Problem Statement-

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.

In [3]:
```
#import all libraries-
#numpy for numerical operations
#pandas for dataframe operations
#matplotlib and seaborn for data visualisation

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
```

In [4]:
```
# read the Jamboree_Admission data comma seperated file and create panda datafran

df=pd.read_csv("/Jamboree_Admission.csv")
```

Observations on shape of data, data types of all the attributes, conversion of categorical attributes to 'category' (If required) , missing value detection, statistical summary.

In [5]:
```
# checking top 5 rows

df.head()
```

Out[5]:

	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

Obs-

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 [6]:
```
# checking bottom 5 rows

df.tail()
```

Out[6]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
495	496	332	108	5	4.5	4.0	9.02	1	0.87
496	497	337	117	5	5.0	5.0	9.87	1	0.96
497	498	330	120	5	4.5	5.0	9.56	1	0.93
498	499	312	103	4	4.0	5.0	8.43	0	0.73
499	500	327	113	4	4.5	4.5	9.04	0	0.84

In [7]:
```
#gives the shape of dataframe

df.shape
```

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

In [8]:
```
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 [9]:
```
for i in df.columns:
    print(i, ' : ',df[i].nunique())
```

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

Statistical Summary

In [10]:
```
#describes statistical summary of dataframe.
#count of no. of records,mean of continuous values,std,
#min continuous value, max continuous value etc

df.describe()
```

Out[10]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Resea
count	500.000000	500.000000	500.000000	500.000000	500.000000	500.00000	500.000000	500.000
mean	250.500000	316.472000	107.192000	3.114000	3.374000	3.48400	8.576440	0.560
std	144.481833	11.295148	6.081868	1.143512	0.991004	0.92545	0.604813	0.496
min	1.000000	290.000000	92.000000	1.000000	1.000000	1.00000	6.800000	0.000
25%	125.750000	308.000000	103.000000	2.000000	2.500000	3.00000	8.127500	0.000
50%	250.500000	317.000000	107.000000	3.000000	3.500000	3.50000	8.560000	1.000
75%	375.250000	325.000000	112.000000	4.000000	4.000000	4.00000	9.040000	1.000
max	500.000000	340.000000	120.000000	5.000000	5.000000	5.00000	9.920000	1.000

Obs-

Mean CGPA is 8.57

Max CGPA is 9.92

Mean GRE is 316.47 and TOEFL is 107.192

Missing value detection

In [11]:
```
# checking null values (missing values)

df.isna().any()

#there is no null values in any column
```

Out[11]:
```
Serial No.           False
GRE Score            False
TOEFL Score          False
University Rating    False
SOP                  False
LOR                  False
CGPA                 False
Research             False
Chance of Admit      False
dtype: bool
```

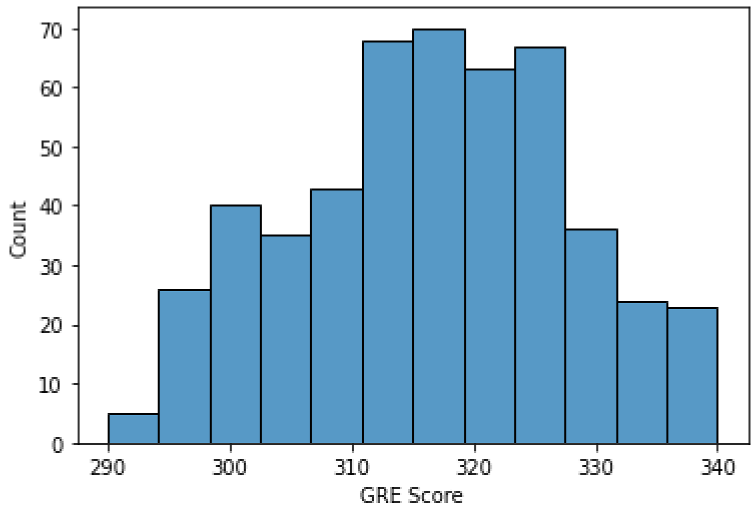
Univariate Analysis (distribution plots of all the continuous variable(s) barplots/countplots of all the categorical variables)

In [12]:
```
df_copy = df.copy(deep=True)
```

In [13]:
```
df.drop(["Serial No."],inplace=True,axis=1)
```

In [14]:
```
sns.histplot(df['GRE Score'])
```

Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x7f17efb4d590>

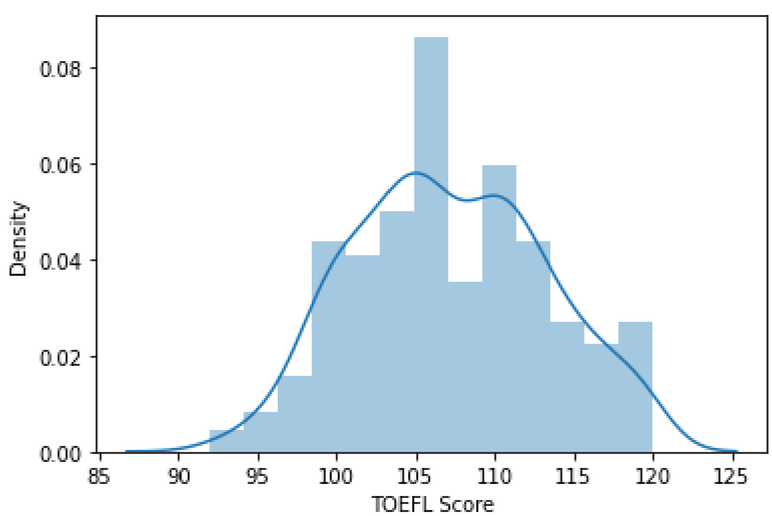


Obs- GRE score is normally distributed having mean of 317 approx.

In [15]:
```
sns.distplot(df['TOEFL Score'])
```

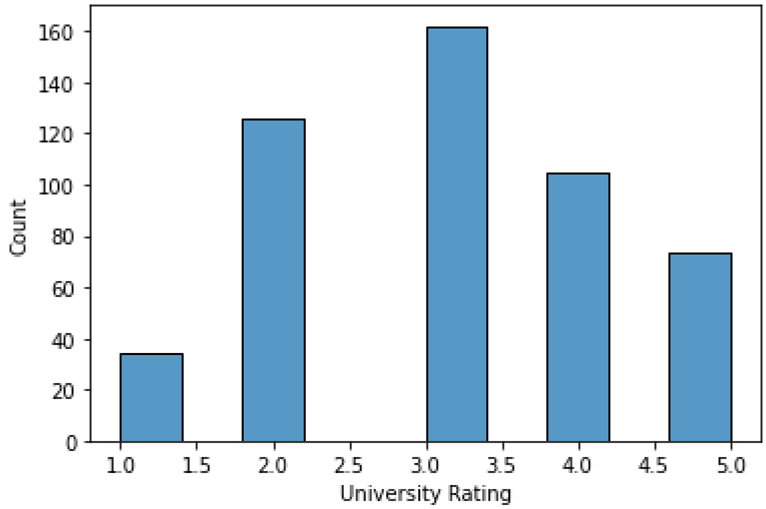
```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWar
ning: `distplot` is a deprecated function and will be removed in a future versi
on. Please adapt your code to use either `displot` (a figure-level function wit
h similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x7f17ef58e6d0>



In [16]:
```
sns.histplot(df['University Rating'])
```

Out[16]: <matplotlib.axes._subplots.AxesSubplot at 0x7f17ef56e590>

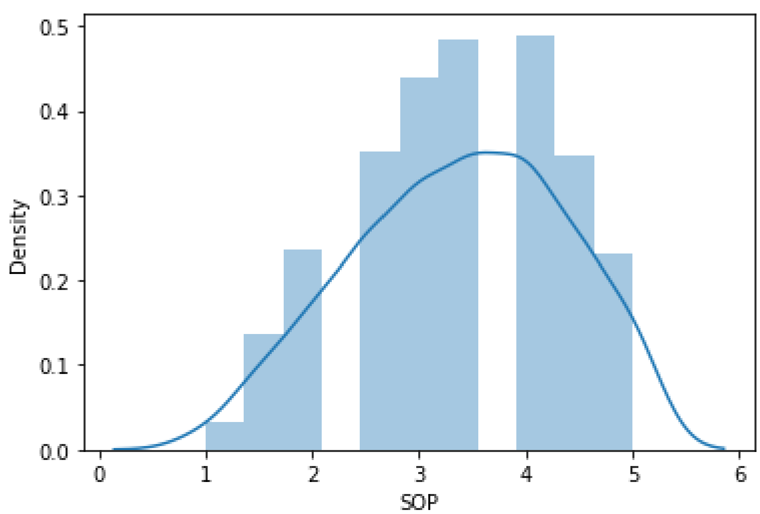


OBS- Most of the University have average rating of 3.

