Data Bootcamp Final Project

UCLA Graduate Admissions Dataset

Team: ***Yuyang Fu*, *Yangming Zhang***

Data Sources: Kaggle https://www.kaggle.com/mohansacharya/graduate-admissions (https://www.kaggle.com/mohansacharya/graduate-admissions)

Citation: Mohan S Acharya, Asfia Armaan, Aneeta S Antony : A Comparison of Regression Models for Prediction of Graduate Admissions, IEEE International Conference on Computational Intelligence in Data Science 2019

Table of Contents

1. Introduction

This project mainly focuses on what parameters are important for a student to get into UCLA graduate school, and how these factors are interrelated among themselves. It will also help predict candidates' chances of admission given the variables.

2. Data Import

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [2]:
```
df1 = pd.read_csv('Admission_Predict.csv')
df2 = pd.read_csv('Admission_Predict_Ver1.1.csv')
```

In [3]:
```
df = pd.concat([df1,df2])
```

Checking data types (which are int64 and float64)

In [4]:
```
df.dtypes
```

Out[4]:
```
Serial No.             int64
GRE Score              int64
TOEFL Score            int64
University Rating      int64
SOP                  float64
LOR                  float64
CGPA                 float64
Research               int64
Chance of Admit      float64
dtype: object
```

The dataset contains several parameters which are considered important during the application for Masters Programs
The parameters included are :

1. GRE Scores (out of 340)
2. TOEFL Scores (out of 120)
3. University Rating (out of 5)
4. Statement of Purpose (out of 5)
5. Letter of Recommendation Strength (out of 5)
6. Undergraduate GPA (out of 10)
7. Research Experience (either 0 or 1)
8. Chance of Admit (ranging from 0 to 1)

3. Data Filtering and Cleaning

Checking if there are any null values in the dataset

In [5]:
```
df.isnull().sum()
```

Out[5]:
```
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
```

In [6]:
```
df.columns
```

Out[6]:
```
Index(['Serial No.', 'GRE Score', 'TOEFL Score', 'University Rating', 'SO
P',
       'LOR ', 'CGPA', 'Research', 'Chance of Admit '],
      dtype='object')
```

Changing the names of columns for future editing

In [7]:
```
df.rename(columns={'GRE Score':'GRE_Score', 'TOEFL Score':'TOEFL_Score'
                   'University Rating':'University_Rating','Chance of A
                  'LOR ':'LOR'},inplace=True)
```

In [8]:
```
df
```

Out[8]:

	Serial No.	GRE_Score	TOEFL_Score	University_Rating	SOP	LOR	CGPA	Research	Chance_of_A
0	1	337	118	4	4.5	4.5	9.65	1	
1	2	324	107	4	4.0	4.5	8.87	1	
2	3	316	104	3	3.0	3.5	8.00	1	
3	4	322	110	3	3.5	2.5	8.67	1	
4	5	314	103	2	2.0	3.0	8.21	0	
5	6	330	115	5	4.5	3.0	9.34	1	
6	7	321	109	3	3.0	4.0	8.20	1	
7	8	308	101	2	3.0	4.0	7.90	0	
8	9	302	102	1	2.0	1.5	8.00	0	
9	10	323	108	3	3.5	3.0	8.60	0	
10	11	325	106	3	3.5	4.0	8.40	1	

Returning a tuple representing the dimensionality of the dataframe

In [9]:
```
df.shape
```

Out[9]: (900, 9)

Grouping the chance of admit into 5 levels(which are HIGH, MEDIA HIGH, MEDIUM, MEDIUM LOW, LOW) by the interval of 0.1. The levels of the admit chance are more understanable and visualized, what's more, differentiating the data by the same interval makes it more convenient to compare with each group.

In [10]:
```
def acl(df):
    if df['Chance_of_Admit'] >= 0.9:
        return 'High'
    elif 0.9 > df['Chance_of_Admit'] >= 0.8:
        return 'Medium High'
    elif 0.8 > df['Chance_of_Admit'] >= 0.7:
        return 'Medium'
    elif 0.7 > df['Chance_of_Admit'] >= 0.6:
        return 'Medium Low'
    else:
        return 'Low'
```

Assuming here that students with 0.7 chance of admission have secured admission. Therefore we create another column named Admit. The value of Admit=1 if Chance>0.7 and Admit=0 if Chance<0.7.

In [11]:
```
def a(row):
    if row['Chance_of_Admit'] >0.7 :
        return 1
    else :
        return 0
```

In [12]:
```
df['Admit_Chance_Level'] = df.apply(acl, axis=1)
df['Admit'] = df.apply(a,axis=1)
```

In [13]:
```
df
```

483	484	304	103	5	5.0	3.0	7.92	0
484	485	317	106	3	3.5	3.0	7.89	1
485	486	311	101	2	2.5	3.5	8.34	1
486	487	319	102	3	2.5	2.5	8.37	0
487	488	327	115	4	3.5	4.0	9.14	0
488	489	322	112	3	3.0	4.0	8.62	1
489	490	302	110	3	4.0	4.5	8.50	0
490	491	307	105	2	2.5	4.5	8.12	1
491	492	297	99	4	3.0	3.5	7.81	0
492	493	298	101	4	2.5	4.5	7.69	1
493	494	300	95	2	3.0	1.5	8.22	1
494	495	301	99	3	2.5	2.0	8.45	1
495	496	332	108	5	4.5	4.0	9.02	1

Merging Enrollment Level, which is the level of a candidate who received an offer and enrolled the school, based on admit chance level

In [14]:
```
enrollment = pd.read_csv('Enrollment.csv')
```

In [15]:
```
merged = pd.merge(df,enrollment, on='Admit_Chance_Level')
merged
```

Out[15]:

	Serial No.	GRE_Score	TOEFL_Score	University_Rating	SOP	LOR	CGPA	Research	Chance_of_A
0	1	337	118	4	4.5	4.5	9.65	1	
1	6	330	115	5	4.5	3.0	9.34	1	
2	23	328	116	5	5.0	5.0	9.50	1	
3	24	334	119	5	5.0	4.5	9.70	1	
4	25	336	119	5	4.0	3.5	9.80	1	
5	26	340	120	5	4.5	4.5	9.60	1	
6	33	338	118	4	3.0	4.5	9.40	1	
7	34	340	114	5	4.0	4.0	9.60	1	
8	35	331	112	5	4.0	5.0	9.80	1	
9	45	326	113	5	4.5	4.0	9.40	1	
10	71	332	118	5	5.0	5.0	9.64	1	

Setting Serial number as index, as it only serves the purpose of identifying entries and would not contribute to data exploration, visualization, and predicitons

In [16]:
```
merged = merged.set_index('Serial No.')
merged
```

Out[16]:

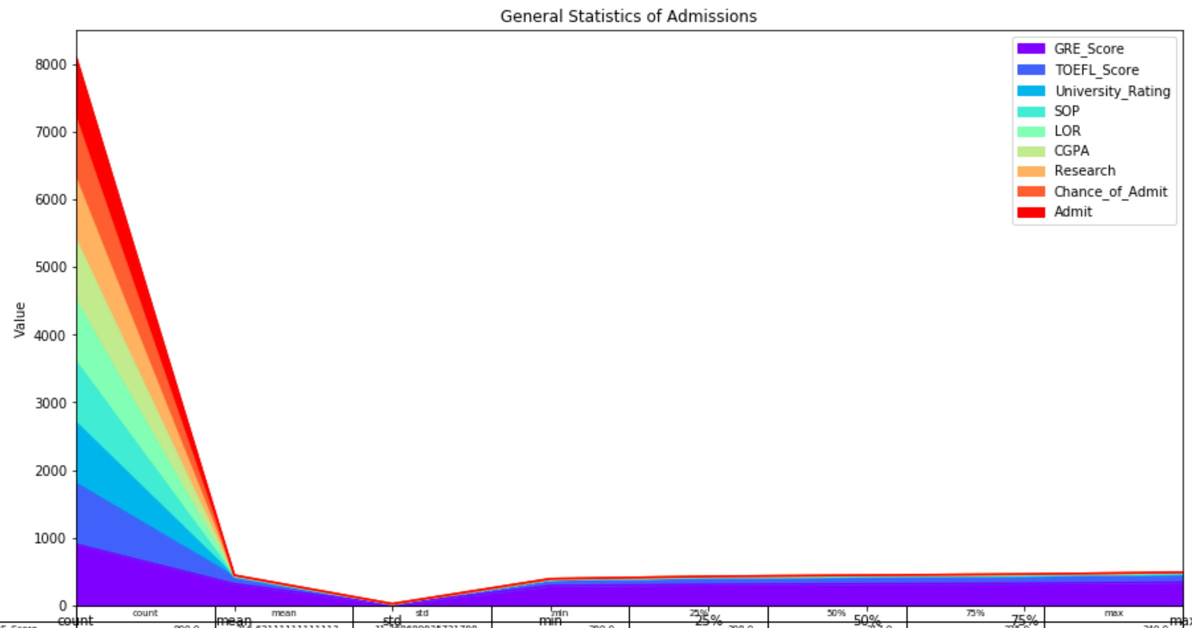
Serial No.	GRE_Score	TOEFL_Score	University_Rating	SOP	LOR	CGPA	Research	Chance_of_Admit
1	337	118	4	4.5	4.5	9.65	1	0.92
6	330	115	5	4.5	3.0	9.34	1	0.90
23	328	116	5	5.0	5.0	9.50	1	0.94
24	334	119	5	5.0	4.5	9.70	1	0.95
25	336	119	5	4.0	3.5	9.80	1	0.97
26	340	120	5	4.5	4.5	9.60	1	0.94
33	338	118	4	3.0	4.5	9.40	1	0.91
34	340	114	5	4.0	4.0	9.60	1	0.90
35	331	112	5	4.0	5.0	9.80	1	0.94
45	326	113	5	4.5	4.0	9.40	1	0.91

4. Data Exploration and Visualization

General Statistics

In [17]:
```
merged.describe().plot(kind = "area",fontsize=10, figsize = (15,8), tab
plt.xlabel('Statistics',)
plt.ylabel('Value')
plt.title("General Statistics of Admissions")
```

