2019

Facilitator: Uzair Ahmad

Group 10
Shrita Gaonkar
100699307

## Table of Content

## Executive Summary

Most students after graduation are open to innumerable job roles. Some of the students are good at analysis while some are good at development. Being open to a bunch of job roles might be confusing as it is difficult to analyze the best fit for yourself and match the interest. The goal of my model would be to find the best job role for students.

On the other hand, my model can also predict student grades and the chances of getting an admit from their desired university

Three major project goal

1. Identify the best job role for students
2. Predict student grade based on tests like math, verbal abilities at age 10
3. Find out whether the student is eligible to apply at a particular university

## Introduction

When it comes to applying for their first job, every newbie in the IT industry, including myself, is puzzled. There are many opportunities in data technology, such as data science engineer or data scientist, and data analyst for freshers.
This can be very difficult to read to align the criteria of the business with your work title. My model makes this method simpler for the organisation and the students. By taking the requirements from the company and some extra information like work location with student's information. After comparing both of them, the model will suggest the best titles for the students

Admission prediction model is simple. It just takes GRE and TOEFL scores along with some other test results and university rating to predict the probability of the student getting the admission from their desired university or college.

Grade prediction takes various test results taken at age 10 and some other variables to predict the grades In English and Math.

## Rationale Statement

Identifying the best job role based on analytical and spatial reasoning. Predicting whether a student gets an admit from their desired learning institution based on their expectation and university requirements. Predicting student grades using many parameters.

## Problem

1. To match the job titles with company requirements can help students to select the job titles correctly.
2. Regular human tendency can be said to be retrieve and understand more information when provided visually rather than just figures. Hence, in my scenario, I am planning to do the same. Looking at the prior stated problems and simplifying them, I can possibly devise a solution to visualise the whole dataset of listed properties and derive to a conclusion where this problem is persistent.
3. The analysis of data set using specific classification and regression algorithm which can provide the information. The regression algorithm will be utilised to predict the grades and admission percentage.

   .

## Data Requirements

1. The first data requirement will be to find out the key variables which define the quality of the data.
2. For developing various types of models, the data should be large and diverse.
3. The data should be in .csv or in excel format for better extraction.
4. The data size must be feasible according to the available machine.
5. The dataset is required to not have special characters in them so the data can be cleaned easily.

## Data

- The first data requirement will be to find out the key variables which define the quality of the data.
- For developing various types of models, the data should be large and diverse.
- The data should be in .csv or in excel format for better extraction.
- The data size must be feasible according to the available machine.
- The dataset is required to not have special characters in them so the data can be cleaned easily.

There are three datasets that I will be using which are downloaded from Kaggle
student-mat.csv

- This dataset has all the features needed to predict student grades
- Most of the columns have categorical values.
- There are 33 columns

Data_job_posts.csv

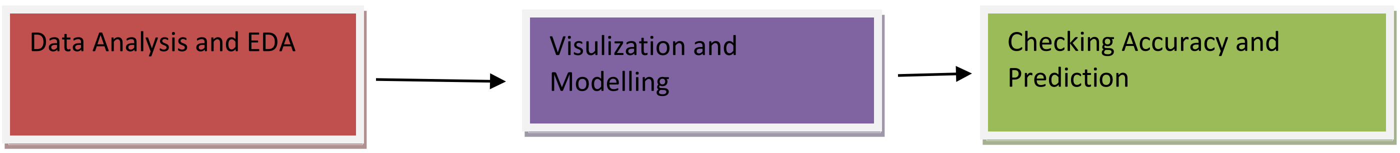
- There are 24 columns of dataset.
- There are 6 columns which have maximum null values.
- Title is the dependent variable

Admission_predict.csv

- The dataset contains GRE, LOR, SOP, IELTS, CGPA, etc.
- These are the independent variables
- The column 'chances of admit' is the dependent variables.

## Model/Architecture Approach

The complete methodology for this project is divided into 3 segments where each has its own set of activities and necessary outcomes.




Even though if the approach and algorithms selected to reach the required output changes, the methodology and flow will remain the same.

## Algorithms Applied:

Our project was first halted for a long time when regular approach was utilised. The regular approach included Loading of data, processing it and after splitting the datapoints, was fed into the machine learning model.

The trained model performed so poorly that only 8% accuracy was obtained which is very less. It can be seen below in the code and plot.

This was a major milestone which took long to cover up. A few different techniques were utilised to improve accuracy such as dropping of features, processing using various algorithms.

For this, first we created a function for each algorithm and then, the same can be called anywhere to implement.

Evaluated algorithms are listed below:

1. Linear Regression
2. SVR
3. Decision Tree Regression
4. Random Forest Classifier

## Exploratory Data Analysis (Student Grade Prediction)

Prior to implementation of any of these models, a few steps are necessary to perform to extrapolate information out of the dataset.

Hence, firstly we are performing the Explanatory Data

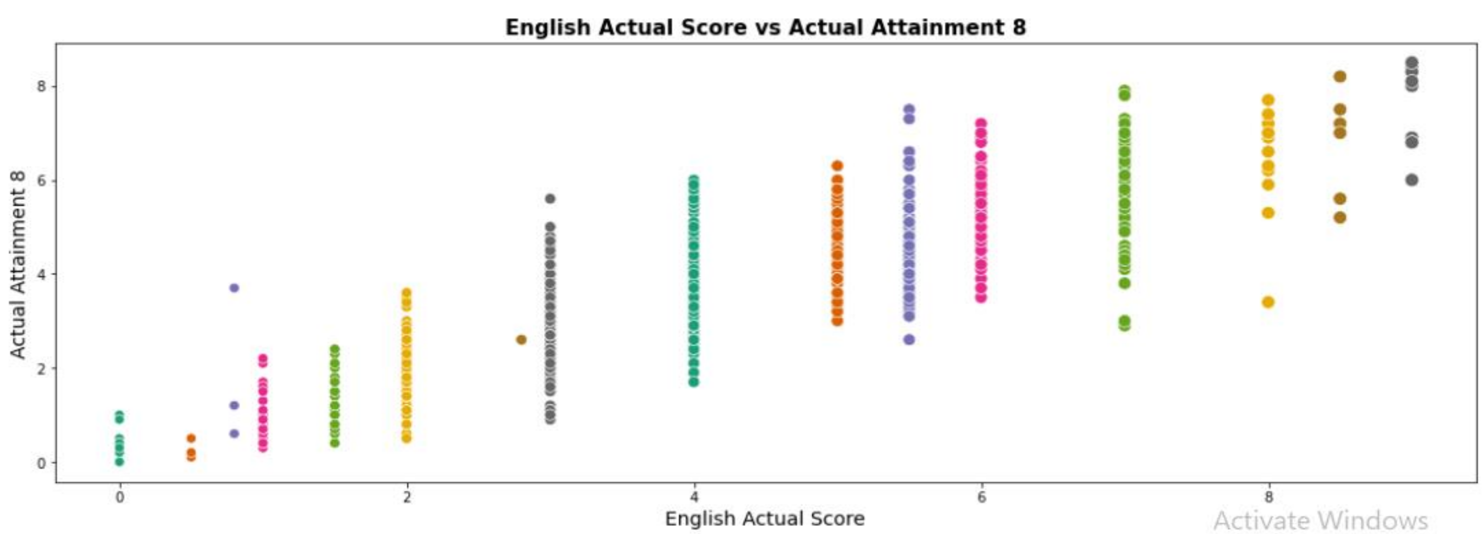
Analysis on the dataset. Loading the dataset using Pandas

For figuring out better relationships between features, we can

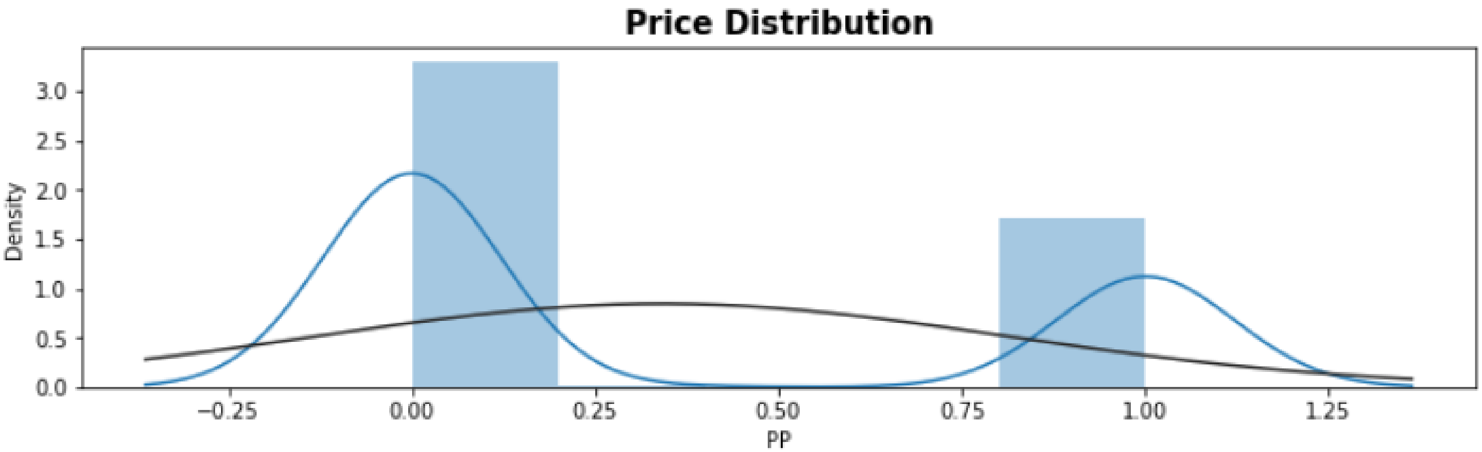
compare them using scatter plot graphs as follows:

