residuals Distribution');
```



Observations

- From above we can see that residuals follow a normal distribution.

Residual Mean

In [ ]:
```
ols_model_2_result.resid.mean()
```

Out[ ]: `-4.1924796967407475e-16`

Observations

Mean of residuals is very close to 0

In [ ]:
```
def adjusted_r2(r2, n, p):
    return 1-((1-r2)*(n-1)/(n-p-1))
```

Lasso Regression

In [ ]:
```
lasso=Lasso(alpha=0.01)
model=lasso.fit(X_train, y_train)
y_train_pred=model.predict(X_train)
y_test_pred=model.predict(X_test)
```

In [ ]:
```
model.coef_
model.feature_names_in_
pd.DataFrame(model.coef_, index=model.feature_names_in_, columns=["importance"]).sort_values("importance", ascending=False)
```

Out[ ]:

	importance
cgpa	0.068957
gre_score	0.026240
toefl_score	0.015137
lor	0.011111
research	0.006099
university_rating	0.000919
sop	0.000000

In [ ]:
```
print("Mean Squared Error: ", mean_squared_error(y_test, y_test_pred))
print("Root Mean Squared Error: ", root_mean_squared_error(y_test, y_test_pred))
print("Mean Absolute Error: ", mean_absolute_error(y_test, y_test_pred))
print("R2 Score: ", r2_score(y_test, y_test_pred))
print("Adjusted R2 Score: ", adjusted_r2(r2_score(y_test, y_test_pred), X_test.shape[0], X_test.shape[1]))
```

```
Mean Squared Error:  0.0038037941002089094
Root Mean Squared Error:  0.06167490656830304
Mean Absolute Error:  0.04270927740179335
R2 Score:  0.8139953985227918
Adjusted R2 Score:  0.799842874497352
```

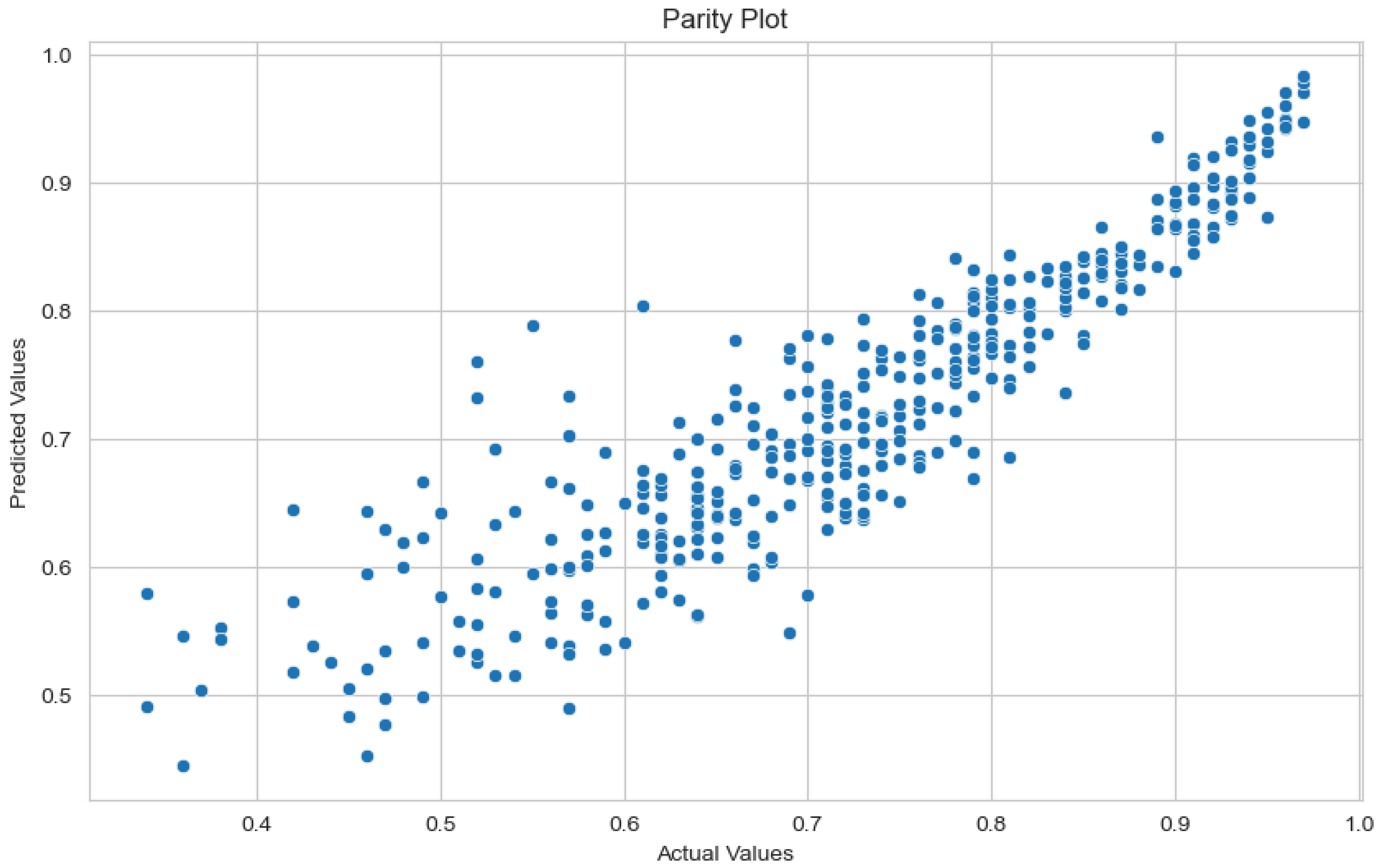
In [ ]:
```