Out[17]: `Text(0.5, 1.0, 'General Statistics of Admissions')`



	count	mean	std	min	25%	50%	75%	max
GRE_Score	900.0	316.62111111111113	11.369699935721798	290.0	308.0	317.0	325.0	340.0
TOEFL_Score	900.0	107.28888888888889	6.073968314772823	92.0	103.0	107.0	112.0	120.0
University_Rating	900.0	3.102222222222222	1.1430476384839672	1.0	2.0	3.0	4.0	5.0
SOP	900.0	3.3855555555555554	0.9976124012000099	1.0	2.5	3.5	4.0	5.0
LOR	900.0	3.47	0.9131897183083166	1.0	3.0	3.5	4.0	5.0
CGPA	900.0	8.586433333333336	0.6008224350361041	6.8	8.14	8.57	9.0525	9.92
Research	900.0	0.5544444444444444	0.4973033199324354	0.0	0.0	1.0	1.0	1.0
Chance_of_Admit	900.0	0.7228999999999998	0.14172201322398056	0.34	0.64	0.73	0.8225	0.97
Admit	900.0	0.58	0.4938329350518604	0.0	0.0	1.0	1.0	1.0

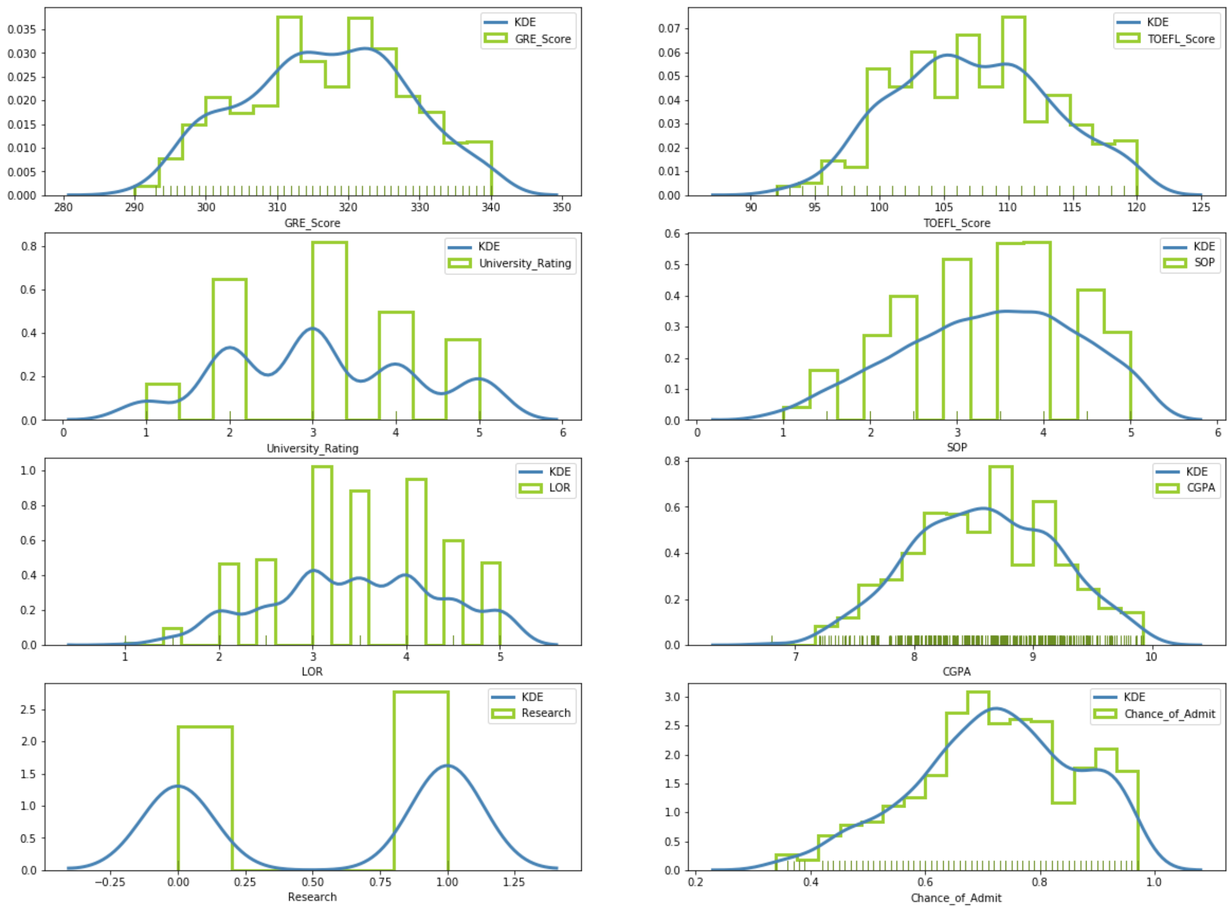
The distributions of different variables

In [18]:

```
#Exclude the last three categorical data
numerical_data = merged.iloc[:,:8]
```

In [19]:

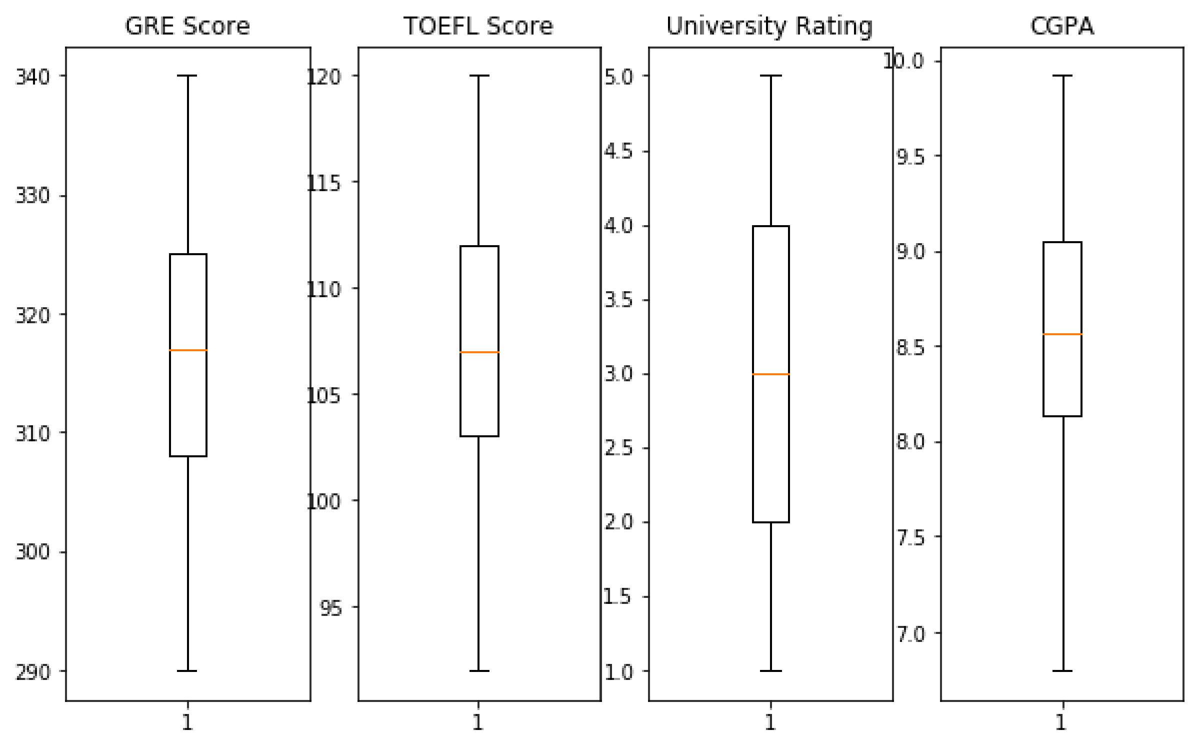
```
plt.figure(figsize=(20,15))
i = 0

for item in numerical_data.columns:
    i += 1
    plt.subplot(4, 2, i)
    sns.distplot(numerical_data[item], rug=True, rug_kws={"color": "oli
                 kde_kws={"color": "steelblue", "lw": 3, "label": "KDE"
                 hist_kws={"histtype": "step", "linewidth": 3,"alpha":
#     sns.distplot(admission_v1[item], kde=True,label="{0}".format(iten

plt.show()
```



TOEFL Score: The density of TOEFL score are between 100 and 105.
GRE Score: There is a density between 310 and 330. Being above this range would be a good feature for a candidate to stand out.
University Rating: Most of candidates come from score 3 university, and the candidates of score 2,3,4 are about half of that of score 3.
Statement of Purpose: The SoPs are mainly distributed between 2.5 and 5.
LOR: For most of candidates, their letters of recommendation are between 3 and 4.
CGPA: The CGPA are mainly distributed between 8.0 to 9.5.

Min,median and max values for GRE,TOEFL,University rating and CGPA.

In [20]:
```
plt.figure(1, figsize=(10,6))
plt.subplot(1,4, 1)
plt.boxplot(merged['GRE_Score'])
plt.title('GRE Score')

plt.subplot(1,4,2)
plt.boxplot(merged['TOEFL_Score'])
plt.title('TOEFL Score')

plt.subplot(1,4,3)
plt.boxplot(merged['University_Rating'])
plt.title('University Rating')

plt.subplot(1,4,4)
plt.boxplot(merged['CGPA'])
plt.title('CGPA')

plt.show()
```



What scores should student get if they want to have an admission chance higher than 0.75?

