Project Report on

ADMISSION PREDICTION

INDEX

TABLE OF FIGURES/GRAPHS :

ABSTRACT :

Problem Statement: To predict the 'Chance of Admit' with minimum MSE and RMSE and maximum R-Square score.

Content:

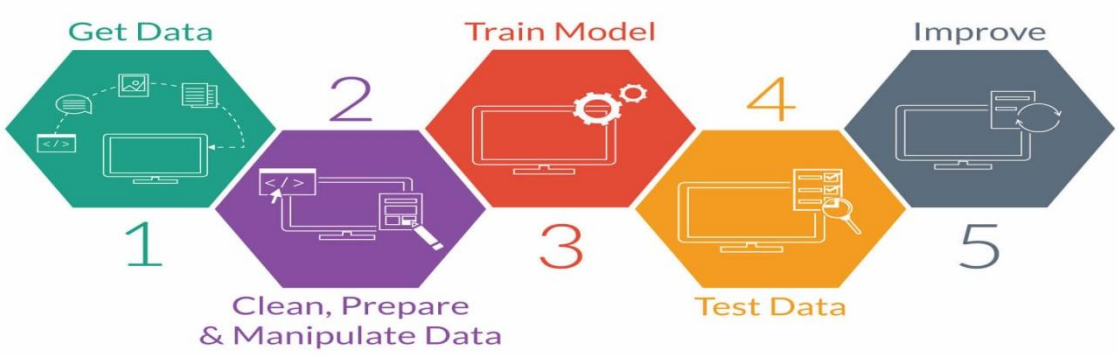
The dataset contains several parameters which are considered important during the application for MastersPrograms.

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 and Letter of Recommendation Strength (out of 5)
5. Undergraduate GPA (out of 10)
6. Research Experience (either 0 or 1)
7. Chance of Admit (ranging from 0 to 1)

INTRODUCTION :

- Many colleges in the U.S. follow similar requirements for student admission. Colleges take different factors into account, such as the ranking on aptitude assessment and academic record review. The command over the English language is calculated on the basis of their performance in the English skills test, such as TOEFL and IELTS. The admission committee of universities takes the decision to approve or reject a specific candidate on the basis of the overall profile of the applicant application.
- Probability of student application into a respective university but the main drawback is they didn't consider all the factors which will contribute in the student admission process like TOEFL/IELTS, SOP, LOR and under graduate score.
-



In this project we have made use of the following algorithms:

Supervised Regression Algorithm-
- Multiple Linear Regression Algorithm.(MLR)
- Random Forest Regression Algorithm.(RFR)

Unsupervised Regression Algorithm-
- MLR with Principle Compenent Analysis(PCA)
- RFR with Principle Compenent Analysis(PCA)

SOFTWARE-LIBRARIES:

- scikit-learn
- seaborn
- numpy
- pandas
- matplotlib

METHODOLOGY:

Problem Understanding: Initially first we have to spend some time on what are the problems or concerns students having during their pre admission period and we should set the solutions to those problems as objectives of this research.

Data Understanding: Data should be collected from multiple sources like yocket and also consider all the factors including which will play a tiny role in student admission process.

Data Preparation: Data should be cleaned that is removing the noise in the data and filling the missing values or extreme values and finalising the attributes/factors which will have crucial importance in student admission process.

Building Models: several ML models have to be developed using various machine learning algorithms for admission to a particular university and the user interface has to be developed to access those models.

Evaluation: Developed models are evaluated according to their accuracy scores. Once the model is finalised that model will be merged with node red for final deployment

VISUALISING THE DATASET

```
import seaborn as sns
```

```
sns.pairplot(df,x_vars=['Serial No.','GRE Score','TOEFL Score','University Rating','SOP','LOR ','CGPA','Research'],y_vars=['
plt.show()
```