1. Math Scores vs Actual Attainment 8

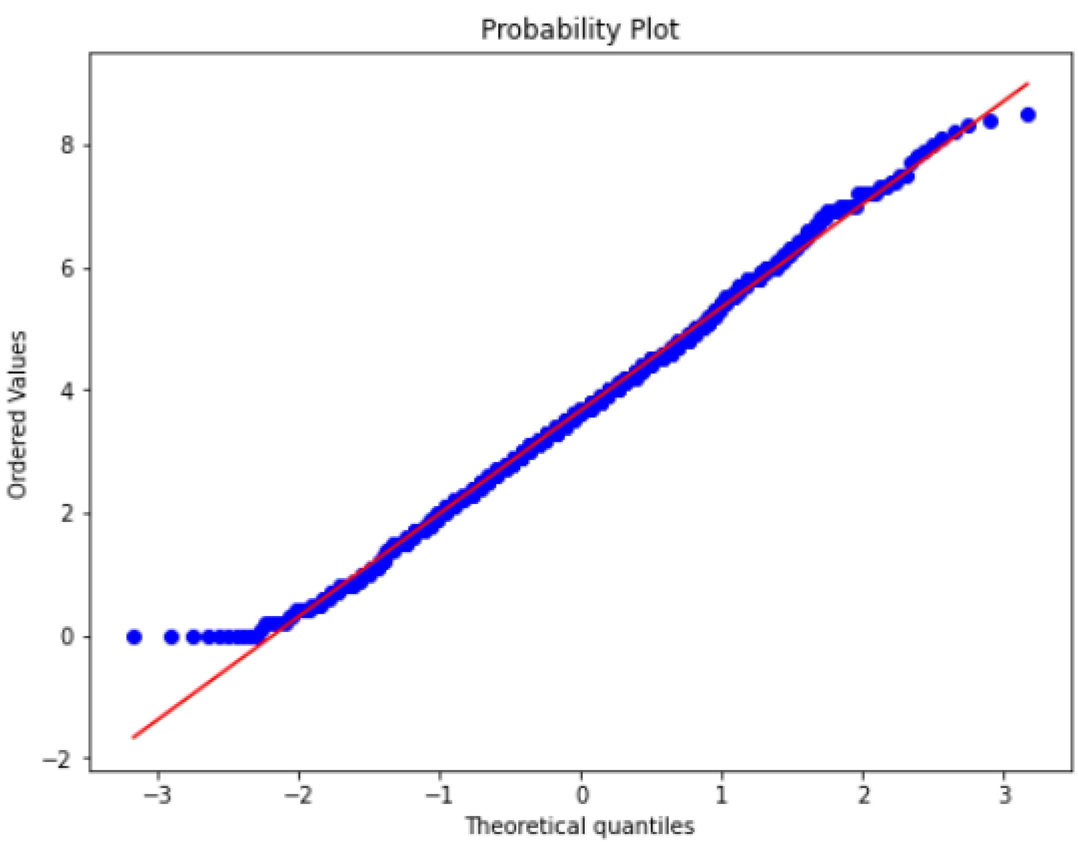

2. English Actual Score vs Actual Attainment 8



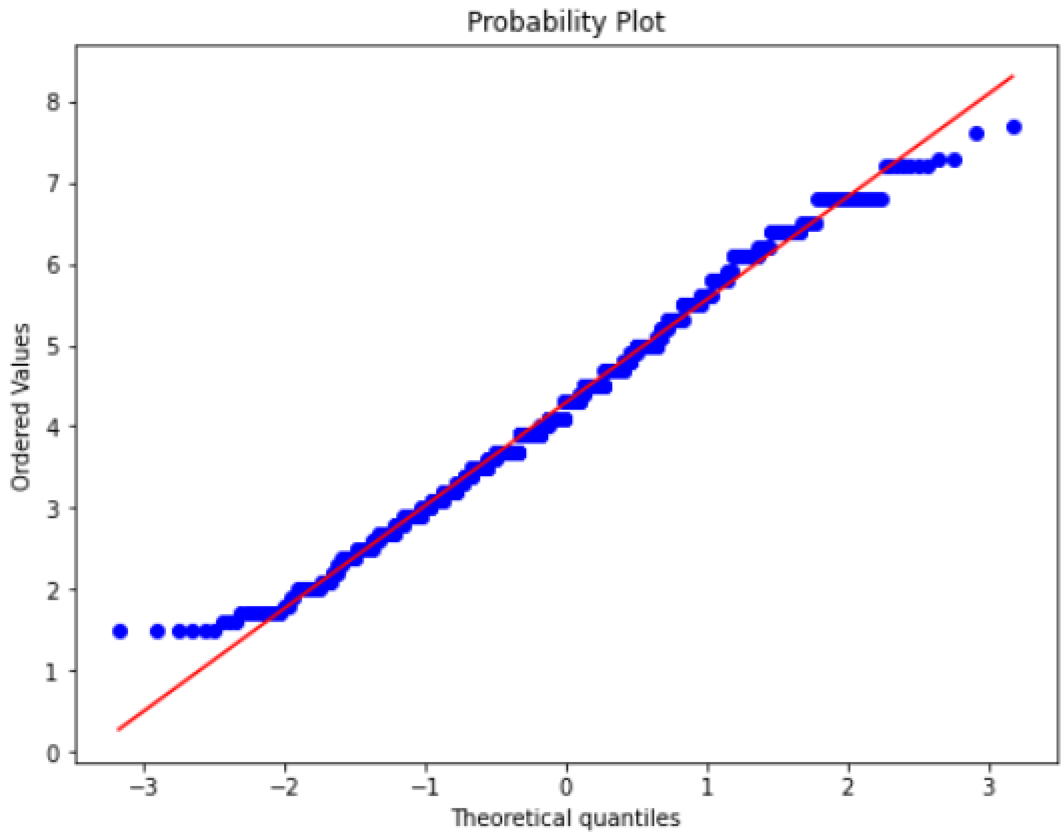
3. HML Distribution



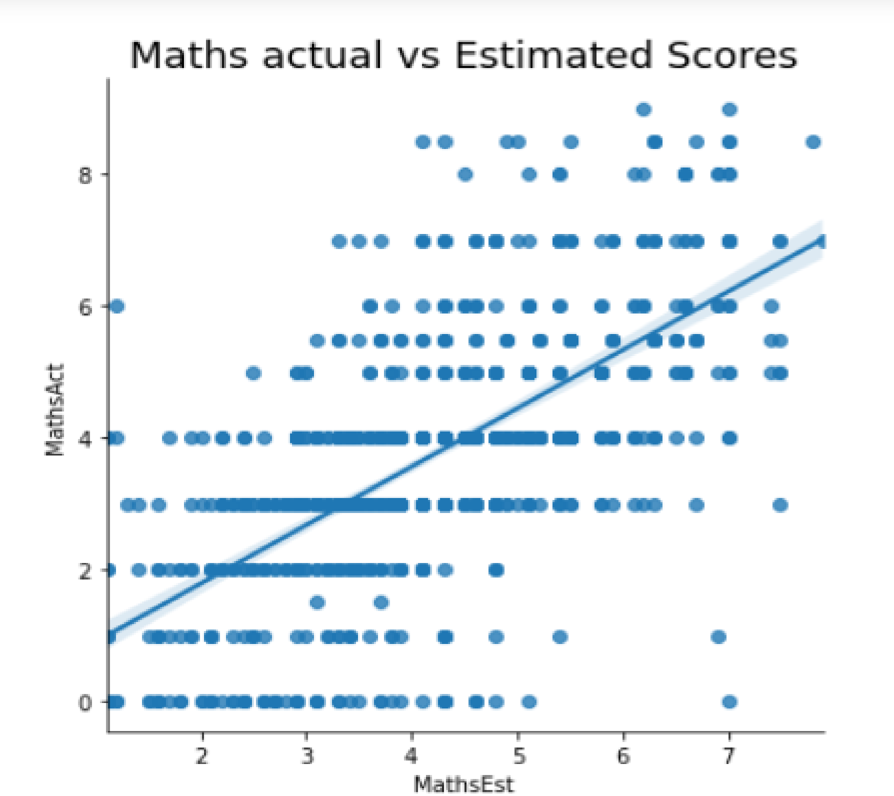

4. Attainment 8 Actual Distribution



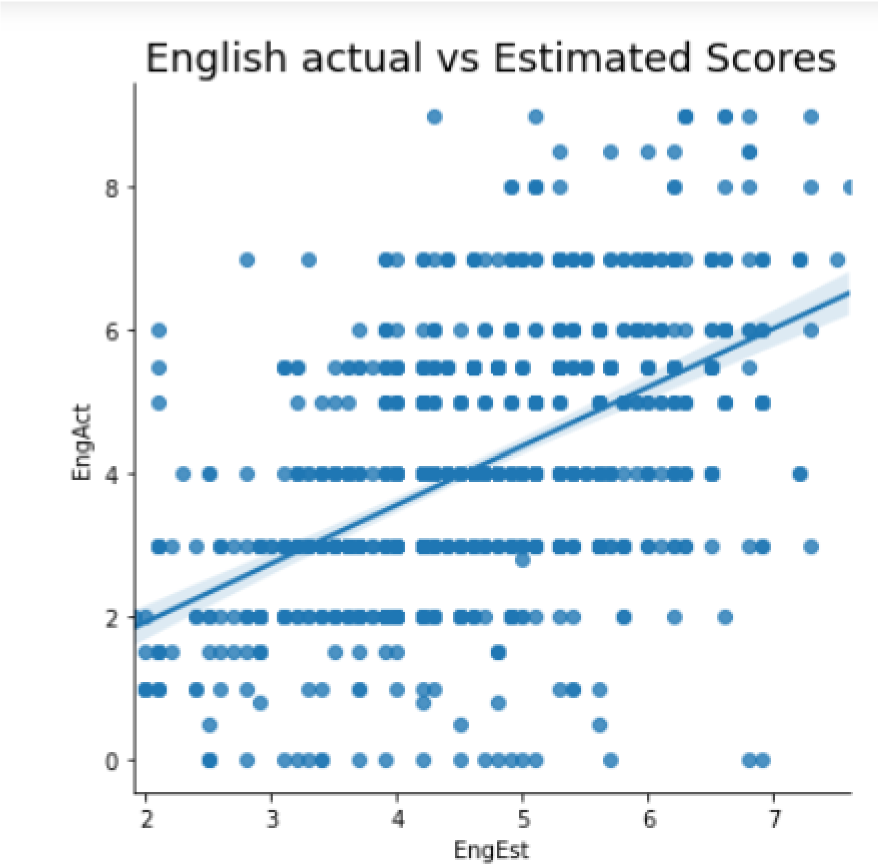

5. Attainment 8 Estimate Distribution


Probability Plot
Ordered Values
Theoretical quantiles
0
1
2
3
4
5
6
7
8
−3
−2
−1
0
1
2
3

6. Math actual vs Estimated Scores

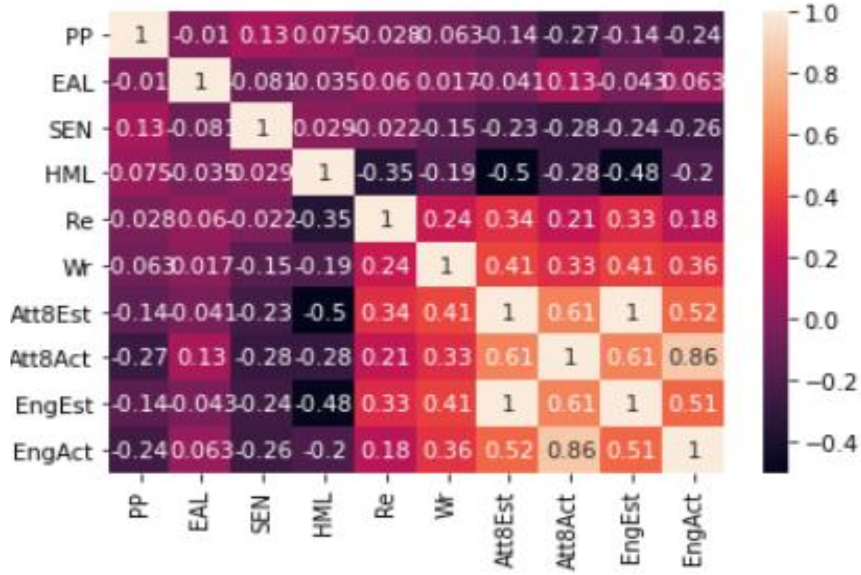