In [21]:
```
merged_sort=merged.sort_values(by=merged.columns[7],ascending=False)
merged_sort.head()
```

Out[21]:

Serial No.	GRE_Score	TOEFL_Score	University_Rating	SOP	LOR	CGPA	Research	Chance_of_Admit
204	334	120	5	4.0	5.0	9.87	1	0.97
144	340	120	4	4.5	4.0	9.92	1	0.97
144	340	120	4	4.5	4.0	9.92	1	0.97
25	336	119	5	4.0	3.5	9.80	1	0.97
25	336	119	5	4.0	3.5	9.80	1	0.97

In [22]:
```
merged_sort[(merged_sort['Chance_of_Admit']>0.75)].mean().reset_index()
```

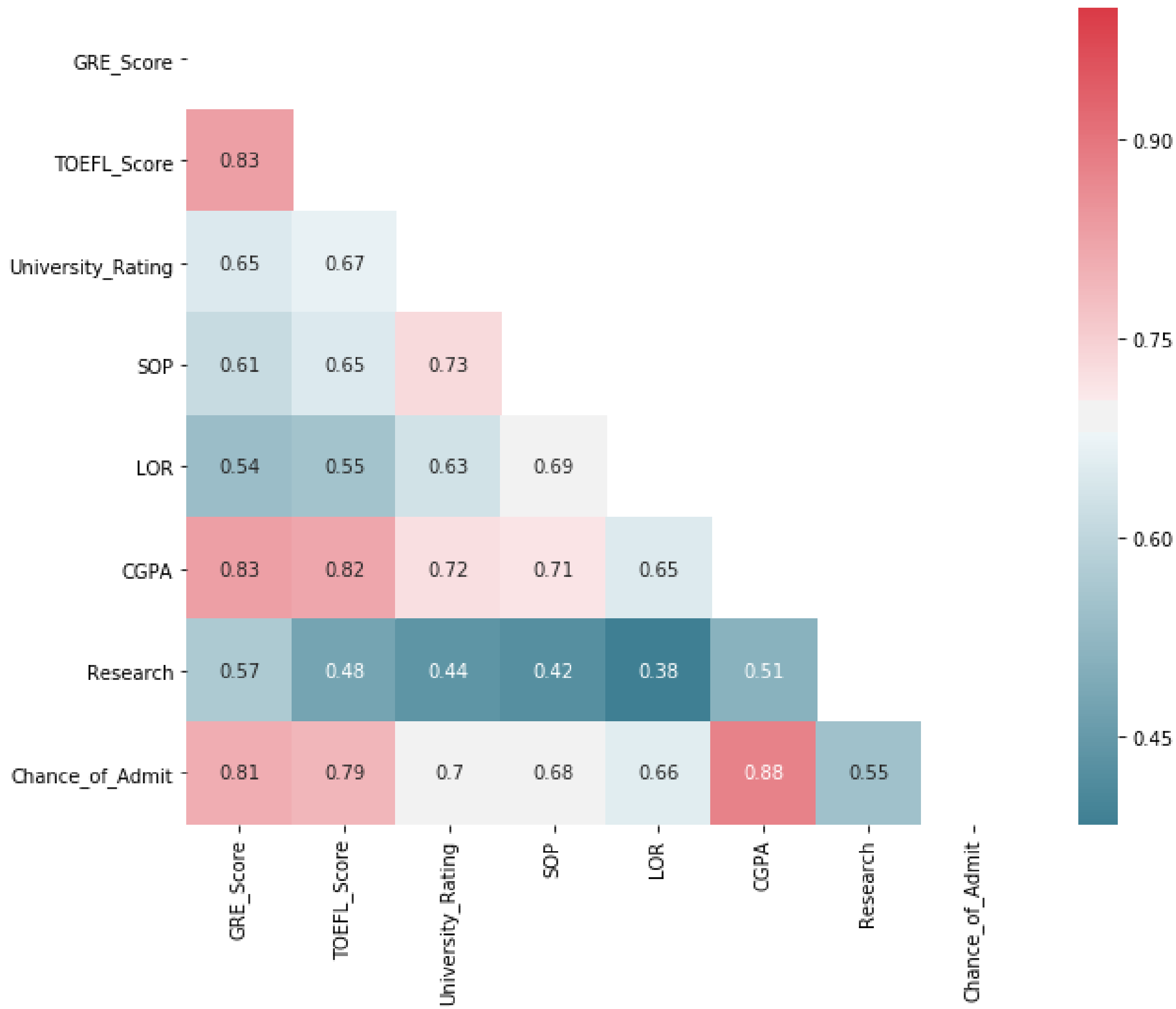
Out[22]:

	index	0
0	GRE_Score	325.884817
1	TOEFL_Score	112.073298
2	University_Rating	3.950262
3	SOP	4.102094
4	LOR	4.061518
5	CGPA	9.114136
6	Research	0.848168
7	Chance_of_Admit	0.854607
8	Admit	1.000000

Assuming students with 0.75 chance of admission have secured admission. To have a 75% Chance to get admission, student should have at least a GRE score of 326, TOEFL score of 112, CGPA of 9.11. Students with scores more than this line have greater chance to get admission.

Correlation between All Columns

In [23]:
```
corr_matrix = numerical_data.corr()
plt.figure(figsize = (10,8))
cmap = sns.diverging_palette(220, 10, as_cmap=True)
mask = np.zeros_like(corr_matrix, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True
sns.heatmap(corr_matrix,cmap=cmap,annot=True,mask=mask);
```



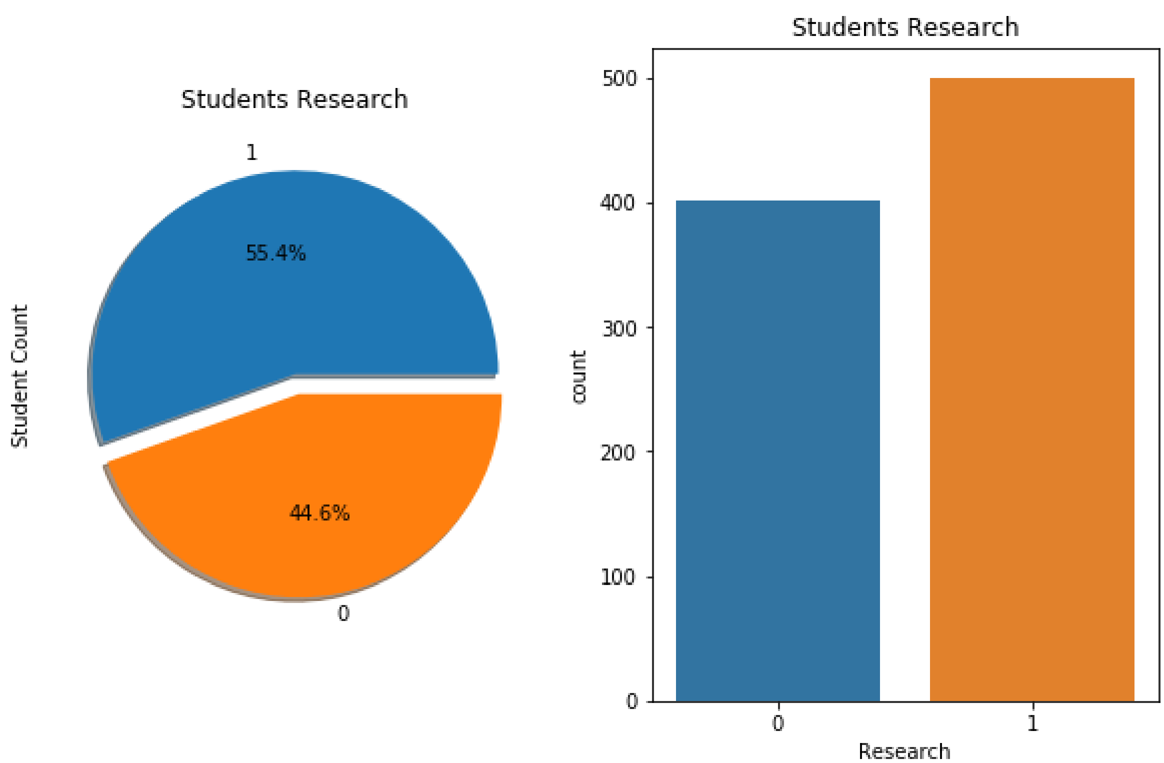
The 3 most important features for admission to the Master: CGPA, GRE SCORE, and TOEFL SCORE
The 3 least important features for admission to the Master: Research, LOR, and SOP

How important is Research to get an Admission?

In [24]:
```
a=len(merged[merged.Research==1])
b=len(merged[merged.Research==0])
print('Total number of students',a+b)
print('Students having Research:',len(merged[merged.Research==1]))
print('Students not having Research:',len(merged[merged.Research==0]))
```

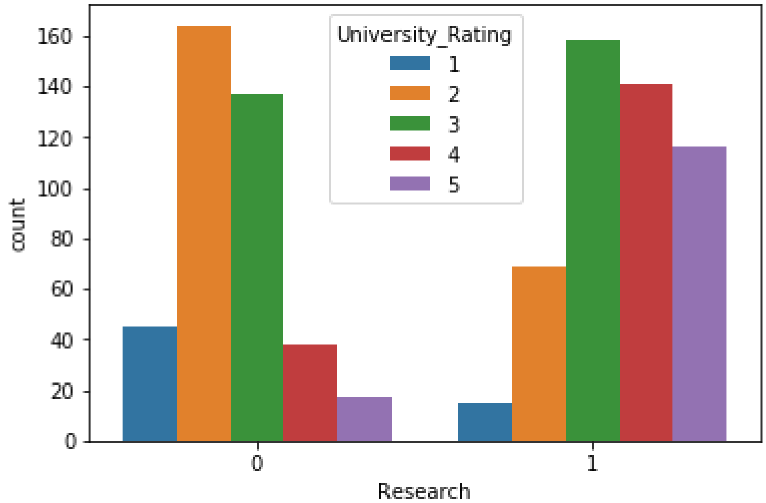
```
Total number of students 900
Students having Research: 499
Students not having Research: 401
```

In [25]:
```
f,ax=plt.subplots(1,2,figsize=(10,6))
merged['Research'].value_counts().plot.pie(explode=[0,0.1],autopct='%1.
ax[0].set_title('Students Research')
ax[0].set_ylabel('Student Count')
sns.countplot('Research',data=merged,ax=ax[1])
ax[1].set_title('Students Research')
plt.show()
```



Around 60% Students have research experience.

