Admission chances prediction

```
#importing necessary libaries
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline
import seaborn as sns
```

Double-click (or enter) to edit

```
df = pd.read_csv("/content/admission_predict (1).csv")
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

```
df.shape
```

```
(500, 9)
```

```
df.rename(columns={'University Rating': 'UniversityRating'}, inplace=True)
```

```
#checking the name of columns
df.columns
```

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

```
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   UniversityRating  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
```

```
df.describe()
```

	Serial No.	GRE Score	TOEFL Score	UniversityRating	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

```
# Returns true for a column having null values, else false
df.isnull().any()
```

```
Serial No.           False
GRE Score            False
TOEFL Score          False
UniversityRating     False
SOP                  False
LOR                  False
CGPA                 False
Research             False
Chance of Admit      False
dtype: bool
```

```
# Returns different datatypes for each columns (float, int, string, bool, etc.)
df.dtypes
```

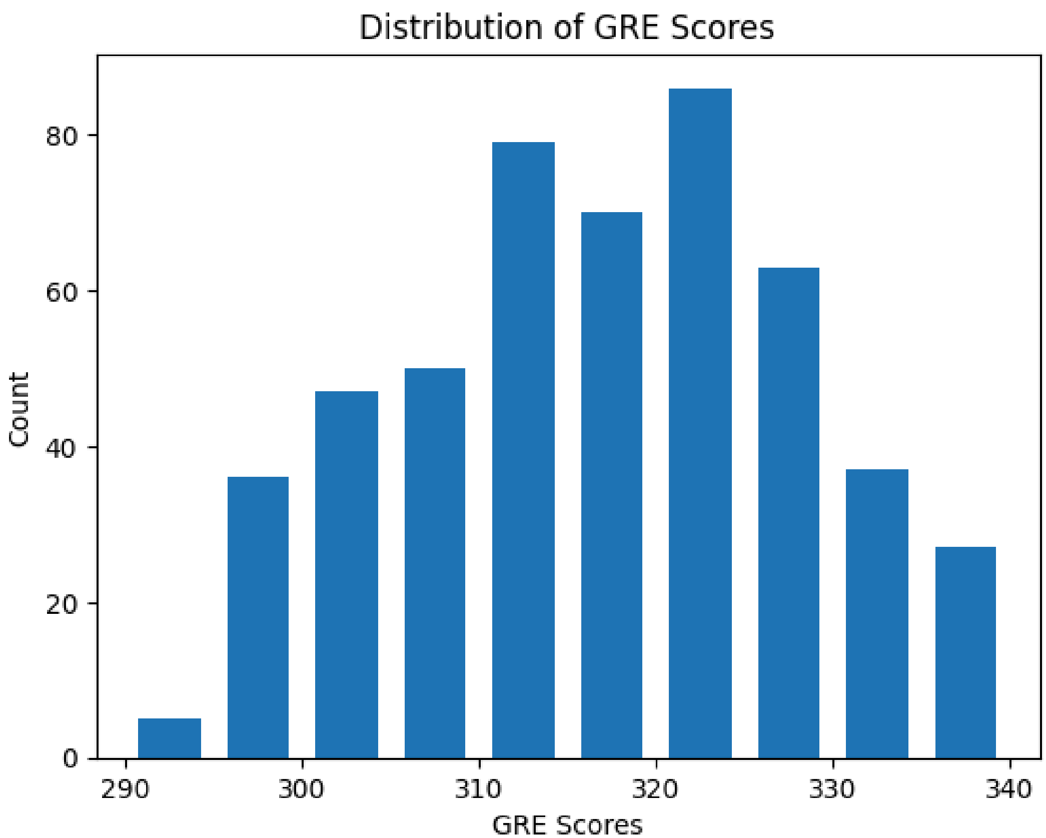
```
Serial No.             int64
GRE Score              int64
TOEFL Score            int64
UniversityRating       int64
SOP                  float64
LOR                  float64
CGPA                 float64
Research               int64
Chance of Admit      float64
dtype: object
```