In [17]:
```
sns.distplot(df['SOP'])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWar
ning: `distplot` is a deprecated function and will be removed in a future versi
on. Please adapt your code to use either `displot` (a figure-level function wit
h similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

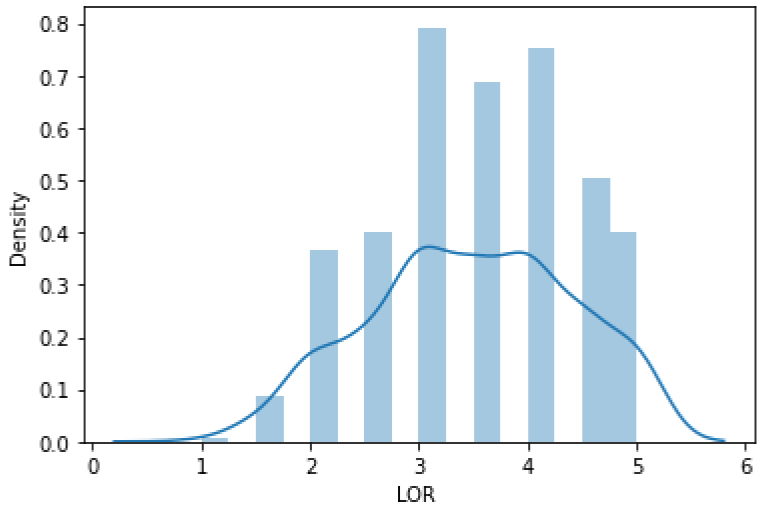
Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x7f17ef456110>



In [18]:
```
sns.distplot(df["LOR "])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

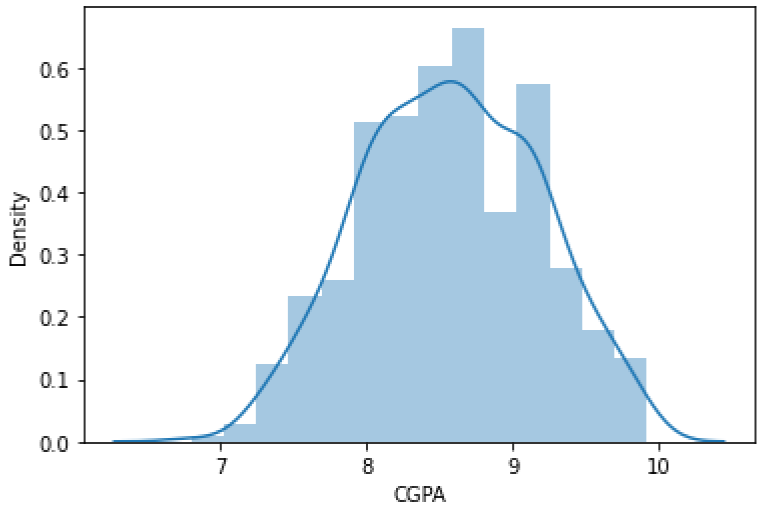
Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x7f17ef3ce210>



In [19]:
```
sns.distplot(df["CGPA"])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWar
ning: `distplot` is a deprecated function and will be removed in a future versi
on. Please adapt your code to use either `displot` (a figure-level function wit
h similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

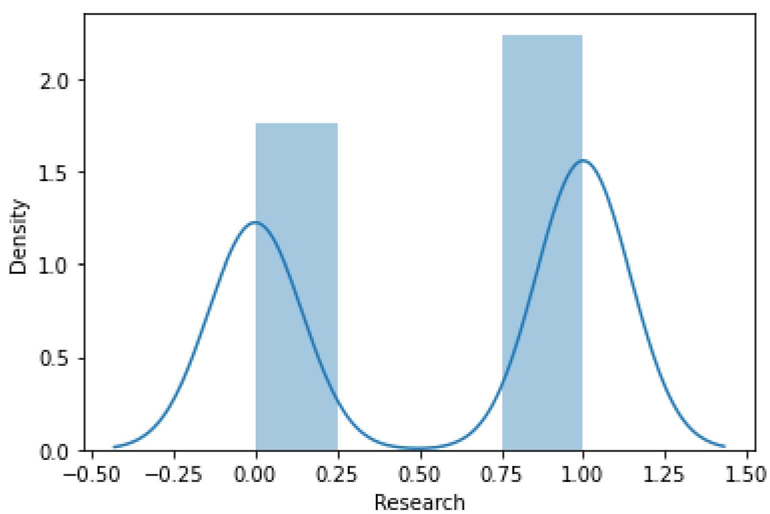
Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x7f17ef397a90>



In [20]:
```
sns.distplot(df["Research"])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWar
ning: `distplot` is a deprecated function and will be removed in a future versi
on. Please adapt your code to use either `displot` (a figure-level function wit
h similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x7f17ef2a6c90>

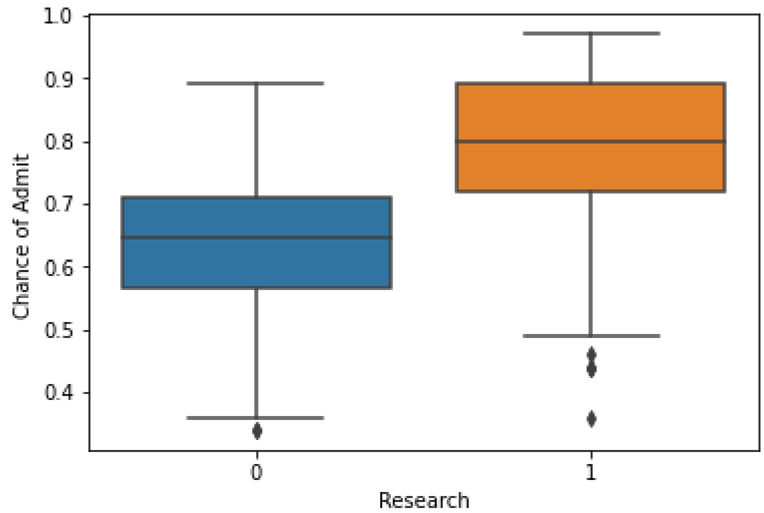


Obs- People have done research count is higher than people who havent done research.

Bivariate Analysis (Relationships between important variables such as workday and count, season and count, weather and count.

In [21]:
```
sns.boxplot(x='Research',y='Chance of Admit ',data=df)
```

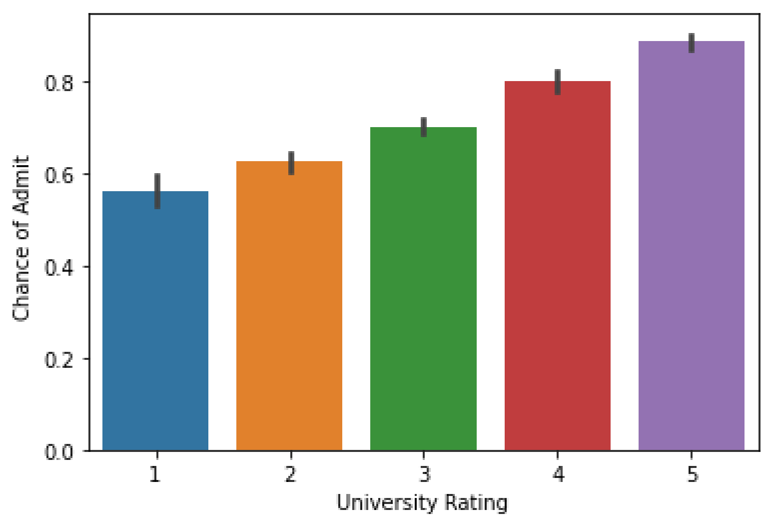
Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0x7f17ef250b50>



Obs- If Research is being done then chances of admission if high.

In [22]:
```
sns.barplot(x='University Rating',y='Chance of Admit ',data=df)
```

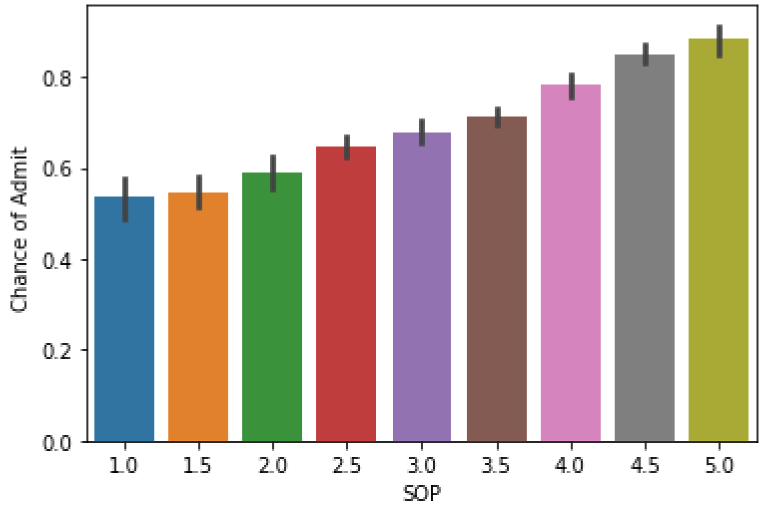
Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x7f17ef1a9110>



Obs- Ratings and chance of admit are corelated

In [23]:
```
sns.barplot(x='SOP',y='Chance of Admit ',data=df)
```

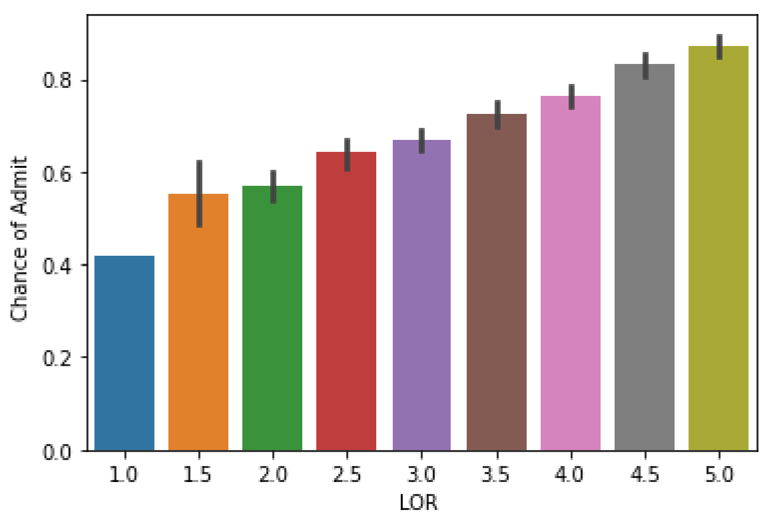
Out[23]: <matplotlib.axes._subplots.AxesSubplot at 0x7f17ef11f090>



Obs- High SOP means High Chance of Admission.