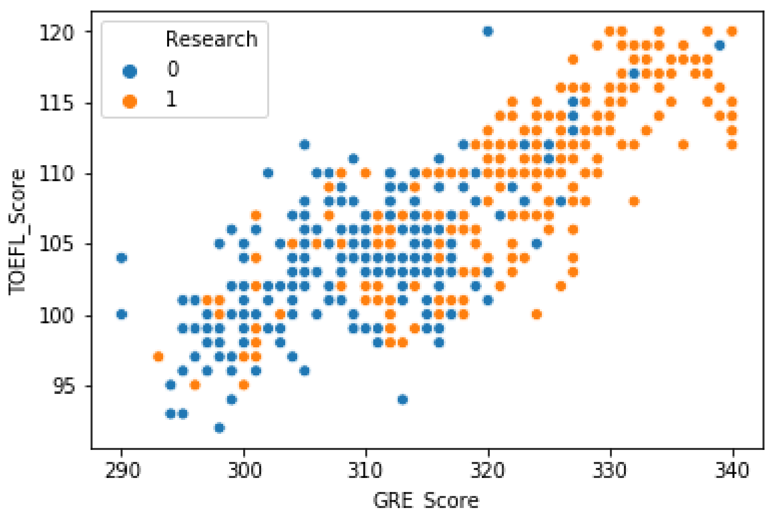
In [26]:
```
sns.countplot(x='Research', hue='University_Rating', data=merged)
plt.show()
```



Students come from university with higher ratings tend to be more possible of having research experience.

In [27]:
```
sns.scatterplot(data=merged,x='GRE_Score',y='TOEFL_Score',hue='Research
```

Out[27]: <matplotlib.axes._subplots.AxesSubplot at 0x1a2244f5f8>



Students with research experience have good GRE scores and TOEFL scores.

Count the percentage of students, in each admission chance level, having research experience.

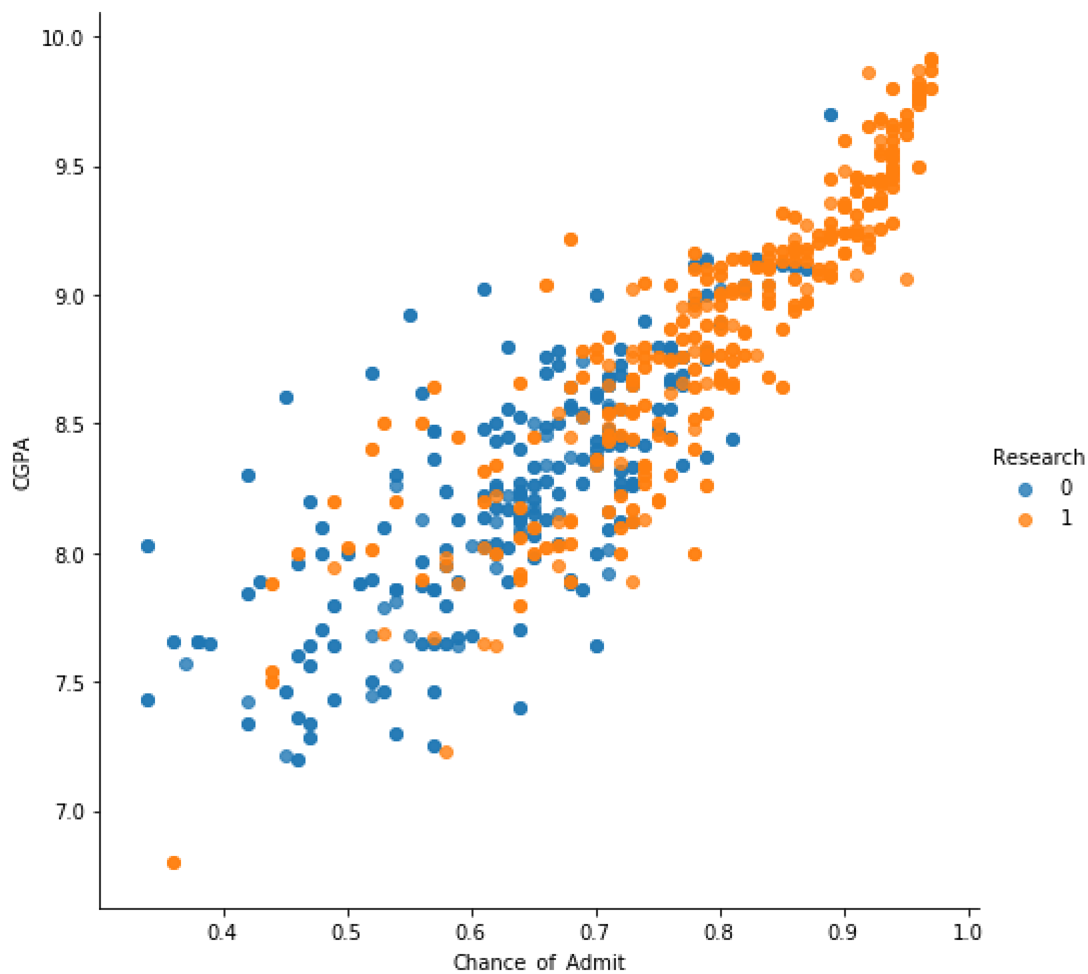
In [28]:
```
groupbyed = merged.groupby('Admit_Chance_Level')
groupbyed['Research'].value_counts(normalize=True) * 100
```

Out[28]:
```
Admit_Chance_Level  Research
High                1           100.000000
Low                 0            77.976190
                    1            22.023810
Medium              1            54.166667
                    0            45.833333
Medium High         1            87.179487
                    0            12.820513
Medium Low          0            69.729730
                    1            30.270270
Name: Research, dtype: float64
```

Percentage of students having research experience goes higher as admission chance level increases.

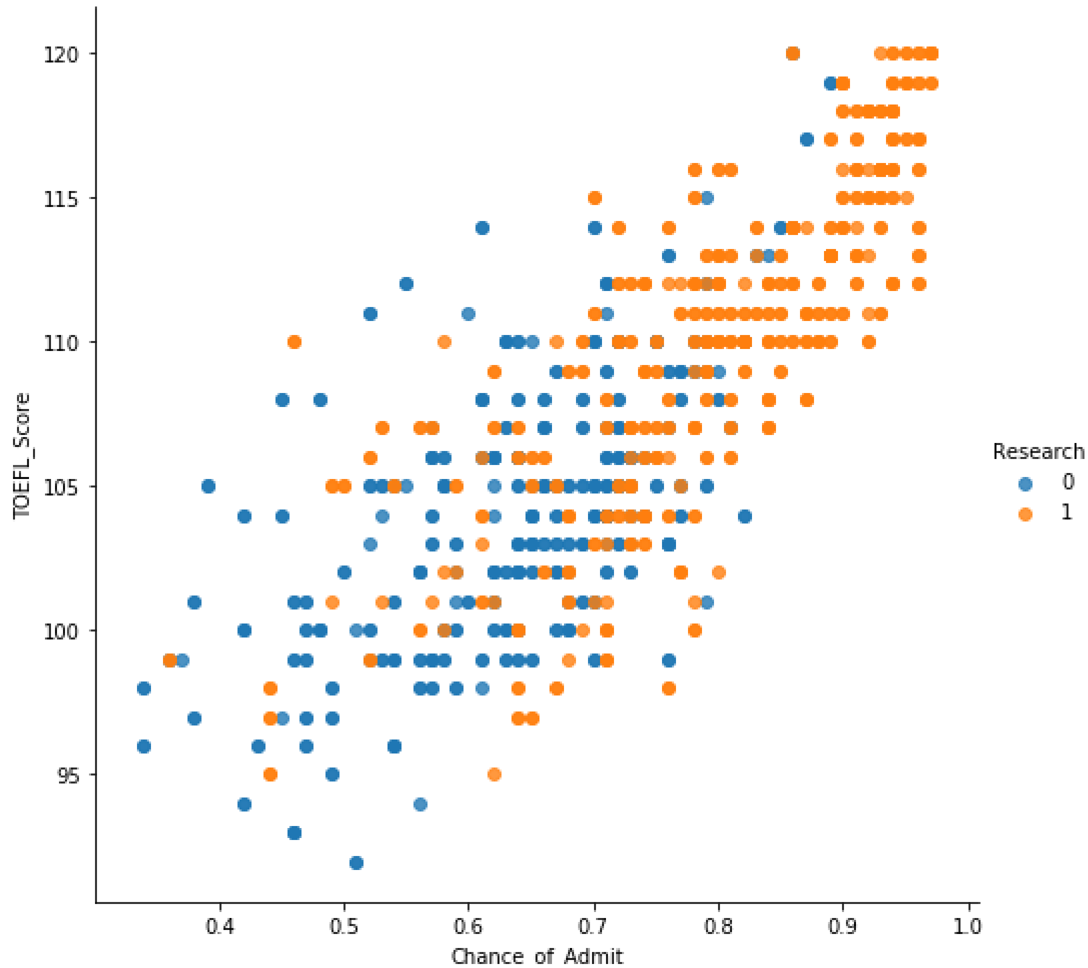
Understanding the relation between different factors responsible for graduate admissions

In [29]:
```
sns.lmplot('Chance_of_Admit', 'CGPA', data=numerical_data, hue='Researc
```