```
# Renaming the columns with appropriate names
df = df.rename(columns={'GRE Score': 'GRE', 'TOEFL Score': 'TOEFL', 'LOR ': 'LOR', 'Chance of Admit ': 'Probability'})
df.head()
```

	Serial No.	GRE	TOEFL	UniversityRating	SOP	LOR	CGPA	Research	Probability
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

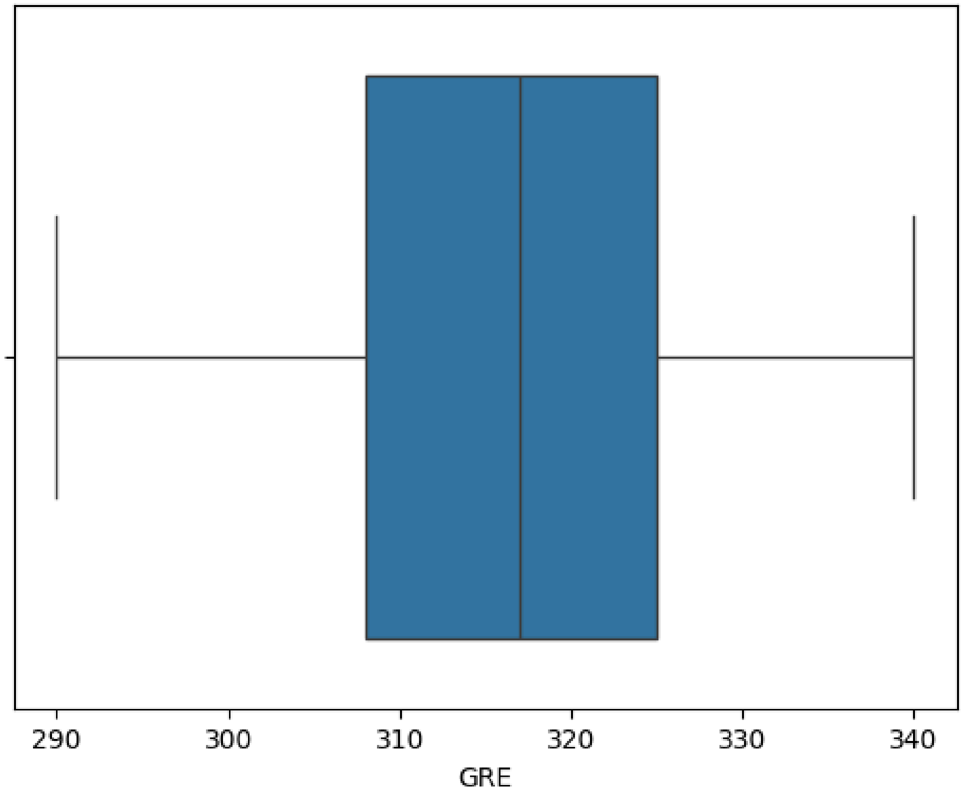
Univariate analysis of columns

```
# Visualizing the feature GRE
fig = plt.hist(df['GRE'], rwidth=0.7)
plt.title("Distribution of GRE Scores")
plt.xlabel('GRE Scores')
plt.ylabel('Count')
plt.show()
```



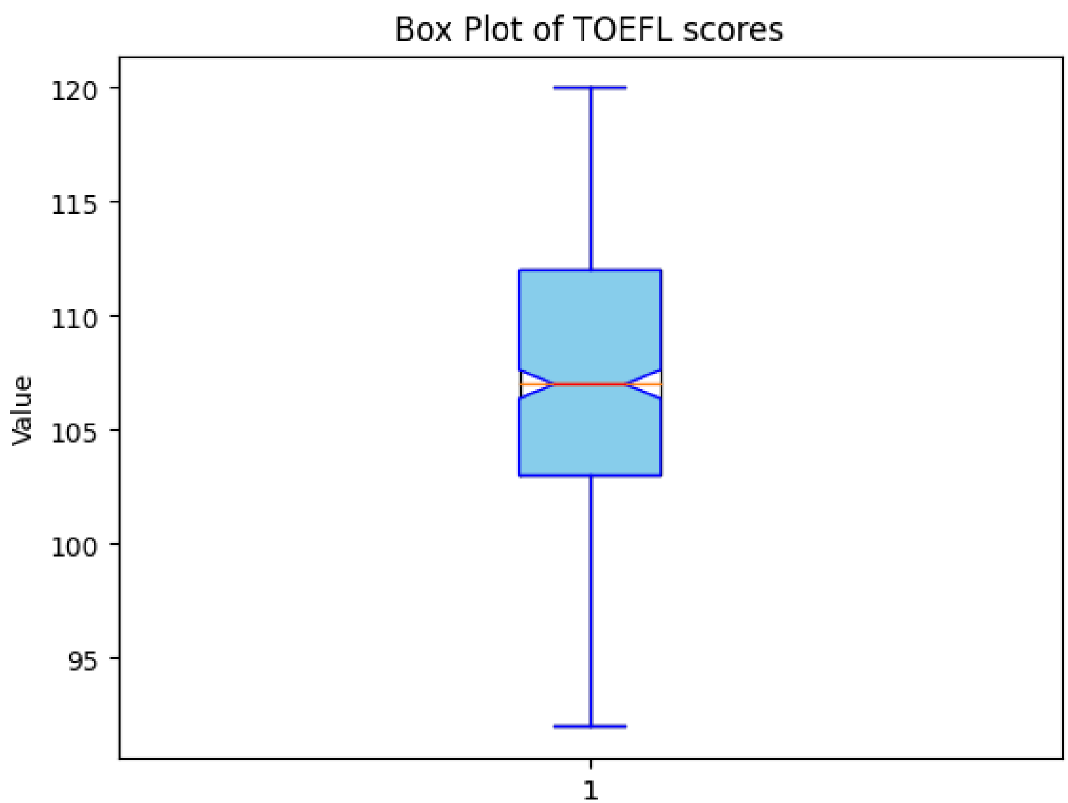

