

▼ Exploratory Data Analysis

Contex:

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.

Column Profiling:

- 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)

Problem Statment:

Predict the chances of graduate admission based on the given features.

Solution Approach:

Understanding what factors are important in graduate admissions and how these factors are interrelated among themselves. So we can predict one's chances of admission using selected variables.

Concept Used:

Exploratory Data Analysis Linear Regression

Double-click (or enter) to edit

```
1 # Importing the libraries
2 import numpy as np
3 import pandas as pd
4 import matplotlib.pyplot as plt
5 import seaborn as sns
6
7 from sklearn.model_selection import train_test_split
8 from sklearn.preprocessing import MinMaxScaler, StandardScaler
9 from sklearn.linear_model import LinearRegression, Ridge, Lasso
10 from sklearn.metrics import r2_score
11
12 from statsmodels.stats.outliers_influence import variance_inflation_factor
13 from scipy import stats
```

```
1 #Loading the data set
2 df = pd.read_csv('/content/Jamboree_Admission.csv')
3 df.head()
```

	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

```
1 # Checking the shape of the data
2 df.shape
```

(500, 9)

```
1 # Checking name of the columns
2 df.columns
```

Index(['Serial No.', 'GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR ', 'CGPA', 'Research', 'Chance of Admit '], dtype='object')

```

1 # Correcting the name of the columns
2 df.rename(columns={'LOR ':'LOR','Chance of Admit ':'Chance of Admit'},inplace=True)
3 df.columns

```

```

Index(['Serial No.', 'GRE Score', 'TOEFL Score', 'University Rating', 'SOP',
      'LOR', 'CGPA', 'Research', 'Chance of Admit'],
      dtype='object')

```

```

1 df.info()
2 # It seems there is no need for changing the datatype of any column.
3 # Note: Although, ['University Rating','SOP','LOR'] columns are numeric,
4 # but we will consider it as an ordinal variable and perform similar analysis as we do for other categorical variables.

```

```

<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

```

```

1 df.isnull().sum()
2 # There are no missing values present in the dataset.

```

```

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

```

```

1 df.nunique()
2 # Serial No. column has all unique values, those values can be considered as row lable index.
3 # But for ML model we do not need Serial No. column so we can remove it.

```

```

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

```

```

1 df.drop("Serial No.",axis=1,inplace=True)
2 # Drop the unique row Identifier and unnecessary columns.
3 # We don't want our model to build some understanding based on row number.

```

```

1 cat_cols = ['University Rating', 'SOP', 'LOR', 'Research']
2 num_cols = ['GRE Score', 'TOEFL Score', 'CGPA']
3 target = 'Chance of Admit'

```

```


1 df.describe()
2 # Chance of Admit ranges from [0.34,0.97]
3 # GRE Score ranges from [290,340]
4 # TOEFL Score ranges from [92,120]
5 # CGPA ranges from [6.80,9.92]
6 # University Rating, SOP, LOR ranges from [0,5]
7 # Mean and Median values are apporximately equal for all columns.
8 # It seems there is no outliers or irrelevant datapoints in all columns.

```

```

    GRE Score      TOEFL Score      University Rating      SOP      LOR      CGPA      Research      Chi
of Ac
count 500 000000    500 000000    500 000000    500 000000    500 00000    500 000000    500 000000    500 000000

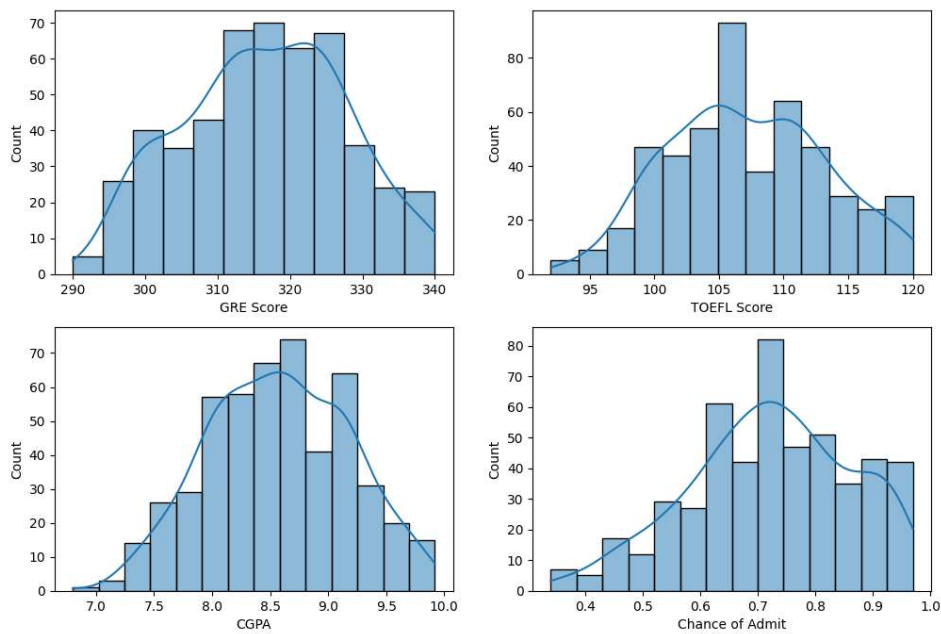
1 #Checking how the data is spread in categoric columns
2 df2=df.copy()
3 df2
4 categ_cols = cat_cols
5 cat_count = df2[categ_cols].melt().groupby(['variable', 'value'])['value'].size().reset_index(name='counts')
6 s = df2[categ_cols].melt().variable.value_counts()
7 cat_count['Percent'] = cat_count['counts'].div(cat_count['variable'].map(s)).mul(100).round().astype('int')
8 cat_count.groupby(['variable', 'value']).first()
9 # Applicants with Research 56%
10 # Applicants with no Research 44%
```

		counts	Percent	
	variable	value		
	LOR	1.0	1	0
		1.5	11	2
		2.0	46	9
		2.5	50	10
		3.0	99	20
		3.5	86	17
		4.0	94	19
		4.5	63	13
		5.0	50	10
	Research	0.0	220	44
		1.0	280	56
	SOP	1.0	6	1
		1.5	25	5
		2.0	43	9
		2.5	64	13
		3.0	80	16
		3.5	88	18
		4.0	89	18
		4.5	63	13
		5.0	42	8
	University Rating	1.0	34	7
		2.0	126	25
		3.0	162	32
		4.0	105	21
		5.0	73	15