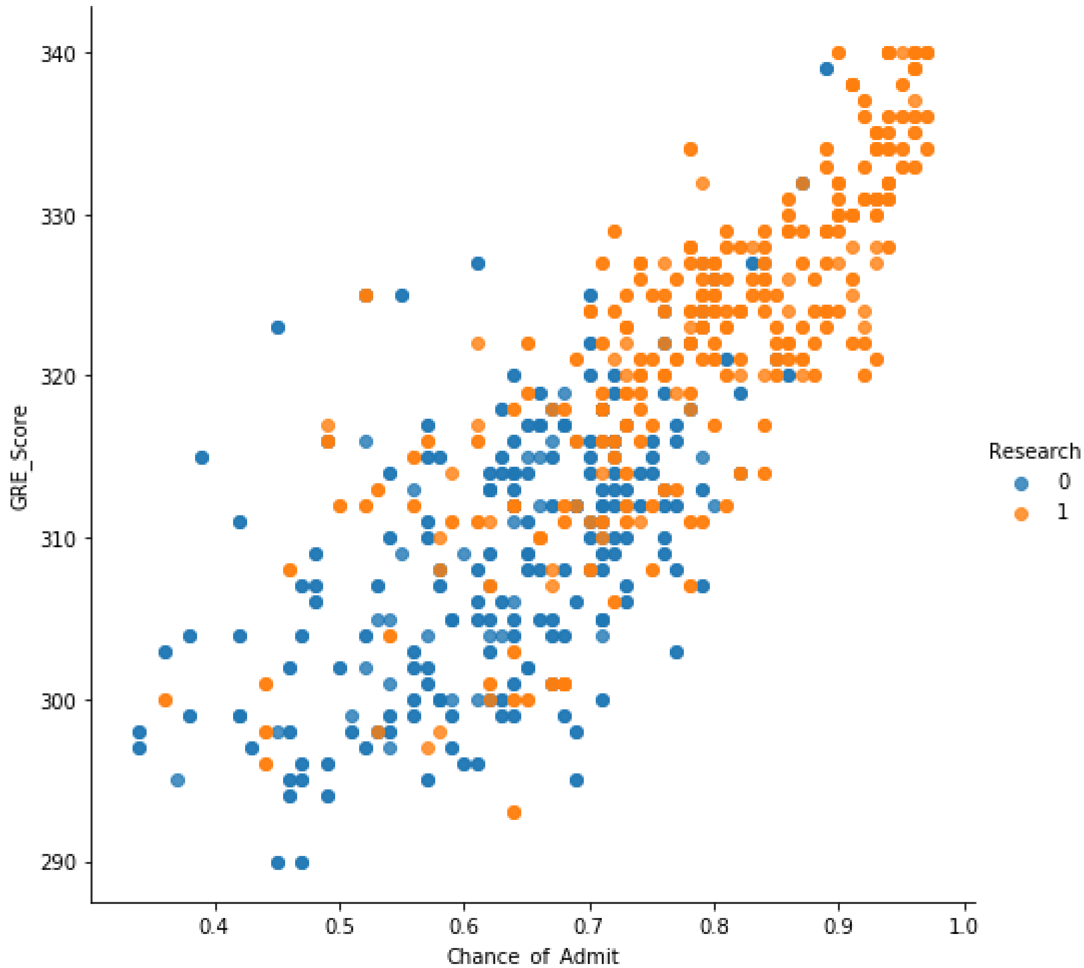
Highest Admission Based on CGPA in Between 8.5 to 9.0, with nearly all students having research experience.

In [30]:
```
1 sns.lmplot('Chance_of_Admit','TOEFL_Score', data=numerical_data, hue='R
```



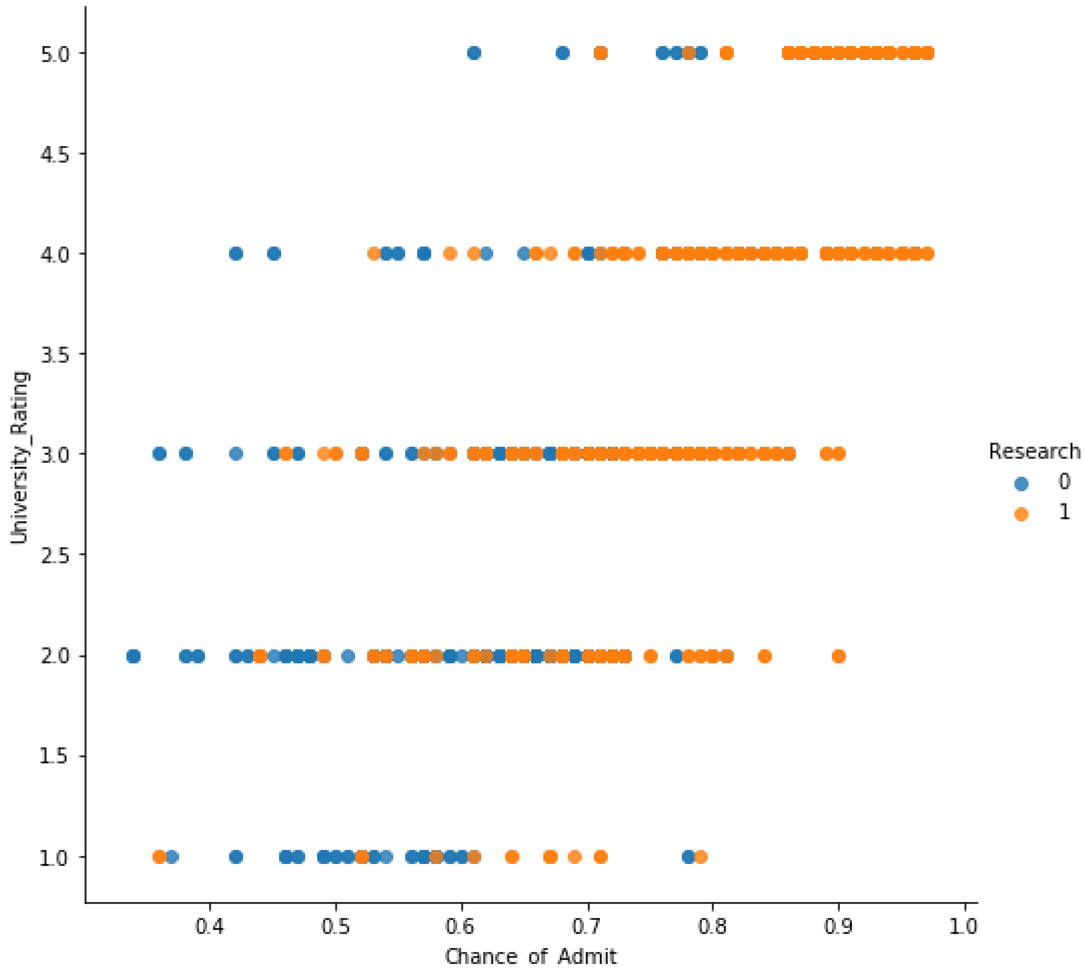
TOEFL Score mostly range from 100 to 120. Student with highest admission rate usually score from 115 to 120.

In [31]:
```
1 sns.lmplot('Chance_of_Admit','GRE_Score', data=numerical_data, hue='Res
```



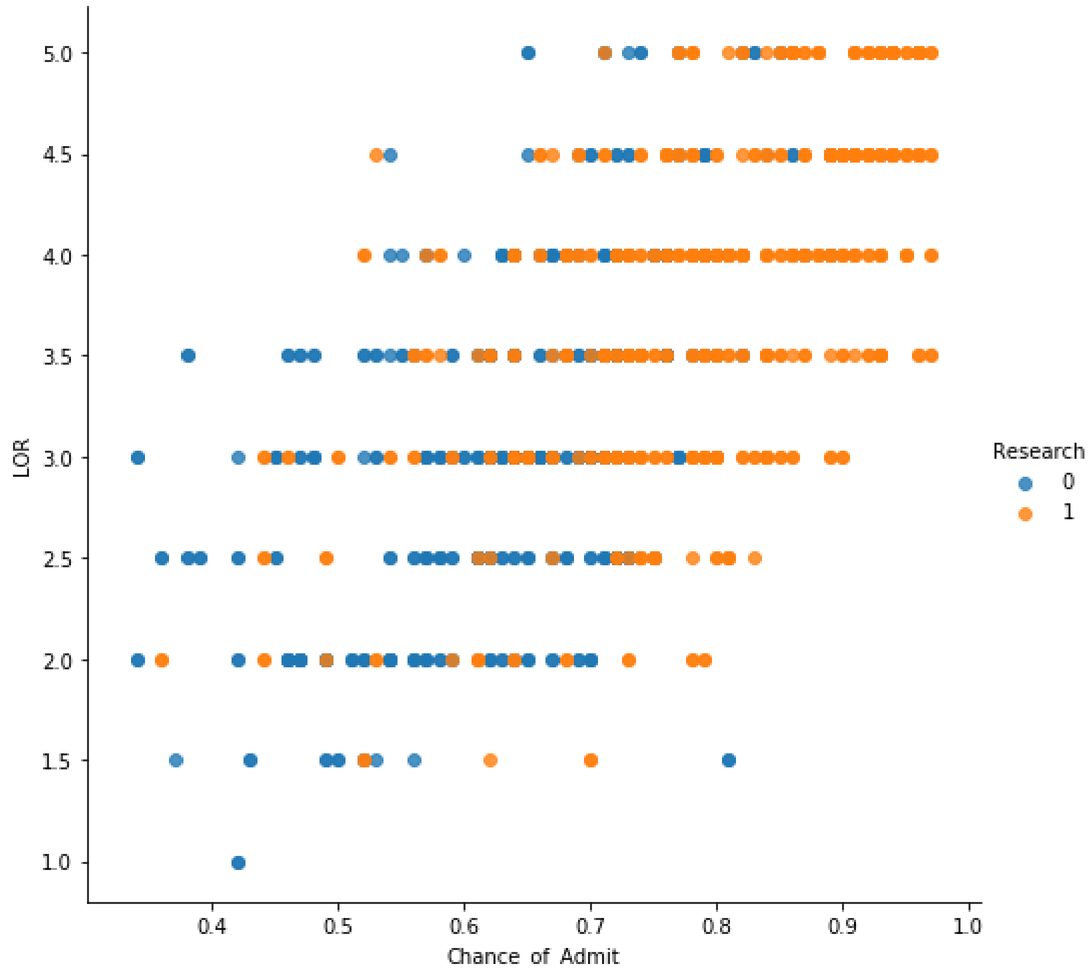
Clutser of GRE Score is Belong to 300 to 330. Students score above 330 have an possibility of admission higher than 0.9. Again, the higher the admission rate, the higher chance students would have research experience.