```
sns.boxplot(x =df['GRE'])
```

<Axes: xlabel='GRE'>

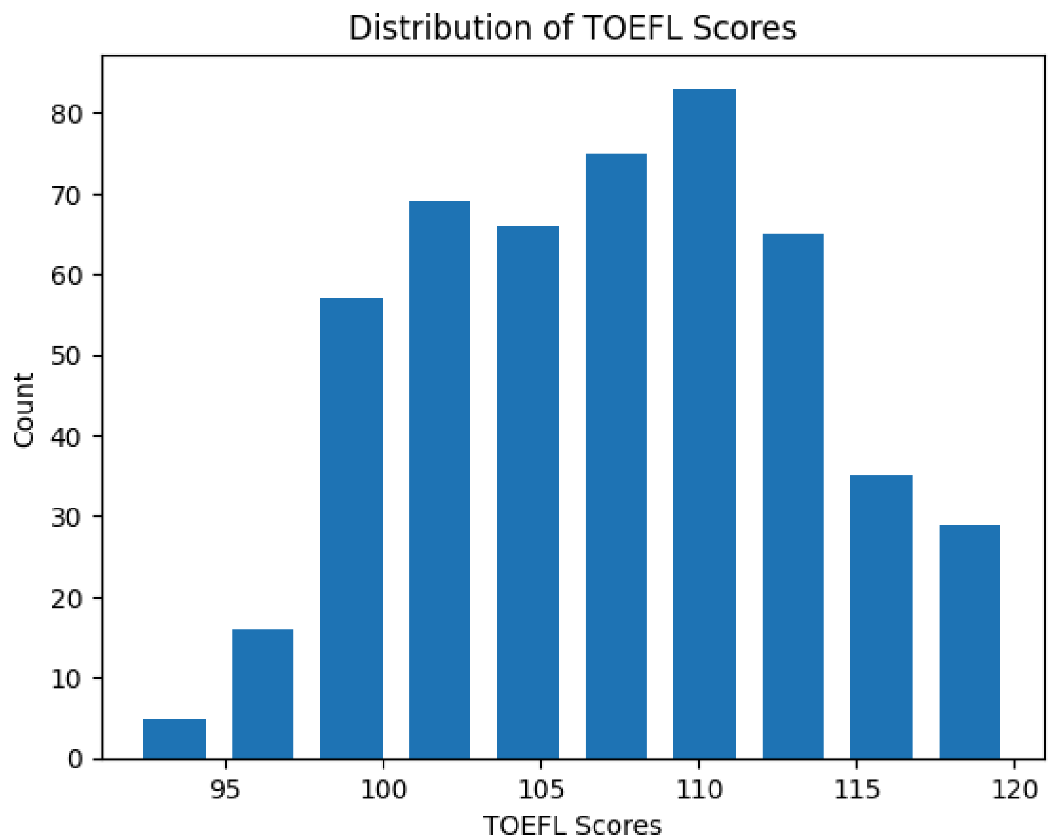


The above boxplot describes the summary statistics of the GRE column,a half of the candidates had scoles less than the 310-320 range while the other half had above that range(i.e median score). The highest score being 340 and 290 being the lowest.