In [24]:
```
sns.barplot(x='LOR ',y='Chance of Admit ',data=df)
```

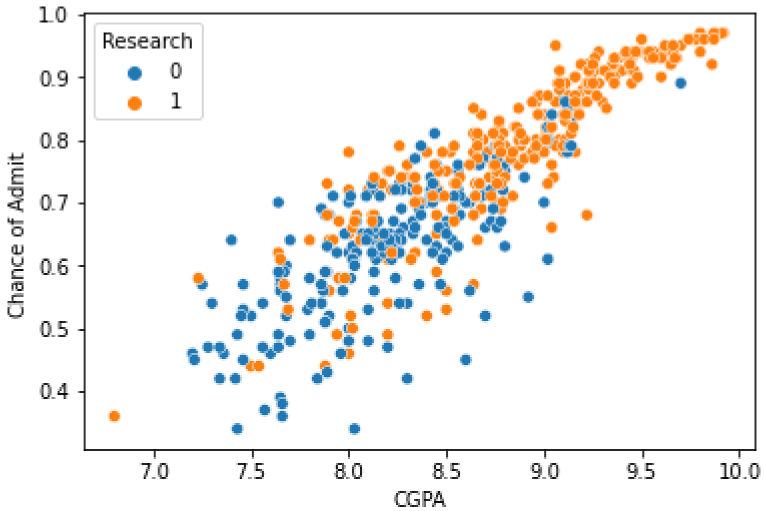
Out[24]: <matplotlib.axes._subplots.AxesSubplot at 0x7f17ef040bd0>



Obs- High LOR means High Chance of Admission.

In [25]:
```
sns.scatterplot(x='CGPA',y='Chance of Admit ',data=df,hue="Research")
```

Out[25]: <matplotlib.axes._subplots.AxesSubplot at 0x7f17eefe7ad0>

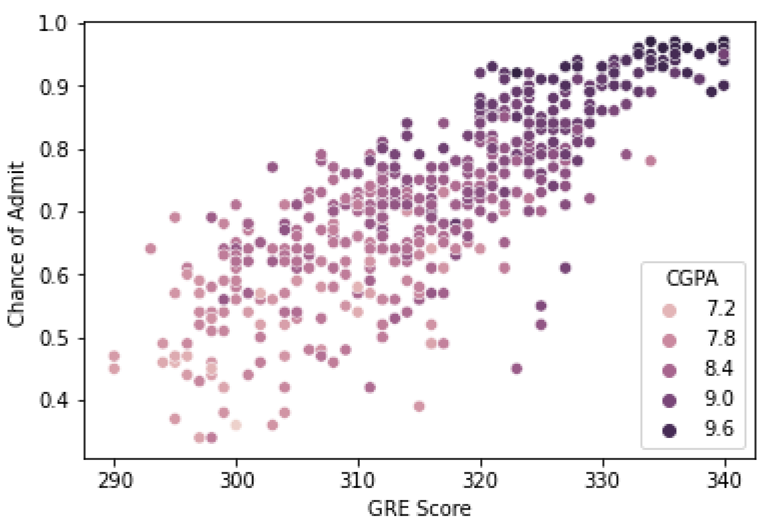


Obs- People who have done research and have high CGPA have high probability to get admission.

```
# This is formatted as code
```

In [26]:
```
sns.scatterplot(x='GRE Score',y='Chance of Admit ',data=df,hue="CGPA")
```

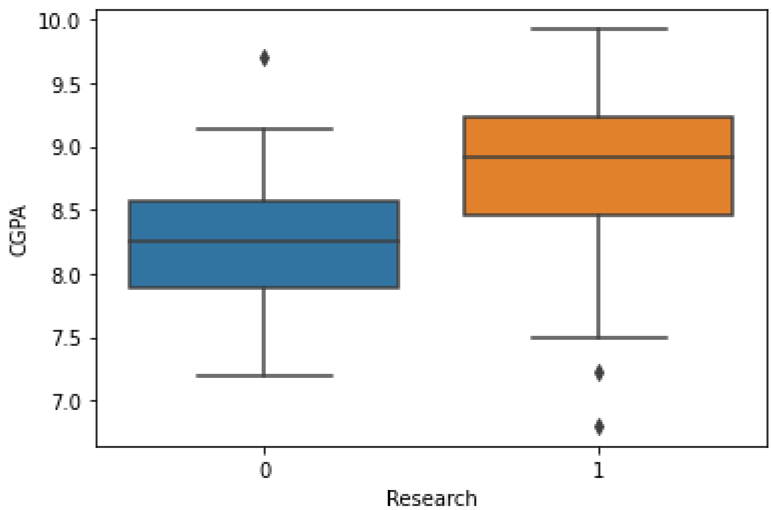
Out[26]: <matplotlib.axes._subplots.AxesSubplot at 0x7f17ef47b510>



Obs- High CGPA and high GRE have high chance of admit.

In [27]:
```
sns.boxplot(x="Research",y='CGPA',data=df)
```

Out[27]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f17efbf6750>
```



OBS- It is clearly visible that people who have done research have high CGPA too.

Data Preprocessing

In [28]:
```
duplicate = df[df.duplicated()]

print("Duplicate Rows :",duplicate)
```

```
Duplicate Rows : Empty DataFrame
Columns: [GRE Score, TOEFL Score, University Rating, SOP, LOR , CGPA, Research, Chance of Admit ]
Index: []
```

In [29]:
```
# checking null values (missing values)

df.isna().any()

#there is no null values in any column
```

Out[29]:
```
GRE Score            False
TOEFL Score          False
University Rating    False
SOP                  False
LOR                  False
CGPA                 False
Research             False
Chance of Admit      False
dtype: bool
```

In [30]:
```
#This dataframe contain no null values...
df.isnull().sum()
```

Out[30]:
```
GRE Score            0
TOEFL Score          0
University Rating    0
SOP                  0
LOR                  0
CGPA                 0
Research             0
Chance of Admit      0
dtype: int64
```

Outlier treatment

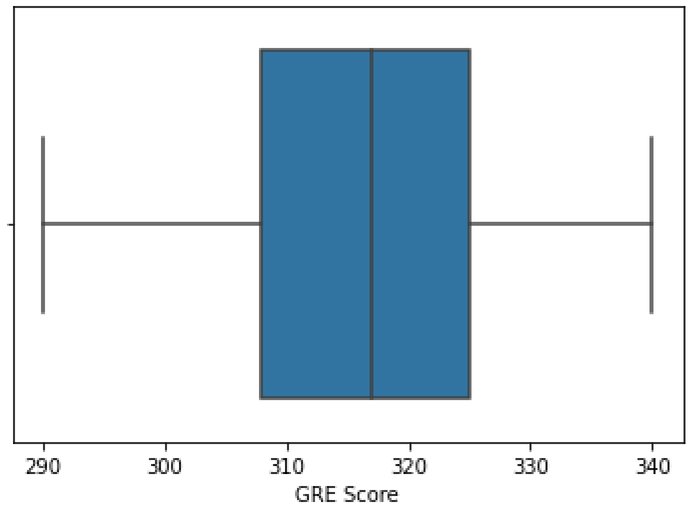
In [31]:
```
df.columns
```

Out[31]:
```
Index(['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR ', 'CGPA',
       'Research', 'Chance of Admit '],
      dtype='object')
```

In [32]:
```
sns.boxplot(df['GRE Score'])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarnin
g: Pass the following variable as a keyword arg: x. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without a
n explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

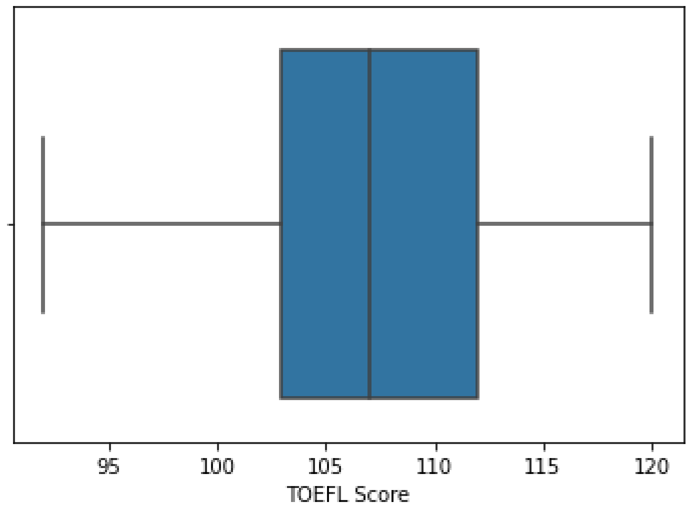
Out[32]: <matplotlib.axes._subplots.AxesSubplot at 0x7f17eedfd090>



