Forecasting Graduate Admission for Master's Application

Jhansi Kurma (jnk22)

jnk22@njit.edu

Option 1: Supervised Data Mining (Classification)

CS 634 - Data Mining

Final Term Project

Table of Contents:

S. No.	Title	Page No.
1.	Executive Summary	3
2.	Problem Definition and Goal	3
3.	Data Description	3
4.	Data Exploration	4
5.	Methods	5
6.	Evaluation	6
7.	Source Code	7
8.	Conclusion	23
9.	Appendix	24
10.	References	27

Executive Summary:

The "Graduate Admisisons"[1] dataset contains several parameters that are important and are used for determining a candidate's chance of acceptance in a graduate institution. It is explicitly created for predicting graduate admission from an Indian perspective. It helps students by giving them a fair idea of how they can be shortlisted and on what basis.

As the number of graduate applications increase every year, so does the arduousness of acceptance. Though there are several other factors that influence the reviewer's decision, some of the crucial parameters are the examination scores and CGPA. This dataset helps students to figure out where they stand and compare their chance of admission rates.

Since the chance of admission ranges from 0 to 1 in this dataset, we could aggregate the values into three bins (or levels) for forecasting the dataset using classification algorithms. We will be using Decision Tree classifier and Random Forest classifier using k-fold cross validation as an approach to compare the algorithms. Once the data is predicted and compared, we can conclude that Random forest is the best algorithm to choose for predicting the chance of admission. Furthermore, as professor permitted, correlation is calculated among all the variables and choose important features, and Random Forest classification is applied again to check for accuracy.

Problem Definition and Goal:

The dataset mainly focuses on various features that are dependent on chances of admission. The data is shown in terms of Indian students' perspective.

Our main goal is to predict the chance of admission for various discrete values and categorical levels, and to finalize the best algorithm based on their average scores.

Data Description:

The dataset has 9 features and 400 observations, in which all the values are numerical. The features include Serial No., GRE Score, TOEFL Score, University Rating, SOP, LOR, CGPA, Research and Chance of Admit.

- GRE Scores are represented out of 340
- TOEFL Scores are represented out of 120
- University Rating, SOP, and LOR ranges from 1 to 5

- Research is either 0 (no) or 1 (yes)
- Chance of Admit ranges from 0 to 1.

	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

Fig 1. First 3 rows of the dataset

Data Exploration:

Before exploring the data, we can delete the Serial No. column as it has little to no impact on the target variable, as shown in Fig 2. To explore the data statistically, we display heatmap of the variables' correlation to understand each other's relationship.

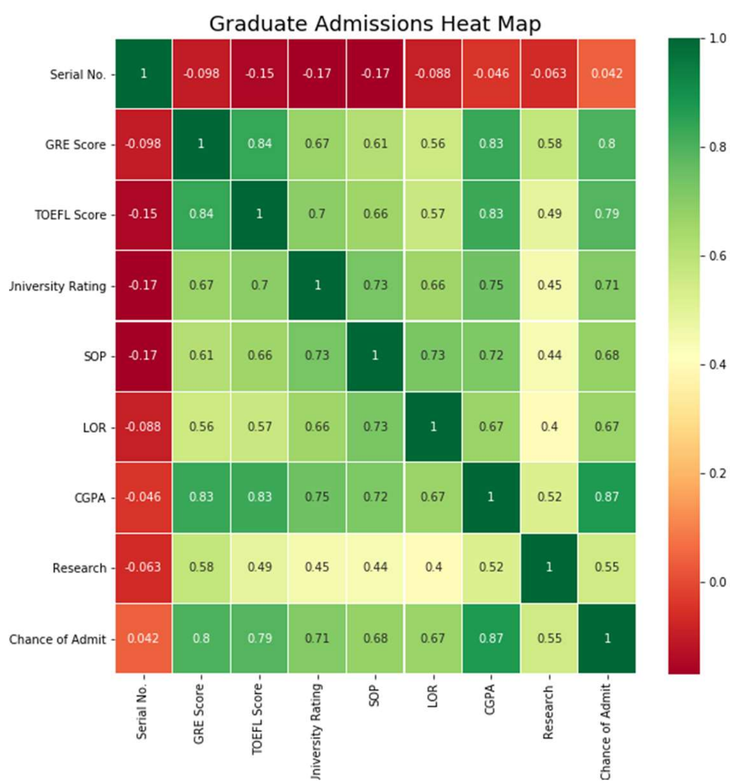


Fig 2. Correlation heatmap of all features

As seen on the heatmap, we can conclude that Serial No. is of no importance to Chance of Admit variable. Moreover, we can also say that CGPA, GRE Score and TOEFL Score are highly correlated to the target variable than the other features. As shown in Appendix, we can see more visualizations to understand how each feature is dependent on one another.

Since the target variable contains values from 0 to 1, we modify the dataset by adding a column "Chance of Admit Bins" that categorizes the target variable values into three levels, namely Low, Medium and High.

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit	Chance of Admit Bins
0	1	337	118	4	4.5	4.5	9.65	1	0.92	High
1	2	324	107	4	4.0	4.5	8.87	1	0.76	Medium
2	3	316	104	3	3.0	3.5	8.00	1	0.72	Medium

Fig 3. Binned dataset

Methods:

To perform k – fold cross validation (k = 10) approach on the dataset, we split the dataset into two data frames, namely x and y, where x data frame contains independent features and y data frame contains the dependent feature. Since the target variable now is 'Chance of Admit Bins', which is labeled and classified into three categories, classification algorithms are applied to the dataset.

```
Data Frame x(independent variables):
    GRE Score  TOEFL Score  University Rating  SOP  LOR  CGPA  Research
0         337          118                  4  4.5  4.5  9.65         1
1         324          107                  4  4.0  4.5  8.87         1
2         316          104                  3  3.0  3.5  8.00         1
Data Frame y(dependent variable):
   Chance of Admit Bins
0                  High
1                Medium
2                Medium
```

Fig 4. x and y data frames

1. **Decision Tree Classifier (Category 3) and Random Forest Classifier (Category 2)**
   The trained and testing data of independent features are normalized and undergo 10-Fold Cross Validation. Their mean scores, predicted values, f1-score, precision, recall and accuracy scores are calculated. For visualization purposes, confusion matrix is depicted to show their true positive, false positive, true negative and false negative values.
2. **Random Forest Classifier for 3 features (Category 2)**

Using the correlation heatmap, it was concluded that CGPA, GRE Score and TOEFL Score are the three features that are closely related to Chance of Admit feature. Hence, dataset undergoes the same process as Decision Tree and Random Forest, but with only 3 independent features.

```
Data frame x with top 3 features:
    GRE Score  TOEFL Score  CGPA
0         337          118  9.65
1         324          107  8.87
2         316          104  8.00
Data frame y:
   Chance of Admit Bins
0                  High
1                Medium
2                Medium
```

Fig 5. Data frame for top 3 features

Evaluation:

The methods that utilize k-fold cross validation are evaluated based on their mean scores. The visualization of the algorithms' average scores is shown in Fig 6 below.