```
#univariate analysis of TOEFL score column
plt.boxplot(df["TOEFL"])

# Add title and labels
plt.title('Box Plot of TOEFL scores')
plt.ylabel('Value')

plt.boxplot(df["TOEFL"], notch=True, patch_artist=True,
            boxprops=dict(facecolor='skyblue', color='blue'),
            capprops=dict(color='blue'),
            whiskerprops=dict(color='blue'),
            flierprops=dict(color='red', markeredgecolor='red'),
            medianprops=dict(color='red'))


# Display the plot
plt.show()
```



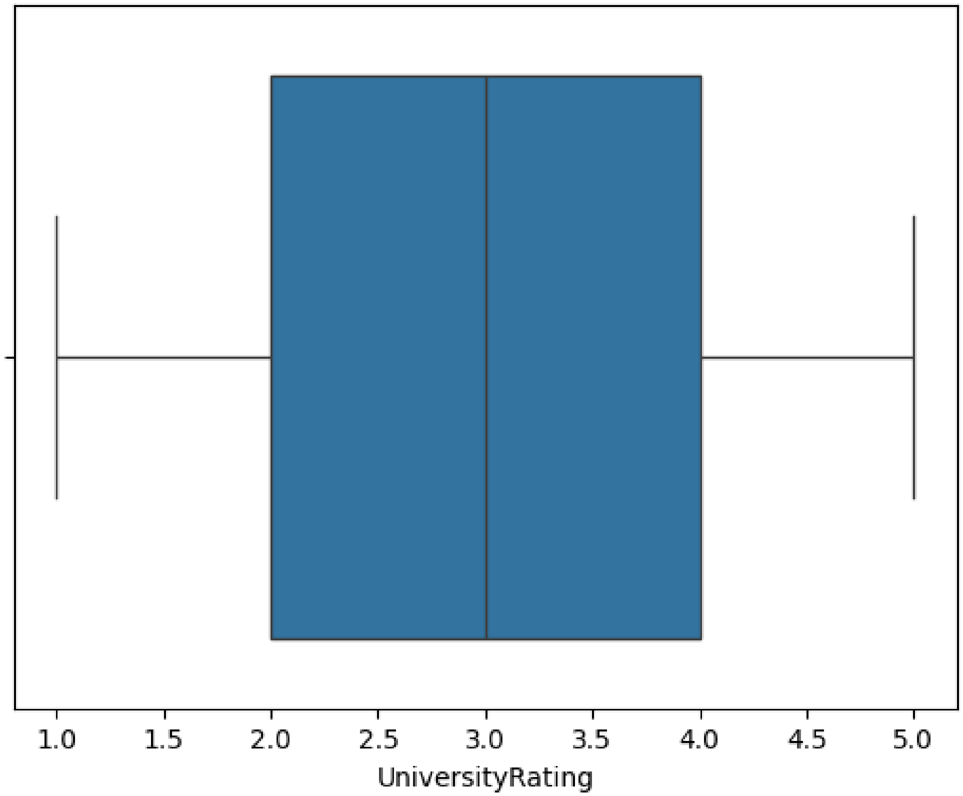
```
fig = plt.hist(df['TOEFL'], rwidth=0.7)
plt.title('Distribution of TOEFL Scores')
plt.xlabel('TOEFL Scores')
plt.ylabel('Count')
plt.show()
```



About 80 candidates had a score of 110 with fewer than 10 students having less than 95.