In [33]:
```
sns.boxplot(df['TOEFL Score'])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarnin
g: Pass the following variable as a keyword arg: x. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without a
n explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

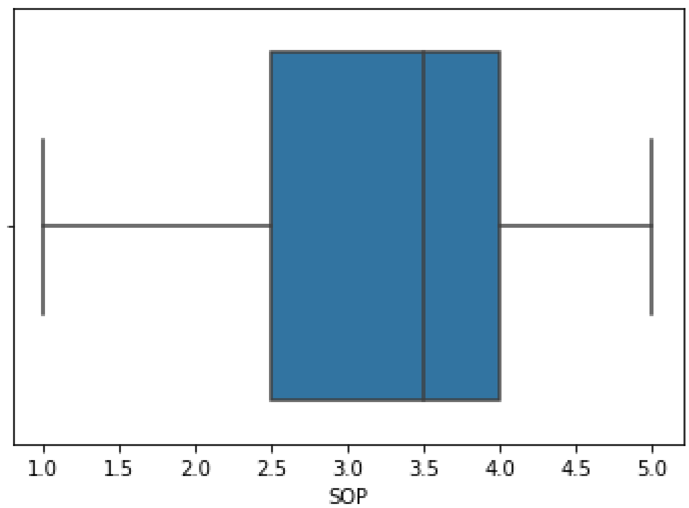
Out[33]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f17eeddb590>
```



In [34]:
```
sns.boxplot(df['SOP'])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarnin
g: Pass the following variable as a keyword arg: x. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without a
n explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

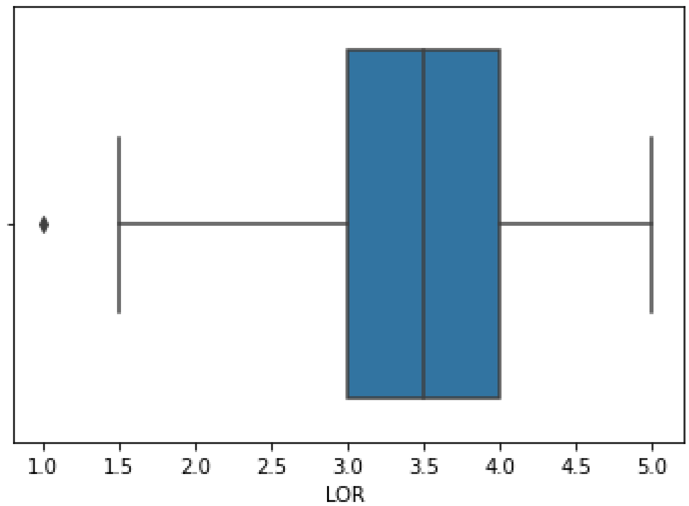
Out[34]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f17eed3e990>
```



In [35]:
```
sns.boxplot(df['LOR '])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarnin
g: Pass the following variable as a keyword arg: x. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without a
n explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

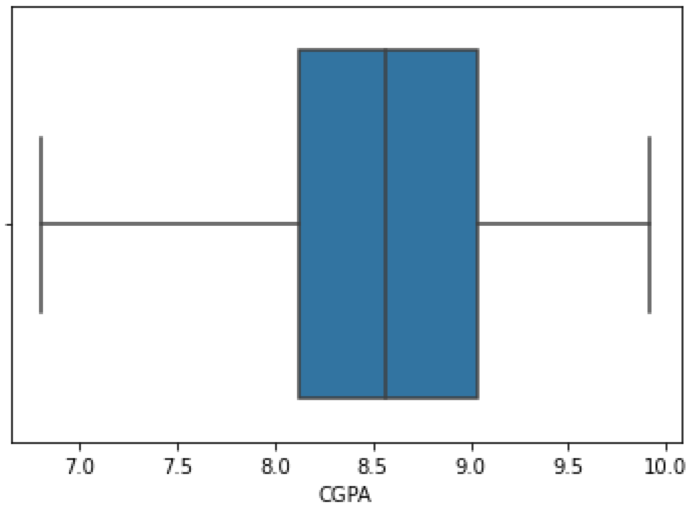
Out[35]: <matplotlib.axes._subplots.AxesSubplot at 0x7f17eecce390>



In [36]:
```
sns.boxplot(df['CGPA'])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarnin
g: Pass the following variable as a keyword arg: x. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without a
n explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

Out[36]: <matplotlib.axes._subplots.AxesSubplot at 0x7f17eec43490>



Obs- It seems that no column have Outliers , so there is no need to treat them

Feature engineering

Data preparation for modeling

In [37]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [38]:
```
scaler = MinMaxScaler()
```

In [39]:
```
df_scale=scaler.fit_transform(df)
```

In [40]:
```
df = pd.DataFrame(df_scale, columns=['GRE Score', 'TOEFL Score', 'University Rat
```

In [41]:
```
df.head()
```

Out[41]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	0.94	0.928571	0.75	0.875	0.875	0.913462	1.0	0.920635
1	0.68	0.535714	0.75	0.750	0.875	0.663462	1.0	0.666667
2	0.52	0.428571	0.50	0.500	0.625	0.384615	1.0	0.603175
3	0.64	0.642857	0.50	0.625	0.375	0.599359	1.0	0.730159
4	0.48	0.392857	0.25	0.250	0.500	0.451923	0.0	0.492063

Obs- Data is scaled and ready for modelling

Model building

Build the Linear Regression model and comment on the model statistics

Train Test Data Split

In [42]:
```
#separating independent and dependent variable
y = df['Chance of Admit']
```

In [43]:
```
df.drop(['Chance of Admit'],inplace=True,axis=1)
```

In [44]:
```
x=df
```

In [45]:
```
#splitting dataset into training and testing dataset
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df, y, test_size = 1/3, rand
```

In [46]:
```
# Import linear regression model

from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)

# Predicting the Test set results
y_pred = regressor.predict(X_test)
```

Display model coefficients with column names

Display model coefficients with column names

In [47]:
```
for idx, col_name in enumerate(X_train.columns):
    print("The coefficient for {} is {}".format(col_name, regressor.coef_[idx]))
```

```
The coefficient for GRE Score is 0.1408790075427227
The coefficient for TOEFL Score is 0.1575040580701022
The coefficient for University Rating is 0.05465035960035644
The coefficient for SOP is -0.00841907706430238
The coefficient for LOR is 0.07045259954550556
The coefficient for CGPA is 0.5833282074356485
The coefficient for Research is 0.037921908968468476
```

In [48]:
```
intercept = regressor.intercept_
print("The intercept for our model is {}".format(intercept))
```

```
The intercept for our model is 0.02880104216445656
```

In [49]:
```
from sklearn.metrics import r2_score
```

In [50]:
```
n=X_test.shape[0]
p=X_test.shape[1]
```

In [51]:
```
r2 = r2_score(y_test, y_pred)
adj_r2 = 1 - ((1 - r2) * (n - 1) / (n-p - 1))
```

In [52]:
```
print("R2 score for basic model is {}".format(r2))
print("Adjusted R2 score for basic model is {}".format(adj_r2))
```

```
R2 score for basic model is 0.824387026476648
Adjusted R2 score for basic model is 0.8166556377051797
```

OBs-Improve this model by checking other parameters and try to find important features.

In [53]:
```
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV
```

In [54]:
```
# list of alphas to tune
params = {'alpha': [0.0001, 0.001, 0.01, 0.05, 0.1,
 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 2.0, 3.0,
 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 20, 50, 100, 500, 1000 ]}


ridge = Ridge()

# cross validation
folds = 5
model_cv = GridSearchCV(estimator = ridge,
                        param_grid = params,
                        scoring= 'neg_mean_absolute_error',
                        cv = folds,
                        return_train_score=True,
                        verbose = 1)
model_cv.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 28 candidates, totalling 140 fits
```

Out[54]:
```
GridSearchCV(cv=5, estimator=Ridge(),
             param_grid={'alpha': [0.0001, 0.001, 0.01, 0.05, 0.1, 0.2, 0.3,
                                   0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 2.0, 3.0,
                                   4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 20, 50,
                                   100, 500, 1000]},
             return_train_score=True, scoring='neg_mean_absolute_error',
             verbose=1)
```

In [55]:
```
cv_results = pd.DataFrame(model_cv.cv_results_)
cv_results = cv_results[cv_results['param_alpha']<=200]
cv_results.head()
```