In [32]:
```
1 sns.lmplot('Chance_of_Admit','University_Rating', data=numerical_data,
```



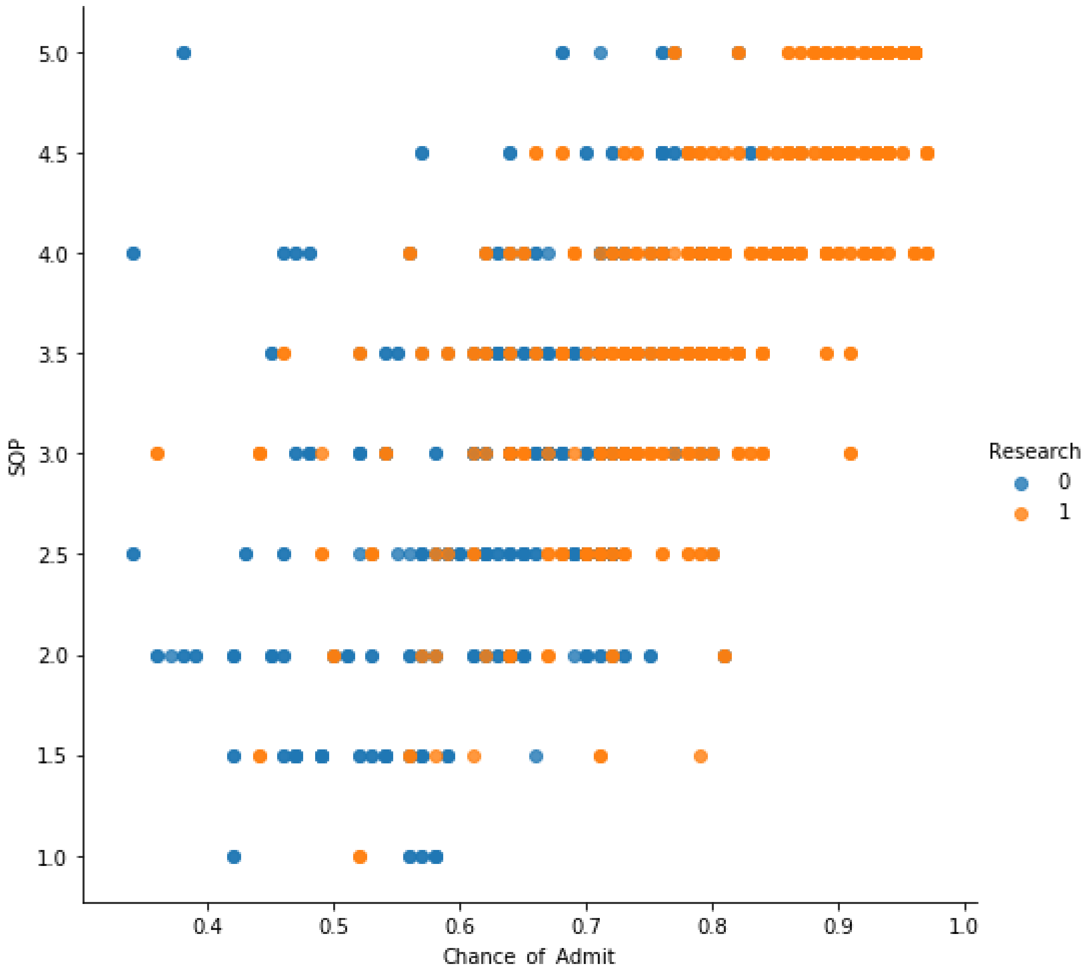
Higer university rating candidates would have a slightly higher chances of admit.

In [33]:
```
1 sns.lmplot('Chance_of_Admit','LOR', data=numerical_data, hue='Research'
```



Higer level LOR candidates would have a higher chances of admit.

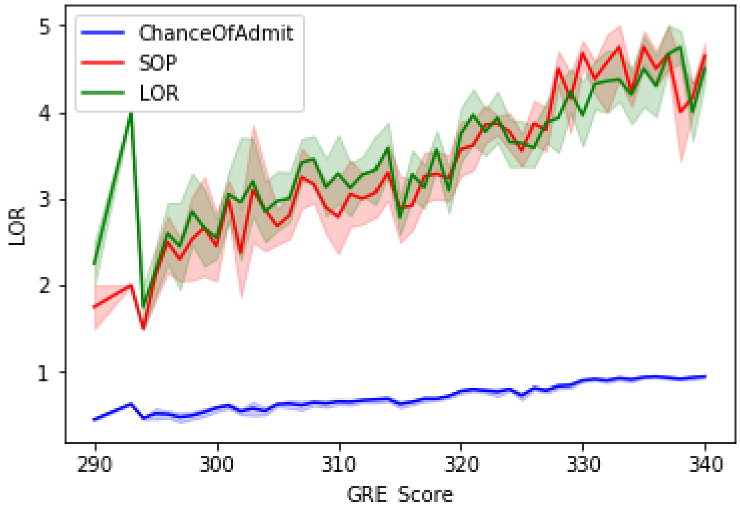
In [34]:
```
1 sns.lmplot('Chance_of_Admit','SOP', data=numerical_data, hue='Research'
```



Higer level SOP candidates would have a higher chances of admit.

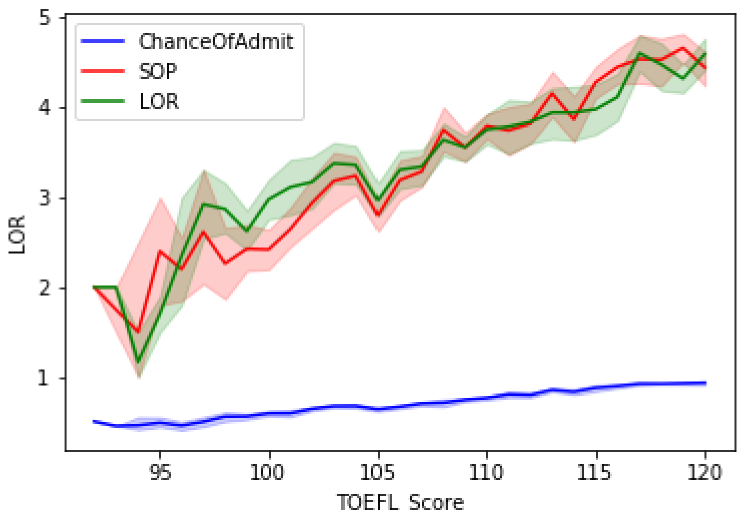
In [35]:

```
sns.lineplot(x="GRE_Score", y="Chance_of_Admit",
                  data=numerical_data,color='b',label='ChanceOfAdmit')
sns.lineplot(x="GRE_Score", y="SOP",
                  data=numerical_data,color='r',label='SOP')
sns.lineplot(x="GRE_Score", y="LOR",
                  data=numerical_data,color='G',label='LOR')
plt.legend(loc=2)
plt.show()
```

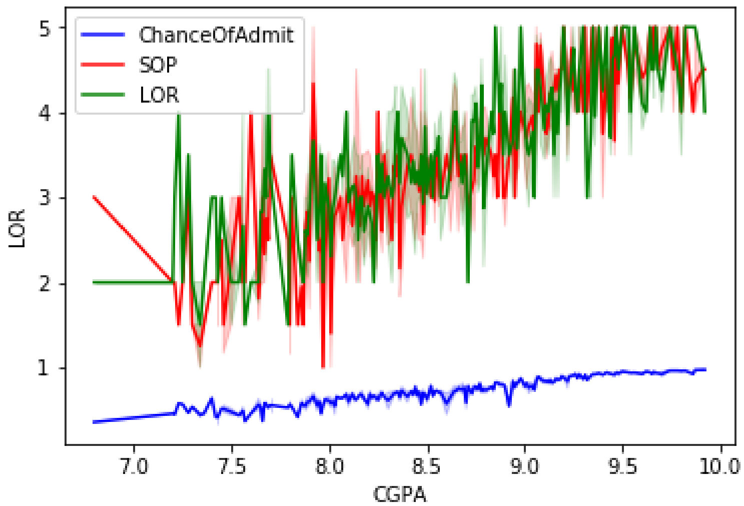


In [36]:

```
sns.lineplot(x="TOEFL_Score", y="Chance_of_Admit",
                  data=numerical_data,color='b',label='ChanceOfAdmit')
sns.lineplot(x="TOEFL_Score", y="SOP",
                  data=numerical_data,color='r',label='SOP')
sns.lineplot(x="TOEFL_Score", y="LOR",
                  data=numerical_data,color='G',label='LOR')
plt.legend(loc=2)
plt.show()
```



In [37]:
```
sns.lineplot(x="CGPA", y="Chance_of_Admit",
                data=numerical_data,color='b',label='ChanceOfAdmit')
sns.lineplot(x="CGPA", y="SOP",
                data=numerical_data,color='r',label='SOP')
sns.lineplot(x="CGPA", y="LOR",
                data=numerical_data,color='G',label='LOR')
plt.legend(loc=2)
plt.show()
```



From the data exploration and visualization above, we can see that student's GRE score, TOEFL score, and CPA having more significant impact on whether they can be admitted or not; while university rating, statement of purpose, letter of recommendation show a weaker influence. Finally, students with higher admission rate usually have research experience. That is to say, research experience, though shows a relatively low correlation, weighs a lot in the admission process.

5. Regression Analysis

train_test_split:

It splits the data into random train (80%) and test (20%) subsets.

In [38]:
```
numerical_data = numerical_data.reset_index()

target = 'Chance_of_Admit'
IDcol = 'Serial No.'
x_columns = [x for x in numerical_data .columns if x not in [target, ID
X = numerical_data[x_columns]
y = numerical_data['Chance_of_Admit']
```

In [39]:
```
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2)
```

Note about r2_score:

We will use R-squared score to compare the accuracy for each regression model as it represents how close the data are to the fitted regression line. That is to say, the higher the R-squared, the better the model fits the data and makes better predictions.The best possible score is 1.0 for r2_score.

5.1 Linear Regression Model

In [40]:
```
from sklearn.linear_model import LinearRegression
lr = LinearRegression().fit(X_train,y_train)
```

In [41]:
```
y_pred_lr = lr.predict(X_test)
r2_score_lr = r2_score(y_test,y_pred_lr)
r2_score_lr
```

Out[41]:
```
0.804590919255666
```

5.2 DecisionTree Regression Model

In [42]:
```
from sklearn.tree import DecisionTreeRegressor
dt = DecisionTreeRegressor().fit(X_train,y_train)
```

In [43]:
```
y_pred_dt = dt.predict(X_test)
r2_score_dt = r2_score(y_test,y_pred_dt)
r2_score_dt
```

Out[43]:
```
0.9375056949439617
```

5.3 Random Forest Regression Model

In [44]:
```
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor()
from pprint import pprint
print('Parameters currently in use:\n')
pprint(rf.get_params())
```

```
Parameters currently in use:

{'bootstrap': True,
 'criterion': 'mse',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 'warn',
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

Tuning the parameters of the model to get more accurate predictions.

In [45]:
```
from sklearn.model_selection import RandomizedSearchCV

# Number of features to consider at every split
max_features = ['auto', 'sqrt','log2']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Method of selecting samples for training each tree
bootstrap = [True, False]

# Create the random grid
random_grid = {'max_features': max_features,
               'max_depth': max_depth,
               'bootstrap': bootstrap}
pprint(random_grid)
```

```
{'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
 'max_features': ['auto', 'sqrt', 'log2']}
```

In [46]:
```
# Use the random grid to search for best hyperparameters
# First create the base model to tune
rf = RandomForestRegressor(n_estimators=100)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = ra
# Fit the random search model
rf_random.fit(X_train,y_train)
```

Out[46]:
```
RandomizedSearchCV(cv=3, error_score='raise-deprecating',
          estimator=RandomForestRegressor(bootstrap=True, criterion='ms
e', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=None,
           oob_score=False, random_state=None, verbose=0, warm_start=Fals
e),
          fit_params=None, iid='warn', n_iter=10, n_jobs=None,
          param_distributions={'max_features': ['auto', 'sqrt', 'log2'],
'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None], 'boots
trap': [True, False]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score='warn', scoring=None, verbose=0)
```

In [47]:
```
print('Best Parameters from fitting the random research:\n')
rf_random.best_params_
```

```
Best Parameters from fitting the random research:
```

Out[47]:
```
{'max_features': 'sqrt', 'max_depth': 110, 'bootstrap': False}
```

In [69]:
```
rf = RandomForestRegressor(max_depth=110, max_features='sqrt', bootstra
rf = rf.fit(X_train,y_train)
```

```
/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/forest.py:246: Fu
tureWarning: The default value of n_estimators will change from 10 in ver
sion 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

In [70]:
```
y_pred_rf = rf.predict(X_test)
r2_score_rf = r2_score(y_test,y_pred_rf)
r2_score_rf
```

Out[70]:
```
0.9488301647111792
```

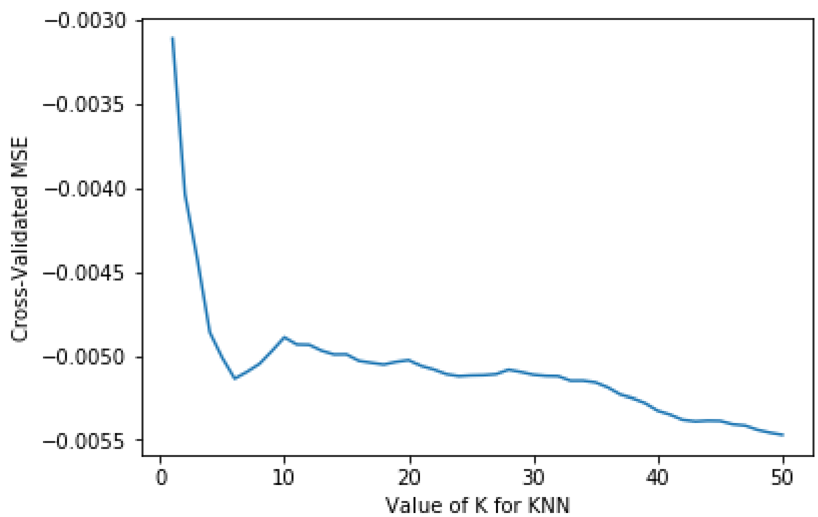
5.4 KNeighbors Model

In [50]:
```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import cross_val_score
```

Finding the optimal K valueto get more accurate predictions.

In [51]:
```
k_list = list(range(1,51))
cv_scores = []

for k in k_list:
    knn = KNeighborsRegressor(n_neighbors=k)
    scores = cross_val_score(knn, X_train, y_train, cv=10, scoring='neg
    cv_scores.append(scores.mean())
```

In [52]:
```
plt.plot(k_list, cv_scores)
plt.xlabel('Value of K for KNN')
plt.ylabel('Cross-Validated MSE')
plt.show()
```



In [53]:
```
MSE = [x for x in cv_scores]
best_k = k_list[MSE.index(min(MSE))]
print("The best number of neighbors K is %d." % best_k)
```

```
The best number of neighbors K is 50.
```

In [54]:
```
knn = KNeighborsRegressor(n_neighbors=50)
knn = knn.fit(X_train,y_train)
```

In [55]:
```
y_pred_knn = knn.predict(X_test)
r2_score_knn = r2_score(y_test,y_pred_knn)
r2_score_knn
```

Out[55]:
```
0.7612420294904299
```

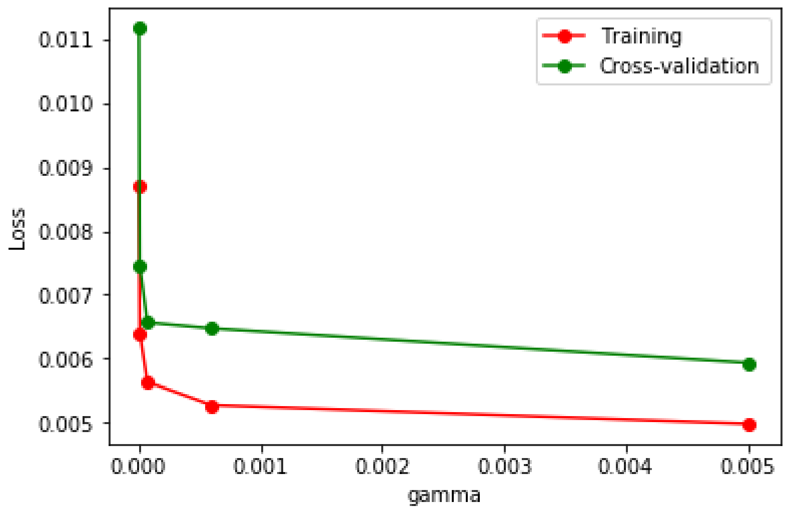
5.5 SVM Model

In [56]:
```
from sklearn.svm import SVR
```

Tuning the parameters of the model to get more accurate predictions.

In [57]:
```
from sklearn.model_selection import  validation_curve

param_range = np.logspace(-6, -2.3, 5)
train_loss, test_loss = validation_curve(
        SVR(), X, y, param_name='gamma', param_range=param_range, cv=10
        scoring='neg_mean_squared_error')
train_loss_mean = -np.mean(train_loss, axis=1)
test_loss_mean = -np.mean(test_loss, axis=1)

plt.plot(param_range, train_loss_mean, 'o-', color="r",label="Training"
plt.plot(param_range, test_loss_mean, 'o-', color="g",label="Cross-vali

plt.xlabel("gamma")
plt.ylabel("Loss")
plt.legend(loc="best")
```

Out[57]: <matplotlib.legend.Legend at 0x1a225525c0>



In [58]:
```
# From the graph above, we can see that the model would have the least
svm = SVR(gamma=0.005).fit(X_train,y_train)
```

In [59]:
```
y_pred_svm = svm.predict(X_test)
r2_score_svm = r2_score(y_test,y_pred_svm)
r2_score_svm
```

Out[59]: 0.7462571620503973

5.6 OLS Model

In [60]:
```
import statsmodels.formula.api as smf
%matplotlib inline
```

In [61]:
```
ols = smf.ols('Chance_of_Admit ~ GRE_Score + TOEFL_Score + University_R
print(ols.summary())
```

```
                            OLS Regression Results
==============================================================================
=====
Dep. Variable:        Chance_of_Admit   R-squared:
0.813
Model:                            OLS   Adj. R-squared:
0.812
Method:                 Least Squares   F-statistic:
555.6
Date:                Tue, 09 Jul 2019   Prob (F-statistic):           4.56
e-320
Time:                        15:35:09   Log-Likelihood:                  1
237.5
No. Observations:                 900   AIC:                             -
2459.
Df Residuals:                     892   BIC:                             -
2421.
Df Model:                           7
Covariance Type:            nonrobust
==============================================================================
============
                        coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
------------
Intercept            -1.2691      0.080    -15.915      0.000      -1.426
-1.113
GRE_Score             0.0018      0.000      4.725      0.000       0.001
0.003
TOEFL_Score           0.0028      0.001      4.146      0.000       0.001
0.004
University_Rating     0.0059      0.003      1.997      0.046       0.000
0.012
SOP                  -0.0004      0.004     -0.106      0.916      -0.007
0.007
LOR                   0.0189      0.003      5.711      0.000       0.012
0.025
CGPA                  0.1187      0.008     15.644      0.000       0.104
0.134
Research              0.0243      0.005      4.792      0.000       0.014
0.034
==============================================================================
=====
Omnibus:                      193.255   Durbin-Watson:
0.817
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               44
1.104
Skew:                          -1.160   Prob(JB):                      1.6
4e-96
Kurtosis:                       5.525   Cond. No.                      1.3
0e+04
==============================================================================
=====
```

```
Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is co
rrectly specified.
[2] The condition number is large, 1.3e+04. This might indicate that ther
e are
strong multicollinearity or other numerical problems.
```