7. English Actual vs Estimate Scores

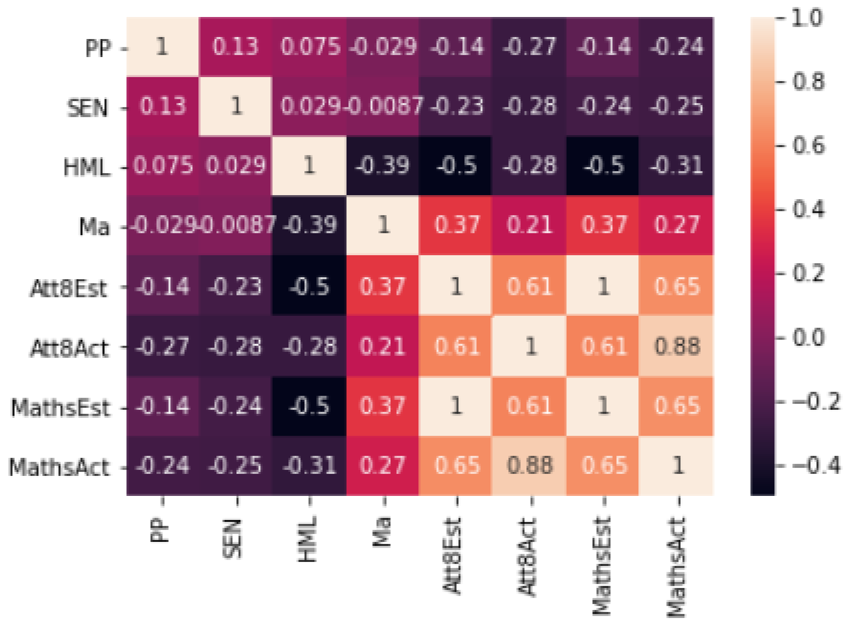


Now, to determine the correlation between all the existing features, a correlation matrix is displayed.

8. Heatmap for English Scores




9. Heatmap for Math Scores



## Exploratory Data Analysis (Student Admission Prediction)

Prior to implementation of any of these models, a few steps are necessary to perform to extrapolate information out of the dataset.

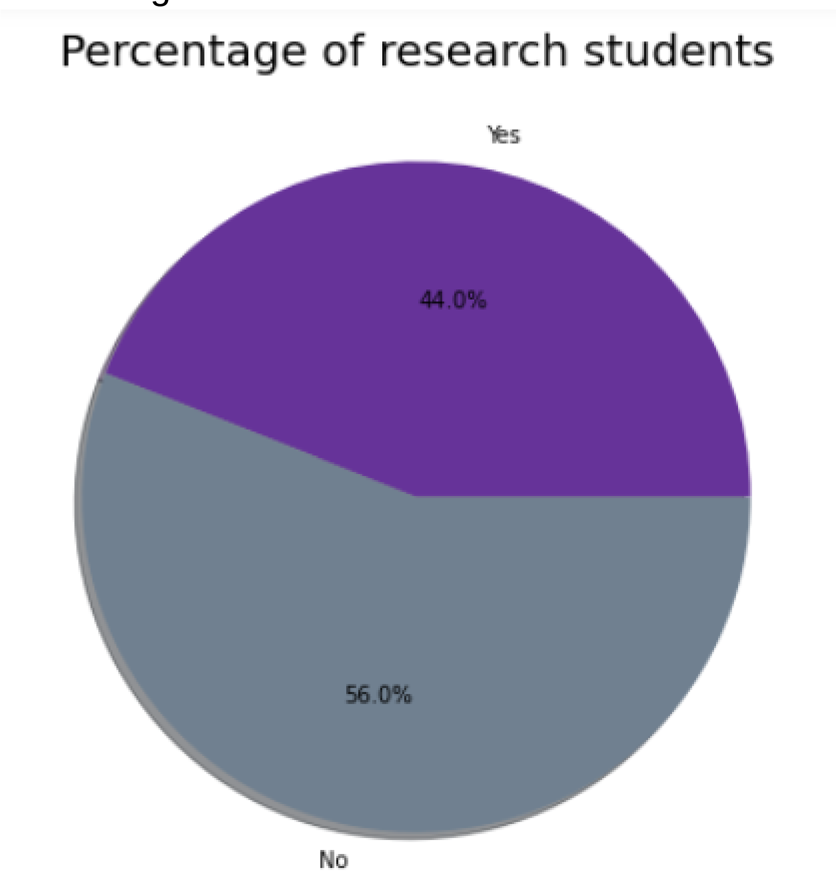
Hence, firstly we are performing the Explanatory Data