```
sns.boxplot(x =df['UniversityRating'])
```

```
<Axes: xlabel='UniversityRating'>
```



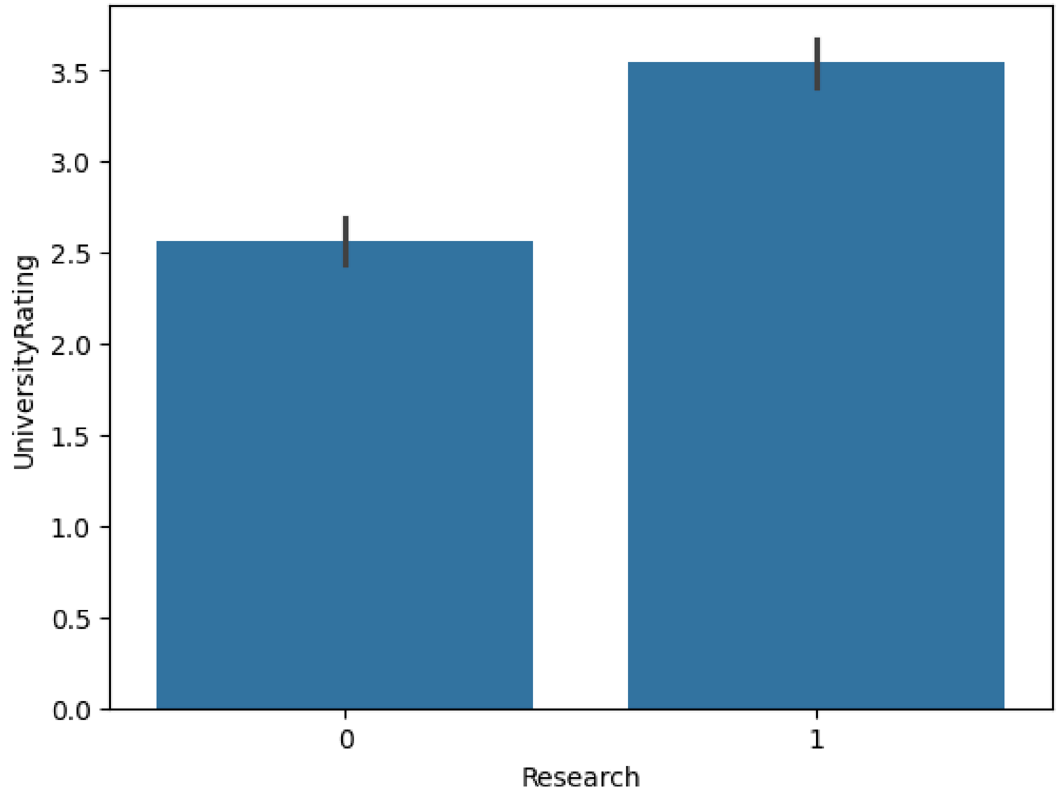

```
df['UniversityRating'].describe()
```

```
count    500.000000
mean       3.114000
std        1.143512
min        1.000000
25%        2.000000
50%        3.000000
75%        4.000000
max        5.000000
Name: UniversityRating, dtype: float64
```

Bivariate analysis

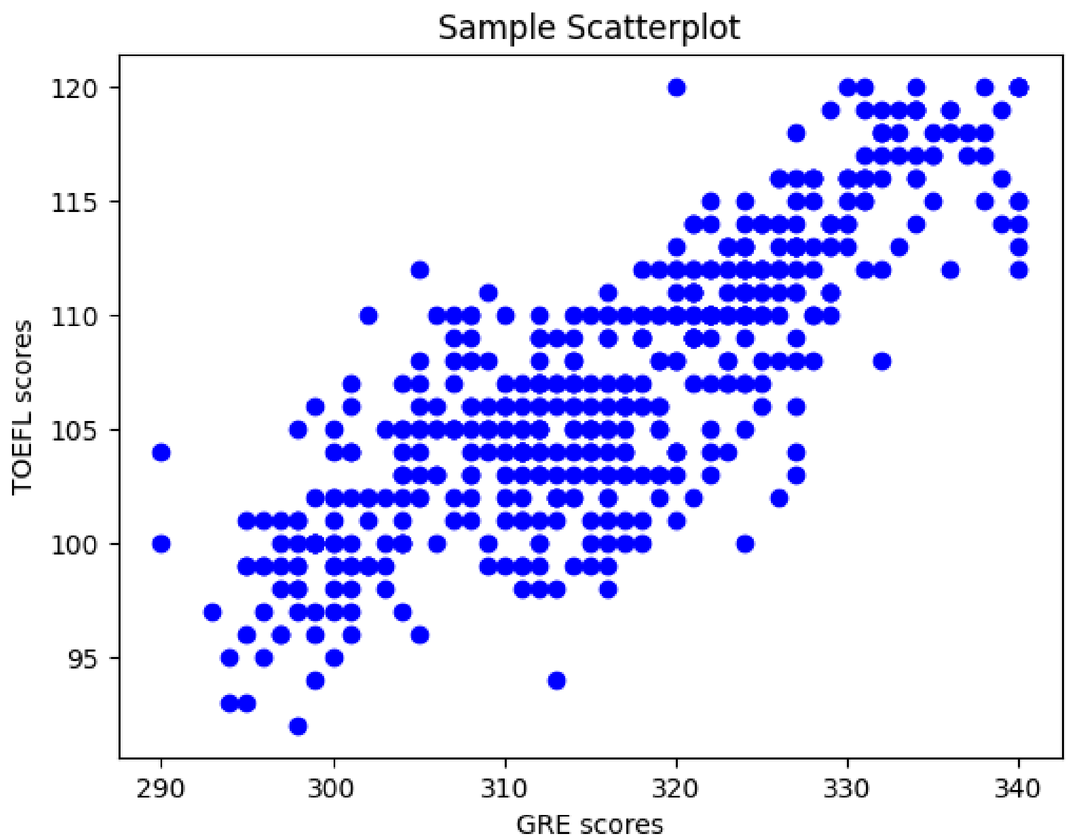
This examines the correlation or relationship between two columns or features. For example, a school with high rating is expected to produce students with good research exposure