print("Mean Squared Error: ", mean_squared_error(y_train, y_train_pred))
print("Root Mean Squared Error: ", root_mean_squared_error(y_train, y_train_pred))
print("Mean Absolute Error: ", mean_absolute_error(y_train, y_train_pred))
print("R2 Score: ", r2_score(y_train, y_train_pred))
print("Adjusted R2 Score: ", adjusted_r2(r2_score(y_train, y_train_pred), X_train.shape[0], X_train.shape[1]))
```

```
Mean Squared Error:  0.0036919803717825094
Root Mean Squared Error:  0.060761668605976496
Mean Absolute Error:  0.04365375627785551
R2 Score:  0.8126736918363732
Adjusted R2 Score:  0.8093285791905941
```

Observations

- There is not much difference in train and test score

In [ ]:
```
# plot a parity plot
plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_train, y=y_train_pred)
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("Parity Plot")
plt.show()
```



Observations

- We can see that there is some variance at the lower end

Ridge Regression

In [ ]:
```
ridge=Ridge(alpha=0.001)
model=ridge.fit(X_train, y_train)
y_train_pred=model.predict(X_train)
y_test_pred=model.predict(X_test)
```

In [ ]:
```
model.coef_
model.feature_names_in_
pd.DataFrame(model.coef_, index=model.feature_names_in_, columns=["importance"]).sort_values("importance", ascending=False)
```

Out[ ]:

	importance
cgpa	0.067580
gre_score	0.026671
toefl_score	0.018226
lor	0.015866
research	0.011941
university_rating	0.002940
sop	0.001788

Observations

- From all above observations we can see that CGPA is the main feature off the dataset.

In [ ]:
```
print("Mean Squared Error: ", mean_squared_error(y_test, y_test_pred))
print("Root Mean Squared Error: ", root_mean_squared_error(y_test, y_test_pred))
print("Mean Absolute Error: ", mean_absolute_error(y_test, y_test_pred))
print("R2 Score: ", r2_score(y_test, y_test_pred))
print("Adjusted R2 Score: ", adjusted_r2(r2_score(y_test, y_test_pred), X_test.shape[0], X_test.shape[1]))
```

```
Mean Squared Error:  0.003704656511294749
Root Mean Squared Error:  0.06086588955478059
Mean Absolute Error:  0.04272267949191486
R2 Score:  0.818843202381675
Adjusted R2 Score:  0.805059532997672
```

In [ ]:
```
print("Mean Squared Error: ", mean_squared_error(y_train, y_train_pred))
print("Root Mean Squared Error: ", root_mean_squared_error(y_train, y_train_pred))
print("Mean Absolute Error: ", mean_absolute_error(y_train, y_train_pred))
print("R2 Score: ", r2_score(y_train, y_train_pred))
print("Adjusted R2 Score: ", adjusted_r2(r2_score(y_train, y_train_pred), X_train.shape[0], X_train.shape[1]))
```

```
Mean Squared Error:  0.0035265554785364438
Root Mean Squared Error:  0.05938480848277987
Mean Absolute Error:  0.04253333646142554
R2 Score:  0.8210671369280614
Adjusted R2 Score:  0.8178719072303482
```

Polynomial Features with Lasso Regression

In [ ]:
```
poly=PolynomialFeatures(degree=2)
X_train_poly=poly.fit_transform(X_train)
X_test_poly=poly.transform(X_test)

lasso=Lasso(alpha=0.0001)

model=lasso.fit(X_train_poly, y_train)
y_train_pred=model.predict(X_train_poly)
y_test_pred=model.predict(X_test_poly)
```

In [ ]:
```
print("Mean Squared Error: ", mean_squared_error(y_train, y_train_pred))
print("Root Mean Squared Error: ", root_mean_squared_error(y_train, y_train_pred))
print("Mean Absolute Error: ", mean_absolute_error(y_train, y_train_pred))
print("R2 Score: ", r2_score(y_train, y_train_pred))
print("Adjusted R2 Score: ", adjusted_r2(r2_score(y_train, y_train_pred), X_train.shape[0], X_train.shape[1]))
```

```
Mean Squared Error:  0.003239799860680364
Root Mean Squared Error:  0.0569192398111602
Mean Absolute Error:  0.040095734168535506
R2 Score:  0.8356167460345217
Adjusted R2 Score:  0.8326813307851382
```

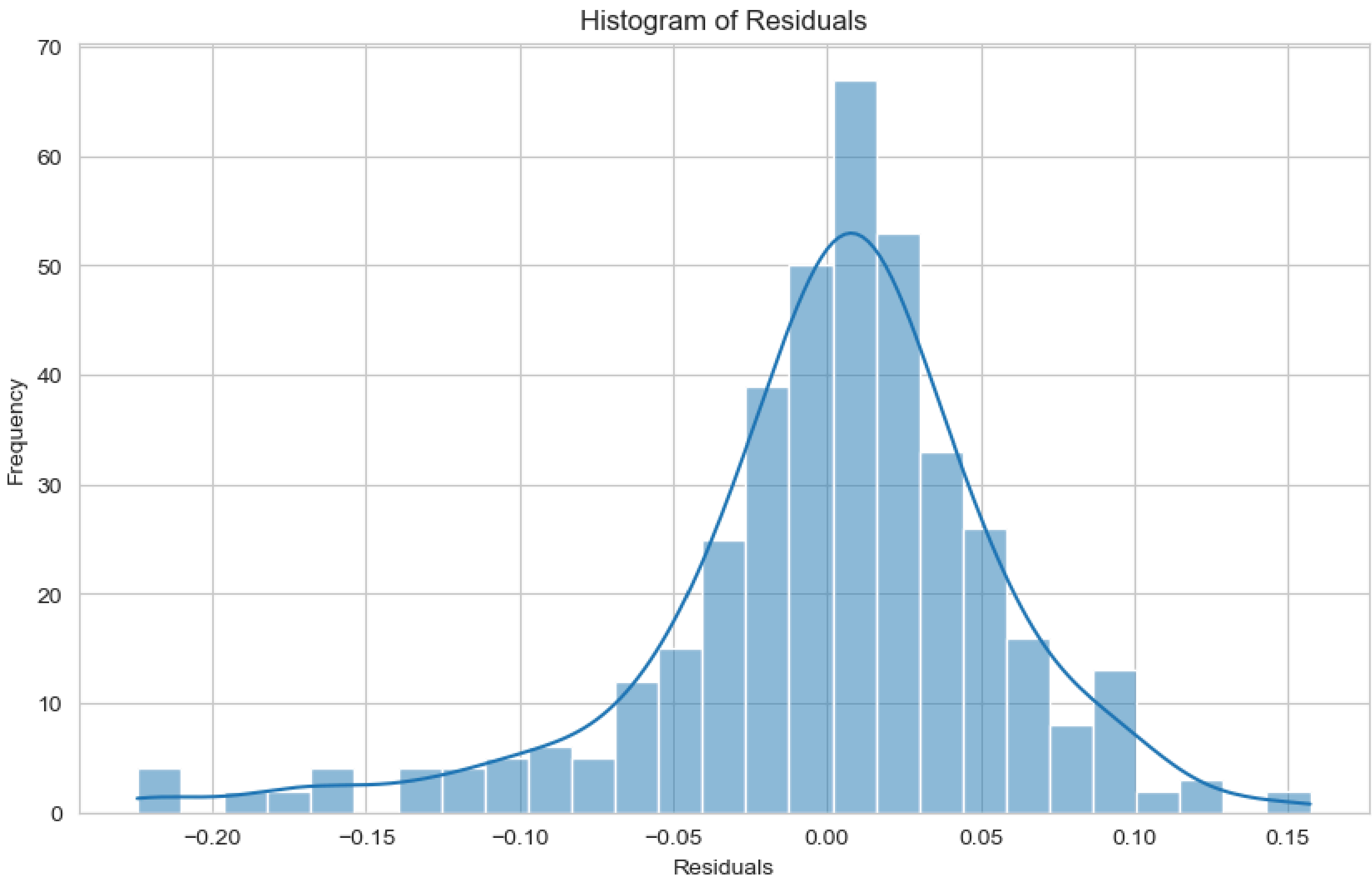
In [ ]:
```
print("Mean Squared Error: ", mean_squared_error(y_test, y_test_pred))
print("Root Mean Squared Error: ", root_mean_squared_error(y_test, y_test_pred))
print("Mean Absolute Error: ", mean_absolute_error(y_test, y_test_pred))
print("R2 Score: ", r2_score(y_test, y_test_pred))
print("Adjusted R2 Score: ", adjusted_r2(r2_score(y_test, y_test_pred), X_test.shape[0], X_test.shape[1]))
```

```
Mean Squared Error:  0.0035844118220144825
Root Mean Squared Error:  0.05986995759155407
Mean Absolute Error:  0.04077857503926597
R2 Score:  0.8247231382878004
Adjusted R2 Score:  0.8113868553314374
```

Observations

- We can see a slight increase in accuracy using Polynomial features

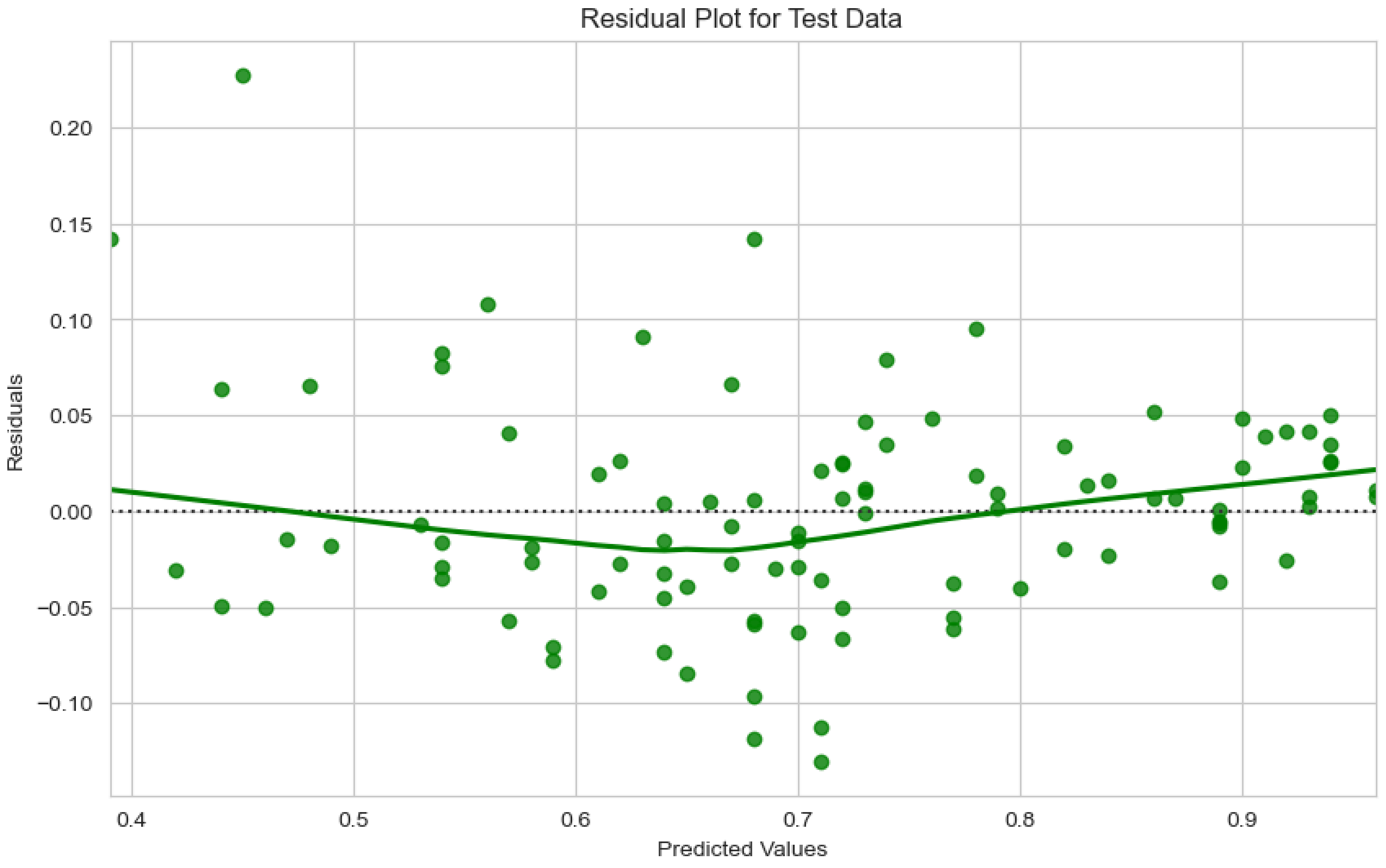
In [ ]:
```
# plot histogram of residuals
plt.figure(figsize=(10, 6))
sns.histplot(y_train-y_train_pred, kde=True)
plt.xlabel("Residuals")
plt.ylabel("Frequency")
plt.title("Histogram of Residuals");
```



Observations

We can see that the residuals follow a normal distribution

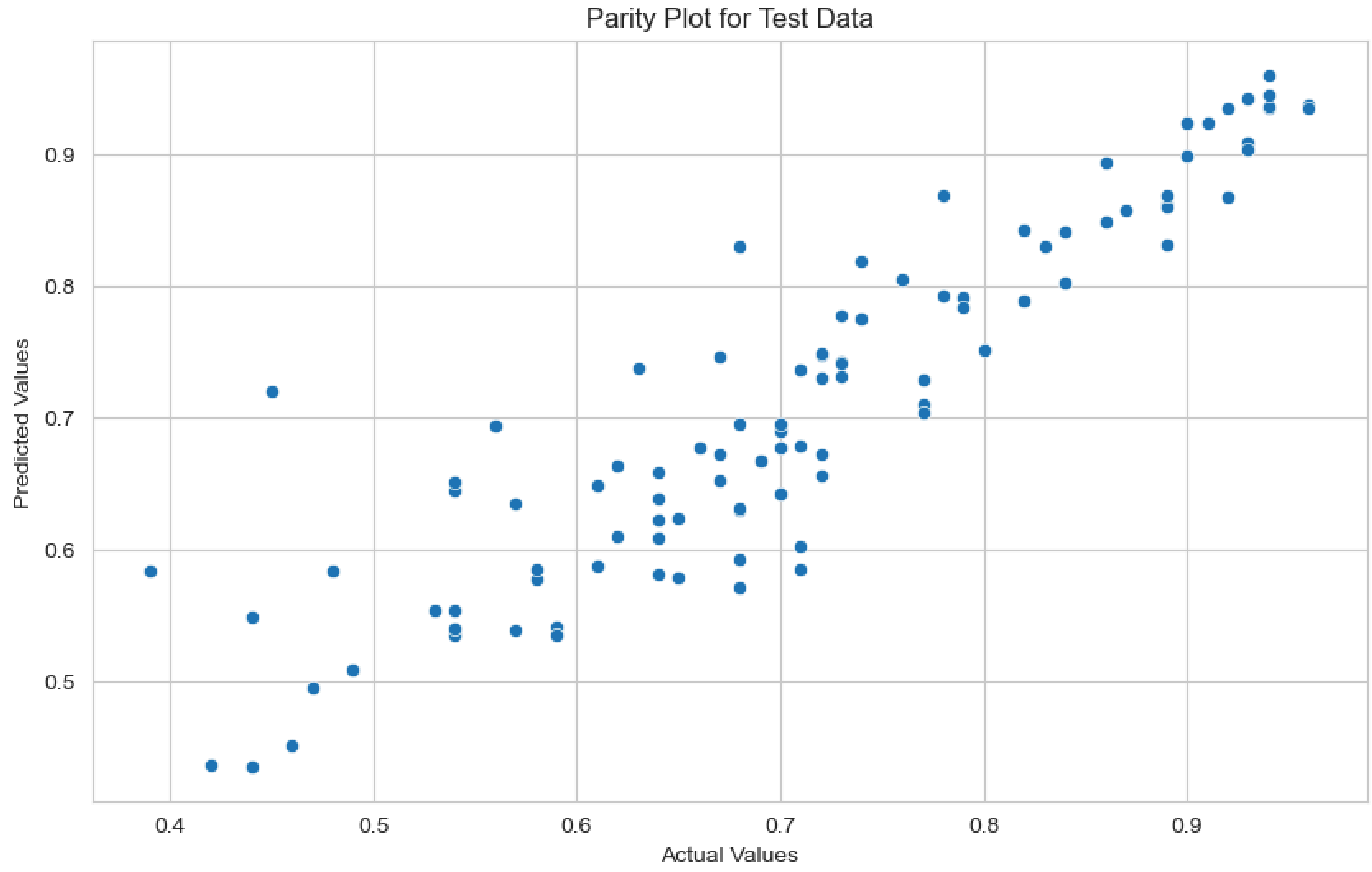
In [ ]:
```
plt.figure(figsize=(10, 6))
sns.residplot(x=y_test, y=y_test_pred, lowess=True, color="g")
plt.xlabel("Predicted Values")
plt.ylabel("Residuals")
plt.title("Residual Plot for Test Data")
plt.show()
```



Observations

From above plot we can see that residual are evenly distributed around the 0 line

In [ ]:
```
# plot a parity plot
plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_test, y=y_test_pred)
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("Parity Plot for Test Data")
plt.show()
```



Observations

- From above plots we can see that there is some variance at the start but decreases towards the end

Recommendations and Insights

For Jamboree

- The Lasso model with polynomial features seems to be best among all models with Adjusted R2 score around 0.82
- University ratings and Strenght of Letter of recomendation have no impact on the chance of admission

For Students

- From above analysis it is clear that smart student, ie students with high CGPA has a higher chance of admission.
- Students must be advised to increase their CGPA score, GRE scrore and TOEFL score as these factors increase the admission chance.