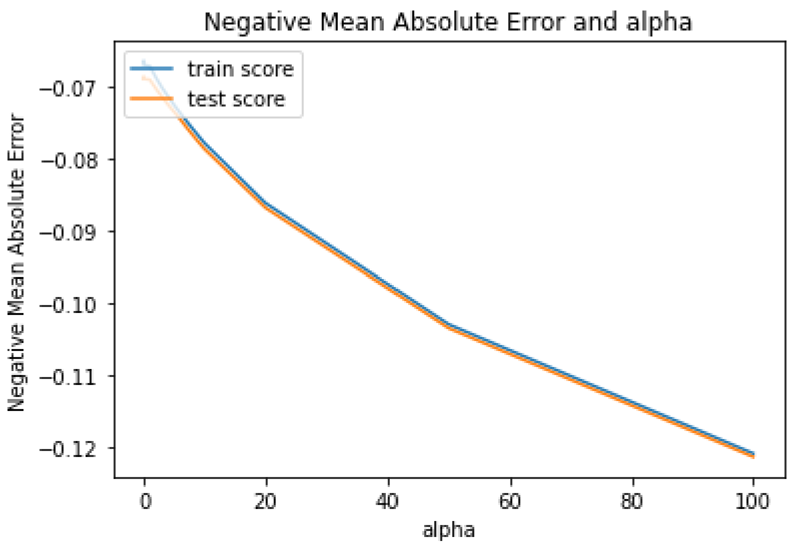
Out[55]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_
0	0.002279	0.000538	0.001484	0.000121	0.0001	{'alpha': 0.0001}	-0.0
1	0.001951	0.000014	0.001407	0.000017	0.001	{'alpha': 0.001}	-0.0
2	0.001924	0.000020	0.001409	0.000019	0.01	{'alpha': 0.01}	-0.0
3	0.002565	0.000901	0.001709	0.000436	0.05	{'alpha': 0.05}	-0.0
4	0.002111	0.000324	0.001449	0.000022	0.1	{'alpha': 0.1}	-0.0

5 rows × 21 columns

In [56]:
```
# plotting mean test and train scoes with alpha
cv_results['param_alpha'] = cv_results['param_alpha'].astype('int32')

# plotting
plt.plot(cv_results['param_alpha'], cv_results['mean_train_score'])
plt.plot(cv_results['param_alpha'], cv_results['mean_test_score'])
plt.xlabel('alpha')
plt.ylabel('Negative Mean Absolute Error')
plt.title("Negative Mean Absolute Error and alpha")
plt.legend(['train score', 'test score'], loc='upper left')
plt.show()
```



In [57]:
```
lasso = Lasso()

# cross validation
model_cv = GridSearchCV(estimator = lasso,
                        param_grid = params,
                        scoring= 'neg_mean_absolute_error',
                        cv = folds,
                        return_train_score=True,
                        verbose = 1)

model_cv.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 28 candidates, totalling 140 fits
```

Out[57]:
```
GridSearchCV(cv=5, estimator=Lasso(),
             param_grid={'alpha': [0.0001, 0.001, 0.01, 0.05, 0.1, 0.2, 0.3,
                                   0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 2.0, 3.0,
                                   4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 20, 50,
                                   100, 500, 1000]},
             return_train_score=True, scoring='neg_mean_absolute_error',
             verbose=1)
```

In [58]:
```
cv_results = pd.DataFrame(model_cv.cv_results_)
cv_results.head()
```

Out[58]:

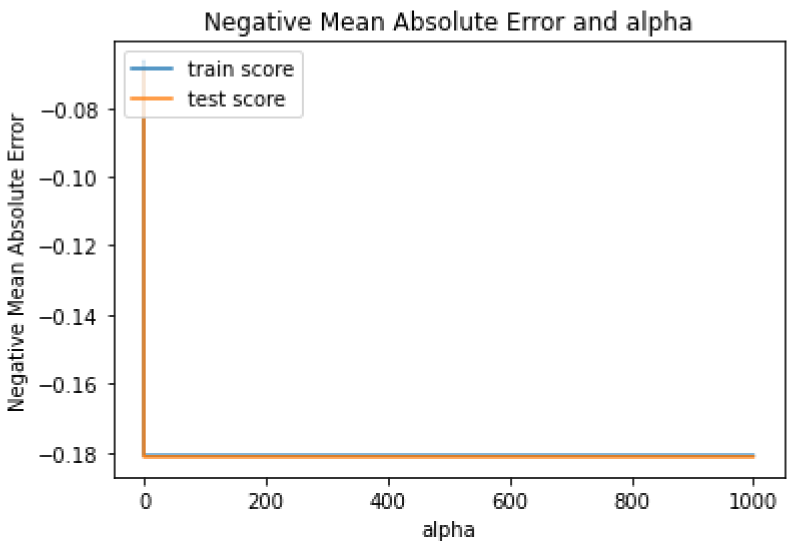
	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_
0	0.004832	0.003166	0.001769	0.000367	0.0001	{'alpha': 0.0001}	-0.0
1	0.002901	0.001672	0.001718	0.000413	0.001	{'alpha': 0.001}	-0.0
2	0.002069	0.000031	0.001637	0.000259	0.01	{'alpha': 0.01}	-0.0
3	0.001994	0.000058	0.001451	0.000023	0.05	{'alpha': 0.05}	-0.1
4	0.002082	0.000138	0.001517	0.000066	0.1	{'alpha': 0.1}	-0.1

5 rows × 21 columns

In [59]:
```
# plotting mean test and train scoes with alpha
cv_results['param_alpha'] = cv_results['param_alpha'].astype('float32')

# plotting
plt.plot(cv_results['param_alpha'], cv_results['mean_train_score'])
plt.plot(cv_results['param_alpha'], cv_results['mean_test_score'])
plt.xlabel('alpha')
plt.ylabel('Negative Mean Absolute Error')

plt.title("Negative Mean Absolute Error and alpha")
plt.legend(['train score', 'test score'], loc='upper left')
plt.show()
```



OLS Model

In [60]:
```
import statsmodels.api as sm
X_train_sm = X_train
#Unlike SKLearn, statsmodels don't automatically fit a constant,
#so you need to use the method sm.add_constant(X) in order to add a constant.
X_train_sm = sm.add_constant(X_train_sm)
# create a fitted model in one line
lm_1 = sm.OLS(y_train,X_train_sm).fit()

# print the coefficients
lm_1.params
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142: FutureW
arning: In a future version of pandas all arguments of concat except for the ar
gument 'objs' will be keyword-only
  x = pd.concat(x[::order], 1)
```

Out[60]:
```
const                0.028801
GRE Score            0.140879
TOEFL Score          0.157504
University Rating    0.054650
SOP                 -0.008419
LOR                  0.070453
CGPA                 0.583328
Research             0.037922
dtype: float64
```

In [61]:
```
print(lm_1.summary())
```

```
                            OLS Regression Results                            
==============================================================================
Dep. Variable:        Chance of Admit   R-squared:                       0.817
Model:                            OLS   Adj. R-squared:                  0.813
Method:                 Least Squares   F-statistic:                     207.2
Date:                Tue, 08 Nov 2022   Prob (F-statistic):          9.14e-116
Time:                        16:20:32   Log-Likelihood:                 312.10
No. Observations:                 333   AIC:                            -608.2
Df Residuals:                     325   BIC:                            -577.7
Df Model:                           7                                         
Covariance Type:            nonrobust                                         
================================================================================
======
                        coef    std err          t      P>|t|      [0.025      
0.975]
--------------------------------------------------------------------------------
------
const                 0.0288      0.018      1.579      0.115      -0.007      
0.065
GRE Score             0.1409      0.049      2.888      0.004       0.045      
0.237
TOEFL Score           0.1575      0.048      3.267      0.001       0.063      
0.252
University Rating     0.0547      0.029      1.901      0.058      -0.002      
0.111
SOP                  -0.0084      0.036     -0.237      0.813      -0.078      
0.061
LOR                   0.0705      0.032      2.184      0.030       0.007      
0.134
CGPA                  0.5833      0.057     10.157      0.000       0.470      
0.696
Research              0.0379      0.013      2.884      0.004       0.012      
0.064
==============================================================================
Omnibus:                       82.757   Durbin-Watson:                   2.001
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              203.687
Skew:                          -1.193   Prob(JB):                     5.89e-45
Kurtosis:                       5.997   Cond. No.                         22.7
==============================================================================

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

Here we can check the p-values which are greater than 0.05 are of no use, so remove them. SOP seems very high so remove it.

Testing the assumptions of the linear regression model

1. Multicollinearity check by VIF score (variables are dropped one-by-one till none has VIF>5)

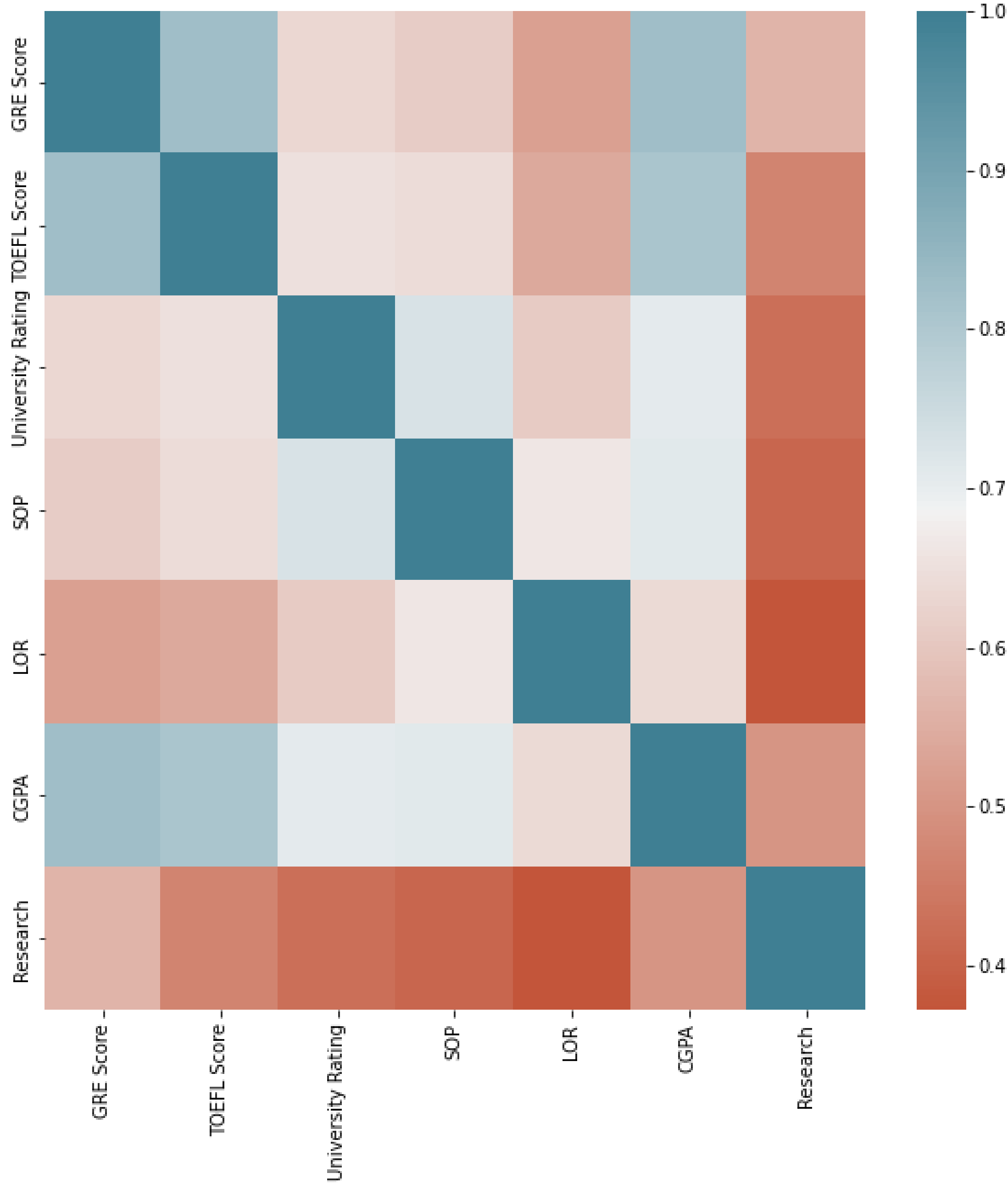
In [62]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor

VIF              = pd.DataFrame()
VIF['feature']   = X_train.columns
VIF['VIF']       = [variance_inflation_factor(X_train.values, i) for i in range(X_
# take a look
VIF
```