In [62]:

```
y_pred_ols = ols.predict(X_test)
ols.rsquared
```

Out[62]:

```
0.8134478843618487
```

Printing R2 Score for each model

Visualizing and comparing results

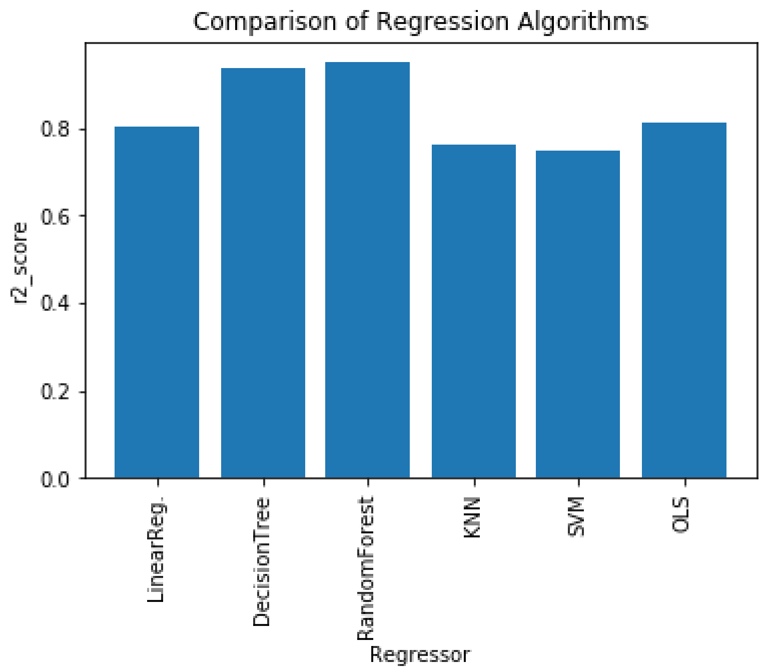
In [71]:

```
models = [['DecisionTree :',dt],
            ['Linear Regression :', lr],
            ['RandomForest :',rf],
            ['KNN :', knn],
            ['SVM :', svm]]

print("R2 Score for each model:")
for name,model in models:
    model = model
    predictions = model.predict(X_test)
    print(name, (r2_score(y_test, predictions)))

print('Ordinary Least Squares:', ols.rsquared)
```

```
R2 Score for each model:
DecisionTree : 0.9375056949439617
Linear Regression : 0.804590919255666
RandomForest : 0.9488301647111792
KNN : 0.7612420294904299
SVM : 0.7462571620503973
Ordinary Least Squares: 0.8134478843618487
```

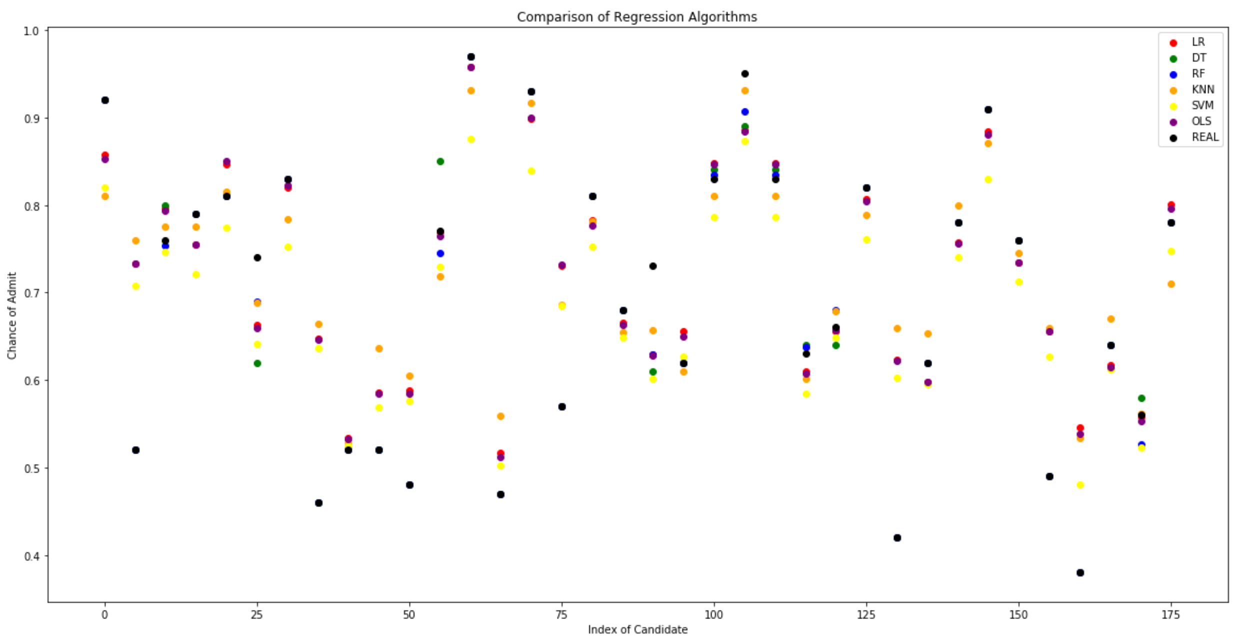
In [72]:

```
y = np.array([r2_score_lr,r2_score_dt,r2_score_rf,r2_score_knn, r2_scor
x = ["LinearReg.","DecisionTree","RandomForest","KNN","SVM","OLS"]
plt.bar(x,y)
plt.title("Comparison of Regression Algorithms")
plt.xlabel("Regressor")
plt.ylabel("r2_score")
plt.xticks(rotation=90)
plt.show()
```



In [73]:

```
plt.figure(figsize=(20,10))
red = plt.scatter(np.arange(0,180,5),y_pred_lr[0:180:5],color = "red")
green = plt.scatter(np.arange(0,180,5),y_pred_dt[0:180:5],color = "gree
blue = plt.scatter(np.arange(0,180,5),y_pred_rf[0:180:5],color = "blue"
orange = plt.scatter(np.arange(0,180,5),y_pred_knn[0:180:5],color = "or
yellow = plt.scatter(np.arange(0,180,5),y_pred_svm[0:180:5],color = "ye
purple = plt.scatter(np.arange(0,180,5),y_pred_ols[0:180:5],color = "pu
black = plt.scatter(np.arange(0,180,5),y_test[0:180:5],color = "black")
plt.title("Comparison of Regression Algorithms")
plt.xlabel("Index of Candidate")
plt.ylabel("Chance of Admit")
plt.legend((red,green,blue,orange,yellow,purple,black),('LR', 'DT', 'RF
plt.show()
```

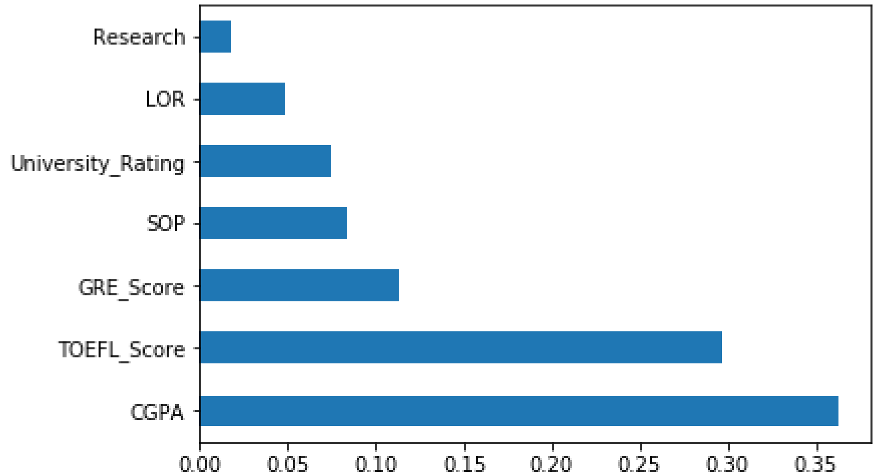


The best model is Random Forest which has the highest R2 score (0.95)

6. Conclusion and Summary

In [74]:
```
1 feature_importances = pd.Series(rf.feature_importances_, index=x_columr
2 feature_importances.nlargest(7).plot(kind='barh')
```

Out[74]: <matplotlib.axes._subplots.AxesSubplot at 0x1a22597e80>



Feature Selection is the process to select those features which contribute most to the prediction variable or output.It reduces overfitting, improves accuracy and reduces training time.

The importances of variables are presented above, and GPA is the most important parameter

CGPA: 0.36

TOEFL SCORE: 0.29

GRE SCORE: 0.11

SOP: 0.08

UNIVERSITY RATING: 0.7

LOR: 0.05

RESEARCH: 0.02

In [75]:

```
# Predicting the Rating values for testing data
PredAdmit = rf.predict(X_test)

# Creating a DataFrame of Testing data
AdmitData=pd.DataFrame(X_test, columns=x_columns)
AdmitData['ChancesOfAdmit']=y_test
AdmitData['PredictedChancesOfAdmit']=PredAdmit
AdmitData.head()
```

Out[75]:

	GRE_Score	TOEFL_Score	University_Rating	SOP	LOR	CGPA	Research	ChancesOfAdmit
71	320	110	5	5.0	4.5	9.22	1	0.92
439	312	106	3	4.0	3.5	8.79	1	0.81
859	297	96	2	2.5	1.5	7.89	0	0.43
176	317	103	3	2.5	3.0	8.54	1	0.73
427	324	110	4	3.0	3.5	8.97	1	0.84

In [76]:

```
# Calculating the Absolute Percentage Error committed in each predictic
AdmitData['APE']=100 * (abs(AdmitData['ChancesOfAdmit'] - AdmitData['Pr
# Final accuracy of the model
print('Mean Absolute Percent Error(MAPE): ',round(np.mean(AdmitData['AP
print('Average Accuracy of the model: ',100 - round(np.mean(AdmitData['
```

```
Mean Absolute Percent Error(MAPE):  2 %
Average Accuracy of the model:  98 %
```

The most important parameter is CGPA
The model is 98% accurate to predict admission status of a candidate