```
#plotting the relationship between research and school rating
sns.barplot(x= "Research", y = "UniversityRating", data=df)
plt.show()
```



As expected, students from schools with high rating has better research experience.

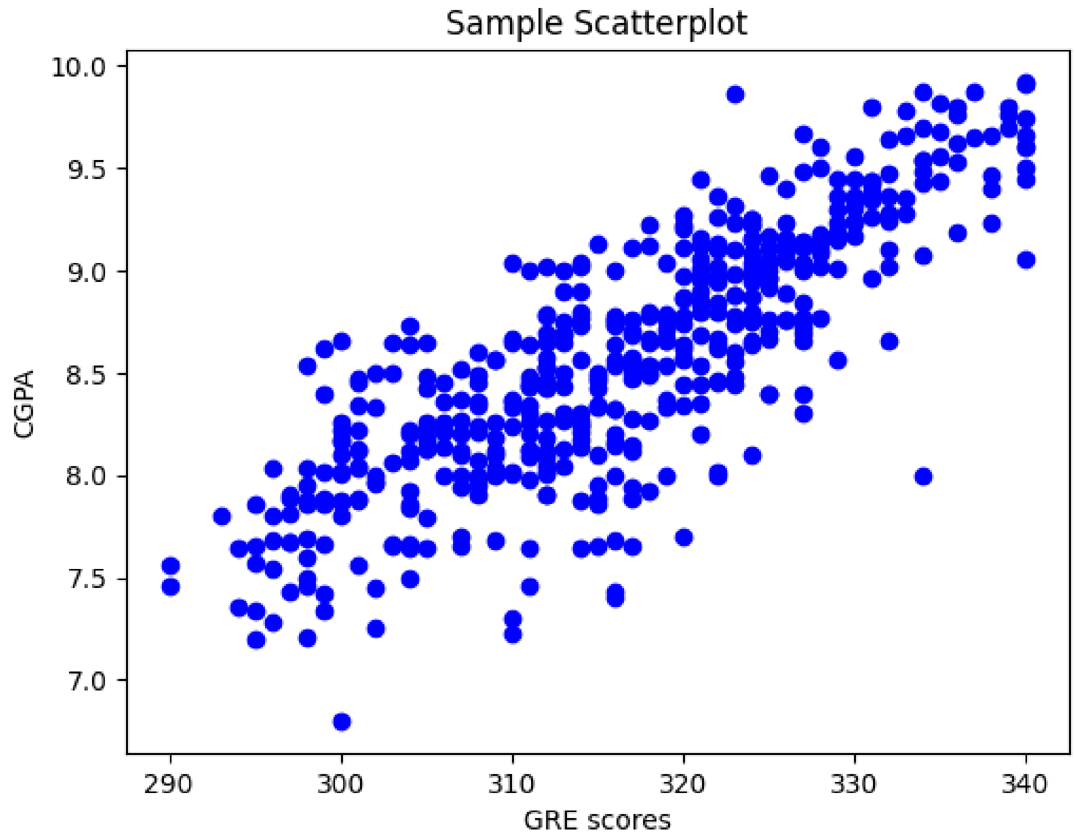
```
#examining the relationship between GRE and TOEFL scores of candidates
plt.scatter(x  = df["GRE"], y = df["TOEFL"], color='blue', marker='o')

# Add title and labels
plt.title('Sample Scatterplot')
plt.xlabel('GRE scores')
plt.ylabel('TOEFL scores')
plt.show()
```