Out[62]:

	feature	VIF
0	GRE Score	29.076515
1	TOEFL Score	29.176248
2	University Rating	10.575487
3	SOP	19.357998
4	LOR	14.530652
5	CGPA	39.800887
6	Research	3.487647

In [63]:
```
corr = df.corr()
# 'sqft_lot15','sqft_lot',
plt.figure(figsize=(10,10))
sns.heatmap(corr, cmap=sns.diverging_palette(20, 220, n=200))
plt.show()
```



Obs- Many independent columns have high correlation which need to be removed.

The mean of residuals is nearly zero

In [64]:
```
residuals = y_test-y_pred
mean_residuals = np.mean(residuals)
print("Mean of Residuals {}".format(mean_residuals))
```

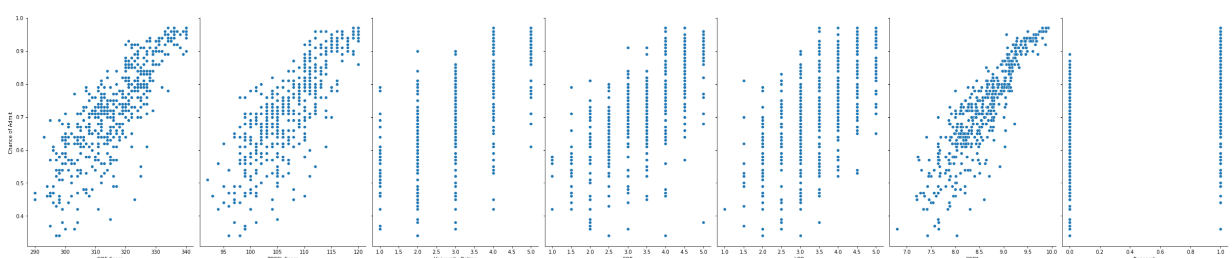
```
Mean of Residuals -0.011714588001401849
```

Obs- Mean of Residuals seems almost zero.

Linearity of variables (no pattern in the residual plot)

In [65]:
```
# visualize the relationship between the features and the response using scatterp
p = sns.pairplot(df_copy, x_vars=['GRE Score', 'TOEFL Score', 'University Rating
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/axisgrid.py:2076: UserWarning: T
he `size` parameter has been renamed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
```



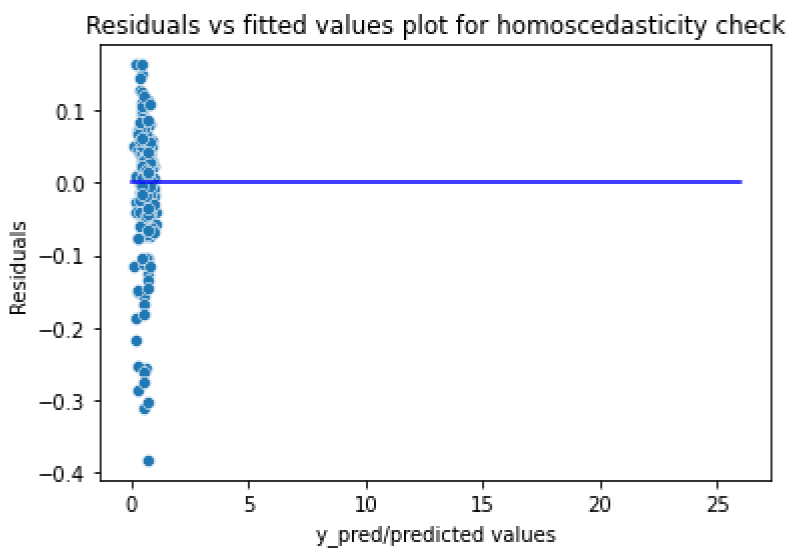
Obs- Linearity of variables is there.

Test for Homoscedasticity

In [66]:
```
p = sns.scatterplot(y_pred,residuals)
plt.xlabel('y_pred/predicted values')
plt.ylabel('Residuals')

p = sns.lineplot([0,26],[0,0],color='blue')
p = plt.title('Residuals vs fitted values plot for homoscedasticity check')
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarnin
g: Pass the following variables as keyword args: x, y. From version 0.12, the o
nly valid positional argument will be `data`, and passing other arguments witho
ut an explicit keyword will result in an error or misinterpretation.
  FutureWarning
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarnin
g: Pass the following variables as keyword args: x, y. From version 0.12, the o
nly valid positional argument will be `data`, and passing other arguments witho
ut an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

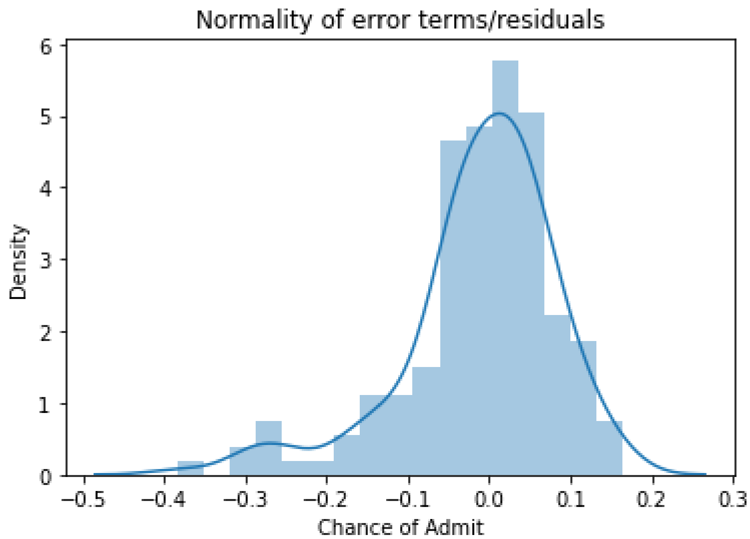


Obs- Test for Homoscedasticity is passed

Normality of residuals (almost bell-shaped curve in residuals distribution, points in QQ plot are almost all on the line)

In [67]:
```
p = sns.distplot(residuals,kde=True)
p = plt.title('Normality of error terms/residuals')
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWar
ning: `distplot` is a deprecated function and will be removed in a future versi
on. Please adapt your code to use either `displot` (a figure-level function wit
h similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



Obs- Normality of Error is maintained.