1. **Decision Tree:**
    - Average score: 0.69250
    - Accuracy: 0.6775
2. **Random Forest:**
    - Average score: 0.7575
    - Accuracy: 0.76
3. **Random Forest for 3 features:**
    - Average score: 0.7700
    - Accuracy: 0.7675

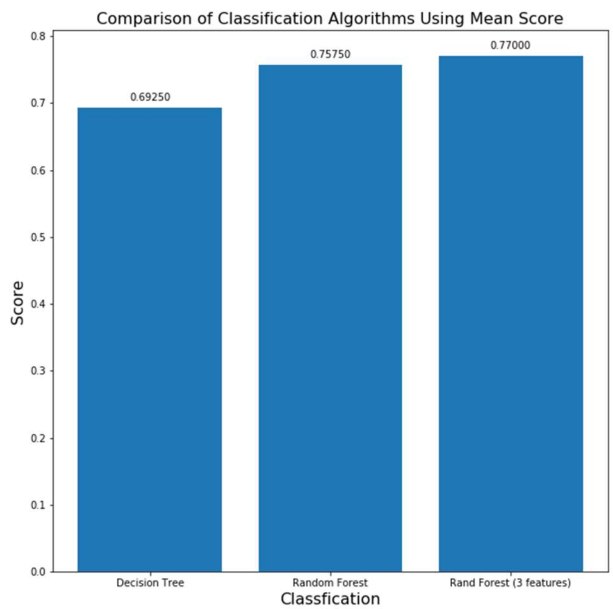


Fig 6. Bar chart of classifications and their mean scores

Source Code:

The chosen dataset underwent Python in Jupyter notebook[2] to perform visualizations and data mining.

1. **Read the data**

```
#importing necessary packages
import numpy as np
import pandas as pd
#reading csv file
df = pd.read_csv('…/Admission_Predict.csv')
#displaying first 3 rows of dataset
df.head(3)
```

Reading the data

```
In [1]: #importing necessary packages
        import numpy as np
        import pandas as pd

        #reading csv file
        df = pd.read_csv('C:/Users/Naveena/Documents/Data Mining/project/graduate-admissions/Admission_Predict.csv')

        #displaying first 3 rows of dataset
        df.head(3)
```

Out[1]:

	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

Fig 7. Reading the data

2. **Understanding data**

   Find out the number of rows and columns and rename columns if necessary.

   print('Shape of data is: \n', df.shape)
   print('Columns of the data are: \n', df.columns)
   #renaming columns
   df.rename(columns = {'LOR ': 'LOR', 'Chance of Admit ': 'Chance of Admit'}, inplace = True)

Understanding the data

```
In [2]: print('Shape of data is: \n', df.shape)
        print('Columns of the data are: \n', df.columns)

Shape of data is:
 (400, 9)
Columns of the data are:
 Index(['Serial No.', 'GRE Score', 'TOEFL Score', 'University Rating', 'SOP',
       'LOR ', 'CGPA', 'Research', 'Chance of Admit '],
      dtype='object')
```

Edit the columns names to make it easier to use
For instance, LOR column has a space and so does Chance of Admit

```
In [3]: df.rename(columns = {'LOR ': 'LOR', 'Chance of Admit ': 'Chance of Admit'}, inplace = True)
```

Fig 8. Understanding and editing the dataset

3. **Handling missing values**

   Check for any missing values in the dataset. If there are missing values that are insignificant, then the entire observation can be deleted. Otherwise, the missing value can be substituted using required measures.

   df.isnull().sum()

Handling missing values

```
In [4]: df.isnull().sum()

Out[4]: 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
```

Since there are no missing values, there will be no substitution required.

Fig 9. Handling missing values

4. **Descriptive Statistics and Visualizations**

   Create visualizations for better understanding of the data.

#importing required packages
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
fig, ax = plt.subplots(figsize = (10,10))
sns.heatmap(df.corr(), ax = ax, annot = True, linewidths = 0.3, cmap = 'RdYlGn')
plt.title('Graduate Admissions Heat Map', fontsize = 18)
plt.show()
fig.savefig('heatmap_of_all_features.png')

Descriptive Statistics and Visualization

Create a heatmap of all the features using their correlations

```
In [5]: import matplotlib.pyplot as plt
        import seaborn as sns
        %matplotlib inline
```

```
In [6]: fig, ax = plt.subplots(figsize = (10,10))
        sns.heatmap(df.corr(), ax = ax, annot = True, linewidths = 0.3, cmap = 'RdYlGn')
        plt.title('Graduate Admissions Heat Map', fontsize = 18)
        plt.show()
        fig.savefig('heatmap_of_all_features.png')
```

Fig 10. Code/Input for creating heatmap

CGPA vs Chance of Admit
fig, ax = plt.subplots(figsize = (10, 10))
plt.title('Regplot of CGPA vs Chance of Admit')
sns.regplot(x = 'CGPA', y = 'Chance of Admit', data = df, ax = ax)
plt.ylim(0,)
fig.savefig('Regplot of CGPA vs Chance of Admit.png')

CGPA vs Chance of Admit

```
In [7]: fig, ax = plt.subplots(figsize = (10, 10))
        plt.title('Regplot of CGPA vs Chance of Admit')
        sns.regplot(x = 'CGPA', y = 'Chance of Admit', data = df, ax = ax)
        plt.ylim(0,)
        fig.savefig('Regplot of CGPA vs Chance of Admit.png')
```

Fig 11. Code/Input for regression plot of CGPA vs Chance of Admit

CGPA vs GRE Score
#regression plot for GRE score and Chance of Admit
fig, ax = plt.subplots(figsize = (10, 10))
plt.title('Regplot of CGPA vs GRE Score')
sns.regplot(x = 'CGPA', y = 'GRE Score', data = df, ax = ax)
plt.ylim(0,)
fig.savefig('Regplot of CGPA vs Chance of Admit.png')

CGPA vs GRE Score

```
In [8]: #regression plot for GRE score and Chance of Admit
        fig, ax = plt.subplots(figsize = (10, 10))
        plt.title('Regplot of CGPA vs GRE Score')
        sns.regplot(x = 'CGPA', y = 'GRE Score', data = df, ax = ax)
        plt.ylim(0,)
        fig.savefig('Regplot of CGPA vs Chance of Admit.png')
```

Fig 12. Code/Input for regression plot of CGPA vs GRE Score

TOEFL Score vs Chance of Admit
#regression plot for TOEFL score and Chance of Admit
fig, ax = plt.subplots(figsize = (10, 10))
plt.title('Regplot of TOEFL Score vs Chance of Admit')
sns.regplot(x = 'TOEFL Score', y = 'Chance of Admit', data = df, ax = ax)
plt.ylim(0,)
fig.savefig('Regplot of TOEFL Score vs Chance of Admit.png')

TOEFL Score vs Chance of Admit

```
In [9]: #regression plot for TOEFL score and Chance of Admit
        fig, ax = plt.subplots(figsize = (10, 10))
        plt.title('Regplot of TOEFL Score vs Chance of Admit')
        sns.regplot(x = 'TOEFL Score', y = 'Chance of Admit', data = df, ax = ax)
        plt.ylim(0,)
        fig.savefig('Regplot of TOEFL Score vs Chance of Admit.png')
```

Fig 13. Code/Input for regression plot of TOEFL Score and Chance of Admit

CGPA vs TOEFL Score
#regression plot for CGPA and TOEFL score
fig, ax = plt.subplots(figsize = (10, 10))
plt.title('Regplot of CGPA vs TOEFL Score')
sns.regplot(x = 'CGPA', y = 'TOEFL Score', data = df, ax = ax)
plt.ylim(0,)
fig.savefig('Regplot of CGPA vs TOEFL Score.png')

CGPA vs TOEFL Score

```
In [10]: #regression plot for CGPA and TOEFL score
         fig, ax = plt.subplots(figsize = (10, 10))
         plt.title('Regplot of CGPA vs TOEFL Score')
         sns.regplot(x = 'CGPA', y = 'TOEFL Score', data = df, ax = ax)
         plt.ylim(0,)
         fig.savefig('Regplot of CGPA vs TOEFL Score.png')
```

Fig 14. Code/Input for regression plot of CGPA and TOEFL Score

Data Visualization for the levels of GRE Scores and TOEFL Scores using plotly
import plotly
plotly.tools.set_credentials_file(username='jnk22', api_key='g3FXYcx8sHq7d6ackjKN')

```
import plotly.plotly as py
import plotly.graph_objs as go
#grouped bar chart for GRE Score and TOEFL Score
x = ['Low', 'Average', 'High']
gre_y = np.array([df['GRE Score'].min(), df['GRE Score'].mean(), df['GRE Score'].max()])
toefl_y = np.array([df['TOEFL Score'].min(), df['TOEFL Score'].mean(), df['TOEFL Score'].max()])

gre_scores = go.Bar(x = x, y = gre_y, text = gre_y, textposition = 'auto', name = 'GRE Scores (out of 340)')
toefl_scores = go.Bar(x = x, y = toefl_y, text = toefl_y, textposition = 'auto', name = 'TOEFL Scores (out of 120)')

data = [gre_scores, toefl_scores]
layout = go.Layout(barmode = 'group', title = 'GRE and TOEFL Scores')

fig = go.Figure(data = data, layout = layout)
py.iplot(fig, filename='GRE and TOEFL Scores')
```

Data Visualization for the levels of GRE Scores and TOEFL Scores using plotly

```
In [11]: import plotly
plotly.tools.set_credentials_file(username='jnk22', api_key='g3FXYcx8sHq7d6ackjKN')
import plotly.plotly as py
import plotly.graph_objs as go

#grouped bar chart for GRE Score and TOEFL Score
x = ['Low', 'Average', 'High']
gre_y = np.array([df['GRE Score'].min(), df['GRE Score'].mean(), df['GRE Score'].max()])
toefl_y = np.array([df['TOEFL Score'].min(), df['TOEFL Score'].mean(), df['TOEFL Score'].max()])

gre_scores = go.Bar(x = x, y = gre_y, text = gre_y, textposition = 'auto', name = 'GRE Scores (out of 340)')
toefl_scores = go.Bar(x = x, y = toefl_y, text = toefl_y, textposition = 'auto', name = 'TOEFL Scores (out of 120)')

data = [gre_scores, toefl_scores]
layout = go.Layout(barmode = 'group', title = 'GRE and TOEFL Scores')

fig = go.Figure(data = data, layout = layout)
py.iplot(fig, filename='GRE and TOEFL Scores')

High five! You successfully sent some data to your account on plotly. View your plot in your browser at https://plot.ly/~jnk22/
0 or inside your plot.ly account where it is named 'GRE and TOEFL Scores'
```

Fig 15. Code/Input for grouped bar chart of GRE and TOEFL scores using plotly

Research Experience
*
Let's see how many people have research experience and how many don't have research experience*

```
y = np.array([len(df[df['Research'] == 0]), len(df[df['Research'] == 1])])
x = ['No', 'Yes']

fig, ax = plt.subplots(figsize = (10, 10))
plt.bar(x, y)
plt.title('Research Frequency')
plt.xlabel('Research Experience')
plt.ylabel('No of students')
plt.show()
fig.savefig('Research Frequency.png')
```

print(df['Research'].value_counts())

Research Experience

Let's see how many people have research experience and how many don't have research experience

```
In [12]: y = np.array([len(df[df['Research'] == 0]), len(df[df['Research'] == 1])])
         x = ['No', 'Yes']

         fig, ax = plt.subplots(figsize = (10, 10))
         plt.bar(x, y)
         plt.title('Research Frequency')
         plt.xlabel('Research Experience')
         plt.ylabel('No of students')
         plt.show()
         fig.savefig('Research Frequency.png')

         print(df['Research'].value_counts())
```

```
1    219
0    181
Name: Research, dtype: int64
```

Fig 16. Code/Input for bar chart to show how many students have research experience (top) and output of how many students have experience and do not (bottom)

CGPA vs Chance of Admit with respect to Research Experience
fig, ax = plt.subplots(figsize = (10, 10))
plt.title('Regplot of CGPA vs Chance of Admit based on Research')
sns.scatterplot(x = 'CGPA', y = 'Chance of Admit', data = df, ax = ax, hue = 'Research')
plt.ylim(0,)
fig.savefig('Regplot of CGPA vs Chance of Admit based on Research.png')

CGPA vs Chance of Admit with respect to Research Experience

```
In [13]: fig, ax = plt.subplots(figsize = (10, 10))
         plt.title('Regplot of CGPA vs Chance of Admit based on Research')
         sns.scatterplot(x = 'CGPA', y = 'Chance of Admit', data = df, ax = ax, hue = 'Research')
         plt.ylim(0,)
         fig.savefig('Regplot of CGPA vs Chance of Admit based on Research.png')
```

Fig 17. Code/Input for scatter plot of CGPA vs Chance of Admit with respect to research experience

GRE Score vs Chance of Admit with respect to Research
fig, ax = plt.subplots(figsize = (10, 10))
plt.title('Regplot of GRE Score vs Chance of Admit based on Research')
sns.scatterplot(x = 'GRE Score', y = 'Chance of Admit', data = df, ax = ax, hue = 'Research')
plt.ylim(0,)
fig.savefig('Regplot of GRE Score vs Chance of Admit based on Research.png')

GRE Score vs Chance of Admit with respect to Research

```
In [14]: fig, ax = plt.subplots(figsize = (10, 10))
         plt.title('Regplot of GRE Score vs Chance of Admit based on Research')
         sns.scatterplot(x = 'GRE Score', y = 'Chance of Admit', data = df, ax = ax, hue = 'Research')
         plt.ylim(0,)
         fig.savefig('Regplot of GRE Score vs Chance of Admit based on Research.png')
```

Fig 18. Code/Input for scatter plot of GRE Score vs Chance of Admit with respect to Research experience

df[df['CGPA'] >= 8.0].plot(kind='scatter', x='GRE Score', y='TOEFL Score')
plt.xlabel("GRE Score")
plt.ylabel("TOEFL SCORE")
plt.title("CGPA>=8.0")
plt.savefig('GRE and TOEFL Scores of CGPA greater than 8.png')
plt.show()

```
In [15]: df[df['CGPA'] >= 8.0].plot(kind='scatter', x='GRE Score', y='TOEFL Score')
         plt.xlabel("GRE Score")
         plt.ylabel("TOEFL SCORE")
         plt.title("CGPA>=8.0")
         plt.savefig('GRE and TOEFL Scores of CGPA greater than 8.png')
         plt.show()
```

Fig 19. Code/Input for scatter plot of GRE Score vs TOEFL Score whose chance of admission is greater than 0.8

#groupby the mean of chances of admit(ca) and university rating(ur)
df_ca_ur = df[['University Rating', 'Chance of Admit']]
df_ca_ur_group = df_ca_ur.groupby(['University Rating'], as_index = False).mean()
df_ca_ur_group.rename(columns = {'Chance of Admit': 'Average_Chance_of_Admit', 'University Rating': 'University_Rating'}, inplace = True)
df_ca_ur_group

```
In [16]: #groupby the mean of chances of admit(ca) and university rating(ur)
         df_ca_ur = df[['University Rating', 'Chance of Admit']]
         df_ca_ur_group = df_ca_ur.groupby(['University Rating'], as_index = False).mean()
         df_ca_ur_group.rename(columns = {'Chance of Admit': 'Average_Chance_of_Admit', 'University Rating': 'University_Rating'}, inplac
         df_ca_ur_group
```

Out[16]:

	University_Rating	Average_Chance_of_Admit
0	1	0.548077
1	2	0.625981
2	3	0.711880
3	4	0.818108
4	5	0.888167

Fig 20. Code/Input and output for calculating average chance of admission for each type of university.

5. Classification Modeling: Decision Tree and Random Forest

i) Preprocessing Data

Add a column for bins in the data frame. Use np.linspace() method to add 3 levels of bins representing the chance of admit.

ca_bins = np.linspace(min(df['Chance of Admit']), max(df['Chance of Admit']), 4)
ca_labels = ['Low', 'Medium', 'High']
df['Chance of Admit Bins'] = pd.cut(df['Chance of Admit'], ca_bins, labels = ca_labels, include_lowest = True)
df.head(3)

Preprocessing the data

In [17]:
```
ca_bins = np.linspace(min(df['Chance of Admit']), max(df['Chance of Admit']), 4)
ca_labels = ['Low', 'Medium', 'High']
df['Chance of Admit Bins'] = pd.cut(df['Chance of Admit'], ca_bins, labels = ca_labels, include_lowest = True)
df.head(3)
```

Out[17]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit	Chance of Admit Bins
0	1	337	118	4	4.5	4.5	9.65	1	0.92	High
1	2	324	107	4	4.0	4.5	8.87	1	0.76	Medium
2	3	316	104	3	3.0	3.5	8.00	1	0.72	Medium

Fig 21. Creating bins for chance of admit

ii) Importing all the required packages

#import required packages
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score, precision_score, recall_score, accuracy_score
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_predict

Import all the required packages

In [18]:
```
#import required packages
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score, precision_score, recall_score, accuracy_score
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_predict
```

Fig 22. Importing all the packages for algorithms

iii) Split the data frame and normalize

Split the df data frame into x and y data frames, where x data frame includes all the columns except Serial No., Chance of Admit and Chance of Admit Bins, and y data frame includes Chance of Admit Bins column.

Once splitting is done successfully, we normalize the data using MinMaxScaler() method for better accuracy.

#assign the data frames
x = df[:]
x.drop(['Serial No.', 'Chance of Admit', 'Chance of Admit Bins'], axis = 1, inplace = True)
print('Data Frame x(independent variables): \n', x.head(3))

y = df[['Chance of Admit Bins']]
print('Data Frame y(dependent variable): \n', y.head(3))

#normalize the data
scalerX = MinMaxScaler(feature_range=(0, 1))
x_scaled = scalerX.fit_transform(x)

```
In [19]: #assign the data frames
x = df[:]
x.drop(['Serial No.', 'Chance of Admit', 'Chance of Admit Bins'], axis = 1, inplace = True)
print('Data Frame x(independent variables): \n', x.head(3))

y = df[['Chance of Admit Bins']]
print('Data Frame y(dependent variable): \n', y.head(3))

#normalize the data
scalerX = MinMaxScaler(feature_range=(0, 1))
x_scaled = scalerX.fit_transform(x)
```

```
Data Frame x(independent variables):
    GRE Score  TOEFL Score  University Rating  SOP  LOR  CGPA  Research
0        337          118                  4  4.5  4.5  9.65         1
1        324          107                  4  4.0  4.5  8.87         1
2        316          104                  3  3.0  3.5  8.00         1
Data Frame y(dependent variable):
   Chance of Admit Bins
0                 High
1               Medium
2               Medium
```

Fig 23. Splitting the data frame and normalizing the independent features.

iv) Applying 10-Fold Cross Validation: Decision Tree

We apply k-Fold Cross validation where k = 10. We calculate the scores using cross_val_score() method and print their mean scores. We also calculate their accuracy score, f1 – score, precision score and recall score as they are few of the evaluation metrics for classification problems. To know compare the actual values and predicted values, we visualize their confusion matrix. Once the confusion matrix is visualized, the x – tick labels and y – tick labels are displayed as 0, 1, 2 (Low, Medium, High)[*].

#Call the Decision Tree Classification
dtc = DecisionTreeClassifier()
print(dtc)

#Apply 10 – fold cross validation
scores_dtc = cross_val_score(dtc, x_scaled, y, cv = 10)
print('Cross Validated Scores: ', scores_dtc)
print('Average of the scores: ', np.mean(scores_dtc))

#predicting data
predictions_dtc = cross_val_predict(dtc, x_scaled, y, cv = 10)
print('Cross Validated predictions: ', predictions_dtc[0:5])

[*]Applicable to all the algorithms' confusion matrix
#Calculating accuracy, f1 score, precision score, and recall score
print('F1 score: ', f1_score(y, predictions_dtc, average = None))
print('Precision: ', precision_score(y, predictions_dtc, average = None))
print('Recall: ', recall_score(y, predictions_dtc, average = None))
print('Accuracy: ', accuracy_score(y, predictions_dtc))

#Visualizing confusion matrix
dtc_cm = confusion_matrix(y, predictions_dtc)
fig, ax = plt.subplots(figsize =(10, 10))
sns.heatmap(dtc_cm, annot = True, linewidths=0.5, linecolor="red", fmt = ".0f", ax=ax)
plt.title("Confusion Matrix for Test Dataset")
plt.xlabel("Predicted y values")
plt.ylabel("Actual y values")
plt.show()

Applying 10-Fold Cross Validation: Decision Tree

In [20]:

```
#Call the Decision Tree Classification
dtc = DecisionTreeClassifier()
print(dtc)

#Apply 10 folds
scores_dtc = cross_val_score(dtc, x_scaled, y, cv = 10)
print('Cross Validated Scores: ', scores_dtc)
print('Average of the scores: ', np.mean(scores_dtc))

#predicting data
predictions_dtc = cross_val_predict(dtc, x_scaled, y, cv = 10)
print('Cross Validated predictions: ', predictions_dtc[0:5])

#Calculating accuracy, f1 score, precision score, and recall score
print('F1 score: ', f1_score(y, predictions_dtc, average = None))
print('Precision: ', precision_score(y, predictions_dtc, average = None))
print('Recall: ', recall_score(y, predictions_dtc, average = None))
print('Accuracy: ', accuracy_score(y, predictions_dtc))

#Visualizing confusion matrix
dtc_cm = confusion_matrix(y, predictions_dtc)
fig, ax = plt.subplots(figsize =(10, 10))
sns.heatmap(dtc_cm, annot = True, linewidths=0.5, linecolor="red", fmt = ".0f", ax=ax)
plt.title("Confusion Matrix for Test Dataset")
plt.xlabel("Predicted y values")
plt.ylabel("Actual y values")
plt.show()
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, 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, presort=False, random_state=None,
            splitter='best')
Cross Validated Scores:  [0.725 0.7   0.675 0.6   0.825 0.775 0.65  0.65  0.6   0.725]
Average of the scores:  0.6924999999999999
Cross Validated predictions:  ['High' 'High' 'Low' 'Medium' 'Medium']
F1 score:  [0.77460317 0.37623762 0.67708333]
Precision:  [0.78709677 0.37254902 0.67010309]
Recall:  [0.7625     0.38       0.68421053]
Accuracy:  0.6775
```

Fig 24. Code for applying 10 – Fold cross validation for Decision Tree (top); Output of the decision tree (bottom)

v) Applying 10-Fold Cross Validation: Random Forest

```
#Call the Decision Tree Classification
rfc = RandomForestClassifier(n_estimators = 100)
print(rfc)

#Apply 10 – fold cross validation
scores_rfc = cross_val_score(rfc, x_scaled, y, cv = 10)
print('Cross Validated Scores: ', scores_rfc)
print('Average of the scores: ', np.mean(scores_rfc))

#predicting data
predictions_rfc = cross_val_predict(rfc, x_scaled, y, cv = 10)
print('Cross Validated predictions: ', predictions_rfc[0:5])

#Calculating accuracy, f1 score, precision score, and recall score
print('F1 score: ', f1_score(y, predictions_rfc, average = None))
print('Precision: ', precision_score(y, predictions_rfc, average = None))
print('Recall: ', recall_score(y, predictions_rfc, average = None))
print('Accuracy: ', accuracy_score(y, predictions_rfc))

#Visualizing confusion matrix
rfc_cm = confusion_matrix(y, predictions_rfc)
fig, ax = plt.subplots(figsize =(10, 10))
sns.heatmap(rfc_cm, annot = True, linewidths=0.5, linecolor="red", fmt = ".0f",
ax=ax)
plt.title("Confusion Matrix for Test Dataset")
plt.xlabel("Predicted y values")
plt.ylabel("Actual y values")
plt.show()
```

Applying 10-Fold Cross Validation: Random Forest

```
In [21]:
#Call the Decision Tree Classification
rfc = RandomForestClassifier(n_estimators = 100)
print(rfc)

#Apply 10 folds
scores_rfc = cross_val_score(rfc, x_scaled, y, cv = 10)
print('Cross Validated Scores: ', scores_rfc)
print('Average of the scores: ', np.mean(scores_rfc))

#predicting data
predictions_rfc = cross_val_predict(rfc, x_scaled, y, cv = 10)
print('Cross Validated predictions: ', predictions_rfc[0:5])

#Calculating accuracy, f1 score, precision score, and recall score
print('F1 score: ', f1_score(y, predictions_rfc, average = None))
print('Precision: ', precision_score(y, predictions_rfc, average = None))
print('Recall: ', recall_score(y, predictions_rfc, average = None))
print('Accuracy: ', accuracy_score(y, predictions_rfc))

#Visualizing confusion matrix
rfc_cm = confusion_matrix(y, predictions_rfc)
fig, ax = plt.subplots(figsize =(10, 10))
sns.heatmap(rfc_cm, annot = True, linewidths=0.5, linecolor="red", fmt = ".0f", ax=ax)
plt.title("Confusion Matrix for Test Dataset")
plt.xlabel("Predicted y values")
plt.ylabel("Actual y values")
plt.show()
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       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=False)
```

```
Cross Validated Scores:  [0.775 0.7   0.725 0.725 0.875 0.75  0.775 0.875 0.625 0.75 ]
Average of the scores:  0.7575
```

```
Cross Validated predictions:  ['High' 'High' 'Medium' 'High' 'Medium']
F1 score:  [0.82692308 0.4691358  0.76658477]
Precision:  [0.84868421 0.61290323 0.71889401]
Recall:  [0.80625    0.38       0.82105263]
Accuracy:  0.76
```

Fig 25. Code for applying 10 – Fold cross validation for Random Forest (top); Output of the random forest (bottom three)

vi) Applying 10-Fold Cross Validation for Top 3 Features: Random Forest

We preprocess the data again by splitting the df data frame into x_top3 and y_top3, where x_top3 includes only GRE Score, TOEFL Score and CGPA, and y_top3 remains same as data frame y. The x_top3 data frame is normalized using MinMaxScaler() method and undergoes 10 – fold cross validation using Random Forest classification algorithm.

#assign the data frames
x_top3 = df[:]

```
x_top3.drop(['Serial No.', 'Chance of Admit', 'Chance of Admit Bins', 'LOR', 'SOP',
'University Rating', 'Research'], axis = 1, inplace = True)
print('Data Frame x(independent variables): \n', x_top3.head(3))

y_top3 = df[['Chance of Admit Bins']]
print('Data Frame y(dependent variable): \n', y_top3.head(3))

#normalize the data
scalerX_top3 = MinMaxScaler(feature_range=(0, 1))
x_scaled_top3 = scalerX_top3.fit_transform(x_top3)

#Call the Decision Tree Classification
rfc_top3 = RandomForestClassifier(n_estimators = 100)
print(rfc_top3)

#Apply 10 – fold cross validation
scores_rfc_top3 = cross_val_score(rfc_top3, x_scaled_top3, y_top3, cv = 10)
print('Cross Validated Scores: ', scores_rfc_top3)
print('Average of the scores: ', np.mean(scores_rfc_top3))

#predicting data
predictions_rfc_top3 = cross_val_predict(rfc_top3, x_scaled_top3, y_top3, cv = 10)
print('Cross Validated predictions: ', predictions_rfc_top3[0:5])

#Calculating accuracy, f1 score, precision score, and recall score
print('F1 score: ', f1_score(y, predictions_rfc_top3, average = None))
print('Precision: ', precision_score(y, predictions_rfc_top3, average = None))
print('Recall: ', recall_score(y, predictions_rfc_top3, average = None))
print('Accuracy: ', accuracy_score(y, predictions_rfc_top3))

#Visualizing confusion matrix
rfc_top3_cm = confusion_matrix(y, predictions_rfc_top3)
fig, ax = plt.subplots(figsize =(10, 10))
sns.heatmap(dtc_cm, annot = True, linewidths=0.5, linecolor="red", fmt = ".0f",
ax=ax)
plt.title("Confusion Matrix for Test Dataset")
plt.xlabel("Predicted y values")
plt.ylabel("Actual y values")
plt.show()
```

```
In [22]:
#assign the data frames
x_top3 = df[:]
x_top3.drop(['Serial No.', 'Chance of Admit', 'Chance of Admit Bins', 'LOR', 'SOP', 'University Rating', 'Research'], axis =
print('Data Frame x(independent variables): \n', x_top3.head(3))

y_top3 = df[['Chance of Admit Bins']]
print('Data Frame y(dependent variable): \n', y_top3.head(3))

#normalize the data
scalerX_top3 = MinMaxScaler(feature_range=(0, 1))
x_scaled_top3 = scalerX_top3.fit_transform(x_top3)

#Call the Decision Tree Classification
rfc_top3 = RandomForestClassifier(n_estimators = 100)
print(rfc_top3)

#Apply 10 folds
scores_rfc_top3 = cross_val_score(rfc_top3, x_scaled_top3, y_top3, cv = 10)
print('Cross Validated Scores: ', scores_rfc_top3)
print('Average of the scores: ', np.mean(scores_rfc_top3))

#predicting data
predictions_rfc_top3 = cross_val_predict(rfc_top3, x_scaled_top3, y_top3, cv = 10)
print('Cross Validated predictions: ', predictions_rfc_top3[0:5])

#Calculating accuracy, f1 score, precision score, and recall score
print('F1 score: ', f1_score(y, predictions_rfc_top3, average = None))
print('Precision: ', precision_score(y, predictions_rfc_top3, average = None))
print('Recall: ', recall_score(y, predictions_rfc_top3, average = None))
print('Accuracy: ', accuracy_score(y, predictions_rfc_top3))

#Visualizing confusion matrix
rfc_top3_cm = confusion_matrix(y, predictions_rfc_top3)
fig, ax = plt.subplots(figsize =(10, 10))
sns.heatmap(dtc_cm, annot = True, linewidths=0.5, linecolor="red", fmt = ".0f", ax=ax)
plt.title("Confusion Matrix for Test Dataset")
plt.xlabel("Predicted y values")
plt.ylabel("Actual y values")
plt.show()
```

```
Data Frame x(independent variables):
    GRE Score  TOEFL Score  CGPA
0        337          118  9.65
1        324          107  8.87
2        316          104  8.00
Data Frame y(dependent variable):
   Chance of Admit Bins
0                 High
1               Medium
2               Medium
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            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=False)
```

```
Cross Validated Scores:  [0.775 0.75  0.75  0.75  0.875 0.75  0.775 0.85  0.7   0.725]
Average of the scores:  0.77
```

```
Cross Validated predictions:  ['High' 'High' 'Medium' 'High' 'Medium']
F1 score:  [0.83601286 0.48837209 0.77419355]
Precision:  [0.86092715 0.58333333 0.73239437]
Recall:  [0.8125     0.42       0.82105263]
Accuracy:  0.7675
```

Fig 25. Code for applying 10 – Fold cross validation for Random Forest for top 3 features(top); Output of the random forest for top 3 features (bottom three)

6. **Compare the algorithms**

Since the k – fold cross validation approach produces scores as their accuracy, we use the algorithms' mean scores to compare the algorithms performance. We visualize their performance using a bar chart.

```
y_axis = np.array([np.mean(scores_dtc), np.mean(scores_rfc), np.mean(scores_rfc_top3)])
x_axis = ["Decision Tree","Random Forest", "Rand Forest (3 features)"]
fig, ax = plt.subplots(figsize = (10, 10))
ax1 = plt.bar(x_axis, y_axis)
plt.title("Comparison of Classification Algorithms Using Mean Score", fontsize = 16)
plt.xlabel("Classfication", fontsize = 16)
plt.ylabel("Score", fontsize = 16)

def add_value_labels(ax1, spacing=5):
    """Add labels to the end of each bar in a bar chart.

    Arguments:
        ax (matplotlib.axes.Axes): The matplotlib object containing the axes
            of the plot to annotate.
        spacing (int): The distance between the labels and the bars.
    """

    # For each bar: Place a label
    for rect in ax.patches:
        # Get X and Y placement of label from rect.
        y_value = rect.get_height()
        x_value = rect.get_x() + rect.get_width() / 2

        # Number of points between bar and label. Change to your liking.
        space = spacing
        # Vertical alignment for positive values
        va = 'bottom'

        # If value of bar is negative: Place label below bar
        if y_value < 0:
            # Invert space to place label below
            space *= -1
            # Vertically align label at top
            va = 'top'

        # Use Y value as label and format number with one decimal place
        label = "{:.5f}".format(y_value)

        # Create annotation
```

```
ax.annotate(
    label,                      # Use `label` as label
    (x_value, y_value),         # Place label at end of the bar
    xytext=(0, space),          # Vertically shift label by `space`
    textcoords="offset points", # Interpret `xytext` as offset in points
    ha='center',                # Horizontally center label
    va=va)                      # Vertically align label differently for
                                # positive and negative values.


# Call the function above
add_value_labels(ax1)
fig.savefig('Comparison of Classification Algorithms Using Accuracy.png')
plt.show()
```

Comparing the algorithms

In [23]:

```python
y_axis = np.array([np.mean(scores_dtc), np.mean(scores_rfc), np.mean(scores_rfc_top3)])
x_axis = ["Decision Tree","Random Forest", "Rand Forest (3 features)"]
fig, ax = plt.subplots(figsize = (10, 10))
ax1 = plt.bar(x_axis, y_axis)
plt.title("Comparison of Classification Algorithms Using Mean Score", fontsize = 16)
plt.xlabel("Classfication", fontsize = 16)
plt.ylabel("Score", fontsize = 16)

def add_value_labels(ax1, spacing=5):
    """Add labels to the end of each bar in a bar chart.

    Arguments:
        ax (matplotlib.axes.Axes): The matplotlib object containing the axes
            of the plot to annotate.
        spacing (int): The distance between the labels and the bars.
    """

    # For each bar: Place a label
    for rect in ax.patches:
        # Get X and Y placement of label from rect.
        y_value = rect.get_height()
        x_value = rect.get_x() + rect.get_width() / 2

        # Number of points between bar and label. Change to your liking.
        space = spacing
        # Vertical alignment for positive values
        va = 'bottom'

        # If value of bar is negative: Place label below bar
        if y_value < 0:
            # Invert space to place label below
            space *= -1
            # Vertically align label at top
            va = 'top'

        # Use Y value as label and format number with one decimal place
        label = "{:.5f}".format(y_value)

        # Create annotation
        ax.annotate(
            label,                      # Use `label` as label
            (x_value, y_value),         # Place label at end of the bar
            xytext=(0, space),          # Vertically shift label by `space`
            textcoords="offset points", # Interpret `xytext` as offset in points
            ha='center',                # Horizontally center label
            va=va)                      # Vertically align label differently for
                                        # positive and negative values.


# Call the function above. All the magic happens there.
add_value_labels(ax1)
fig.savefig('Comparison of Classification Algorithms Using Accuracy.png')
plt.show()
```

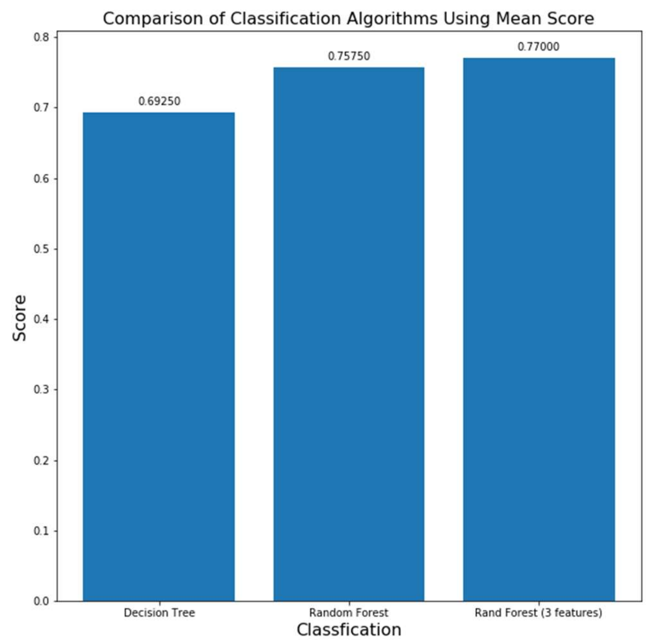
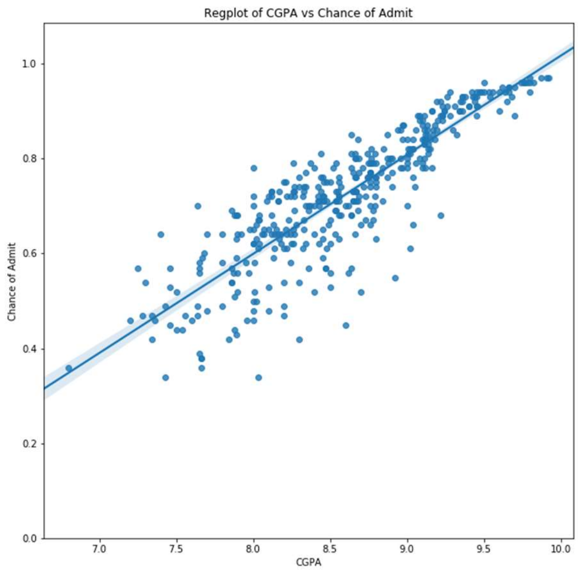


Fig 26. Code for creating bar chart to compare the algorithms (top) and bar chart output (bottom)

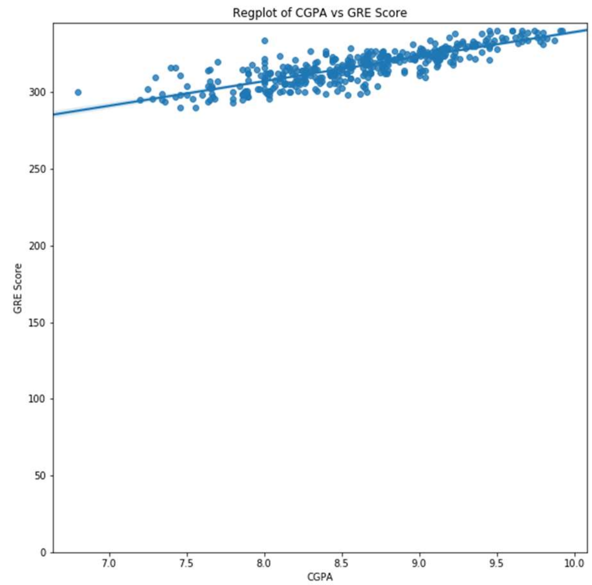
Conclusion:

From the above results and evaluation, it is concluded that Random Forest algorithm is better than Decision tree algorithm when 10 – fold cross validation is implemented, as it has the average score of 0.7575 for all independent features and 0.77 for three independent features.

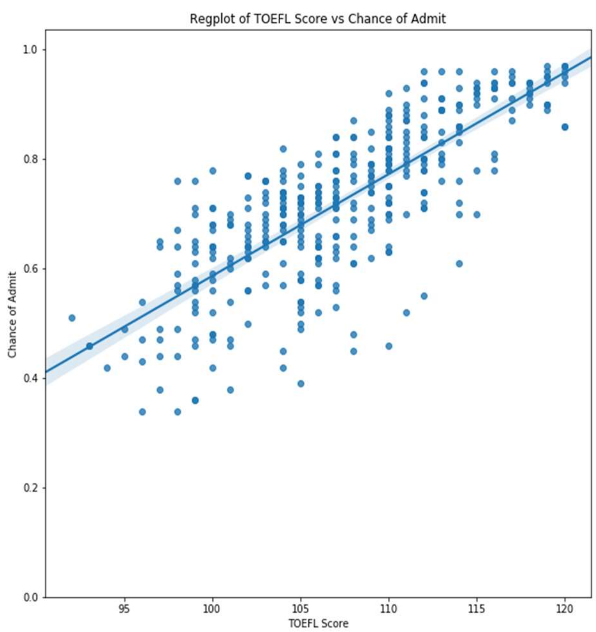
Appendix:



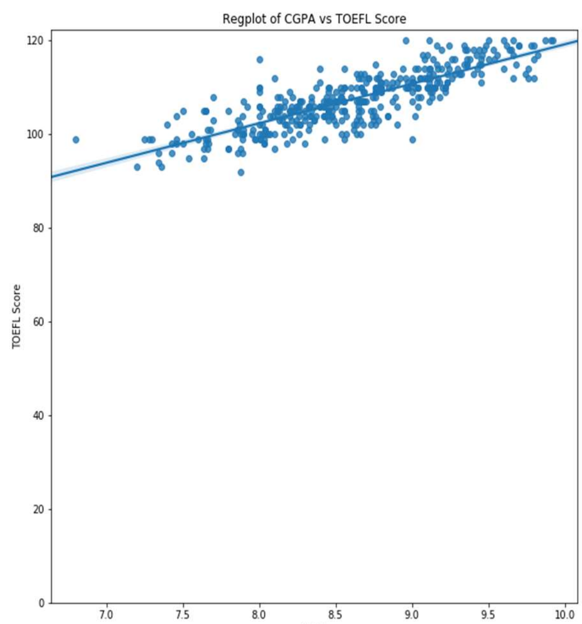
Regplot of CGPA vs Chance of Admit



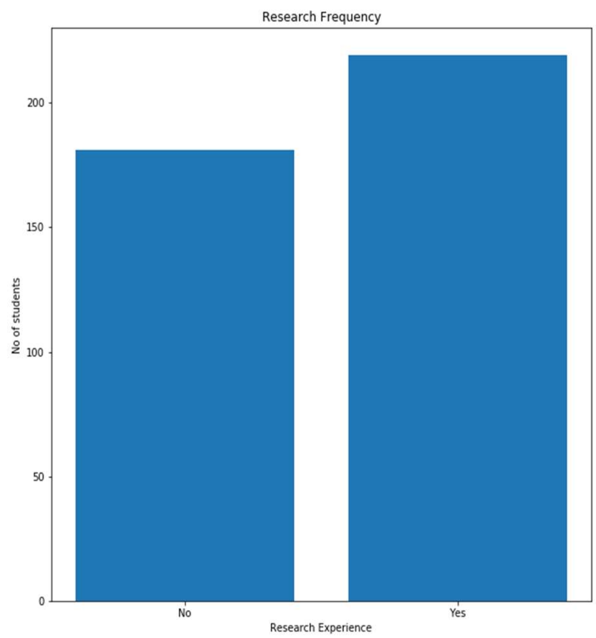
Regplot of CGPA vs GRE Score



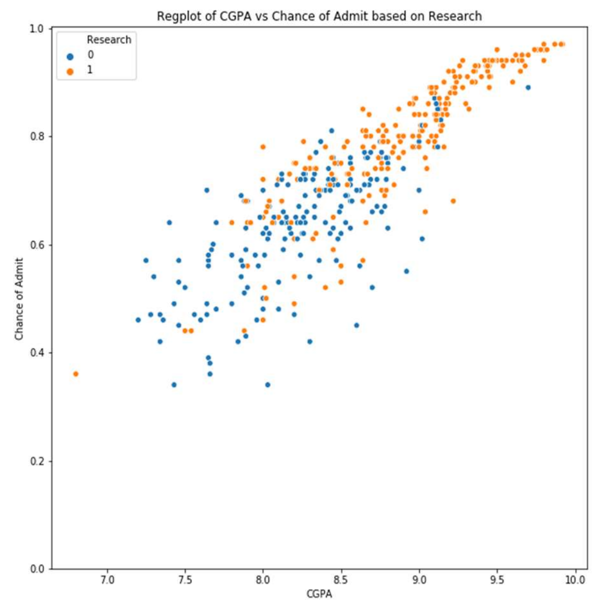
Regplot of TOEFL Score vs Chance of Admit



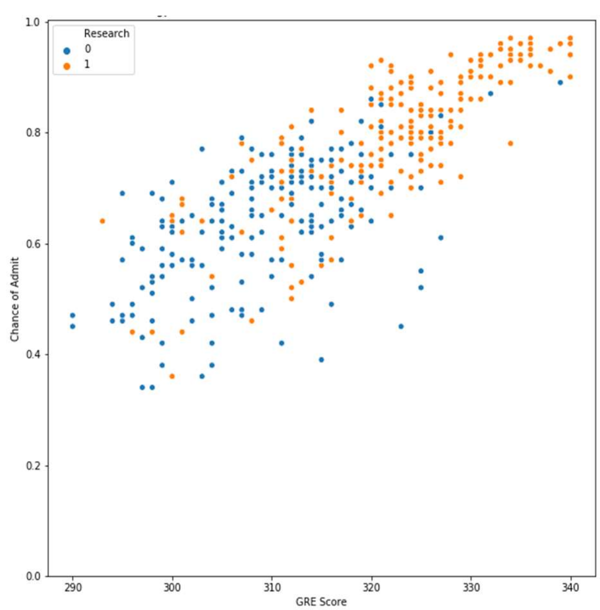
Regplot of CGPA vs TOEFL Score



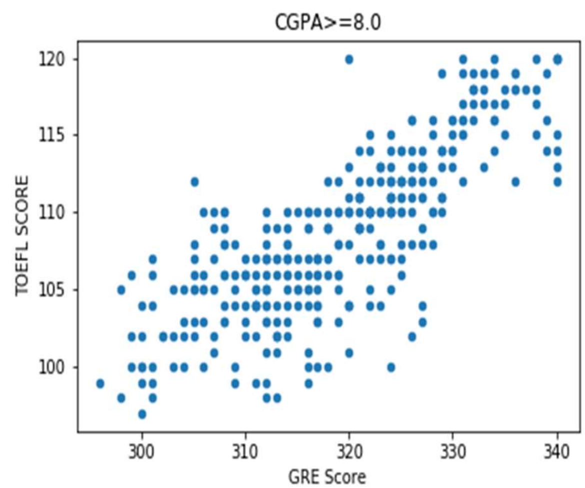
Bar chart of Research frequency



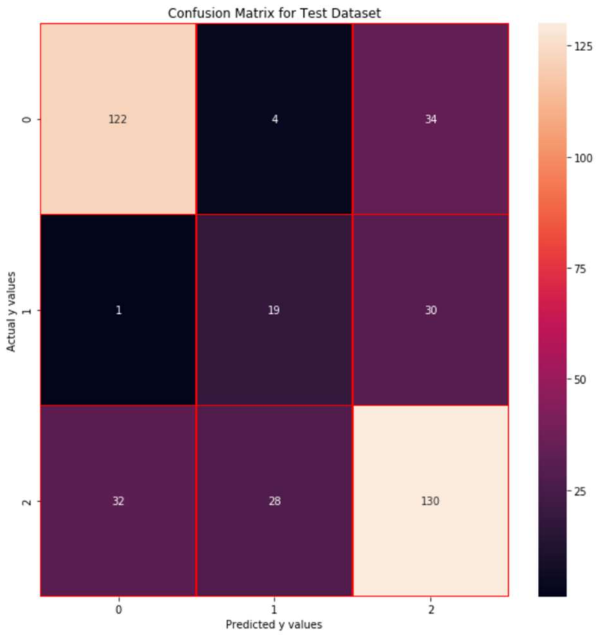
Scatter plot of CGPA vs Chance of Admit with respect to Research



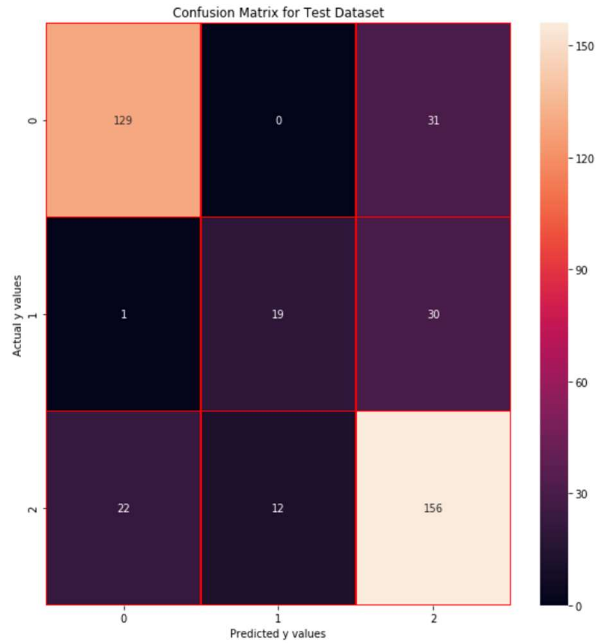
Scatter plot of GRE Score vs Chance of Admit with respect to Research



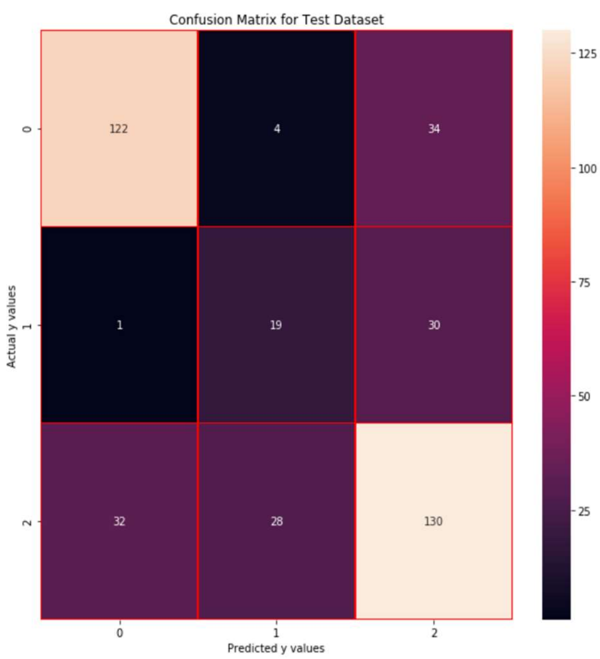
Scatter plot of GRE Score vs TOEFL Score of students who have chance of admission greater than equal to 0.8



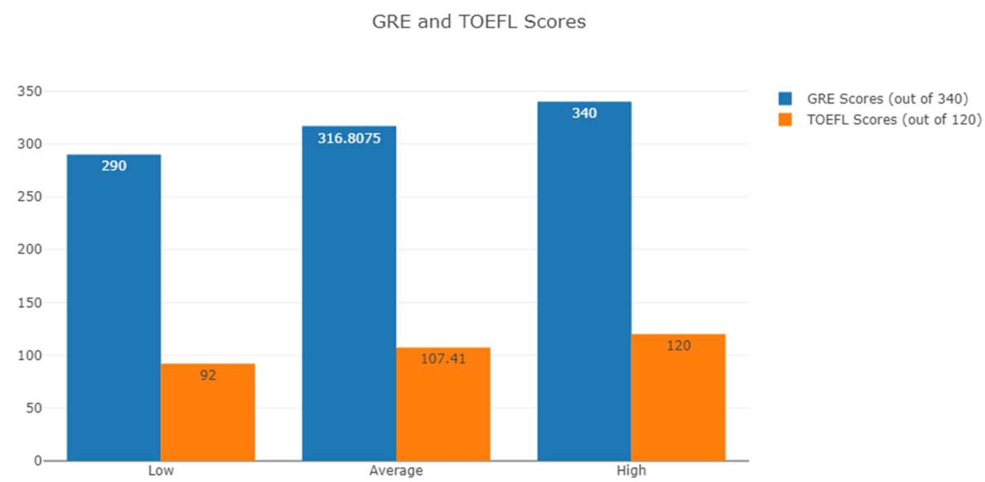
Confusion Matrix for Test Dataset: Decision Tree



Confusion Matrix for Test Dataset: Random Forest



Confusion Matrix for Test Dataset: Random Forest for top 3 features



Grouped bar chart of low, average and high GRE and TOEFL Scores

References:

1. "Graduate Admissions: Predicting admission from important parameters", Mohan S Acharya, https://www.kaggle.com/mohansacharya/graduate-admissions
2. Anaconda software for Python 3.7 Version (Windows): https://www.anaconda.com/distribution/