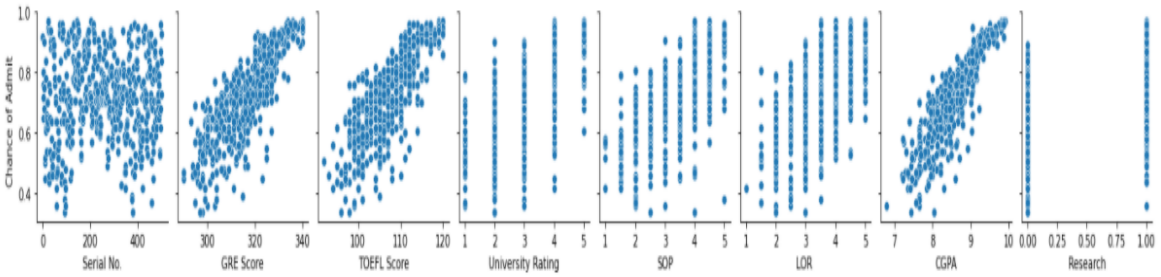


Fig4: pairplot of the analysed data

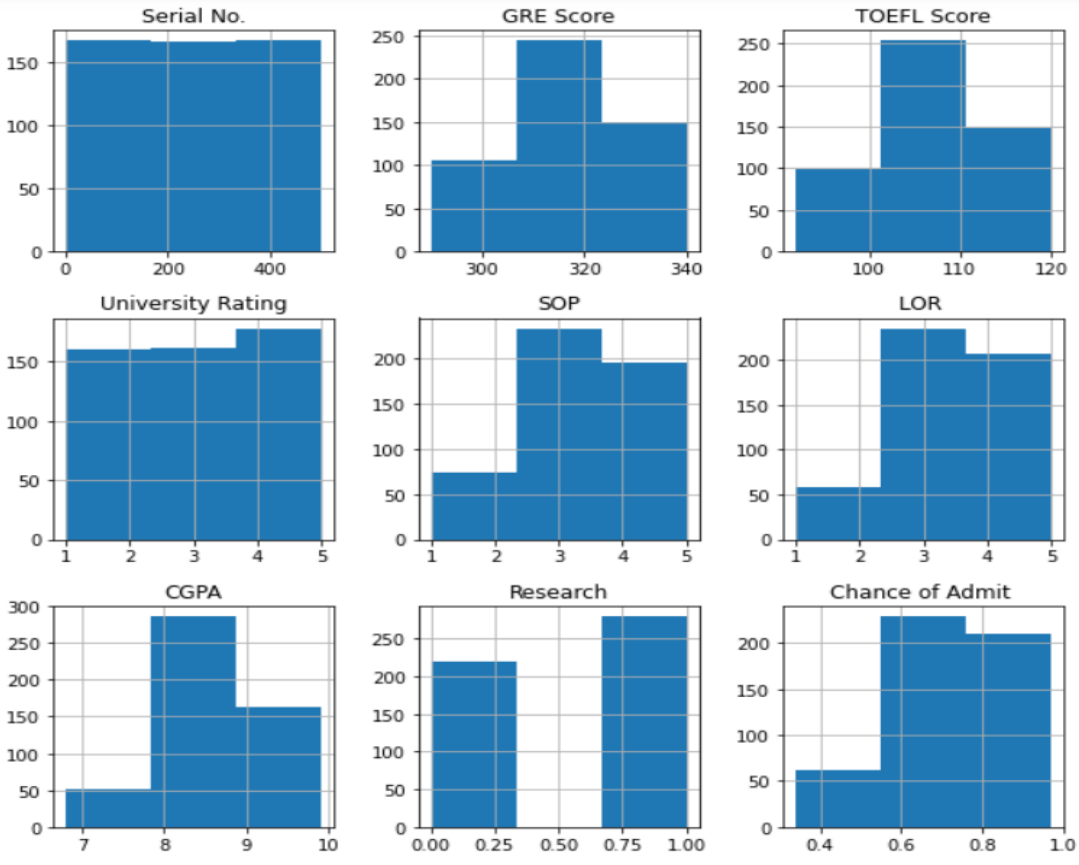


Fig5: histogram of the analysed data

DATA ANALYSIS AND DATA PREPROCESSING

First we import the libraries and load the data.

- Numpy: to work with arrays.
- Pandas: to work with csv files and dataframes
- Matplotlib: to create plots and graphs using pyplot

```
import pandas as pd
import numpy as np
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings(action="ignore")
```

To read the dataset:

```
df = pd.read_csv('Admission_Predict_Ver1.1.csv',encoding = 'utf-8')
print(df)
```

```
     Serial No.  GRE Score  TOEFL Score  University Rating  SOP  LOR   CGPA  \
0             1        337          118                  4  4.5   4.5  9.65
1             2        324          107                  4  4.0   4.5  8.87
2             3        316          104                  3  3.0   3.5  8.00
3             4        322          110                  3  3.5   2.5  8.67
4             5        314          103                  2  2.0   3.0  8.21
..          ...        ...          ...                ...  ...   ...   ...
495         496        332          108                  5  4.5   4.0  9.02
496         497        337          117                  5  5.0   5.0  9.87
497         498        330          120                  5  4.5   5.0  9.56
498         499        312          103                  4  4.0   5.0  8.43
499         500        327          113                  4  4.5   4.5  9.04

     Research  Chance of Admit
0           1              0.92
1           1              0.76
2           1              0.72
3           1              0.80
4           0              0.65
..        ...               ...
495         1              0.87
496         1              0.96
497         1              0.93
498         0              0.73
499         0              0.84

[500 rows x 9 columns]
```

Information on the dataset:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Serial No.         500 non-null    int64
 1   GRE Score          500 non-null    int64
 2   TOEFL Score        500 non-null    int64
 3   University Rating  500 non-null    int64
 4   SOP                500 non-null    float64
 5   LOR                500 non-null    float64
 6   CGPA               500 non-null    float64
 7   Research           500 non-null    int64
 8   Chance of Admit    500 non-null    float64
dtypes: float64(4), int64(5)
memory usage: 35.3 KB
```

Description of the dataset

```
1 df.describe()
```

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
count	500.000000	500.000000	500.000000	500.000000	500.000000	500.00000	500.000000	500.000000	500.00000
mean	250.500000	316.472000	107.192000	3.114000	3.374000	3.48400	8.576440	0.560000	0.72174
std	144.481833	11.295148	6.081868	1.143512	0.991004	0.92545	0.604813	0.496884	0.14114
min	1.000000	290.000000	92.000000	1.000000	1.000000	1.00000	6.800000	0.000000	0.34000
25%	125.750000	308.000000	103.000000	2.000000	2.500000	3.00000	8.127500	0.000000	0.63000
50%	250.500000	317.000000	107.000000	3.000000	3.500000	3.50000	8.560000	1.000000	0.72000
75%	375.250000	325.000000	112.000000	4.000000	4.000000	4.00000	9.040000	1.000000	0.82000
max	500.000000	340.000000	120.000000	5.000000	5.000000	5.00000	9.920000	1.000000	0.97000

Scaling the data:

StandardScaler: scale features so that the algorithm/model works efficiently on the dataset

```
1 from sklearn.preprocessing import StandardScaler
2 scaler = StandardScaler()
3 x = scaler.fit_transform(x)
4
```

Get x and y . Split the data into test set and train set

Train_test_split: split data frame into test set and training set

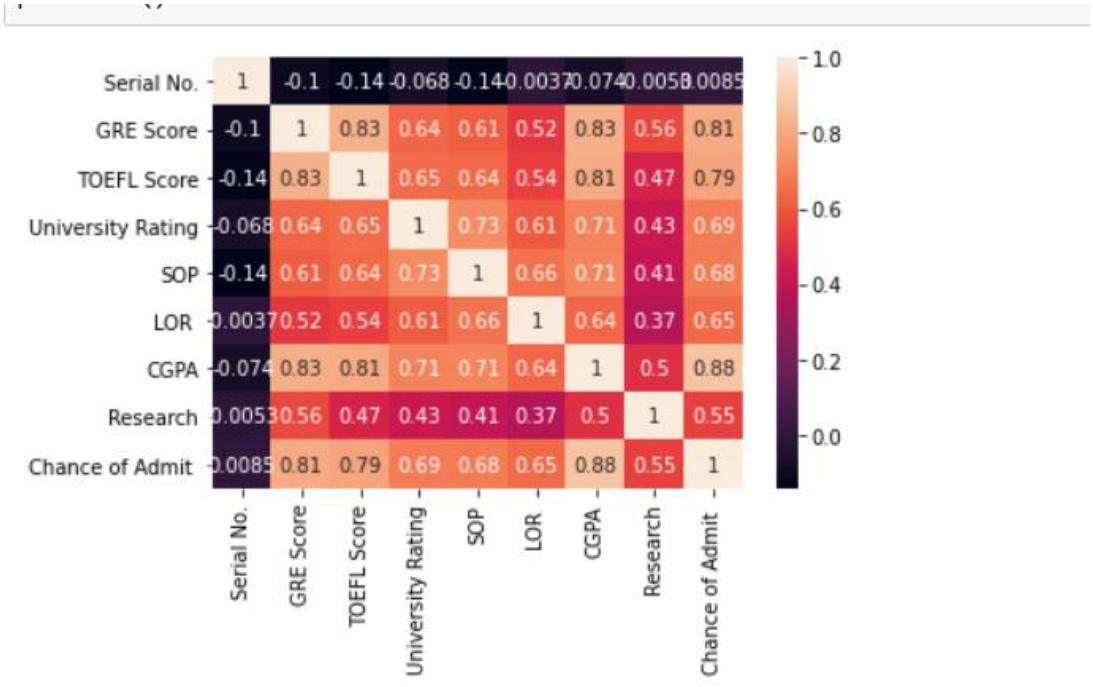


Fig2: heat map of the analysed csv data.

]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
Serial No.	1.000000	-0.103839	-0.141696	-0.067641	-0.137352	-0.003694	-0.074289	-0.005332	0.008505
GRE Score	-0.103839	1.000000	0.827200	0.635376	0.613498	0.524679	0.825878	0.563398	0.810351
TOEFL Score	-0.141696	0.827200	1.000000	0.649799	0.644410	0.541563	0.810574	0.467012	0.792228
University Rating	-0.067641	0.635376	0.649799	1.000000	0.728024	0.608651	0.705254	0.427047	0.690132
SOP	-0.137352	0.613498	0.644410	0.728024	1.000000	0.663707	0.712154	0.408116	0.684137
LOR	-0.003694	0.524679	0.541563	0.608651	0.663707	1.000000	0.637469	0.372526	0.645365
CGPA	-0.074289	0.825878	0.810574	0.705254	0.712154	0.637469	1.000000	0.501311	0.882413
Research	-0.005332	0.563398	0.467012	0.427047	0.408116	0.372526	0.501311	1.000000	0.545871
Chance of Admit	0.008505	0.810351	0.792228	0.690132	0.684137	0.645365	0.882413	0.545871	1.000000

Fig3: Finalised Data

MODELS AND ALGORITHMS USED:

1.MULTILINEAR REGRESSION:

Multiple Linear Regression is one of the important regression algorithms which models the linear relationship between a single dependent continuous variable and more than one independent variable.

For MLR, the dependent or target variable(Y) must be the continuous/real, but the predictor or independent variable may be of continuous or categorical form.

Each feature variable must model the linear relationship with the dependent variable.

MLR tries to fit a regression line through a multidimensional space of data-points.

Mathematical formulae for MLR:

Y = b0 + b1 * x1 + b2 * x2 + b3 * x3 + …… bn * xn
Y = Dependent variable and x1, x2, x3, …… xn = multiple independent variables

CODE:

```
1 from sklearn.linear_model import LinearRegression
2 mlr = LinearRegression()
3 mlr.fit(x_tr,y_tr)
```

```
LinearRegression()
```

```
1 Y_pred = mlr.predict(x_te)
2 Y_pred.shape
```

```
(165,)
```

```
from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score
mse= mean_squared_error(y_te,Y_pred)
rmse=np.sqrt(mse)
print("Mean squared error:",mse)
print("Mean absolute error:",mean_absolute_error(y_te,Y_pred))
print("Root mean squared error:",np.sqrt(mse))
print("R-Square:",r2_score(y_te,Y_pred)*100)
```

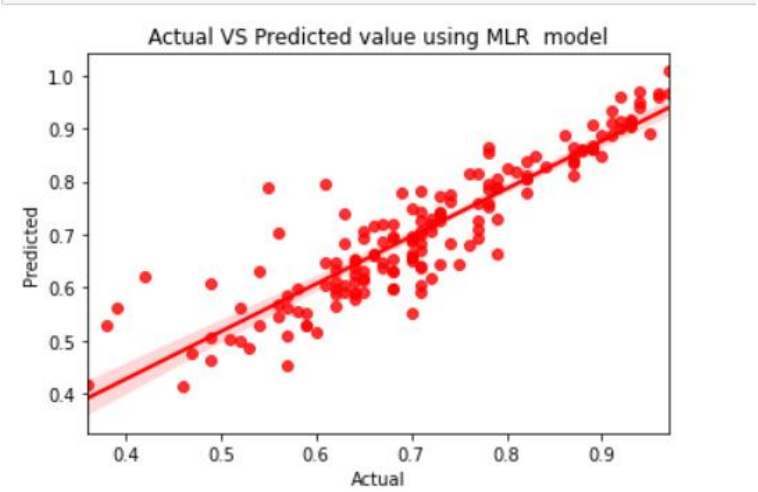


FIG4:Actual vs Predicted value using MLR model.

Inference:

Mean squared error: 0.0033427527326492425
Mean absolute error: 0.04110446284565613
Root mean squared error: 0.05781654376257061
R-Square: 80.74119498003456
Accuracy: 0.8074119498003456

2.RANDOM FOREST :

Random forest is a machine learning algorithm which is a combined effect of classification and regression and other tasks which operate by erection of decision trees at training time and outputs the class that is the mode of the classes or mean value of individual trees

The training algorithm for random forests applies the general technique of bootstrap aggregating, or bagging, to tree learners. Given a training set $X = x_1, ..., x_n$ with responses $Y = y_1, ..., y_n$, bagging repeatedly (B times) selects a random sample with replacement of the training set and fits trees to these samples:

For $b = 1, ..., B$:

1. Sample, with replacement, n training examples from X, Y; call these X_b, Y_b.
2. Train a classification or regression tree f_b on X_b, Y_b.

After training, predictions for unseen samples x' can be made by averaging the predictions from all the individual regression trees on x':

This bootstrapping procedure leads to better model performance because it decreases the variance of the model, without increasing the bias. This means that while the predictions of a single tree are highly sensitive to noise in its training set, the average of many trees is not, as long as the trees are not correlated. Simply training many trees on a single training set would give strongly correlated trees (or even the same tree many times, if the training algorithm is deterministic); bootstrap sampling is a way of de-correlating the trees by showing them different training sets.

Additionally, an estimate of the uncertainty of the prediction can be made as the standard deviation of the predictions from all the individual regression trees on x':

```
from sklearn.ensemble import RandomForestRegressor
rfr=RandomForestRegressor(n_estimators=180,max_depth=6,random_state=0)
rfr.fit(x_tr,y_tr)
y_pred=rfr.predict(x_te)
rfr
```

```
RandomForestRegressor(max_depth=6, n_estimators=180, random_state=0)
```

```
from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score
Y_pred = rfr.predict(x_te)
mse= mean_squared_error(y_te,y_pred)
rmse=np.sqrt(mse)
print("Mean squared error:",mse)
print("Mean absolute error:",mean_absolute_error(y_te,Y_pred))
print("Root mean squared error:",np.sqrt(mse))
print("R-Square:",r2_score(y_te,y_pred))
```

```
Mean squared error: 0.0037818572113340393
Mean absolute error: 0.042493314988917334
Root mean squared error: 0.061496806513298224
R-Square: 0.7993785650993054
```

```
accuracy= rfr.score(x_te,y_te)
print("Accuracy:",accuracy)
```

```
Accuracy: 0.7993785650993054
```

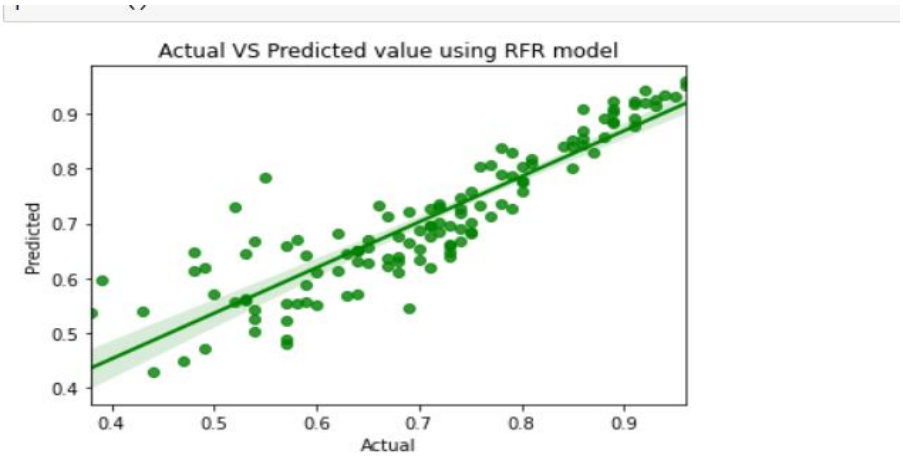


Fig5:Actual VS Predicted value using RFR model

Inference:

Mean squared error: 0.0037818572113340393
Mean absolute error: 0.042493314988917334
Root mean squared error: 0.061496806513298224
R-Square: 0.7993785650993054
Accuracy: 0.7993785650993054

3.PCA :

Principle component analysis, or colloquially known as dimensionality reduction, is a statistical procedure that uses eigenvalue decomposition to generalize the most important features in a dataset. PCA simplifies the complexity of high dimensional (many features) data while retaining trends and patterns. By simplifying the dataset into principle components, we can observe features that contribute more information to the dataset than others, speed up process time if the dimensionality reduced is significant, and visualize trends and patterns of datasets that have many features.

PCA is not ideal for non-continuous, discrete dataset attributes. The 10 continuous values out of the total 14 are the only attributes that can be used for PCA analysis. In order to obtain the most accurate PCA analysis results, the data needs to be scaled. If scaling is overlooked, then features that have higher quantitative values will influence the results far more than the other features. There are several ways to scale data, but the most common is to subtract each observation by the overall feature mean, then divide the difference by the feature standard deviation.

Libraries:

sklearn.decomposition - PCA

Mathematics:

The whole process of obtaining principle components from a raw dataset can be simplified in six parts :

- Take the whole dataset consisting of *d+1 dimensions* and ignore the labels such that our new dataset becomes *d dimensional.*
- Compute the *mean* for every dimension of the whole dataset.
- Compute the *covariance matrix* of the whole dataset. We can compute the covariance of two variables **X** and **Y** using the following formula:

$$cov(X,Y) = \frac{1}{n-1}\sum_{i=1}^{n} (Xi - \bar{x})(Yi - \bar{y})$$

Using the above formula, we can find the covariance matrix of **A.** Also, the result would be a *square matrix of d ×d dimensions.*

- Compute *eigenvectors* and the their *eigenvalues*.

  *Let **A** be a square matrix, **v** a vector and **λ** a scalar that satisfies **A**v = **λ**v, then **λ** is called eigenvalue associated with eigenvector **v** of **A**.*

  The eigenvalues of ***A*** are roots of the characteristic equation

$$\det(A-\lambda I) = 0$$

- Sort the eigenvectors by decreasing eigenvalues and choose k eigenvectors with the largest eigenvalues to form a ***d* × *k dimensional*** matrix W.

  Transform the samples onto the new subspace.

Principal Component Analysis (PCA) is an unsupervised, non-parametric

In [12]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=x.shape[1])
x= pca.fit_transform(x)
ratios = pca.explained_variance_ratio_
```

In [ ]:
```
pca = PCA(n_components=7)
x = pca.fit_transform(x)
```

In [16]:
```
explained_variance = pca.explained_variance_ratio_
print(explained_variance)
print(pca.singular_values_)
```

```
[0.67519343 0.10596446 0.08023255 0.0543379  0.03766808 0.02546844
 0.02113513]
[48.61251905 19.25813119 16.75750346 13.79067208 11.4820858   9.44137474
  8.60075376]
```

statistical technique primarily used for dimensionality reduction in machine learning.

MLR with PCA model:

CODE:

```
from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score
Y_pred = mlr.predict(x_te)
mse= mean_squared_error(y_te,Y_pred)
rmse=np.sqrt(mse)
print("Mean squared error:",mse)
print("Mean absolute error:",mean_absolute_error(y_te,Y_pred))
print("Root mean squared error:",np.sqrt(mse))
print("R-Square:",r2_score(y_te,Y_pred)*100)
```

```
Mean squared error: 0.0033427527326492473
Mean absolute error: 0.041104462845656255
Root mean squared error: 0.05781654376257065
R-Square: 80.74119498003452
```

```
accuracy= mlr.score(x_te,y_te)
print("Accuracy:",accuracy)
```

```
Accuracy: 0.8074119498003453
```

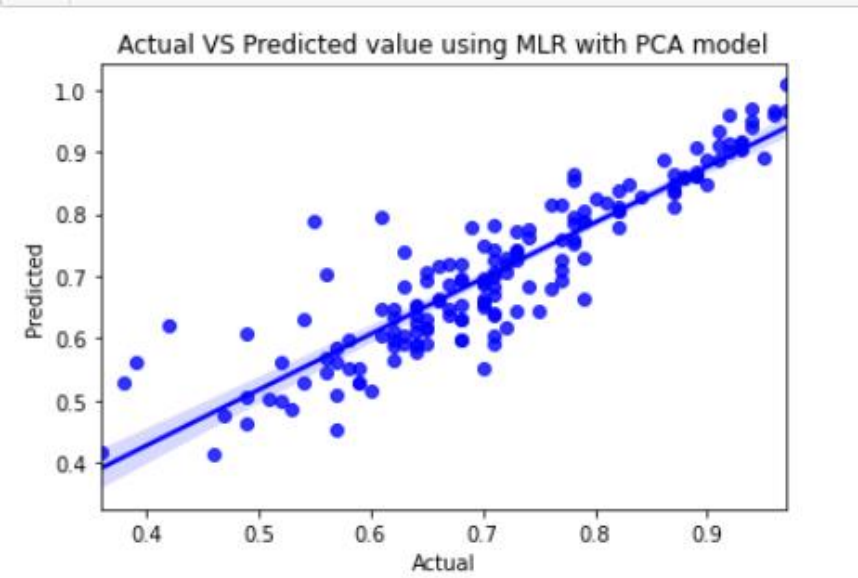


Fig6:Actual vs Predicted value using MLR with PCA model.

Inference:

Mean squared error: 0.0033427527326492473
Mean absolute error: 0.041104462845656255
Root mean squared error: 0.05781654376257065
R-Square: 80.74119498003452
Accuracy: 0.8074119498003453

RFR with PCA model:

CODE:

```
from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score
Y_pred = rfr.predict(x_te)
mse= mean_squared_error(y_te,y_pred)
rmse=np.sqrt(mse)
print("Mean squared error:",mse)
print("Mean absolute error:",mean_absolute_error(y_te,Y_pred))
print("Root mean squared error:",np.sqrt(mse))
print("R-Square:",r2_score(y_te,y_pred))
```

```
Mean squared error: 0.0037242510841204515
Mean absolute error: 0.04370242678304667
Root mean squared error: 0.06102664241231408
R-Square: 0.7854324527987849
```

```
accuracy= rfr.score(x_te,y_te)
print("Accuracy:",accuracy)
```

```
Accuracy: 0.7854324527987849
```

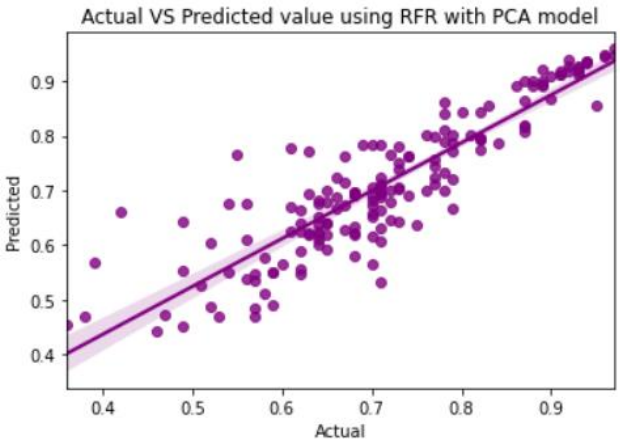


Fig7:Actual vs Predicted value using RFR with PCA model.

Inference:

Mean squared error: 0.0037242510841204515
Mean absolute error: 0.04370242678304667
Root mean squared error: 0.06102664241231408
R-Square: 0.7854324527987849
Accuracy: 0.7854324527987849

CONCLUSION:

The project involved analysis of the admission prediction dataset with proper data processing. Then, 4 models were trained and tested with maximum scores as follows:

Multiple Linear Regression: 80.74119498003456%

Random Forest Regression: 79.93785650993054%

Multiple Linear Regression with PCA: 80.74119498003453%

Random Forest Regression with PCA: 78.54324527987849%

MODELS	Mean squared error	Root mean squared error	R-Square:
Multiple Linear Regression	0.0033427527326492425	0.05781654376257061	0.8074119498003456
Random Forest Regression	0.0037818572113340393	0.061496806513298224	0.7993785650993054
Multiple Linear Regression with PCA	0.0033427527326492473	0.041104462845656255	0.8074119498003453
Random Forest Regression with PCA	0.0037242510841204515	0.06102664241231408	0.7854324527987849