Build Model Again after feature selection

After Running OLS model, it seems that SOP have high p-value and VIF too and its coefficent value is small too , so remove it and again run the model

In [68]:
```
df_copy.drop("SOP",inplace=True,axis=1)
df_copy.drop("Serial No.",inplace=True,axis=1)
```

In [69]:
```
y = df_copy['Chance of Admit ']
x=df_copy.drop("Chance of Admit ",axis=1)
```

In [70]:
```
#splitting dataset into training and testing dataset
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = 1/3, rando
```

In [71]:
```
import statsmodels.api as sm
X_train_sm = X_train
#Unlike SKLearn, statsmodels don't automatically fit a constant,
#so you need to use the method sm.add_constant(X) in order to add a constant.
X_train_sm = sm.add_constant(X_train_sm)
# create a fitted model in one line
lm_1 = sm.OLS(y_train,X_train_sm).fit()

# print the coefficients
lm_1.params
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142: FutureW
arning: In a future version of pandas all arguments of concat except for the ar
gument 'objs' will be keyword-only
  x = pd.concat(x[::order], 1)
```

Out[71]:
```
const                -1.297827
GRE Score             0.001779
TOEFL Score           0.003521
University Rating     0.008171
LOR                   0.010778
CGPA                  0.117212
Research              0.023866
dtype: float64
```

In [72]:
```
print(lm_1.summary())
```

```
                            OLS Regression Results                            
==============================================================================
Dep. Variable:      Chance of Admit    R-squared:                       0.817
Model:                          OLS    Adj. R-squared:                  0.814
Method:               Least Squares    F-statistic:                     242.5
Date:              Tue, 08 Nov 2022    Prob (F-statistic):          5.78e-117
Time:                      16:20:35    Log-Likelihood:                 465.93
No. Observations:               333    AIC:                            -917.9
Df Residuals:                   326    BIC:                            -891.2
Df Model:                         6                                         
Covariance Type:          nonrobust                                         
================================================================================
======
                        coef    std err          t      P>|t|      [0.025      
0.975]
--------------------------------------------------------------------------------
------
const                -1.2978      0.124    -10.458      0.000      -1.542      
-1.054
GRE Score             0.0018      0.001      2.899      0.004       0.001      
0.003
TOEFL Score           0.0035      0.001      3.264      0.001       0.001      
0.006
University Rating     0.0082      0.004      1.978      0.049    4.37e-05      
0.016
LOR                   0.0108      0.005      2.203      0.028       0.001      
0.020
CGPA                  0.1172      0.011     10.351      0.000       0.095      
0.139
Research              0.0239      0.008      2.885      0.004       0.008      
0.040
==============================================================================
Omnibus:                     83.508    Durbin-Watson:                   2.003
Prob(Omnibus):                0.000    Jarque-Bera (JB):              206.340
Skew:                        -1.203    Prob(JB):                     1.56e-45
Kurtosis:                     6.014    Cond. No.                     1.26e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctl
y specified.
[2] The condition number is large, 1.26e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [73]:
```
#from statsmodels.stats.outliers_influence import variance_inflation_factor

VIF               = pd.DataFrame()
VIF['feature']    = X_train.columns
VIF['VIF']        = [variance_inflation_factor(X_train.values, i) for i in range(X_
# take a look
VIF
```

Out[73]:

	feature	VIF
0	GRE Score	1245.613776
1	TOEFL Score	1213.118015
2	University Rating	16.074717
3	LOR	28.516201
4	CGPA	856.546077
5	Research	2.942324

In [74]:
```
df_copy.head()
```

Out[74]:

	GRE Score	TOEFL Score	University Rating	LOR	CGPA	Research	Chance of Admit
0	337	118	4	4.5	9.65	1	0.92
1	324	107	4	4.5	8.87	1	0.76
2	316	104	3	3.5	8.00	1	0.72
3	322	110	3	2.5	8.67	1	0.80
4	314	103	2	3.0	8.21	0	0.65

After Running OLS model, it seems that GRE Score have high VIF too, so remove it and again run the model

In [75]:
```
df_copy.drop("GRE Score",inplace=True,axis=1)
```

In [76]:
```
x=df_copy.drop("Chance of Admit ",axis=1)
```

In [77]:
```
y=df_copy["Chance of Admit "]
```

In [78]:
```
#splitting dataset into training and testing dataset
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = 1/3, randon
```

In [79]:
```
import statsmodels.api as sm
X_train_sm = X_train
#Unlike SKLearn, statsmodels don't automatically fit a constant,
#so you need to use the method sm.add_constant(X) in order to add a constant.
X_train_sm = sm.add_constant(X_train_sm)
# create a fitted model in one line
lm_1 = sm.OLS(y_train,X_train_sm).fit()

# print the coefficients
lm_1.params
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142: FutureW
arning: In a future version of pandas all arguments of concat except for the ar
gument 'objs' will be keyword-only
  x = pd.concat(x[::order], 1)
```

Out[79]:
```
const                -1.010088
TOEFL Score           0.005133
University Rating     0.008456
LOR                   0.009530
CGPA                  0.129094
Research              0.030851
dtype: float64
```

In [80]:
```
print(lm_1.summary())
```

```
                            OLS Regression Results                            
==============================================================================
Dep. Variable:      Chance of Admit    R-squared:                       0.812
Model:                          OLS    Adj. R-squared:                  0.809
Method:               Least Squares    F-statistic:                     282.9
Date:              Tue, 08 Nov 2022    Prob (F-statistic):          2.05e-116
Time:                      16:20:35    Log-Likelihood:                 461.69
No. Observations:               333    AIC:                            -911.4
Df Residuals:                   327    BIC:                            -888.5
Df Model:                         5                                         
Covariance Type:          nonrobust                                         
================================================================================
======
                        coef    std err          t      P>|t|      [0.025      
0.975]
--------------------------------------------------------------------------------
------
const                -1.0101      0.075    -13.409      0.000      -1.158      
-0.862
TOEFL Score           0.0051      0.001      5.490      0.000       0.003      
0.007
University Rating     0.0085      0.004      2.025      0.044       0.000      
0.017
LOR                   0.0095      0.005      1.934      0.054      -0.000      
0.019
CGPA                  0.1291      0.011     12.094      0.000       0.108      
0.150
Research              0.0309      0.008      3.855      0.000       0.015      
0.047
==============================================================================
Omnibus:                     74.520    Durbin-Watson:                   2.036
Prob(Omnibus):                0.000    Jarque-Bera (JB):              156.948
Skew:                        -1.143    Prob(JB):                     8.30e-35
Kurtosis:                     5.466    Cond. No.                     2.44e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctl
y specified.
[2] The condition number is large, 2.44e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [81]:
```
#from statsmodels.stats.outliers_influence import variance_inflation_factor

VIF             = pd.DataFrame()
VIF['feature']  = X_train.columns
VIF['VIF']      = [variance_inflation_factor(X_train.values, i) for i in range(X_
# take a look
VIF
```

Out[81]:

	feature	VIF
0	TOEFL Score	609.235738
1	University Rating	14.332736
2	LOR	27.662518
3	CGPA	688.616447
4	Research	2.926751

In [82]:
```
df_copy.drop("LOR ",inplace=True,axis=1)
```

In [83]:
```
x=df_copy.drop("Chance of Admit ",axis=1)
```

In [84]:
```
y=df_copy["Chance of Admit "]
```

In [85]:
```
#splitting dataset into training and testing dataset
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = 1/3, randor
```

In [86]:
```
import statsmodels.api as sm
X_train_sm = X_train
#Unlike SKLearn, statsmodels don't automatically fit a constant,
#so you need to use the method sm.add_constant(X) in order to add a constant.
X_train_sm = sm.add_constant(X_train_sm)
# create a fitted model in one line
lm_1 = sm.OLS(y_train,X_train_sm).fit()

# print the coefficients
lm_1.params
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142: FutureW
arning: In a future version of pandas all arguments of concat except for the ar
gument 'objs' will be keyword-only
  x = pd.concat(x[::order], 1)
```

Out[86]:
```
const               -1.032058
TOEFL Score          0.005098
University Rating    0.010475
CGPA                 0.135170
Research             0.031839
dtype: float64
```