Analysis on the dataset. Loading the dataset using Pandas

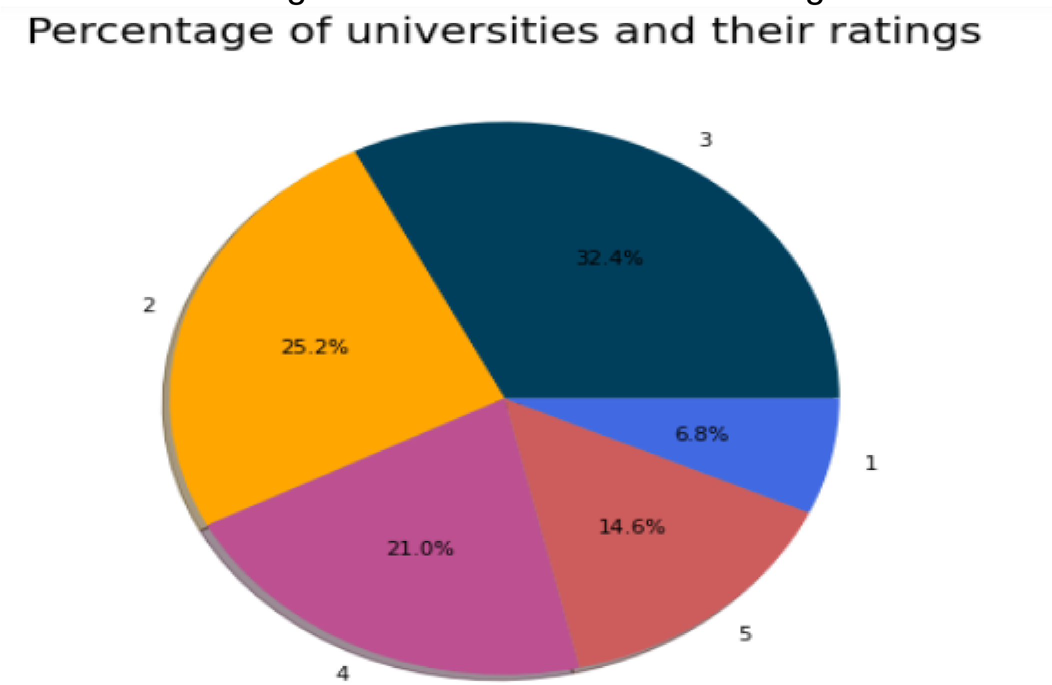
For figuring out better relationships between features, we can

compare them using scatter plot graphs as follows:

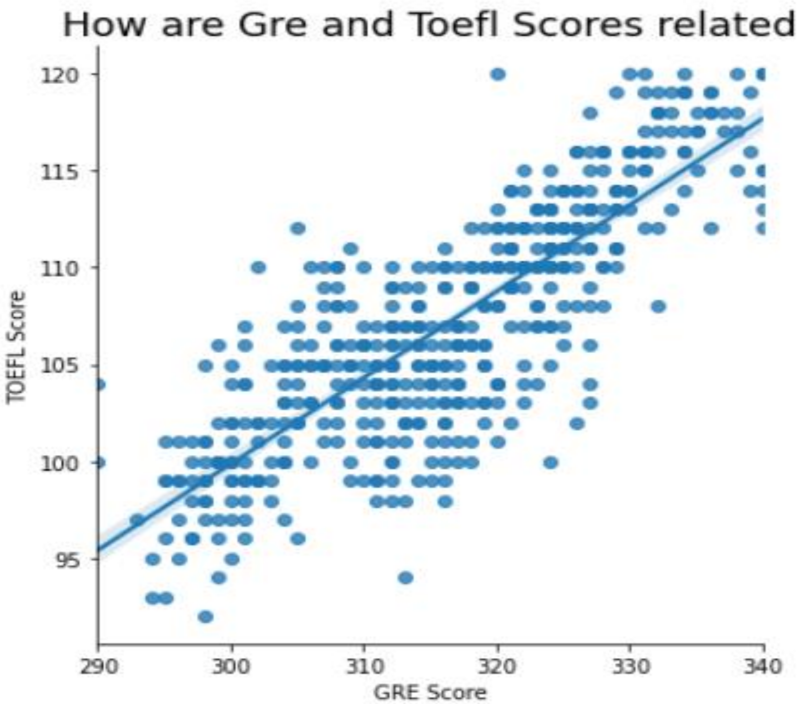
1. Percentage of Research students



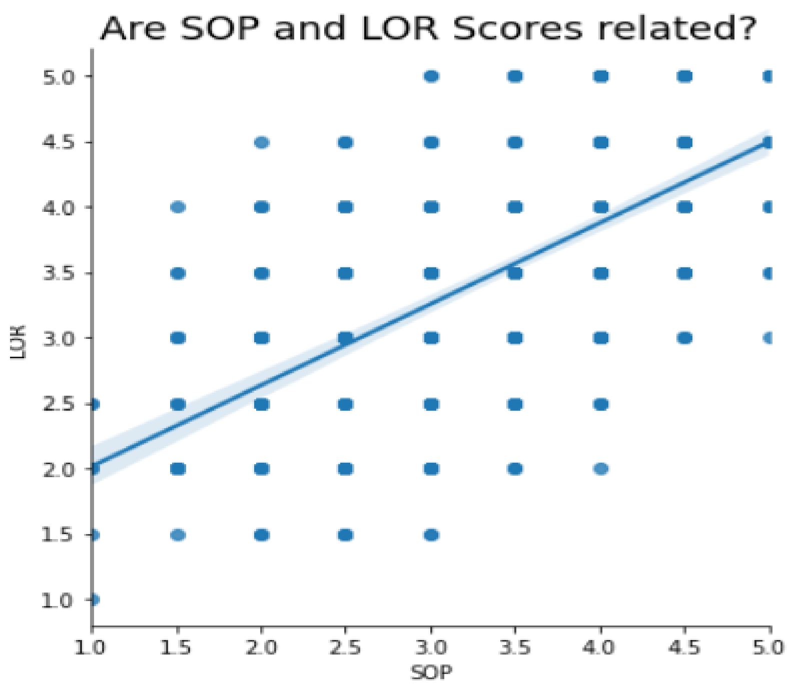
2. Percentage of universities and their ratings