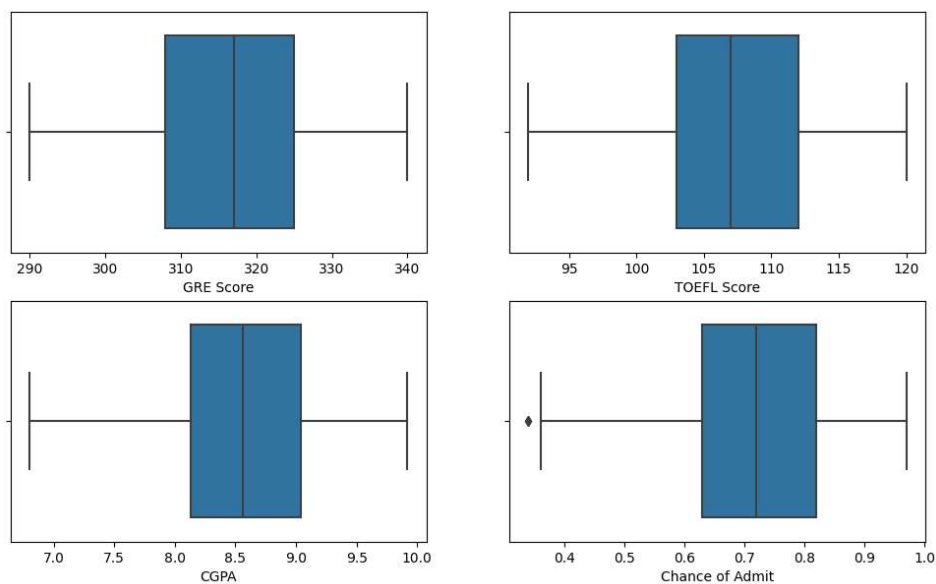
▼ Univariate Analysis

```

1 # check distribution of each numerical variable
2 rows, cols = 2, 2
3 fig, axs = plt.subplots(rows,cols, figsize=(12, 8))
4 index = 0
5 for row in range(rows):
6     for col in range(cols):
7         sns.histplot(df[num_cols[index]], kde=True, ax=axs[row,col])
8         index += 1
9     break
10
11 sns.histplot(df[num_cols[-1]], kde=True, ax=axs[1,0])
12 sns.histplot(df[target], kde=True, ax=axs[1,1])
13 plt.show()
14 # We can see approximately Normal distribution in all continous columns
```



```
1 # check for outliers using boxplots
2 rows, cols = 2, 2
3 fig, axs = plt.subplots(rows, cols, figsize=(12, 7))
4
5 index = 0
6 for col in range(cols):
7     sns.boxplot(x=num_cols[index], data=df, ax=axs[0,index])
8     index += 1
9
10 sns.boxplot(x=num_cols[-1], data=df, ax=axs[1,0])
11 sns.boxplot(x=target, data=df, ax=axs[1,1])
12 plt.show()
13 # We can see there is no outliers present in any column
```



```
1 # check unique values in categorical variables
2 for col in cat_cols:
3     print("Column: {:18} Unique values: {}".format(col, df[col].nunique()))
```

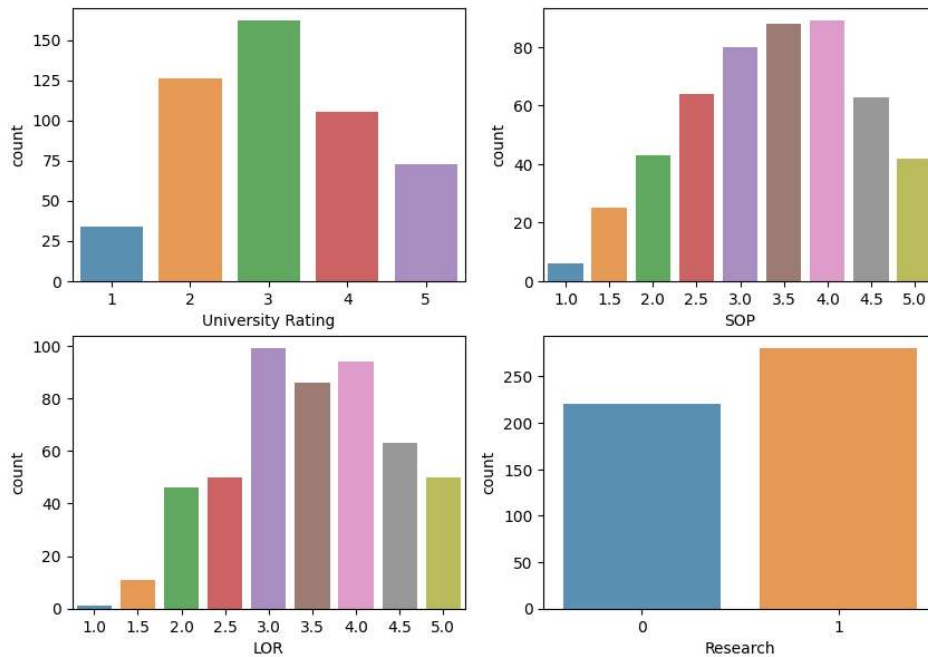
```
Column: University Rating Unique values: 5
Column: SOP               Unique values: 9
Column: LOR               Unique values: 9
Column: Research          Unique values: 2
```

```
1 # countplots for categorical variables
2 cols, rows = 2, 2
```

```

3 fig, axs = plt.subplots(rows, cols, figsize=(10, 7))
4
5 index = 0
6 for row in range(rows):
7     for col in range(cols):
8         sns.countplot(x=cat_cols[index], data=df, ax=axs[row, col], alpha=0.8)
9         index += 1
10
11 plt.show()
12
13 # Among students who have done research vs those who did not, 56 % said Yes and 44 % said No. We can make pie plot.

```

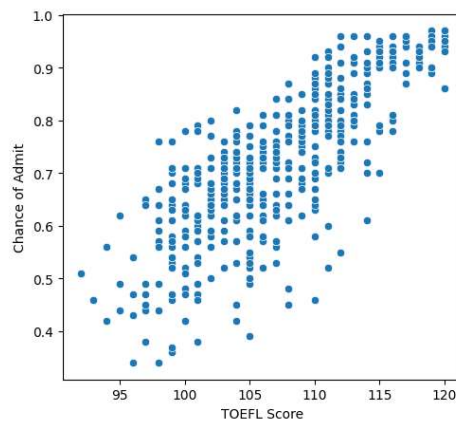
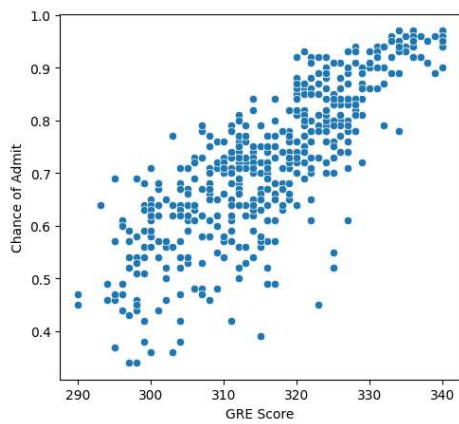


▼ Bivariate Analysis

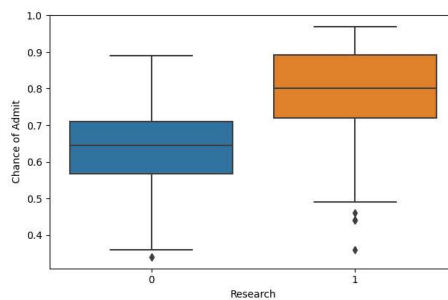
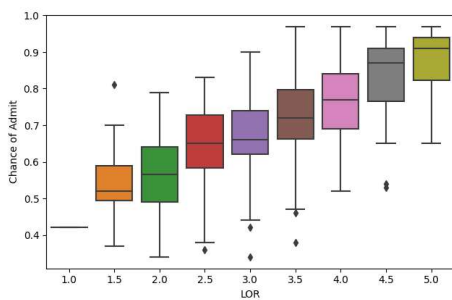
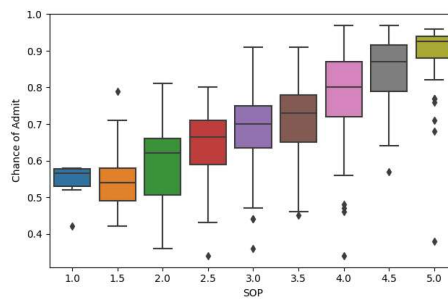
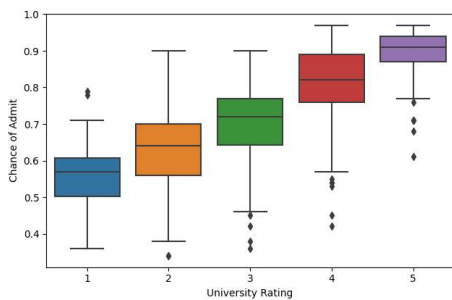
```

1 # check relation bw continuous variables & target variable. We can also use sns.regplot()
2 fig, axs = plt.subplots(1, 2, figsize=(12,5))
3
4 sns.scatterplot(x=num_cols[0], y=target, data=df, ax=axs[0])
5 sns.scatterplot(x=num_cols[1], y=target, data=df, ax=axs[1])
6 plt.show()
7 sns.scatterplot(x=num_cols[2], y=target, data=df)
8 plt.show()

```

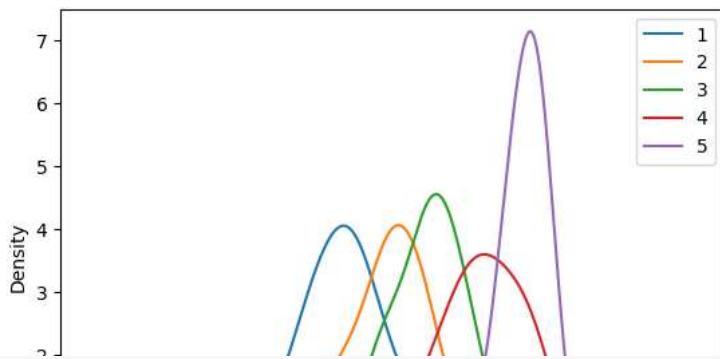


```
1 rows, cols = 2,2
2 fig, axs = plt.subplots(rows, cols, figsize=(16,10))
3
4 index = 0
5 for row in range(rows):
6     for col in range(cols):
7         sns.boxplot(x=cat_cols[index], y=target, data=df, ax=axs[row,col])
8         index += 1
```

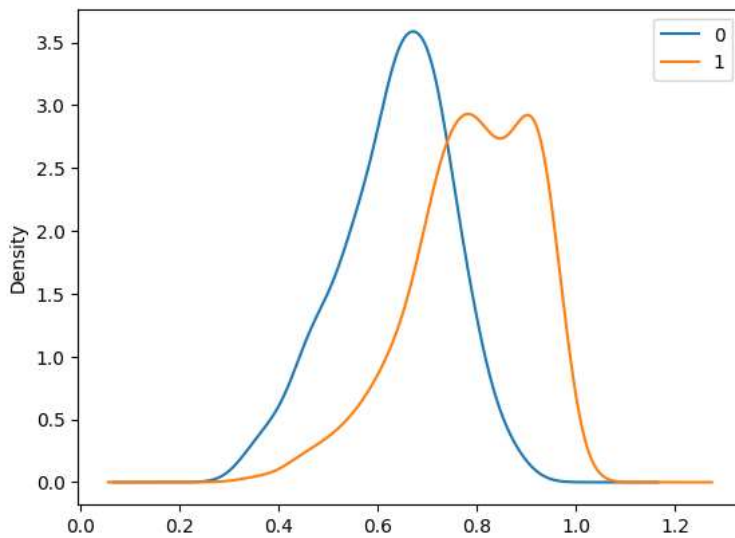


▼ Multivariate Analysis

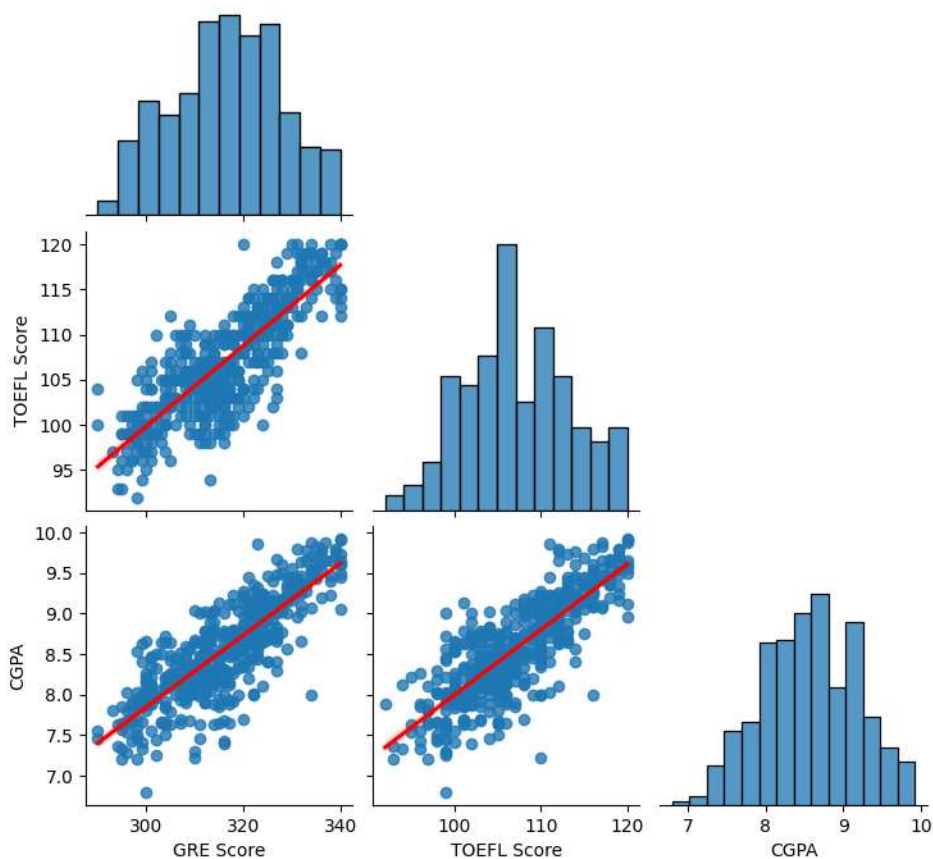
```
1 df.groupby('University Rating')['Chance of Admit'].plot.density();
2 plt.legend();
3 # Looks like for different University Rating, the Chance of Admit distribution is different.
4 # Hence one intuition that can be developed from this plot is that the University Rating variable may turn out to be very important for the ML model
```



```
1 df.groupby('Research')['Chance of Admit'].plot.density();
2 plt.legend();
3 # Looks like for different Research type, the Chance of Admit distrubution is different.
4 # Hence one intuition that can be developed from this plot is that the Researcg variable will not turn out to be very important for the ML model. (W
```



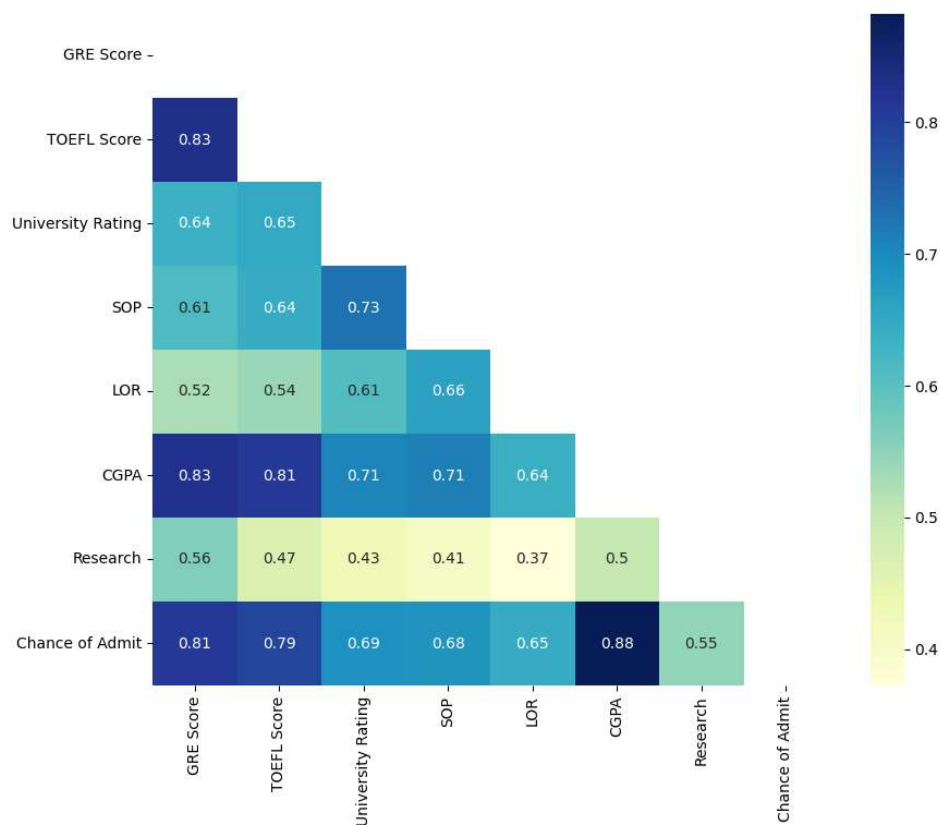
```
1 sns.pairplot(df[num_cols], kind="reg", plot_kws={'line_kws':{'color':'red'}}, corner=True)
2 plt.show()
3 # All variables are in positive correlation with Chance of Admit variable.
4 # IF value of any of variable increase then Chance of Admit will also increase.
```



```

1 # creating mask for plotting a triangle correlation heatmap
2 plt.figure(figsize=(10,8))
3 mask = np.triu(np.ones_like(df.corr()))
4 sns.heatmap(df.corr(), cmap="YlGnBu", annot=True, mask=mask)
5 plt.show()
6 # Checking correlation among independent variables and how they interact with each other.
7 # CGPA Score, GRE Score, TOEFL Score variables are highly correlated with each other.
8 # University Rating, SOP LOR variables have some correlation with each other.

```



▼ Data Preprocessing

```

1 # check for duplicates
2 df.duplicated().sum()

```

0

```

1 # check for missing values
2 df.isna().sum().sum()

```

0

```

1 # There are no missing values, outliers and duplicates present in the dataset.

```

```

1 #Data preparation for model building
2 X = df.drop(columns=[target])
3 y = df[target]

```

```

1 # standardize the dataset
2 sc = StandardScaler()
3 X = sc.fit_transform(X)

```

```

1 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)

```

```

1 print(X_train.shape, y_train.shape)
2 print(X_test.shape, y_test.shape)

```

```

(350, 7) (350,)
(150, 7) (150,)

```


▼ LR Model Building

```
1 def adjusted_r2(r2, p, n):
2     """
3     n: no of samples
4     p: no of predictors
5     r2: r2 score
6     """
7     adj_r2 = 1 - ((1-r2)*(n-1) / (n-p-1))
8     return adj_r2
9
10 def get_metrics(y_true, y_pred, p=None):
11     n = y_true.shape[0]
12     mse = np.sum((y_true - y_pred)**2) / n
13     rmse = np.sqrt(mse)
14     mae = np.mean(np.abs(y_true - y_pred))
15     score = r2_score(y_true, y_pred)
16     adj_r2 = None
17     if p is not None:
18         adj_r2 = adjusted_r2(score, p, n)
19
20     res = {
21         "mean_absolute_error": round(mae, 2),
22         "rmse": round(rmse, 2),
23         "r2_score": round(score, 2),
24         "adj_r2": round(adj_r2, 2)
25     }
26     return res
```

```
1 def train_model(X_train, y_train, X_test, y_test,cols, model_name="linear", alpha=1.0):
2     model = None
3     if model_name == "lasso":
4         model = Lasso(alpha=alpha)
5     elif model_name == "ridge":
6         model = Ridge(alpha=alpha)
7     else:
8         model = LinearRegression()
9
10    model.fit(X_train, y_train)
11    y_pred_train = model.predict(X_train)
12    y_pred_test = model.predict(X_test)
13    p = X_train.shape[1]
14    train_res = get_metrics(y_train, y_pred_train, p)
15    test_res = get_metrics(y_test, y_pred_test, p)
16
17    print(f"\n---    {model_name.title()} Regression Model    ---\n")
18    print(f"Train MAE: {train_res['mean_absolute_error']} Test MAE: {test_res['mean_absolute_error']}")
19    print(f"Train RMSE: {train_res['rmse']} Test RMSE: {test_res['rmse']}")
20    print(f"Train R2_score: {train_res['r2_score']} Test R2_score: {test_res['r2_score']}")
21    print(f"Train Adjusted_R2: {train_res['adj_r2']} Test Adjusted_R2: {test_res['adj_r2']}")
22    print(f"Intercept: {model.intercept_}")
23    #print(len(df.columns), len(model.coef_))
24    coef_df = pd.DataFrame({"Column": cols, "Coef": model.coef_})
25    print(coef_df)
26    print("-"*50)
27    return model
```

```
1 train_model(X_train, y_train, X_test, y_test,df.columns[:-1], "linear")
2 train_model(X_train, y_train, X_test, y_test,df.columns[:-1], "ridge")
3 train_model(X_train, y_train, X_test, y_test,df.columns[:-1], "lasso", 0.001)
```

---- Linear Regression Model ----

```
Train MAE: 0.04 Test MAE: 0.04
Train RMSE: 0.06 Test RMSE: 0.06
Train R2_score: 0.82 Test R2_score: 0.82
Train Adjusted_R2: 0.82 Test Adjusted_R2: 0.81
Intercept: 0.724978121476996
      Column      Coef
0      GRE Score  0.018657
1      TOEFL Score 0.023176
2 University Rating 0.011565
3          SOP -0.000999
4          LOR  0.012497
5          CGPA  0.064671
6      Research  0.013968
```

---- Ridge Regression Model ----

```
Train MAE: 0.04 Test MAE: 0.04
Train RMSE: 0.06 Test RMSE: 0.06
Train R2_score: 0.82 Test R2_score: 0.82
Train Adjusted_R2: 0.82 Test Adjusted_R2: 0.81
Intercept: 0.7249823645841696
      Column      Coef
0      GRE Score  0.018902
1      TOEFL Score 0.023252
2 University Rating 0.011594
3          SOP -0.000798
4          LOR  0.012539
5          CGPA  0.064004
```

- 1 # Since model is not overfitting, Results for Linear, Ridge and Lasso are the same.
- 2 # R2_score and Adjusted_r2 are almost the same. Hence there are no unnecessary independent variables in the data.

---- Lasso Regression Model ----

```
1 # Statmodels implementation of Linear regression
2
3 import statsmodels.api as sm
4
5 X_sm = sm.add_constant(X) #Statmodels default is without intercept, to add intercept we need to add constant
6
7 sm_model = sm.OLS(y, X_sm).fit()
8
9 print(sm_model.summary())
```

OLS Regression Results

```
=====
Dep. Variable:      Chance of Admit    R-squared:                0.822
Model:              OLS                Adj. R-squared:          0.819
Method:             Least Squares      F-statistic:             324.4
Date:               Wed, 14 Jun 2023    Prob (F-statistic):      8.21e-180
Time:               17:52:27            Log-Likelihood:          701.38
No. Observations:   500                AIC:                   -1387.
Df Residuals:       492                BIC:                   -1353.
Df Model:           7
Covariance Type:    nonrobust
=====
              coef      std err          t      P>|t|      [0.025      0.975]
-----
const         0.7217      0.003     269.039      0.000      0.716      0.727
x1            0.0210      0.006      3.700      0.000      0.010      0.032
x2            0.0169      0.005      3.184      0.002      0.006      0.027
x3            0.0068      0.004      1.563      0.119     -0.002      0.015
x4            0.0016      0.005      0.348      0.728     -0.007      0.010
x5            0.0156      0.004      4.074      0.000      0.008      0.023
x6            0.0715      0.006     12.198      0.000      0.060      0.083
x7            0.0121      0.003      3.680      0.000      0.006      0.019
=====
Omnibus:                 112.770    Durbin-Watson:           0.796
Prob(Omnibus):            0.000    Jarque-Bera (JB):         262.104
Skew:                    -1.160    Prob(JB):                 1.22e-57
Kurtosis:                 5.684    Cond. No.                  5.65
=====
```

Notes:

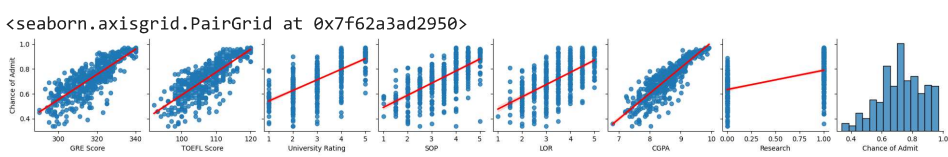
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

▼ LR Assumption Tests

▼ Linearity of variables

```
1 # It is quite clear from EDA and visual analysis that independent variables are linearly dependent on the target variables.
```

```
1 sns.pairplot(df, y_vars=["Chance of Admit"], kind="reg", plot_kws={'line_kws':{'color':'red'}})
2
3 # All variables are in positive correlation with Chance of Admit variable.
4 # IF value of any of variable increase then Chance of Admit will also increase.
```

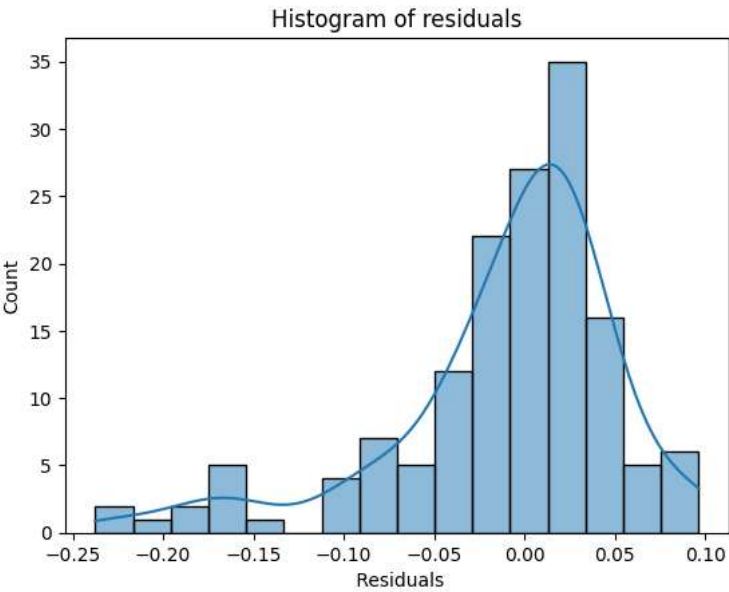


▼ Normality of Residuals

```

1 model = LinearRegression()
2 model.fit(X_train, y_train)
3 y_pred = model.predict(X_test)
4 residuals = (y_test - y_pred)
5 sns.histplot(residuals, kde=True)
6 plt.xlabel(" Residuals")
7 plt.title("Histogram of residuals")
8 plt.show()

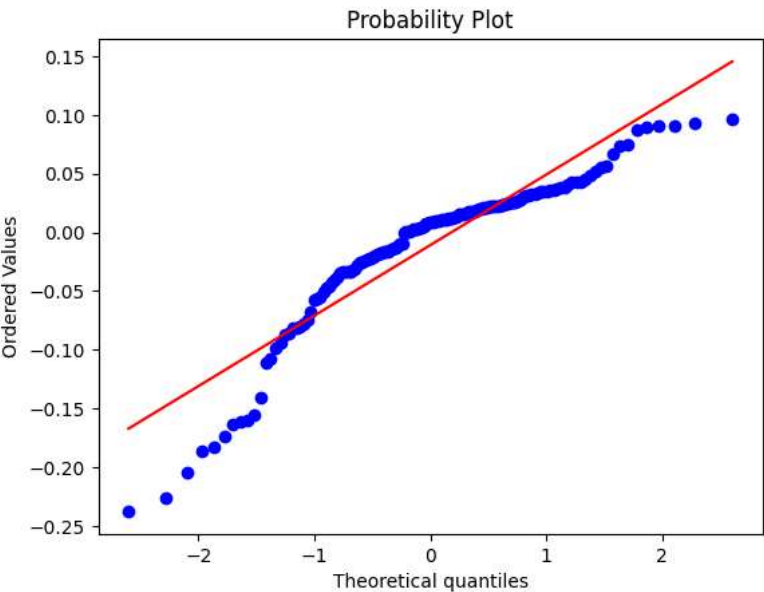
```



```

1 stats.probplot(residuals, plot=plt)
2 plt.show()
3 # Data that aligns closely to the dotted line indicates a normal distribution.

```



▼ Mean of Residuals

```

1 # It is clear from RMSE that Mean of Residuals is almost zero.
2 abs(residuals.mean())

0.010793738256654513

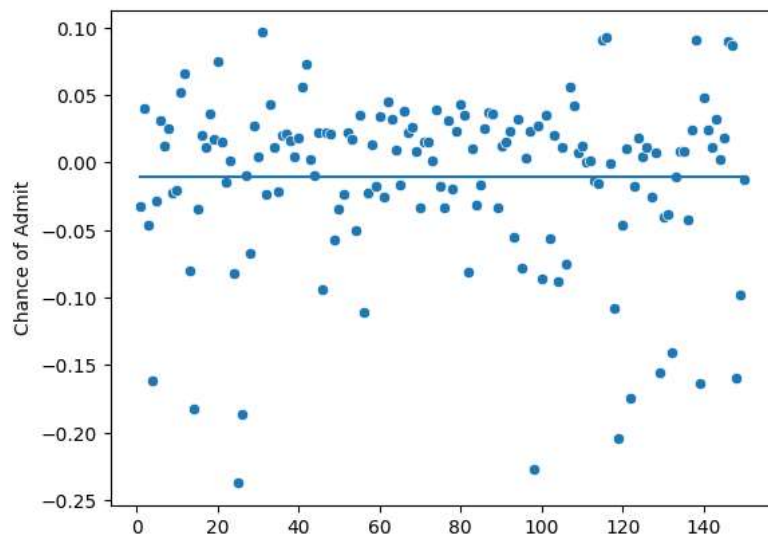
```

```

1 sns.scatterplot(x=np.arange(1,len(y_test)+1,1),y=residuals)
2 sns.lineplot(x=np.arange(1,len(y_test)+1,1),y=residuals.mean())
3 # They are pretty symmetrically distributed

```

<Axes: ylabel='Chance of Admit'>

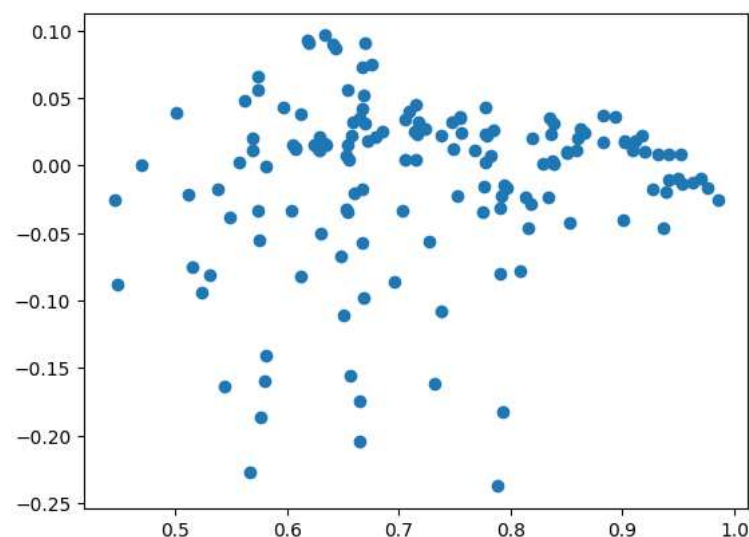


▼ Test for Homoscedasticity

```

1 plt.scatter(y_pred, residuals)
2 plt.show()

```



```

1 # Since the plot is not creating a cone type shape. Hence there is no homoscedasticity present in the data.

```

▼ Muticollinearity Check

```

1 def vif(newdf):
2     # VIF dataframe
3     vif_data = pd.DataFrame()
4     vif_data["feature"] = newdf.columns
5
6     # calculating VIF for each feature
7     vif_data["VIF"] = [variance_inflation_factor(newdf.values, i) for i in range(len(newdf.columns))]
8     return vif_data

```

```

1 res = vif(df.iloc[:, :-1])
2 res

```

	feature	VIF
--	---------	-----



0	GRF Score	1308.061089
---	-----------	-------------

```
1 # drop GRE Score and again calculate the VIF
2 res = vif(df.iloc[:, 1:-1])
3 res
```

	feature	VIF
--	---------	-----



0	TOEFL Score	639.741892
1	University Rating	19.884298
2	SOP	33.733613
3	LOR	30.631503
4	CGPA	728.778312
5	Research	2.863301

```
1 # # drop TOEFL Score and again calculate the VIF
2 res = vif(df.iloc[:,2:-1])
3 res
```

	feature	VIF
--	---------	-----



0	University Rating	19.777410
1	SOP	33.625178
2	LOR	30.356252
3	CGPA	25.101796
4	Research	2.842227

```
1 # Now lets drop the SOP and again calculate VIF
2 res = vif(df.iloc[:,2:-1].drop(columns=['SOP']))
3 res
```

	feature	VIF
--	---------	-----



0	University Rating	15.140770
1	LOR	26.918495
2	CGPA	22.369655
3	Research	2.819171

```
1 # lets drop the LOR as well
2 newdf = df.iloc[:,2:-1].drop(columns=['SOP'])
3 newdf = newdf.drop(columns=['LOR'], axis=1)
4 res = vif(newdf)
5 res
```

	feature	VIF
--	---------	-----



0	University Rating	12.498400
1	CGPA	11.040746
2	Research	2.783179

```
1 # drop the University Rating
2 newdf = newdf.drop(columns=['University Rating'])
3 res = vif(newdf)
4 res
```

	feature	VIF
--	---------	-----



0	CGPA	2.455008
1	Research	2.455008

▼ Model performance evaluation

```
1 # now again train the model with these only two features
2 X = df[['CGPA', 'Research']]
3 sc = StandardScaler()
4 X = sc.fit_transform(X)
5
6 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
```

```

1 model = train_model(X_train, y_train, X_test, y_test, ['CGPA', 'Research'], "linear")
2 train_model(X_train, y_train, X_test, y_test, ['CGPA', 'Research'], "ridge")
3 train_model(X_train, y_train, X_test, y_test, ['CGPA', 'Research'], "lasso", 0.001)

```

---- Linear Regression Model ----

```

Train MAE: 0.05 Test MAE: 0.05
Train RMSE: 0.06 Test RMSE: 0.07
Train R2_score: 0.78 Test R2_score: 0.81
Train Adjusted_R2: 0.78 Test Adjusted_R2: 0.81
Intercept: 0.7247774222727991
      Column      Coef
0      CGPA  0.112050
1  Research  0.020205
-----

```

---- Ridge Regression Model ----

```

Train MAE: 0.05 Test MAE: 0.05
Train RMSE: 0.06 Test RMSE: 0.07
Train R2_score: 0.78 Test R2_score: 0.81
Train Adjusted_R2: 0.78 Test Adjusted_R2: 0.81
Intercept: 0.7247830300095277
      Column      Coef
0      CGPA  0.111630
1  Research  0.020362
-----

```

---- Lasso Regression Model ----

```

Train MAE: 0.05 Test MAE: 0.05
Train RMSE: 0.06 Test RMSE: 0.07
Train R2_score: 0.78 Test R2_score: 0.81
Train Adjusted_R2: 0.78 Test Adjusted_R2: 0.81
Intercept: 0.7247713356661623
      Column      Coef
0      CGPA  0.111344
1  Research  0.019571
-----

```

▼ Lasso
Lasso(alpha=0.001)

```

1 # After removing collinear features using VIF and using only two features.
2 # R2_score and Adjusted_r2 are still the same as before the testing dataset.

```

▼ Insights

- Based on the EDA: There are no missing values, outliers and duplicates present in the dataset.
- Among students who have done research vs those who did not, 56 % said Yes and 44 % said No
- More than 50% of the data has a university rating of 3 or 2
- A majority of students (56%) have letter of recommendation values between 3.0 and 4.5
- A strong positive relationship exists between Chance of admit and numerical variables (GRE & TOEFL score and CGPA).
- GRE Score , TOFEL Score and CGPA are highly correlated (0.80). We should drop two of these.
- Based on the analysis an upward trend for each categorical variable. A higher rating value increases the chance of admission.
- Independent variables are linearly correlated with dependent variables.
- Multicollinearity was present in the within independent variables.
- After removing collinear features there are only two variables which are important in making predictions for the target variables.
- Following are the final model results on the test data:
- RMSE: 0.07
- MAE: 0.05
- R2_score: 0.81
- Adjusted_R2: 0.81

▼ Recommendations

- CGPA and Research are two variables which are important in making the prediction for Chance of Admit.
- CGPA is the most important variable in making the prediction for the Chance of Admit.

- from CGPA, TOEFL Score, GRE Score any one of varibale can be helpful in the prediction for the Chance of Admit.
- We can use any of model like Linear regression, Ridge regression or Lasso regression etc. All those models have shown appoximately similar performace.
- We can use any of above model, where students/learners can come to the website and check their probability of getting into the admission in IVY league college.

✓ 0s completed at 11:25 PM