```
Start coding or generate with AI.
```

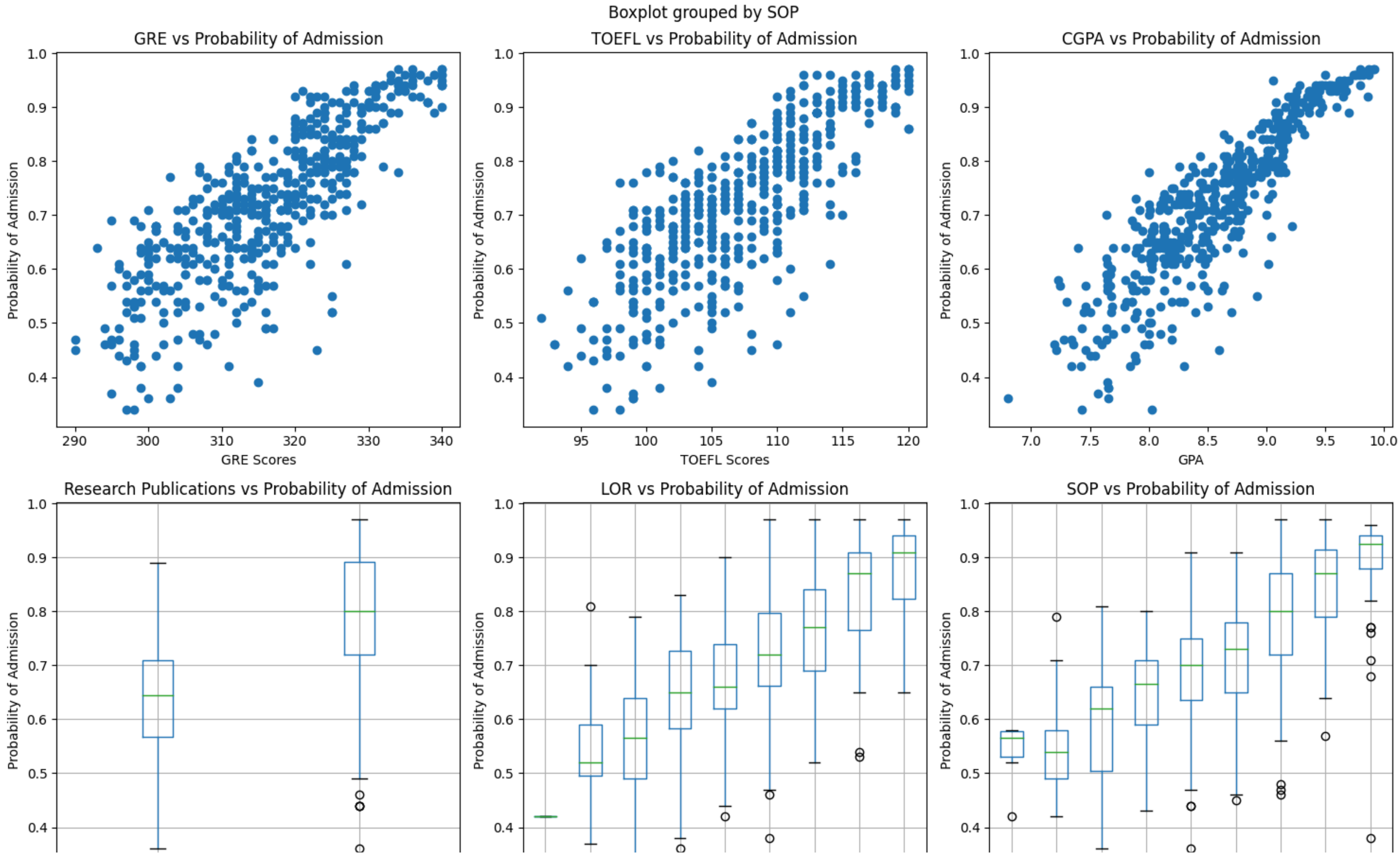
The above scatter plot indicates a positive linear relationship between the GRE and TOEFL scores of candidates, meaning that with candidates with high GRE scores also most likely have high TOEFL scores.

```
#GRESCORES VS CGPA comparison
plt.scatter(x  = df["GRE"], y = df["CGPA"], color='blue', marker='o')

# Add title and labels
plt.title('Sample Scatterplot')
plt.xlabel('GRE scores')
plt.ylabel('CGPA')
plt.show()
```



There is also a positive linear rellationship between GRE SCORES and CGPA, therefore, candidates with high cgpa will most likely have high GRE scores

```
#checking the relationship between various admission requirement and chance of admit
fig, axs = plt.subplots(2, 3, figsize=(15, 10))

# Scatter plot for GRE vs Probability of Admission
axs[0, 0].scatter(df['GRE'], df['Probability'])
axs[0, 0].set_title('GRE vs Probability of Admission')
axs[0, 0].set_xlabel('GRE Scores')
axs[0, 0].set_ylabel('Probability of Admission')

# Scatter plot for TOEFL vs Probability of Admission
axs[0, 1].scatter(df['TOEFL'], df['Probability'])
axs[0, 1].set_title('TOEFL vs Probability of Admission')
axs[0, 1].set_xlabel('TOEFL Scores')
axs[0, 1].set_ylabel('Probability of Admission')

# Scatter plot for GPA vs Probability of Admission
axs[0, 2].scatter(df['CGPA'], df['Probability'])
axs[0, 2].set_title('CGPA vs Probability of Admission')
axs[0, 2].set_xlabel('GPA')
axs[0, 2].set_ylabel('Probability of Admission')

# Box plot for Research Publications vs Probability of Admission
df.boxplot(column='Probability', by='Research', ax=axs[1, 0])
axs[1, 0].set_title('Research Publications vs Probability of Admission')
axs[1, 0].set_xlabel('Research Publications')
axs[1, 0].set_ylabel('Probability of Admission')

# Box plot for LOR vs Probability of Admission
df.boxplot(column='Probability', by='LOR', ax=axs[1, 1])
axs[1, 1].set_title('LOR vs Probability of Admission')
axs[1, 1].set_xlabel('LOR')
axs[1, 1].set_ylabel('Probability of Admission')

# Box plot for SOP vs Probability of Admission
df.boxplot(column='Probability', by='SOP', ax=axs[1, 2])
axs[1, 2].set_title('SOP vs Probability of Admission')
axs[1, 2].set_xlabel('SOP')
axs[1, 2].set_ylabel('Probability of Admission')

# Adjust layout
plt.tight_layout()
plt.show()
```



Double-click (or enter) to edit

Data Cleaning

```
# Removing the serial no, column
df.drop('Serial No.', axis='columns', inplace=True)
df.head()
```

	GRE	TOEFL	UniversityRating	SOP	LOR	CGPA	Research	Probability
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

```
#remove outliers using the IQR method
def remove_outliers(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]

# Apply the function to relevant numeric columns

numeric_columns = df.select_dtypes(include=['number']).columns

for column in numeric_columns:
    df = remove_outliers(df, column)

# Display the shape of the dataframe to see how many rows were removed
df.shape
```

```
(497, 8)
```

3 rows have been removed from the dataset because they contain outliers

```
#Standardization of numerical features
from sklearn.preprocessing import StandardScaler

#scaler = StandardScaler()
#df = pd.DataFrame(scaler.fit_transform(df))

# Check the transformed data
df.head()
```

	GRE	TOEFL	UniversityRating	SOP	LOR	CGPA	Research	Probability
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

```
#import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
import joblib


# Data Preprocessing
# Assuming there are no missing values and no categorical variables to encode

# Features and target
X = df[['GRE', 'TOEFL', 'CGPA', 'Research', 'LOR', 'SOP']]
y = df['Probability']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Feature Scaling
```

```
# Feature Scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Model Selection and Training
# Linear Regression
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)

# Decision Tree Regressor
dt_model = DecisionTreeRegressor(random_state=42)
dt_model.fit(X_train, y_train)

# Random Forest Regressor
rf_model = RandomForestRegressor(random_state=42, n_estimators=100)
rf_model.fit(X_train, y_train)

# Model Evaluation
models = {'Linear Regression': lr_model, 'Decision Tree': dt_model, 'Random Forest': rf_model}

for name, model in models.items():
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    print(f'{name}:')
    print(f'Mean Squared Error: {mse:.4f}')
    print(f'R-squared: {r2:.4f}')
    print('---')

# Hyperparameter Tuning (example for Random Forest)
from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid, cv=3, n_jobs=-1, verbose=2)
grid_search.fit(X_train, y_train)

print(f'Best Parameters: {grid_search.best_params_}')
```

```
# Best model after hyperparameter tuning
best_rf_model = grid_search.best_estimator_

# Evaluate the best model
y_pred_best = best_rf_model.predict(X_test)
mse_best = mean_squared_error(y_test, y_pred_best)
r2_best = r2_score(y_test, y_pred_best)

print('Best Random Forest Model:')
print(f'Mean Squared Error: {mse_best:.4f}')
print(f'R-squared: {r2_best:.4f}')

# Save the best model
joblib.dump(best_rf_model, 'best_rf_model.pkl')
joblib.dump(scaler, 'scaler.pkl')
```

```
Linear Regression:
Mean Squared Error: 0.0037
```