In [87]:
```
print(lm_1.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:      Chance of Admit   R-squared:                       0.810
Model:                          OLS   Adj. R-squared:                  0.808
Method:               Least Squares   F-statistic:                     349.7
Date:              Tue, 08 Nov 2022   Prob (F-statistic):          6.56e-117
Time:                      16:20:35   Log-Likelihood:                 459.79
No. Observations:               333   AIC:                            -909.6
Df Residuals:                   328   BIC:                            -890.5
Df Model:                         4
Covariance Type:          nonrobust
================================================================================
======
                        coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
------
const                -1.0321      0.075    -13.801      0.000      -1.179
-0.885
TOEFL Score           0.0051      0.001      5.431      0.000       0.003
0.007
University Rating     0.0105      0.004      2.580      0.010       0.002
0.018
CGPA                  0.1352      0.010     13.195      0.000       0.115
0.155
Research              0.0318      0.008      3.970      0.000       0.016
0.048
==============================================================================
Omnibus:                     75.113   Durbin-Watson:                   2.058
Prob(Omnibus):                0.000   Jarque-Bera (JB):              156.930
Skew:                        -1.156   Prob(JB):                     8.38e-35
Kurtosis:                     5.442   Cond. No.                     2.41e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctl
y specified.
[2] The condition number is large, 2.41e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

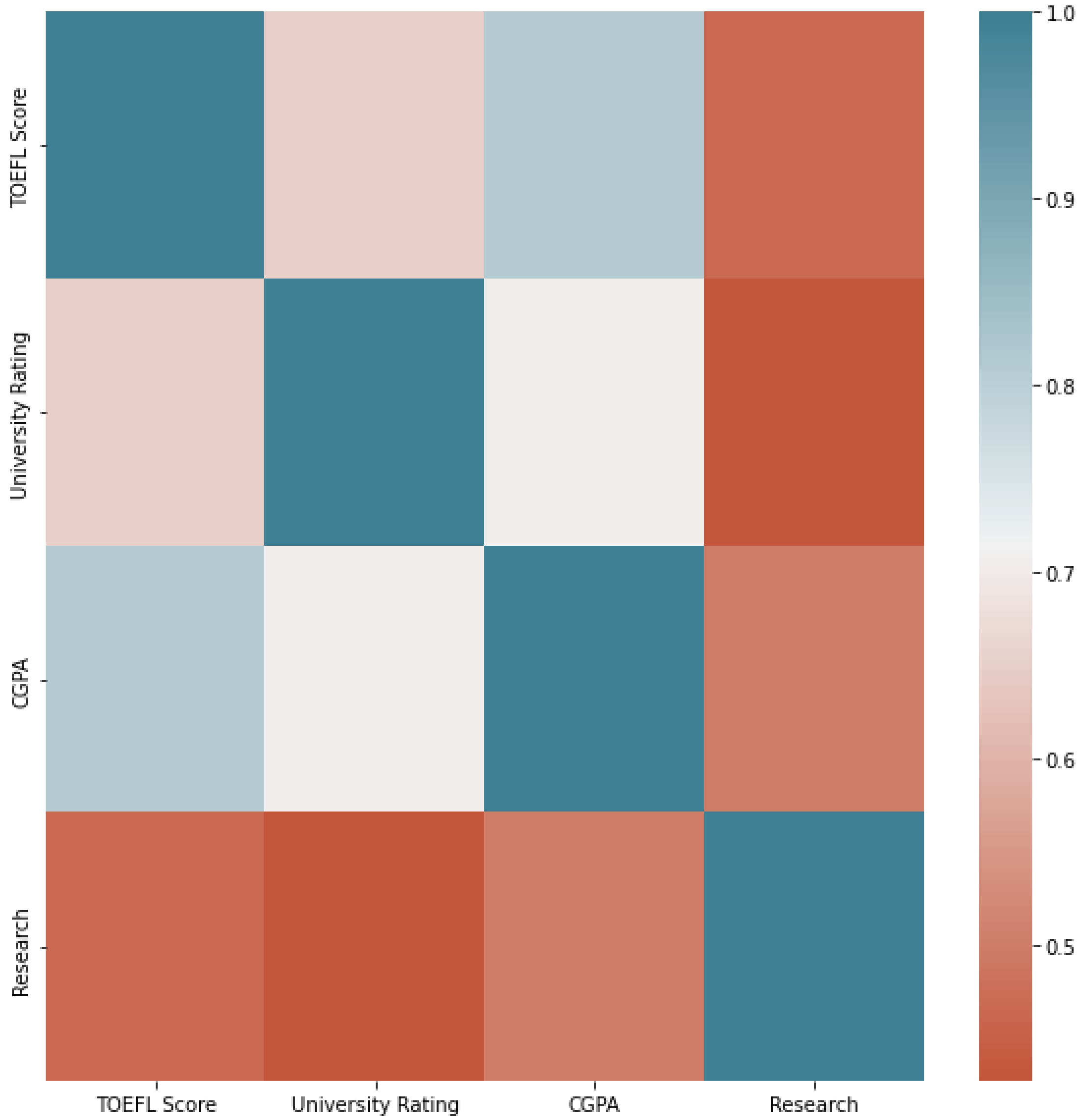
In [88]:
```
#from statsmodels.stats.outliers_influence import variance_inflation_factor

VIF             = pd.DataFrame()
VIF['feature']  = X_train.columns
VIF['VIF']      = [variance_inflation_factor(X_train.values, i) for i in range(X_
# take a look
VIF
```

Out[88]:

	feature	VIF
0	TOEFL Score	598.937177
1	University Rating	12.602519
2	CGPA	640.516327
3	Research	2.885511

In [89]:
```
corr = df_copy.drop("Chance of Admit ",axis=1).corr()
# 'sqft_lot15','sqft_lot',
plt.figure(figsize=(10,10))
sns.heatmap(corr, cmap=sns.diverging_palette(20, 220, n=200))
plt.show()
```



In [90]:
```
df_copy.drop("TOEFL Score",inplace=True,axis=1)
```

In [91]:
```
x=df_copy.drop("Chance of Admit ",axis=1)
```

In [92]:
```
y=df_copy["Chance of Admit "]
```

In [93]:
```
#splitting dataset into training and testing dataset
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = 1/3, rando
```

In [94]:
```
import statsmodels.api as sm
X_train_sm = X_train
#Unlike SKLearn, statsmodels don't automatically fit a constant,
#so you need to use the method sm.add_constant(X) in order to add a constant.
X_train_sm = sm.add_constant(X_train_sm)
# create a fitted model in one line
lm_1 = sm.OLS(y_train,X_train_sm).fit()

# print the coefficients
lm_1.params
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142: FutureW
arning: In a future version of pandas all arguments of concat except for the ar
gument 'objs' will be keyword-only
  x = pd.concat(x[::order], 1)
```

Out[94]:
```
const                -0.796318
University Rating     0.013801
CGPA                  0.169782
Research              0.038513
dtype: float64
```

In [95]:
```
print(lm_1.summary())
```

```
                            OLS Regression Results                            
==============================================================================
Dep. Variable:       Chance of Admit   R-squared:                       0.793
Model:                           OLS   Adj. R-squared:                  0.791
Method:                Least Squares   F-statistic:                     420.1
Date:               Tue, 08 Nov 2022   Prob (F-statistic):          3.92e-112
Time:                       16:20:36   Log-Likelihood:                 445.46
No. Observations:                333   AIC:                            -882.9
Df Residuals:                    329   BIC:                            -867.7
Df Model:                          3                                         
Covariance Type:           nonrobust                                         
================================================================================
======
                        coef    std err          t      P>|t|      [0.025      
0.975]
--------------------------------------------------------------------------------
------
const                -0.7963      0.063    -12.545      0.000      -0.921      
-0.671
University Rating     0.0138      0.004      3.298      0.001       0.006      
0.022
CGPA                  0.1698      0.008     20.308      0.000       0.153      
0.186
Research              0.0385      0.008      4.662      0.000       0.022      
0.055
==============================================================================
Omnibus:                      66.611   Durbin-Watson:                   2.041
Prob(Omnibus):                 0.000   Jarque-Bera (JB):              128.449
Skew:                         -1.066   Prob(JB):                     1.28e-28
Kurtosis:                      5.171   Cond. No.                         169.
==============================================================================

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

In [96]:
```
#from statsmodels.stats.outliers_influence import variance_inflation_factor

VIF               = pd.DataFrame()
VIF['feature']  = X_train.columns
VIF['VIF']      = [variance_inflation_factor(X_train.values, i) for i in range(X_
# take a look
VIF
```

Out[96]:

	feature	VIF
0	University Rating	12.229630
1	CGPA	10.623491
2	Research	2.878537

In [97]:
```
ytrain_predict = lm_1.predict(X_train_sm)
```

In [98]:
```
x_test = sm.add_constant(X_test)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142: FutureW
arning: In a future version of pandas all arguments of concat except for the ar
gument 'objs' will be keyword-only
  x = pd.concat(x[::order], 1)
```

In [99]:
```
ytest_predict = lm_1.predict(x_test)
```

Model Evaluation

In [100]:
```
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test,ytest_predict))
print('Mean Squared Error:', metrics.mean_squared_error(y_test,ytest_predict))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test,ytes
```

```
Mean Absolute Error: 0.04461028957806784
Mean Squared Error: 0.00391070947098406
Root Mean Squared Error: 0.0625356655915971
```

In [101]:
```
n=x_test.shape[0]
p=x_test.shape[1]
r2 = r2_score(y_test, ytest_predict)
adj_r2 = 1 - ((1 - r2) * (n - 1) / (n-p - 1))

print("R2 score for basic model is {}".format(r2))
print("Adjusted R2 score for basic model is {}".format(adj_r2))
```

```
R2 score for basic model is 0.8090048139650706
Adjusted R2 score for basic model is 0.8042888834456896
```

Final model is having good accuracy with no multicollinearity.

Actionable Insights

If Research is being done then chances of admission if high.

GRE score is normally distributed having mean of 317 approx.

Most of the University have average rating of 3.

People have done research count is higher than people who havent done research.

Ratings and chance of admit are corelated.

People who have done research and have high CGPA have high probability to get admission.

High CGPA and high GRE have high chance of admit.

It seems that no column have Outliers , so there is no need to treat them.

Many independent columns have high correlation which need to be removed.

Mean of Residuals seems almost zero.

Linearity of variables is there.

Normality of Error is maintained.

Recommendations

1. CGPA plays an important role, so having high CGPA with Research means high Chance of Admission.
2. TOEFL score and GRE score have almost same importance, so having good marks in any one of them would give high probability of admission.
3. Statement of Purpose and Letter of Recommendation Strength are not that much important factor to get admission as per analysis.

In [101]: