**

Project Report on

**TO PREDICT THE ‘CHANCE OF ADMIT’**

GRADUATE ADMISSIONS

**Submitted by:**

**TEAM 36**

|  |  |  |  |
| --- | --- | --- | --- |
| NAME | MADHUSHREE T P | VACHEL CAETANO DE SOUZA | SAIENI ALANKRUTHI |
| COLLEGE | leftlogo.png (1344×1344)KS INSTITUTE OF  TECHNOLOGY,  BANGLORE | ACHARYA INSTITUTE OF TECHNOLOGY  BANGLORE | G.NARAYANAMMA INSTITUTE OF TECHNOLOGY AND SCIENCE, HYDERABAD. |

**Submitted To:**

Gurvansh Singh,

Mtech,

Knowledge solutions India.

**TABLE OF CONTENT**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **TITLE** | **PAGE NO** |
| 1 | Table of content | 2 |
| 2 | Table of figure | 3 |
| 3 | Abstract | 4 |
| 4 | **chapters** |  |
| Introduction | 5 |
| Software Requirements | 6 |
| Methodology |  |
| Summary of Workflow | 7 |
| Models used   1. MLR 2. RFR 3. MLR and RFR with PCA | 8 |
| 5 | Regression Code | 13 |
| 6 | Coding Result and testing | 23 |
| 7 | Conclusion | 27 |
| 8 | **Reference** | 28 |

**TABLE OF FIGURE**

|  |  |  |
| --- | --- | --- |
| **FIG** | **TITLE** | **PAGE NO** |
| 1. | Work-Flow Diagram | 6 |
| 2. | MLR test vs predicted graphs | 23 |
| 3. | RFR test vs predicted graphs | 24 |
| 4. | MLR with PCA test vs predicted graphs | 25 |
| 5. | RFR with PCA test vs predicted graphs | 25 |
| 6. | Model Comparison | 26 |
| 7. | Testing through Tkinter | 26 |

# ABSTRACT

Students are often worried about their chance of admission in graduate school. The aim of this project is to help students in shortlisting universities with their profile. The predicted output gives them a fair idea about their admission chances in a particular university. This analysis should also help students who are currently preparing or will be preparing to get a better idea.

We are used some libraries in the machine learning using python code to create the plot of student’s profiles against the predicted values.

We used the following models to create the code,

Multiple linear regressor (MLR) Model, Random Forest Regressor (RFR) Model, MLR with PCA Model, RFR with PCA Model

We have used different algorithm for solving this regression problem. The random forest combines hundreds or thousands of decision trees, trains each one on a slightly different set of the observations, splitting nodes in each tree considering a limited number of the features. The final predictions of the random forest are made by averaging the predictions of each individual tree.

RFs train each tree independently, using a random sample of the data. This randomness helps to make the model more robust than a single decision tree, and less likely to overfit on the training data

**CHAPTERS**

**INTRODUCTION**

**What is Regression?**

Regression is a statistical method used in finance, investing, and other disciplines that attempts to determine the strength and character of the relationship between one dependent variable (usually denoted by Y) and a series of other variables (known as independent variables).

Regression analysis is a reliable method of identifying which variables have impact on a topic of interest. The process of performing a regression allows you to confidently determine which factors matter most, which factors can be ignored, and how these factors influence each other.

* **Dependent Variable:** This is the main factor that you’re trying to understand or predict.
* **Independent Variables:** These are the factors that you hypothesize have an impact on your dependent variable.

**Why to use Regression Analysis?**

Regression analysis is helpful statistical method that can be leveraged across an organization to determine the degree to which particular independent variables are influencing dependent variables.

The next time someone in your business is proposing a hypothesis that states that one factor, whether you can control that factor or not, is impacting a portion of the business, suggest performing a regression analysis to determine just how confident you should be in that hypothesis! This will allow you to make more informed business decisions, allocate resources more efficiently, and ultimately boost your bottom line.

Regression Analysis has several applications in finance. It is used to calculate the Beta (Volatility of returns relative to the overall market) for a stock. When forecasting financial statements for a company, it may be useful to do a multiple regression analysis to determine how changes in certain assumptions or drivers of the business will impact revenue or expenses in future.

# SOFTWARE REQUIREMENTS

**Software:**

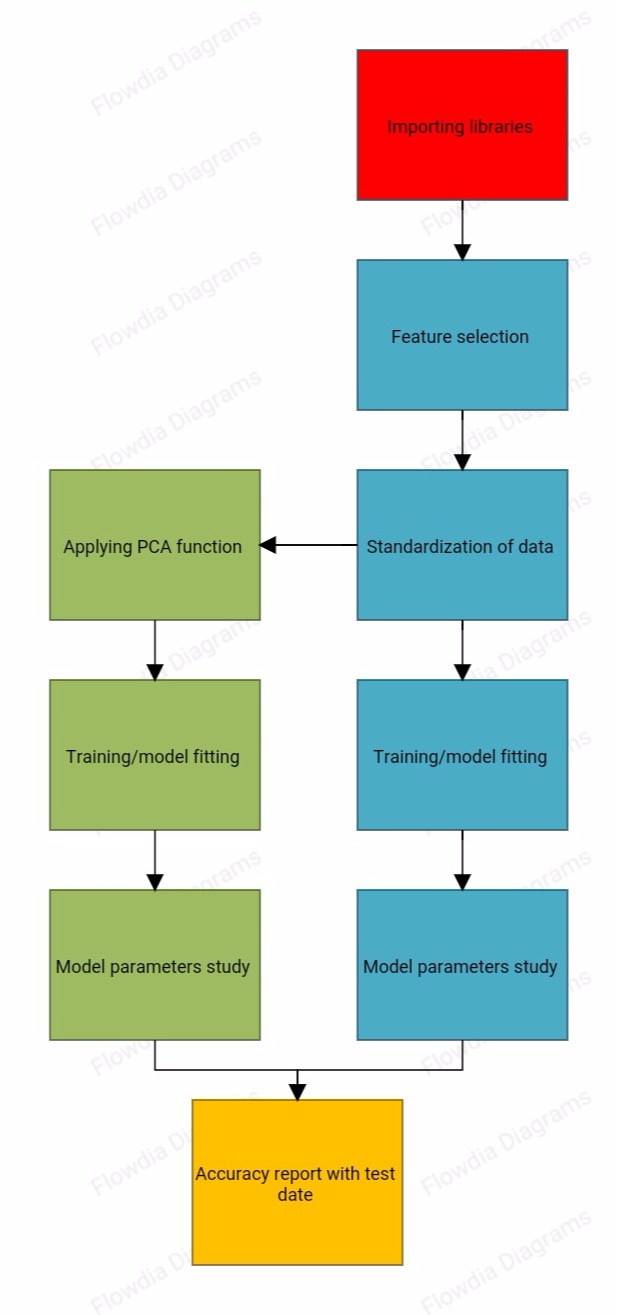
* Python

**Libraries Required:**

* Numpy
* Pandas
* Matplotlib
* Seaborn
* Sklearn

# METHODOLOGY

**Summary of Workflow:**

****

**Models used in the project:**

**MLR**

Introduction:

Multiple linear regression allows the investigator to account for all of these potentially important factors in one model. The advantages of this approach are that this may lead to a more accurate and precise understanding of the association of each individual factor with the outcome

The multiple regression model is based on the following assumptions:

* The multiple regression model is based on the following assumptions:

There is a [linear relationship](https://www.investopedia.com/terms/l/linearrelationship.asp) between the dependent variables and the independent variables.

* The independent variables are not too highly [correlated](https://www.investopedia.com/terms/c/correlation.asp) with each other.
* yi observations are selected independently and randomly from the population.
* Residuals should be [normally distributed](https://www.investopedia.com/terms/n/normaldistribution.asp) with a mean of 0 and [variance](https://www.investopedia.com/terms/v/variance.asp) *σ.*

Algorithm:

1. Import necessary libraries
2. Feature selection
3. Normalize the data
4. Splitting the data set into train and test
5. Training/model fitting (LinearRegression())
6. Model parameters study
7. Accuracy report with test data

Mathematical formula

*yi*​=*β*0​+*β*1​*xi*1​+*β*2​*xi*2​+...+*βp*​*xip*​+*ϵ*

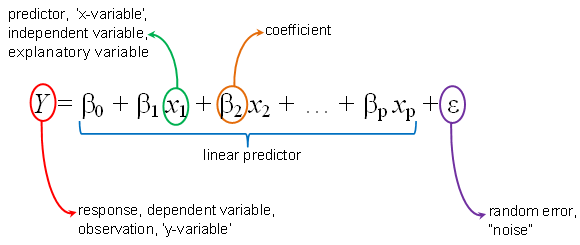
where, for *i*=*n* observations:

*yi*​=dependent variable

*xi*​=explanatory variables

*β*0​=y-intercept (constant term)

*βp*​=slope coefficients for each explanatory variable

*ϵ*=the model’s error term (also known as the residuals)​

**RFR**

Introduction:

Random Forest is a flexible, easy to use machine learning algorithm that produces great results most of the time with minimum time spent on hyper-parameter tuning. It has gained popularity due to its simplicity and diversity.

Application of RFR:

* The random forest algorithm is used in a lot of different fields, like banking, the stock market, medicine and e-commerce.
* In finance, for example, it is used to detect customers more likely to repay their debt on time, or use a bank's services more frequently. In this domain it is also used to detect fraudsters out to scam the bank. In trading, the algorithm can be used to determine a stock's future behaviour.
* In the healthcare domain it is used to identify the correct combination of components in medicine and to analyse a patient’s medical history to identify diseases.
* Random forest is used in e-commerce to determine whether a customer will actually like the product or not

ENSEMBLE METHODS:

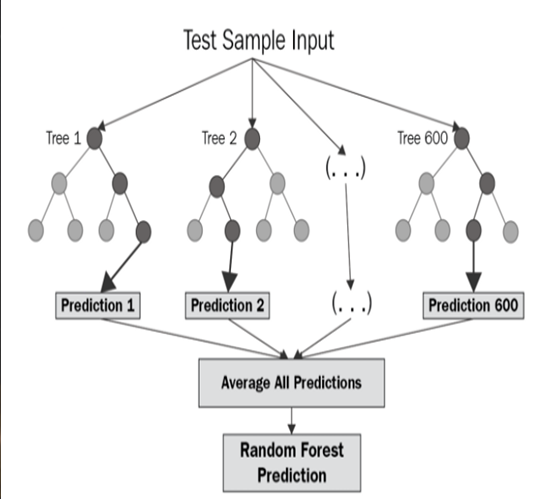
**Bagging**:-

It often considers homogeneous weak learners, learns them independently from each other in parallel and combines them following some kind of deterministic averaging process

**Boosting**:-

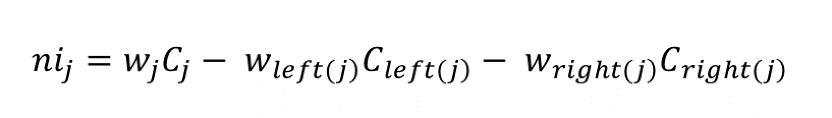
It often considers homogeneous weak learners, learns them sequentially in a very adaptative way (a base model depends on the previous ones) and combines them following a deterministic strategy

**Stacking**:-

It often considers heterogeneous weak learners, learns them in parallel and combines them by training a meta-model to output a prediction based on the different weak models predictions

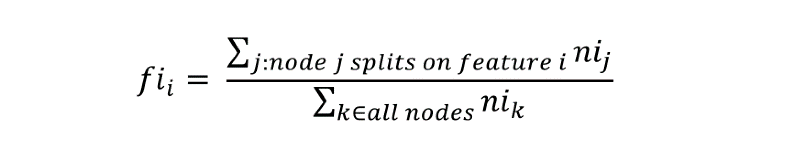
Algorithm:

1. Import necessary libraries
2. Feature selection
3. Normalize the data
4. Splitting the data set into train and test
5. Training/model fitting (RandomForestRegressor())
6. Model parameters study
7. Accuracy report with test data

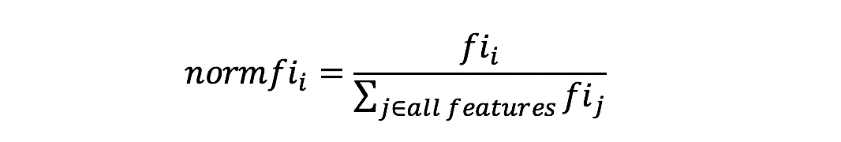
Mathematical formulae:

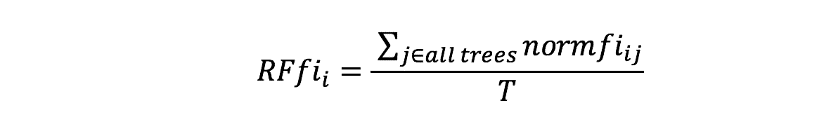
For each decision tree, Scikit-learn calculates a nodes importance using Gini Importance, assuming only two child nodes (binary tree):

* nij= the importance of node j
* wj = weighted number of samples reaching node j
* Cj=the impurity value of node j
* wleft(j)= child node from left split on node j
* wright(j)= = child node from right split on node j

The importance for each feature on a decision tree is then calculated as:

* fi sub(i)= the importance of feature i
* ni sub(j)= the importance of node j

These can then be normalized to a value between 0 and 1 by dividing by the sum of all feature importance values:

The final feature importance, at the Random Forest level, is it’s average over all the trees. The sum of the feature’s importance value on each trees is calculated and divided by the total number of trees:

* RFfi sub(i)= the importance of feature i calculated from all trees in the Random Forest model
* normfi sub(ij)= the normalized feature importance for i in tree j
* T = total number of trees

**MLR & RFR with PCA:**

Introduction:

Principal component analysis, or **PCA**, is a statistical technique to convert high dimensional data to low dimensional data by selecting the most important features that capture maximum information about the dataset. The features are selected on the basis of variance that they cause in the output. The feature that causes highest variance is the first principal component. The feature that is responsible for second highest variance is considered the second principal component, and so on. It is important to mention that principal components do not have any correlation with each other.

There are two main advantages of [dimensionality reduction](https://en.wikipedia.org/wiki/Dimensionality_reduction) with PCA.

* The training time of the algorithms reduces significantly with less number of features.
* It is not always possible to analyse data in high dimensions. For instance if there are 100 features in a dataset. Total number of scatter plots required to visualize the data would be 100(100-1)2 = 4950. Practically it is not possible to analyse data this way.

Goal of PCA:

* It identifies the patterns in the data.
* Detect the correlation between variables.
* Reduce the dimensions of a d-dimensional dataset by projecting it
* onto a k-dimensional subspace(where k&lt;d).

Algorithm:

1. Import necessary libraries
2. Feature selection
3. Normalize the data
4. Applying PCA function
5. Splitting the data set into train and test
6. Training/model fitting(RandomForestRegressor()/LinearRegression())
7. Model parameters study
8. Accuracy report with test data

Mathematical formulae:

we have an equation like

λu1= 𝛿u1

(The above equation looks similar to the equation of Eigen vectors i.e λ u1= 𝛿v1 where λ=Eigen values of 𝛿d\*d= λ1, λ2…. λd with corresponding v1,v2,…vd such that λ1≥λ2≥λ3≥λ4… λd)

𝛿 = (XTX)/ n

where 𝛿 = covariance matrix of size d\*d (i.e. for d features)

Therefore, finally u1 is nothing but the Eigen vector v1 of 𝛿d\*d corresponding to λ1 (i.e. corresponding to maximal Eigen values). Each λ corresponding to u &λ describes the variance. So by then we got u1=v1 to find the u2 we have the property of Eigen vector such as if vi ⊥ r vj for all i& j.

Then v2 ⊥rv1 (also u1 ⊥ru2 )

∴u2 = v2

Therfore, we can say that V1 = direction with maximal variance,

v2= direction with 2nd most maximal variance and so on..

Eigen values λ can help us to determine

% variance explained = λi/Σλi (for all i)

λ tells us how much information is left over after reducing the information.

From the m independent variables of your dataset, PCA extracts p<=m new independent variables that explain the most variance of the dataset regardless of the dependent variable. The fact is that dependent variable is not considered. So, it makes PCA as unsupervised algorithm.

# REGRESSION CODE

**#**importing libraries

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

import seaborn as sns

import tkinter as tk

#importing dataset

df=pd.read\_csv('Admission\_Predict\_Ver1.1.csv')

print(df)

sns.pairplot(df)

fig, ax = plt.subplots(figsize=(15,10))

sns.heatmap(df.corr(), annot=True, cmap='Blues')

x=df.iloc[:,1:-1].values

y=df.iloc[:,-1].values

#performing feature scalling

from sklearn.preprocessing import StandardScaler

sc=StandardScaler()

x=sc.fit\_transform(x)

#splitting into train and test data

from sklearn.model\_selection import train\_test\_split

x\_tr,x\_te,y\_tr,y\_te=train\_test\_split(x,y,test\_size=0.2,random\_state=11)

# multiple Linear Regression

from sklearn.linear\_model import LinearRegression

lr = LinearRegression()

lr.fit(x\_tr,y\_tr)

mlr = (lr.score(x\_te, y\_te))\*100

y\_pred1 = lr.predict(x\_te)

from sklearn.metrics import r2\_score,mean\_squared\_error

mlr\_mse = mean\_squared\_error(y\_te,y\_pred1)

mlr\_r2 = r2\_score(y\_te,y\_pred1)

mlr\_ = (lr.score(x\_te, y\_te))\*100

print("MSE: ",mlr\_mse)

print("r2\_score: ",mlr\_r2)

dt=[x\_te[:,0],x\_te[:,1],x\_te[:,2],x\_te[:,3],x\_te[:,4],x\_te[:,5],x\_te[:,6],y\_te,y\_pred1]

f, ax = plt.subplots(7,1,figsize=(10, 31))

sns.regplot(x=x\_te[:,0],y=y\_te,data=dt,color='blue',label=' actual ',ax=ax[0])

sns.regplot(x=x\_te[:,0],y=y\_pred1,data=dt,color='red',label='predicted',ax=ax[0])

ax[0].legend()

ax[0].set\_title('GRE Score vs Chance of admit')

ax[0].set\_xlabel('GRE Score ')

ax[0].set\_ylabel('chance of admit')

ax[0].axis('tight')

sns.regplot(x=x\_te[:,1],y=y\_te,data=dt,color='blue',label=' actual ',ax=ax[1])

sns.regplot(x=x\_te[:,1],y=y\_pred1,data=dt,color='red',label='predicted',ax=ax[1])

ax[1].legend()

ax[1].set\_title('TOEFL score vs Chance of admit')

ax[1].set\_xlabel('TOEFL score')

ax[1].set\_ylabel('chance of admit')

ax[1].axis('tight')

sns.regplot(x=x\_te[:,2],y=y\_te,data=dt,color='blue',label=' actual ',ax=ax[2])

sns.regplot(x=x\_te[:,2],y=y\_pred1,data=dt,color='red',label='predicted',ax=ax[2])

ax[2].set\_title('University Rating vs Chance of admit')

ax[2].set\_xlabel('University Rating')

ax[2].set\_ylabel('chance of admit')

ax[2].axis('tight')

ax[2].legend()

sns.regplot(x=x\_te[:,3],y=y\_te,data=dt,color='blue',label=' actual ',ax=ax[3])

sns.regplot(x=x\_te[:,3],y=y\_pred1,data=dt,color='red',label='predicted',ax=ax[3])

ax[3].set\_title(' SOP vs Chance of admit')

ax[3].set\_xlabel(' SOP')

ax[3].set\_ylabel('chance of admit')

ax[3].axis('tight')

ax[3].legend()

sns.regplot(x=x\_te[:,4],y=y\_te,data=dt,color='blue',label=' actual ',ax=ax[4])

sns.regplot(x=x\_te[:,4],y=y\_pred1,data=dt,color='red',label='predicted',ax=ax[4])

ax[4].set\_title('LOR vs Chance of admit')

ax[4].set\_xlabel('LOR ')

ax[4].set\_ylabel('chance of admit')

ax[4].axis('tight')

ax[4].legend()

sns.regplot(x=x\_te[:,5],y=y\_te,data=dt,color='blue',label=' actual ',ax=ax[5])

sns.regplot(x=x\_te[:,5],y=y\_pred1,data=dt,color='red',label='predicted',ax=ax[5])

ax[5].set\_title(' CGPA vs Chance of admit')

ax[5].set\_xlabel(' CGPA ')

ax[5].set\_ylabel('chance of admit')

ax[5].axis('tight')

ax[5].legend()

sns.regplot(x=x\_te[:,6],y=y\_te,data=dt,color='blue',label=' actual ',ax=ax[6])

sns.regplot(x=x\_te[:,6],y=y\_pred1,data=dt,color='red',label='predicted',ax=ax[6])

ax[6].set\_title('Research vs Chance of admit')

ax[6].set\_xlabel(' Research ')

ax[6].set\_ylabel('chance of admit')

ax[6].axis('tight')

ax[6].legend()

fig.tight\_layout()

plt.tight\_layout()

# Random Forest Regression Model

from sklearn.ensemble import RandomForestRegressor

rfr=RandomForestRegressor(n\_estimators=100,random\_state=11)

rfr.fit(x\_tr,y\_tr)

rfr\_ = (rfr.score(x\_te, y\_te))\*100

#predicting values

y\_pred=rfr.predict(x\_te)

#Calculating accuracy score and error

from sklearn.metrics import r2\_score,mean\_absolute\_error,mean\_squared\_error

r2=r2\_score(y\_te,y\_pred)

mae=mean\_absolute\_error(y\_te,y\_pred)

mse=mean\_squared\_error(y\_te,y\_pred)

rmse = np.sqrt(mse)

print ("r2\_score: ", r2)

print ("MAE: ", mae)

print ("MSE: ", mse)

print ("RMSE: ", rmse)

dt=[x\_te[:,0],x\_te[:,1],x\_te[:,2],x\_te[:,3],x\_te[:,4],x\_te[:,5],x\_te[:,6],y\_te,y\_pred2]

f, ax = plt.subplots(7,1,figsize=(10, 31))

sns.regplot(x=x\_te[:,0],y=y\_te,data=dt,color='blue',label=' actual ',ax=ax[0])

sns.regplot(x=x\_te[:,0],y=y\_pred2,data=dt,color='red',label='predicted',ax=ax[0])

ax[0].legend()

ax[0].set\_title('GRE Score vs Chance of admit')

ax[0].set\_xlabel('GRE Score ')

ax[0].set\_ylabel('chance of admit')

ax[0].axis('tight')

sns.regplot(x=x\_te[:,1],y=y\_te,data=dt,color='blue',label=' actual ',ax=ax[1])

sns.regplot(x=x\_te[:,1],y=y\_pred2,data=dt,color='red',label='predicted',ax=ax[1])

ax[1].legend()

ax[1].set\_title('TOEFL score vs Chance of admit')

ax[1].set\_xlabel('TOEFL score')

ax[1].set\_ylabel('chance of admit')

ax[1].axis('tight')

sns.regplot(x=x\_te[:,2],y=y\_te,data=dt,color='blue',label=' actual ',ax=ax[2])

sns.regplot(x=x\_te[:,2],y=y\_pred2,data=dt,color='red',label='predicted',ax=ax[2])

ax[2].set\_title('University Rating vs Chance of admit')

ax[2].set\_xlabel('University Rating')

ax[2].set\_ylabel('chance of admit')

ax[2].axis('tight')

ax[2].legend()

sns.regplot(x=x\_te[:,3],y=y\_te,data=dt,color='blue',label=' actual ',ax=ax[3])

sns.regplot(x=x\_te[:,3],y=y\_pred2,data=dt,color='red',label='predicted',ax=ax[3])

ax[3].set\_title(' SOP vs Chance of admit')

ax[3].set\_xlabel(' SOP')

ax[3].set\_ylabel('chance of admit')

ax[3].axis('tight')

ax[3].legend()

sns.regplot(x=x\_te[:,4],y=y\_te,data=dt,color='blue',label=' actual ',ax=ax[4])

sns.regplot(x=x\_te[:,4],y=y\_pred2,data=dt,color='red',label='predicted',ax=ax[4])

ax[4].set\_title('LOR vs Chance of admit')

ax[4].set\_xlabel('LOR ')

ax[4].set\_ylabel('chance of admit')

ax[4].axis('tight')

ax[4].legend()

sns.regplot(x=x\_te[:,5],y=y\_te,data=dt,color='blue',label=' actual ',ax=ax[5])

sns.regplot(x=x\_te[:,5],y=y\_pred2,data=dt,color='red',label='predicted',ax=ax[5])

ax[5].set\_title(' CGPA vs Chance of admit')

ax[5].set\_xlabel(' CGPA ')

ax[5].set\_ylabel('chance of admit')

ax[5].axis('tight')

ax[5].legend()

sns.regplot(x=x\_te[:,6],y=y\_te,data=dt,color='blue',label=' actual ',ax=ax[6])

sns.regplot(x=x\_te[:,6],y=y\_pred2,data=dt,color='red',label='predicted',ax=ax[6])

ax[6].set\_title('Research vs Chance of admit')

ax[6].set\_xlabel(' Research ')

ax[6].set\_ylabel('chance of admit')

ax[6].axis('tight')

ax[6].legend()

fig.tight\_layout()

plt.tight\_layout()

#MLR with PCA

from sklearn.decomposition import PCA

pca\_= PCA(n\_components=6)

x\_1= pca\_.fit\_transform(x)

variance=pca\_.explained\_variance\_ratio\_

var=np.cumsum(np.round(variance, decimals=3)\*100)

nc=np.array([1,2,3,4,5,6])

plt.ylabel('% Variance Explained')

plt.xlabel('Number of Features')

plt.title('PCA Analysis')

plt.ylim(60,100)

plt.plot(nc,var)

plt.grid(True)

from sklearn.decomposition import PCA

pca\_= PCA(n\_components=2)

x1= pca\_.fit\_transform(x)

print(pca\_.explained\_variance\_ratio\_)

```

from sklearn.model\_selection import train\_test\_split

x\_train,x\_test,y\_train,y\_test = train\_test\_split(x1,y,test\_size = 0.2,random\_state = 11)

from sklearn.linear\_model import LinearRegression

mlr = LinearRegression()

mlr.fit(x\_test,y\_test)

mlr\_pca=(mlr.score(x\_test, y\_test))\*100

y\_pred3 = mlr.predict(x\_test)

from sklearn.metrics import r2\_score,mean\_squared\_error

mlr\_pca\_mse = mean\_squared\_error(y\_test,y\_pred3)

mlr\_pca\_r2 = r2\_score(y\_test,y\_pred3)

print("MSE: ",mlr\_pca\_mse)

print("r2\_score: ",mlr\_pca\_r2)

dt2=[x\_test[:,0],x\_test[:,1],y\_test,y\_pred3]

f, ax = plt.subplots(1, 2,figsize=(12, 4))

sns.regplot(x=x\_test[:,0],y=y\_test,data=dt2,color='blue',label='test',ax=ax[0],)

sns.regplot(x=x\_test[:,0],y=y\_pred3,data=dt2,color='red',label='predicted',ax=ax[0])

ax[0].legend()

ax[0].set\_title('GRE Score vs Chance of admit')

ax[0].set\_xlabel('GRE Score ')

ax[0].set\_ylabel('chance of admit')

ax[0].axis('tight')

sns.regplot(x=x\_test[:,1],y=y\_test,data=dt2,color='blue',label='test',ax=ax[1])

sns.regplot(x=x\_test[:,1],y=y\_pred3,data=dt2,color='red',label='predicted',ax=ax[1])

ax[1].legend()

ax[1].set\_title('TOEFL score vs Chance of admit')

ax[1].set\_xlabel('TOEFL score')

ax[1].set\_ylabel('chance of admit')

ax[1].axis('tight')

# RFR with PCA

from sklearn.decomposition import PCA

pca = PCA(n\_components=2)

x2 = pca.fit\_transform(x)

print(pca.explained\_variance\_ratio\_)

from sklearn.model\_selection import train\_test\_split

x\_train,x\_test,y\_train,y\_test = train\_test\_split(x2,y,test\_size = 0.2,random\_state = 11)

from sklearn.ensemble import RandomForestRegressor

rfr2 = RandomForestRegressor(n\_estimators=100, random\_state=0)

rfr2.fit(x\_train,y\_train)

rfr\_pca=(rfr2.score(x\_test, y\_test))\*100

y\_pred4 = rfr2.predict( x\_test)

from sklearn.metrics import r2\_score,mean\_squared\_error

rfr\_pca\_mse = mean\_squared\_error(y\_test,y\_pred4)

rfr\_pca\_r2 = r2\_score(y\_test,y\_pred4)

print("MSE: ",rfr\_pca\_mse)

print("r2\_score: ",rfr\_pca\_r2)

dt=[x\_test[:,0],x\_test[:,1],y\_test,y\_pred4]

f, ax = plt.subplots(1, 2,figsize=(12, 4))

sns.regplot(x=x\_test[:,0],y=y\_test,data=dt,color='blue',label='test',ax=ax[0],)

sns.regplot(x=x\_test[:,0],y=y\_pred4,data=dt,color='red',label='predicted',ax=ax[0])

ax[0].legend()

ax[0].set\_title('GRE Score vs Chance of admit')

ax[0].set\_xlabel('GRE Score ')

ax[0].set\_ylabel('chance of admit')

ax[0].axis('tight')

sns.regplot(x=x\_test[:,1],y=y\_test,data=dt,color='blue',label='test',ax=ax[1])

sns.regplot(x=x\_test[:,1],y=y\_pred4,data=dt,color='red',label='predicted',ax=ax[1])

ax[1].legend()

ax[1].set\_title('TOEFL score vs Chance of admit')

ax[1].set\_xlabel('TOEFL score')

ax[1].set\_ylabel('chance of admit')

ax[1].axis('tight')

# Result

Methods=['MLR', 'RFR', 'MLR with PCA','RFR with PCA']

Scores1=np.array([mlr\_,rfr\_,mlr\_pca,rfr\_pca])

Scores2=np.array([mlr\_mse,rfr\_mse,rfr\_pca\_mse,rfr\_pca\_mse])

def show\_values\_on\_bars(axs):

def \_show\_on\_single\_plot(ax):

for p in ax.patches:

\_x = p.get\_x() + p.get\_width() / 2

\_y = p.get\_y() + p.get\_height()

value = '{:.4f}'.format(p.get\_height())

ax.text(\_x, \_y, value, ha="center")

if isinstance(axs, np.ndarray):

for idx, ax in np.ndenumerate(axs):

\_show\_on\_single\_plot(ax)

else:

\_show\_on\_single\_plot(axs)

fig, ax = plt.subplots(1,2,figsize=(12,4))

sns.barplot(Methods, Scores1,ax=ax[0])

ax[0].set\_title("Algorithm's Accuracies")

ax[0].set\_ylabel('Accuracy')

ax[0].axis('tight')

ax[0].grid(True)

show\_values\_on\_bars(ax[0])

sns.barplot(Methods, Scores2,ax=ax[1])

ax[1].set\_title("Algorithm's MSE")

ax[1].set\_ylabel('MSE')

ax[1].axis('tight')

ax[1].grid(True)

show\_values\_on\_bars(ax[1])

`

based on the above graph Multiple Linear Regression is the most preferable model

x2=df.iloc[:,1:-1].values

y2=df.iloc[:,-1].values

from sklearn.model\_selection import train\_test\_split

x\_tr1,x\_te1,y\_tr1,y\_te1=train\_test\_split(x2,y2,test\_size=0.2,random\_state=11)

from sklearn.linear\_model import LinearRegression

lr\_ = LinearRegression()

lr\_.fit(x\_tr1,y\_tr1)

root= tk.Tk()

canvas1 = tk.Canvas(root, width = 550, height = 300)

canvas1.pack()

# GRE Score

label1 = tk.Label(root, text='GRE Score:')

canvas1.create\_window(137, 60, window=label1)

entry1 = tk.Entry(root)

canvas1.create\_window(280, 60, window=entry1)

# TOEFL Score:

label2 = tk.Label(root, text='TOEFL Score:')

canvas1.create\_window(142, 80, window=label2)

entry2 = tk.Entry(root)

canvas1.create\_window(280, 80, window=entry2)

# University Rating

label3 = tk.Label(root, text=' University Rating:')

canvas1.create\_window(150, 100, window=label3)

entry3 = tk.Entry(root)

canvas1.create\_window(280, 100, window=entry3)

# SOP

label4 = tk.Label(root, text='SOP:')

canvas1.create\_window(120, 120, window=label4)

entry4 = tk.Entry(root)

canvas1.create\_window(280, 120, window=entry4)

#LOR

label5 = tk.Label(root, text='LOR:')

canvas1.create\_window(120, 140, window=label5)

entry5 = tk.Entry(root)

canvas1.create\_window(280, 140, window=entry5)

#CGPA

label6 = tk.Label(root, text='CGPA:')

canvas1.create\_window(125, 160, window=label6)

entry6 = tk.Entry(root)

canvas1.create\_window(280, 160, window=entry6)

#Research

label7 = tk.Label(root, text='Research:')

canvas1.create\_window(125, 180, window=label7)

entry7 = tk.Entry(root)

canvas1.create\_window(280, 180, window=entry7)

def values():

global GRE\_Score

GRE\_Score = float(entry1.get())

global TOEFL\_Score

TOEFL\_Score = float(entry2.get())

global University\_Rating

University\_Rating = float(entry3.get())

global SOP

SOP = float(entry4.get())

global LOR

LOR = float(entry5.get())

global CGPA

CGPA = float(entry6.get())

global Research

Research = float(entry7.get())

Prediction\_result = (' Predicted Result:’

,lr\_.predict([[GRE\_Score,TOEFL\_Score,University\_Rating,SOP,LOR,CGPA,Research]]),)

label\_Prediction = tk.Label(root, text= Prediction\_result, bg='sky blue')

label\_Prediction.config(font=("Courier", 15))

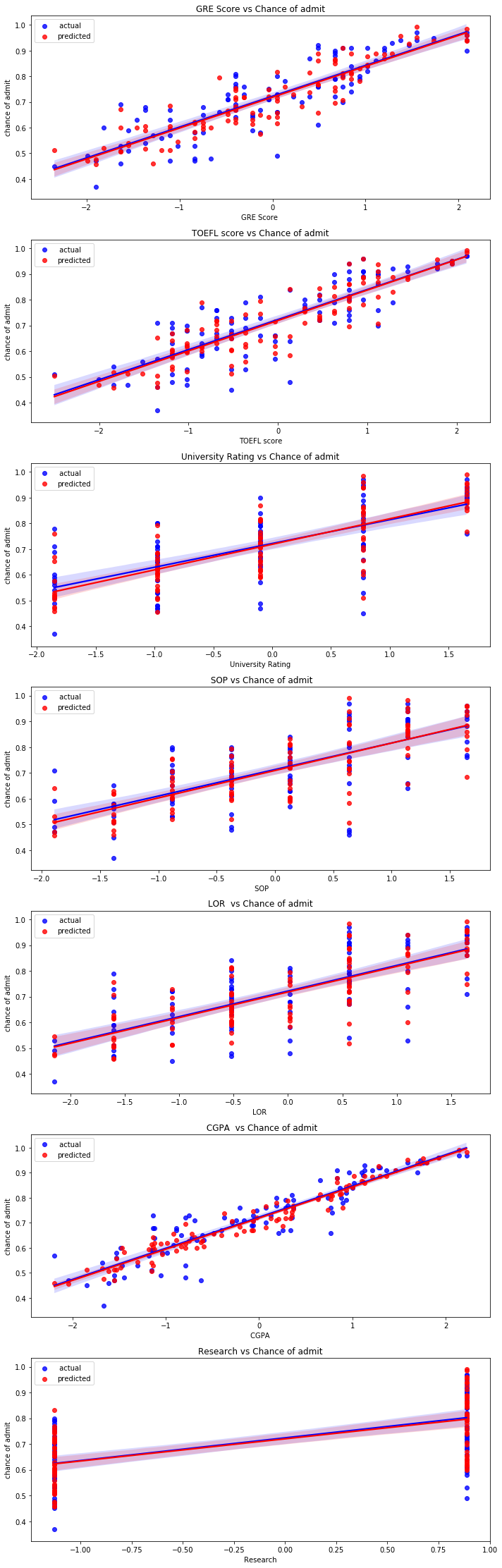
canvas1.create\_window(270, 280, window=label\_Prediction)

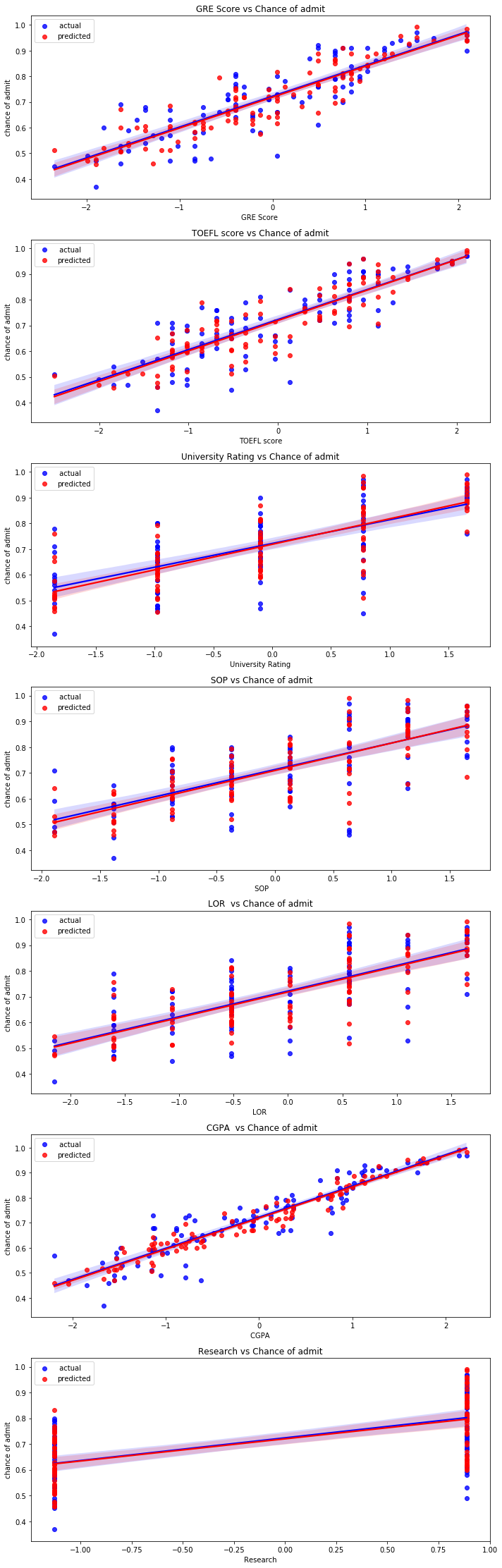
button1 = tk.Button (root, text=' Predict ',command=values, bg='green', fg='white', font=11)

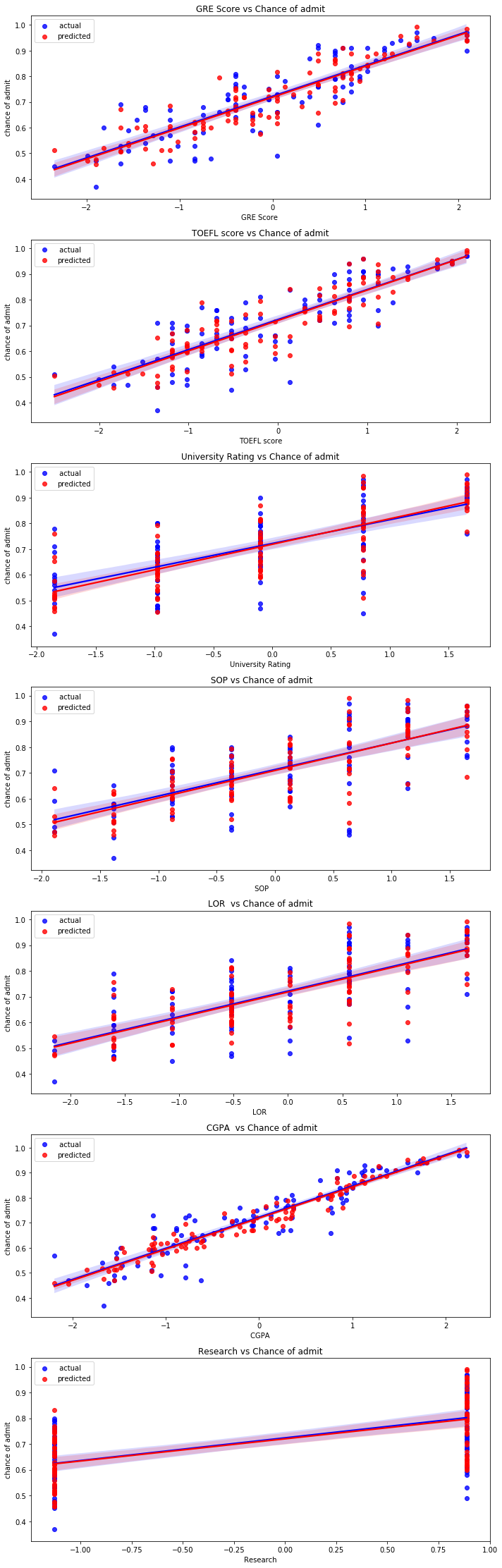
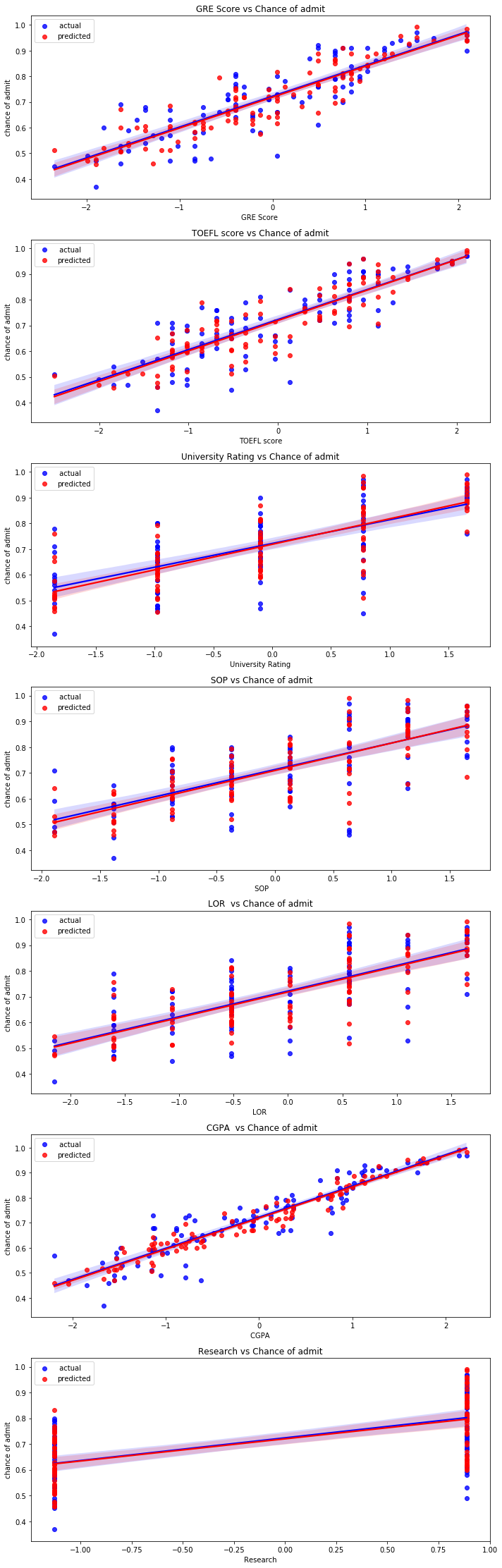
canvas1.create\_window(270, 220, window=button1)

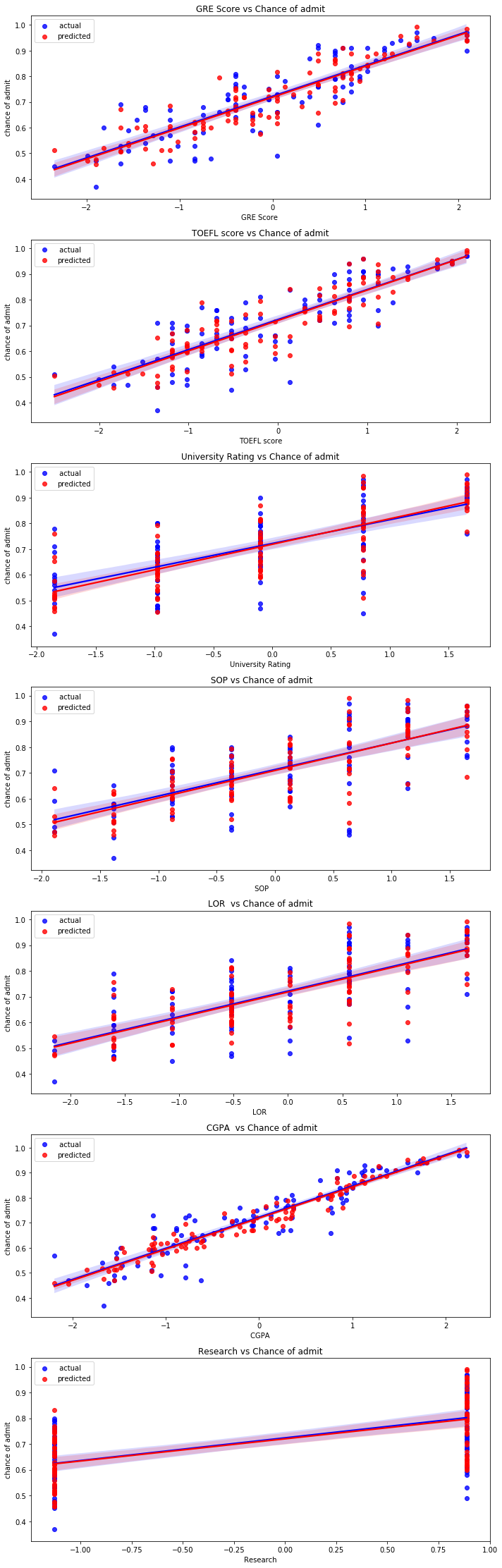
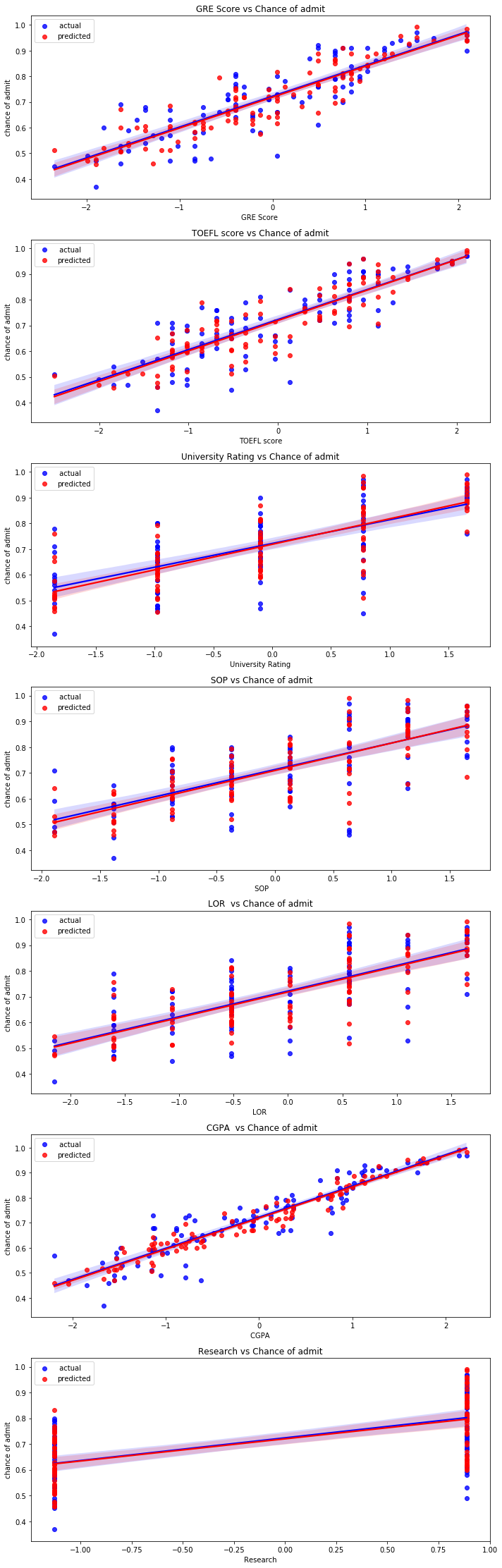
root.mainloop()

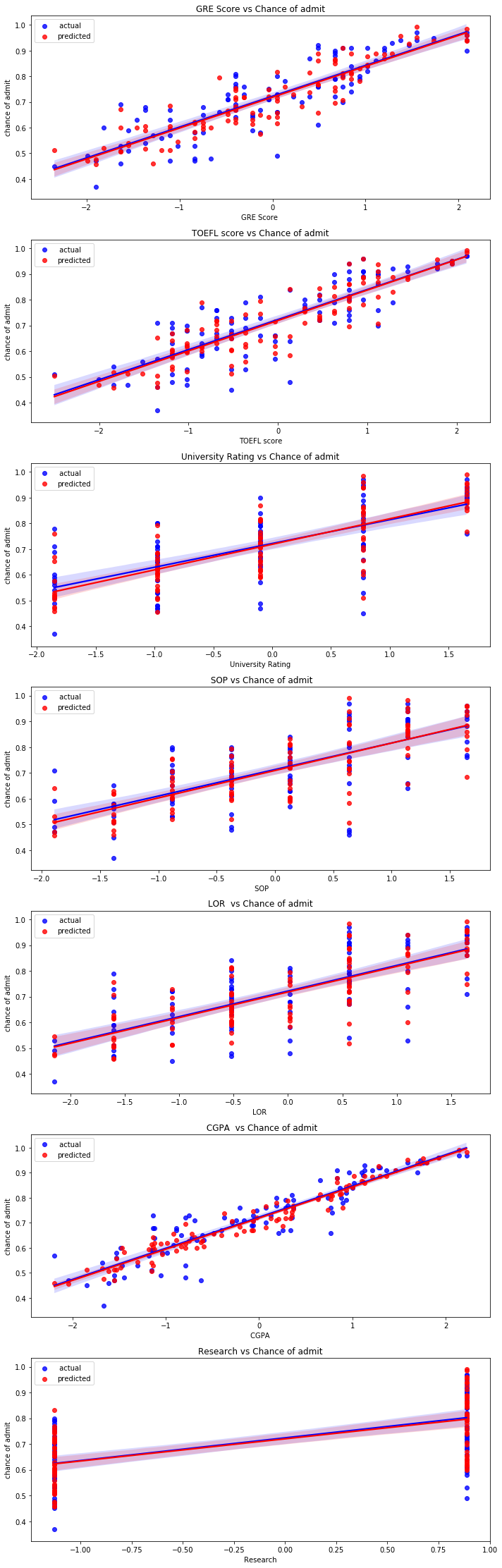
# TESTING AND RESULT

**Result of MLR for test and predicted:**

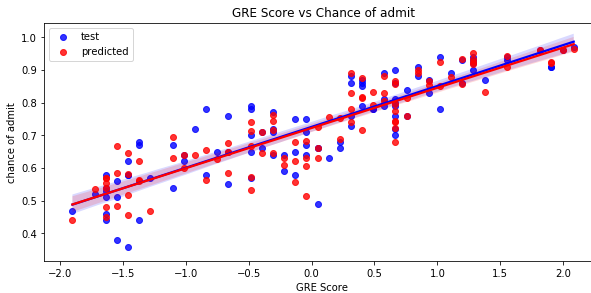


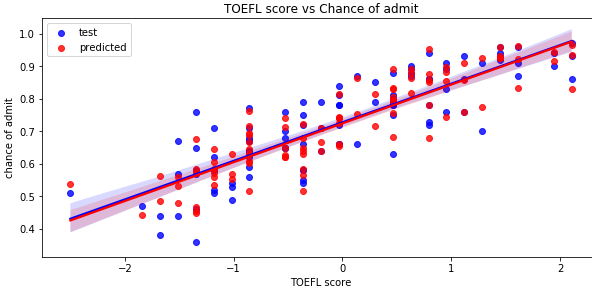


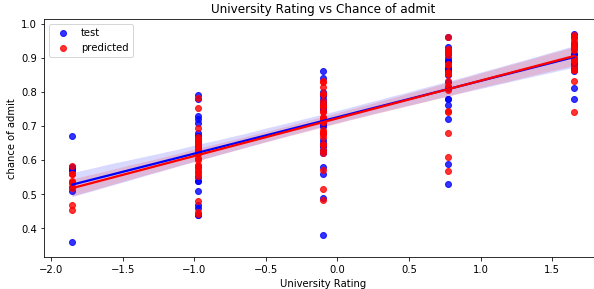
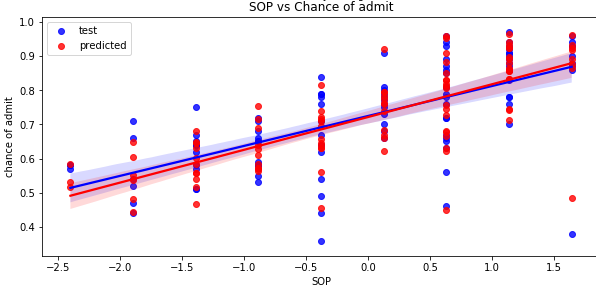
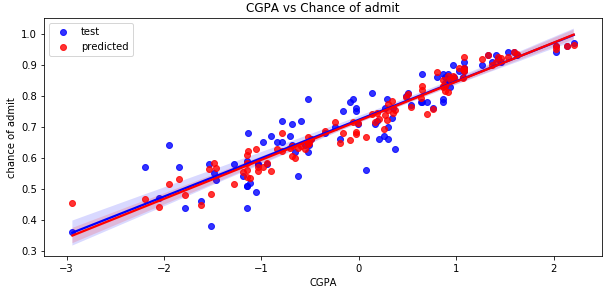
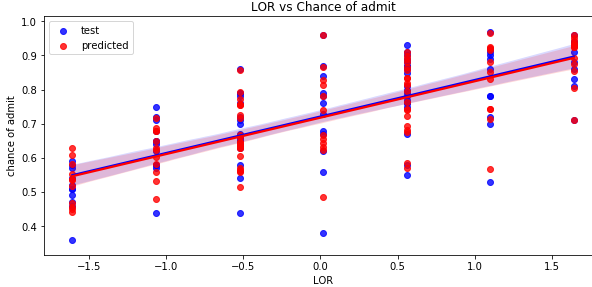
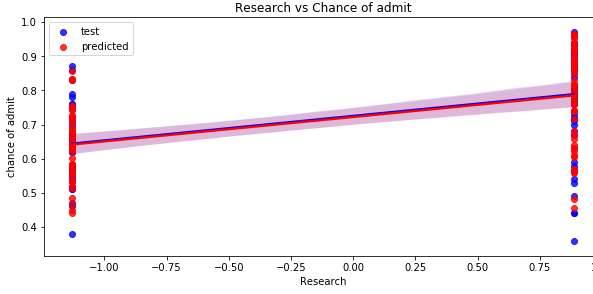


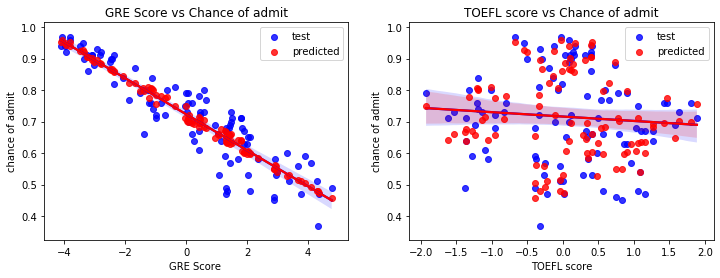


**Result of RFR for test and predicted:**

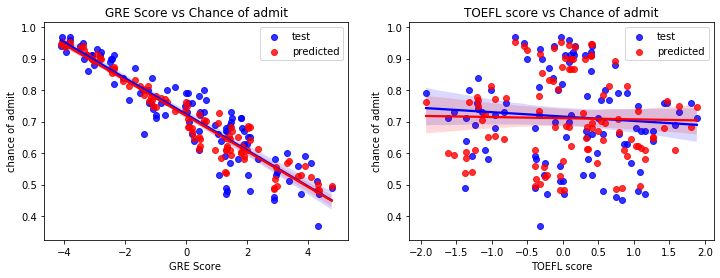
**

****

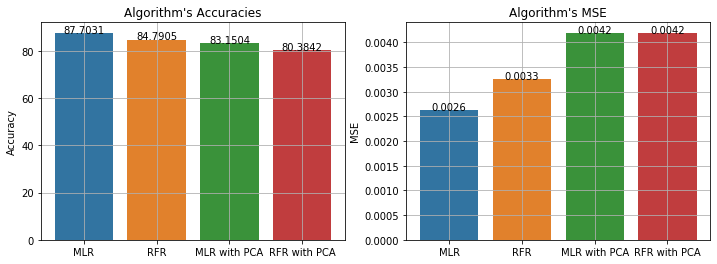
****

**Result of MLR with PCA for test and predicted:**

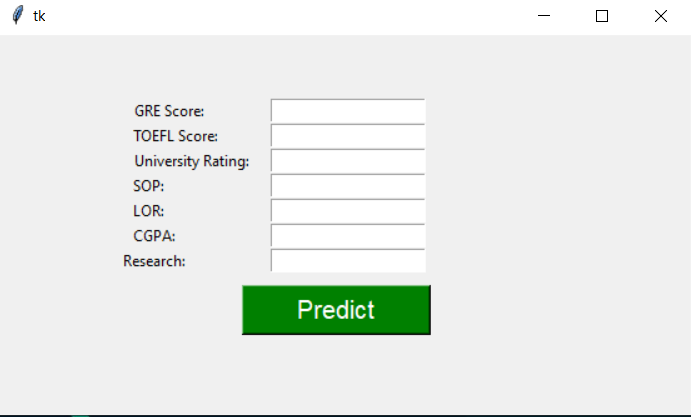
**Result of RFR with PCA for test and predicted:**

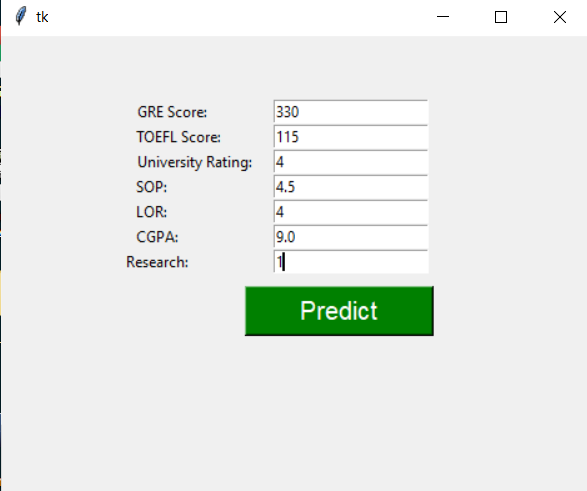
****

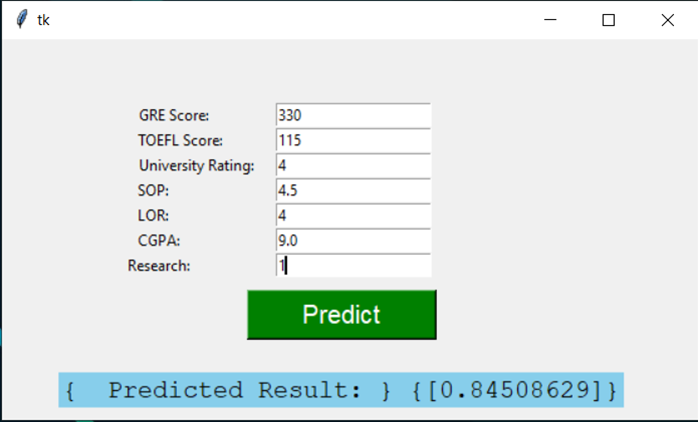
**Model comparison:**



**TESTING:**

****

****



# Conclusion

The main goal of this work is to create a Machine Learning model which could be used by students who want to pursue their education in the US. Many machine learning algorithms were utilized for this research. Linear Regression model compared to other ones. Students can use the model to assess their chances of getting admission into a particular university with an average accuracy of 87 percent. A GUI was developed to make the program, from a non-technical perspective, usable and user-friendly. Using node-red the user interface was developed. The ultimate goal of research will be accomplished successfully, as the system allows students to save the lot of time and money that they would spend on educational mentors and application fees for colleges where they have less chances of getting admissions. The main limitation of this research is we developed models based solely on data from Indian Students studying Masters in Computer Science in the United States, we considered only few universities with different rankings. More information relating to new colleges and courses can be added to the curriculum in the future. The system may also be modified to a web-based application by making node-red modifications. To solve the problem, it is possible to test other classification algorithms if they have high accuracy score than the current algorithm, the framework can be easily modified to support the new algorithm by changing the server code in the Node Red. Finally students can have an open source machine Learning model which will help the students to know their chance of admission into a particular university with high accuracy.

# reference

**Web page:**

* <https://www.geeksforgeeks.org/python-tkinter-tutorial/>
* <https://github.com/girishkuniyal/Predict-chance-of-getting-admission>
* <http://zerospectrum.com/2019/06/02/the-maths-behind-multiple-linear-regression/>
* <https://seaborn.pydata.org/generated/seaborn.regplot.html>