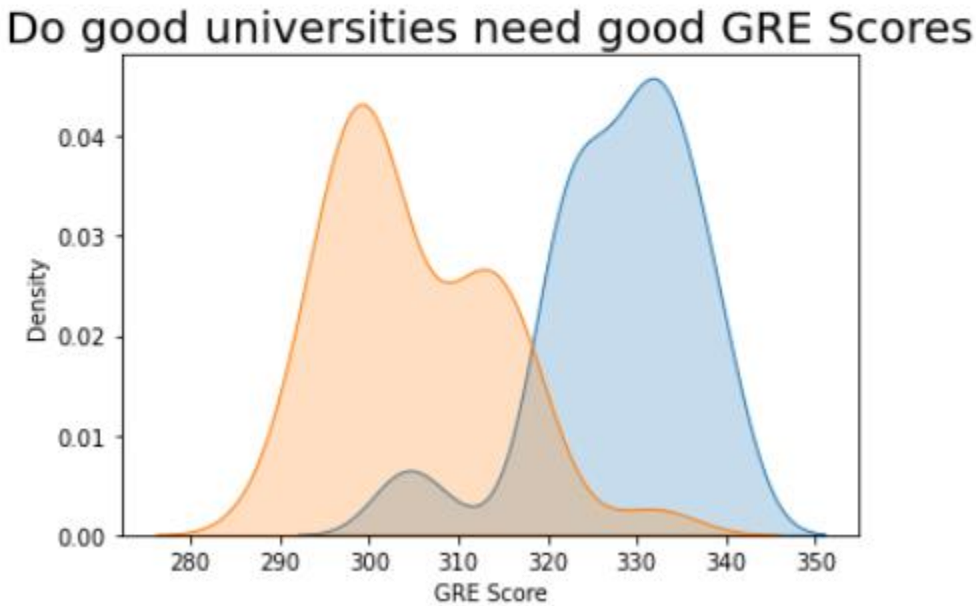
3. How are Gre and Toefl Scores related



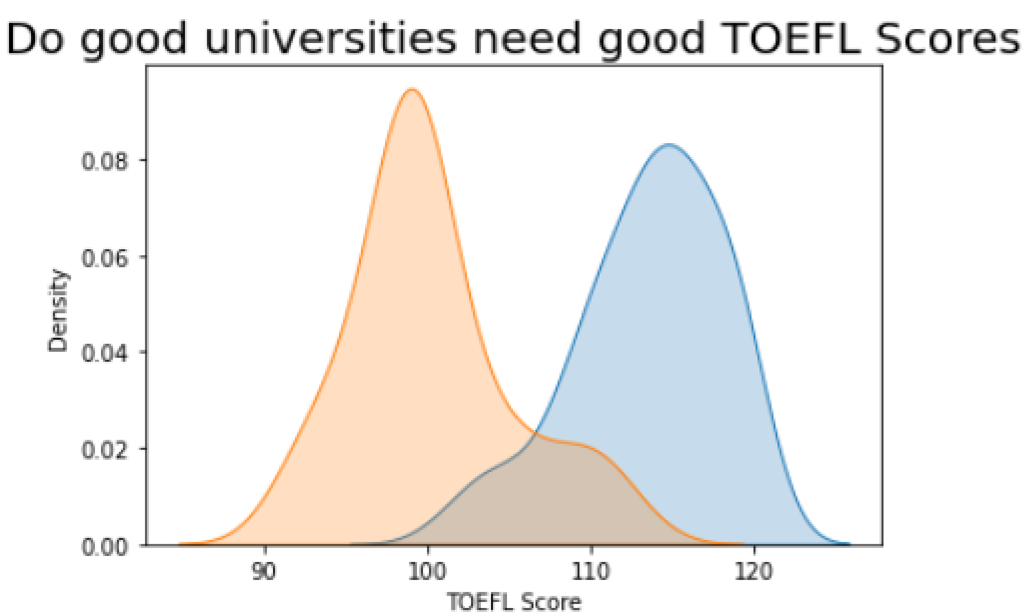

4. Are SOP and LOR Scores related?



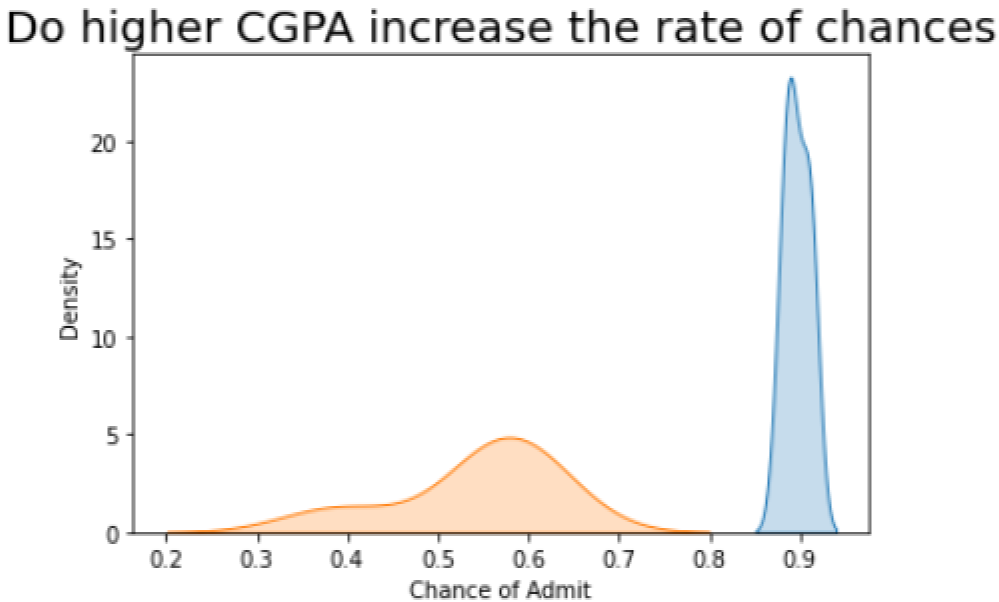
5. Do good universities need good GRE Scores



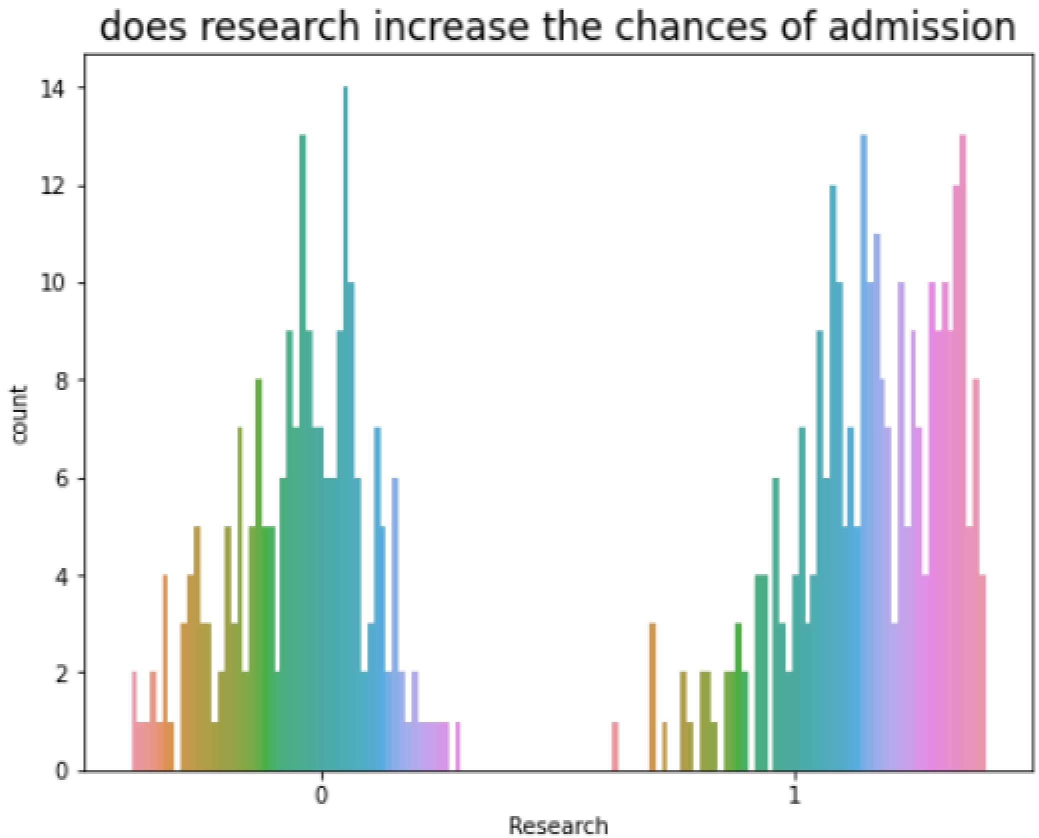

6. Do good universities need good TOEFL Scores



7. Do higher CGPA increase the rate of chances

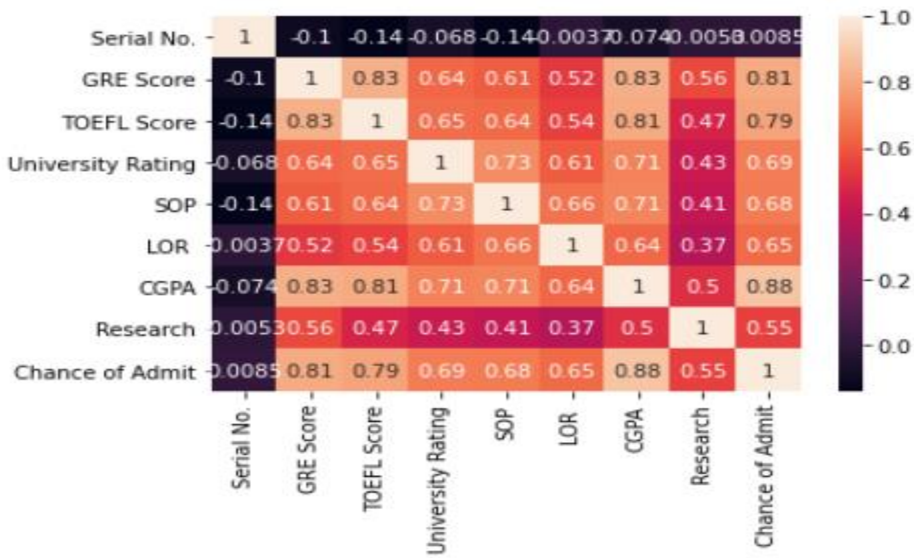


8. Does research increase the chances of admission



Now, to determine the correlation between all the existing features, a correlation matrix is displayed.

9. Heatmap for Admission



## Data Preprocessing Pipeline (Student Grade Prediction)

### Splitting Math and English Datasets

```
df.shape
```

```
(915, 20)
```

```
df1 = df.iloc[:,[1,3,4,7,8,9,14,15]]
```

```
df1.shape
```

```
(915, 8)
```

```
df2 = df.iloc[:,[1,2,3,4,5,6,8,9,11,12]]
```

```
df2.shape
```

```
(915, 10)
```

```
df1.head(3)
```

| | PP | SEN | HML | Ma | Att8Est | Att8Act | MathsEst | MathsAct |
|---|---|---|---|---|---|---|---|---|
| **4** | 0.0 | 0.0 | 1 | 14 | 1.7 | 2.9 | 1.3 | 3.0 |
| **5** | 0.0 | 0.0 | 0 | 9 | 5.8 | 6.6 | 5.9 | 7.0 |
| **8** | 0.0 | 0.0 | 2 | 8 | 4.9 | 4.8 | 4.8 | 7.0 |

```
df2.head(3)
```

| | PP | EAL | SEN | HML | Re | Wr | Att8Est | Att8Act | EngEst | EngAct |
|---|---|---|---|---|---|---|---|---|---|---|
| **4** | 0.0 | 0 | 0.0 | 1 | 2 | 3 | 1.7 | 2.9 | 2.2 | 3.0 |
| **5** | 0.0 | 0 | 0.0 | 0 | 7 | 12 | 5.8 | 6.6 | 6.0 | 7.0 |
| **8** | 0.0 | 0 | 0.0 | 2 | 4 | 2 | 4.9 | 4.8 | 5.3 | 5.5 |

The dataset is normalized using the StandardScalar() and split into 70% for training of models and 30% for testing of models.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 0)
```

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

## Label Encoding

```
from sklearn import preprocessing

categorical = ['HML','Ma']
for feature in categorical:
        le = preprocessing.LabelEncoder()
        df1[feature] = le.fit_transform(df1[feature])
```

```
C:\Users\Shrita\Anaconda3\lib\site-packages\ipykernel_launcher.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
rsus-a-copy
```

```
from sklearn import preprocessing

categorical = ['HML','Re','Wr']
for feature in categorical:
        le = preprocessing.LabelEncoder()
        df2[feature] = le.fit_transform(df2[feature])
```



## Data Preprocessing Pipeline (Student Admission Prediction)
## Dropping unnecessary columns

```
df1.columns
df1 = df1.drop('Serial No.',axis = 1)
```

```
X10 = df1.iloc[:,:7].values
y10 = df1['Chance of Admit'].values
X10
```

```
array([[337.  , 118.  ,   4.  , ...,   4.5 ,   9.65,   1.  ],
       [324.  , 107.  ,   4.  , ...,   4.5 ,   8.87,   1.  ],
       [316.  , 104.  ,   3.  , ...,   3.5 ,   8.  ,   1.  ],
       ...,
       [330.  , 120.  ,   5.  , ...,   5.  ,   9.56,   1.  ],
       [312.  , 103.  ,   4.  , ...,   5.  ,   8.43,   0.  ],
       [327.  , 113.  ,   4.  , ...,   4.5 ,   9.04,   0.  ]])
```

The dataset is normalized using the StandardScalar() and split into 70% for training of models and 30% for testing of models.

```
from sklearn.model_selection import train_test_split
X_train10, X_test10, y_train10, y_test10 = train_test_split(X10, y10, test_size = 0.2 , random_state = 50)
```

No Label encoding needed as all the columns are integer types

## Data Preprocessing Pipeline (Job Recommendation):

```
df1 = df1[df1['IT'] == True]
```

```
df1
```

```
df1
```

| Term | Eligibility | Audience | StartDate | Duration | ... | Salary | ApplicationP | OpeningDate | Deadline | Notes | AboutC | Attach | Year | Month | IT |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| NaN | NaN | NaN | NaN | NaN | ... | NaN | Successful candidates should submit\r\n-CV; \... | NaN | 20 January 2004, 18:00 | NaN | NaN | NaN | 2004 | 1 | True |
| NaN | NaN | NaN | NaN | NaN | ... | NaN | Successful candidates should submit CV and 1-2... | NaN | 28 February 2004, 18:00 | NaN | NaN | NaN | 2004 | 1 | True |
| NaN | NaN | NaN | NaN | NaN | ... | NaN | Interested applicants should send CVs by email... | NaN | 26 January 2004 | NaN | NaN | NaN | 2004 | 1 | True |
| NaN | NaN | NaN | NaN | 3 week | ... | NaN | If you are interested in this course and feel\... | NaN | 30 January 2004 | NaN | NaN | NaN | 2004 | 1 | True |

```
df1.shape
```

```
(3759, 24)
```

```
col = ['RequiredQual', 'Eligibility', 'Title', 'JobDescription', 'JobRequirment']
df1 = df1[col]
```

```
df1
```

| | RequiredQual | Eligibility | Title | JobDescription | JobRequirment |
|---|---|---|---|---|---|
| 4 | - University degree; economical background is ... | NaN | Software Developer | NaN | - Rendering technical assistance to Database M... |
| 15 | - Excellent knowledge of Windows 2000 Server, ... | NaN | Network Administrator | NaN | - Network monitoring and administration;\r\n- ... |
| 19 | As a GD you are creative, innovative and | NaN | Graphic Designer | The position of Graphic Designer (GD) | Graphic Designer will be responsible for |

```
df1['Title'].value_counts()
```

```
Software Developer                          134
Web Developer                               101
Java Developer                               88
Graphic Designer                             75
Software Engineer                            69
                                           ...
C#/ .NET Developer                            1
Junior C++ Developer                          1
Senior Software Engineer, I                   1
ASP Developer                                 1
Project Manager, Software Development         1
Name: Title, Length: 1272, dtype: int64
```

```
df1['Title'].value_counts().head(30)
```

```
def repl(title):
    tokens = title.split()
    for i,x in enumerate(tokens):
        if x == 'Senior':
            tokens = tokens[1:]
        elif x == 'Junior':
            tokens = tokens[1:]

    title2 = ' '.join(tokens)
    return title2

df1['Title'] = df1['Title'].apply(lambda x: repl(x))
```

```python
df1['Title'].value_counts()
```

```
Software Developer                                         181
Java Developer                                             165
Software Engineer                                          141
Web Developer                                              132
PHP Developer                                              103
                                                          ...
IT Network Administrator, Administration Unit, Network       1
Technical Specialist in Gyumri                               1
Web-Master/ IT Specialist                                    1
WPF Developers                                               1
Project Manager, Software Development                        1
Name: Title, Length: 1126, dtype: int64
```

```python
def repl(s):
    if 'C++ Software Developer' in s:
        s = s.replace('C++ Software Developer', 'C++ Developer')
    elif ('Quality Assurance Engineer' in s) or ('Software QA Engineer' in s):
        s = s.replace('Quality Assurance Engineer', 'QA Engineer')
        s = s.replace('Software QA Engineer', 'QA Engineer')
    elif '.Net Developer' in s:
        s = s.replace('.Net Developer', '.NET Developer')
    elif 'Java Software Developer' in s:
        s = s.replace('Java Software Developer', 'Java Developer')
    elif 'Senior Software Developer' in s:
        s = s.replace('Senior Software Developer', 'Software Developer')
    elif 'Database Administrator' in s:
        s = s.replace('Database Administrator', 'Database Developer')
    elif 'PHP Software Developer' in s:
        s = s.replace('PHP Software Developer', 'PHP Developer')
    elif '.NET Software Developer' in s:
        s = s.replace('.NET Software Developer', '.NET Developer')
    elif 'Java Software Engineer' in s:
        s = s.replace('Java Software Engineer', 'Java Developer')
    elif ('C#.NET Developer' in s) or ('C# .NET Developer' in s):
        s = s.replace('C#.NET Developer', '.NET Developer')
        s = s.replace('C# .NET Developer', '.NET Developer')
    elif 'ASP.NET Developer' in s:
        s = s.replace('ASP.NET Developer', 'Java Developer')


    else:
        pass

    return s


df1['Title'] = df1['Title'].apply(lambda x: repl(x))
```

```python
df1['Title'].value_counts()
```

```
Java Developer                             239
Software Developer                         214
Software Engineer                          141
Web Developer                              132
.NET Developer                             122
                                          ...
Software Developer for Unix (Intern)         1
Search Engine Optimization Specialist        1
Delphi Software Developer                    1
C#. Net Developer                            1
Project Manager, Software Development        1
Name: Title, Length: 1103, dtype: int64
```

```
classes = df1['Title'].value_counts()[:14]
keys = classes.keys().to_list()

df1 = df1[df1['Title'].isin(keys)]
df1['Title'].value_counts()
```

```
Java Developer        239
Software Developer    214
Software Engineer     141
Web Developer         132
.NET Developer        122
PHP Developer         116
QA Engineer           107
Graphic Designer       78
Database Developer     69
Android Developer      60
Programmer             58
IT Specialist          56
iOS Developer          54
C++ Developer          41
Name: Title, dtype: int64
```

```
from sklearn.feature_extraction import DictVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory
import nltk
from nltk.corpus import stopwords
from nltk import word_tokenize
from nltk.stem import WordNetLemmatizer
class LemmaTokenizer(object):
    def __init__(self):
        # lemmatize text - convert to base form
        self.wnl = WordNetLemmatizer()
        # creating stopwords list, to ignore lemmatizing stopwords
        self.stopwords = stopwords.words('english')
    def __call__(self, doc):
        return [self.wnl.lemmatize(t) for t in word_tokenize(doc) if t not in self.stopwords]

# removing new line characters, and certain hypen patterns
df1['RequiredQual']=df1['RequiredQual'].apply(lambda x: x.replace('\n', ' ').replace('\r', '').replace('- ', ''). replace(' - ',
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import LabelEncoder
# train features and labels
y = df1['Title']
X = df1['RequiredQual']
# tdif feature rep
vectorizer = TfidfVectorizer(tokenizer=LemmaTokenizer(), stop_words='english')
vectorizer.fit(X)
# transoforming text to tdif features
tfidf_matrix = vectorizer.transform(X)
# sparse matrix to dense matrix for training
X_tdif = tfidf_matrix.toarray()
# encoding text labels in categories
enc = LabelEncoder()
enc.fit(y.values)
y_enc=enc.transform(y.values)

X_train_words, X_test_words, y_train, y_test = train_test_split(X, y_enc, test_size=0.15, random_state=10)

X_train = vectorizer.transform(X_train_words)
X_train = X_train.toarray()

X_test = vectorizer.transform(X_test_words)
X_test = X_test.toarray()
```

## Algorithm Evaluation (Student Grade Prediction)

### 1. Linear Regression:

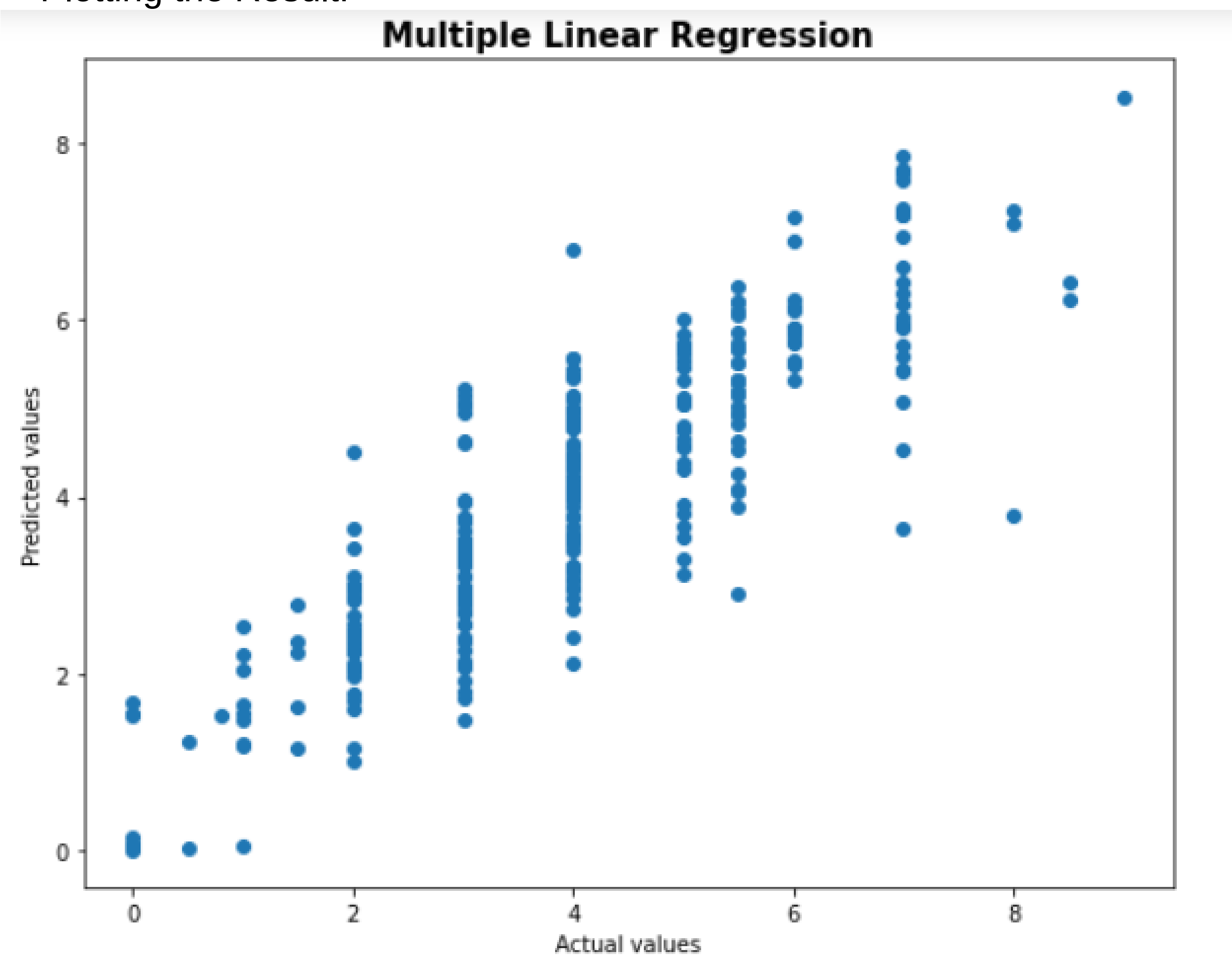
```
y_pred = regressor.predict(X_test)
```

```
```

```
from sklearn.metrics import r2_score
r2_score(y_test, y_pred)
```

```
0.8061568538868469
```

Plotting the Result:



Benefits:

- Works well in multiple data
- Easy to learn and implement with

simplified Data Disadvantages:

- Over-simplifies real world data
- Usually, co-variates and response variables don't exhibit a linear relationship
- Not ideal for real world applications

## 2. SVR

```
from sklearn.svm import SVR
regressor = SVR(kernel = 'rbf')
regressor.fit(X_train, y_train)
```

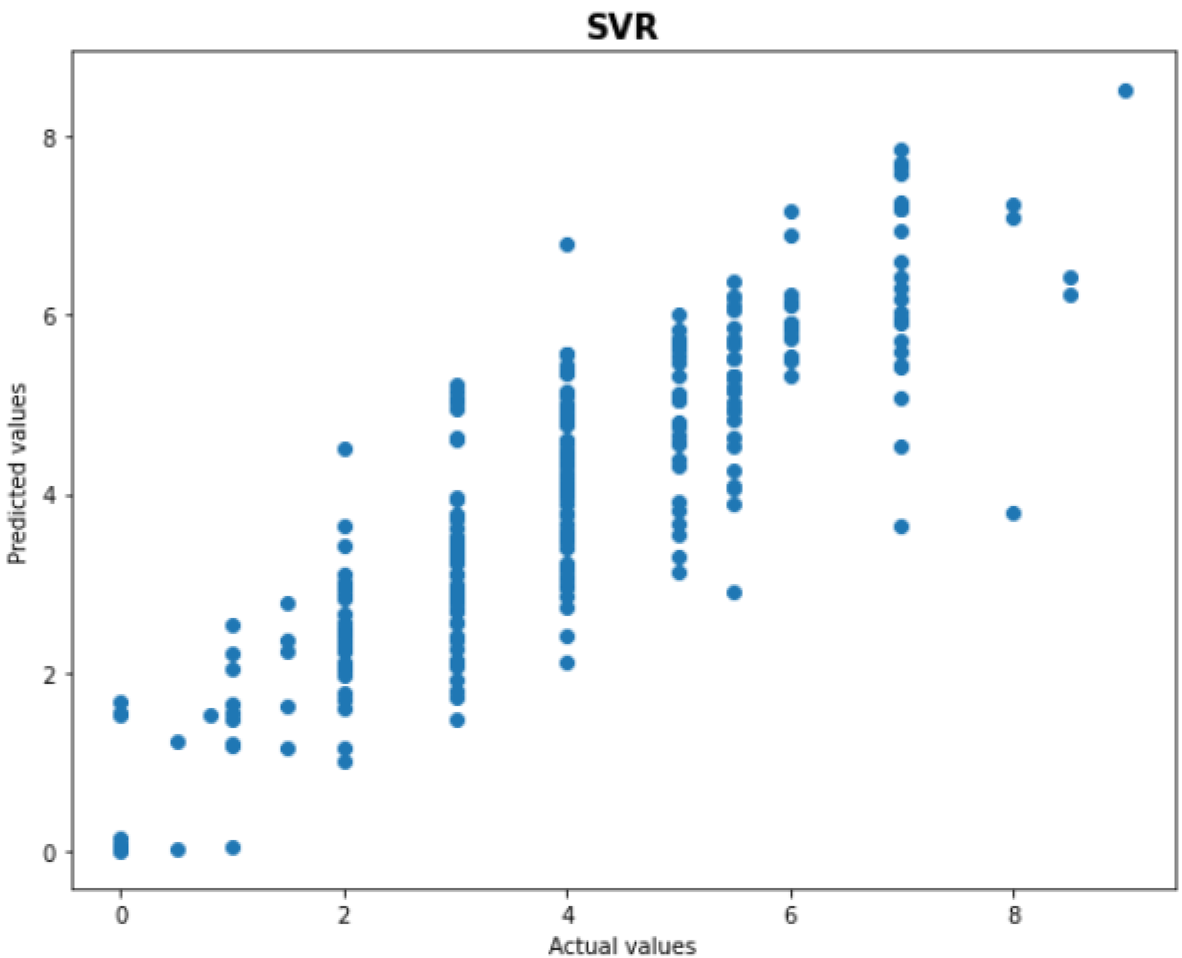
```
SVR()
```

```
y_pred = regressor.predict(X_test)
```

```
from sklearn.metrics import r2_score
r2_score(y_test, y_pred)
```

```
0.745622520025346
```

Plotting the Results:




Benefits:

- Simplistic algorithm
- Efficient

computations

- Disadvantages:

## 3. Random Forest Regression:

```
from sklearn.ensemble import RandomForestRegressor
regressor = RandomForestRegressor(n_estimators = 500, random_state = 0)
regressor.fit(X_train, y_train)
```

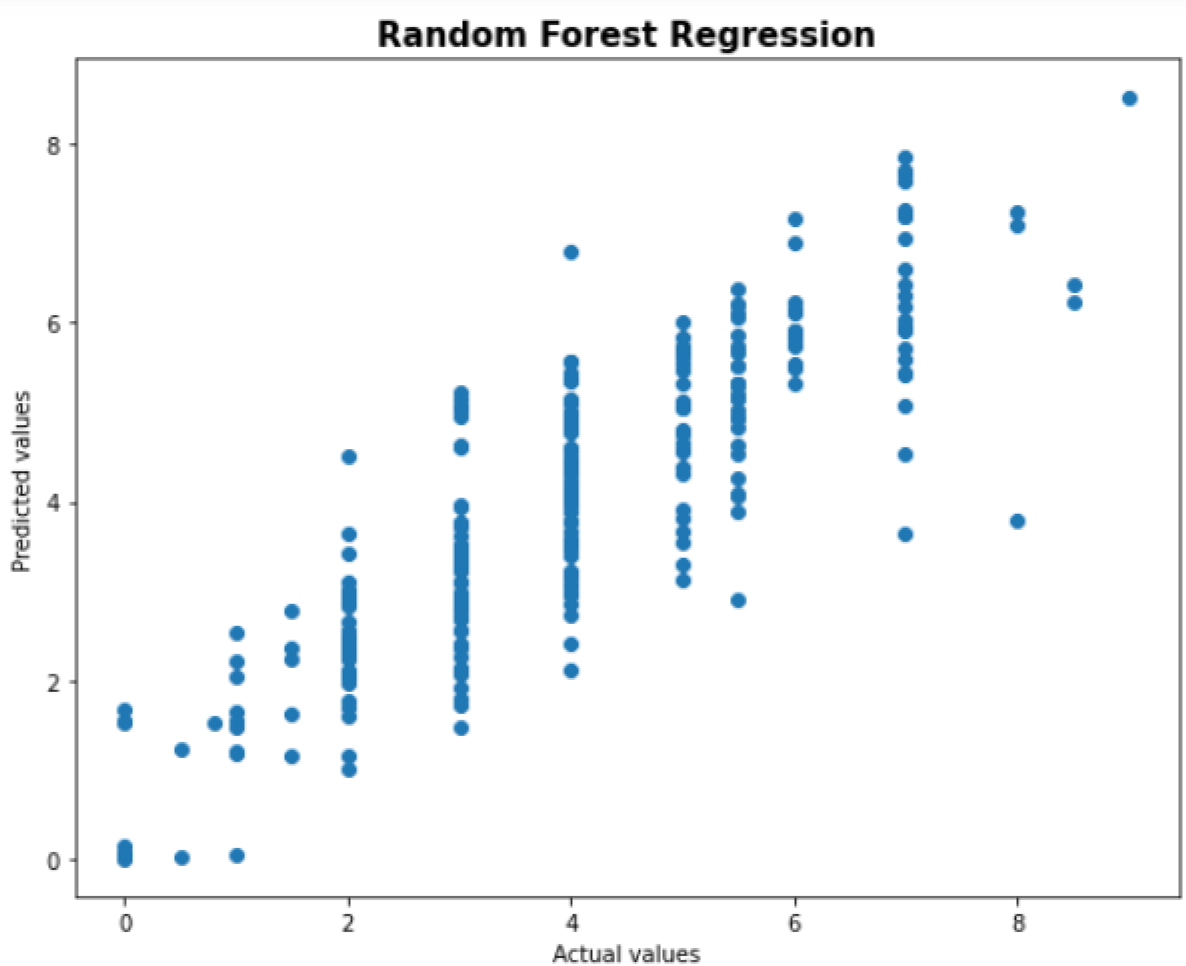
```
RandomForestRegressor(n_estimators=500, random_state=0)
```

```
y_pred = regressor.predict(X_test)
```

```
from sklearn.metrics import r2_score
r2_score(y_test, y_pred)
```

```
0.7428964569049179
```

Plotting the Results:



Benefits:

- Can train model with relatively small number of samples.
- Can solve both, classification and regression problems
- Effective method for estimation of missing information and

maintaining of accuracy

- Disadvantages:

- Performs better on classification problem than regression problem
- Can prove to be like black box approach for statistical modelers

## Algorithm Evaluation (Student English Grade Prediction)

### 1. MultipleLinear Regression:

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

```
]: LinearRegression()
```

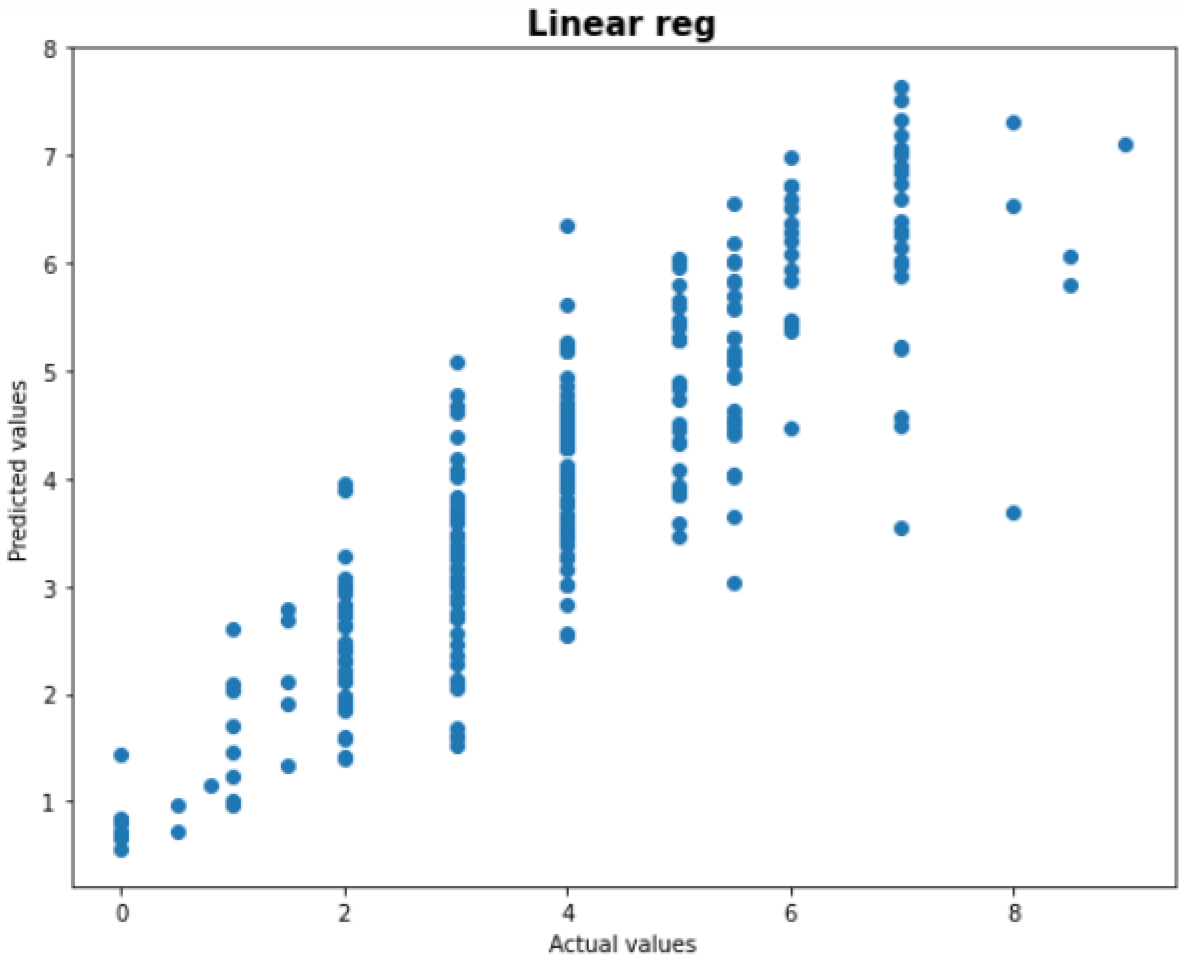
```
]: y_pred = regressor.predict(X_test)
```

```
]: rom sklearn.metrics import r2_score, mean_squared_error
nport numpy as np
r_r2 = r2_score(y_test, y_pred)
r_mse = mean_squared_error(y_test, y_pred)
r_rmse = np.sqrt(lr_mse)
rint('Linear Regression R2 Score: {0} \nLinear Regression MSE: {1}, \nLinear Regression RMSE:{2}'.format(lr_r2,
```

```
Linear Regression R2 Score: 0.7596358012725606
Linear Regression MSE: 0.8405543498667046,
Linear Regression RMSE:0.9168175117583132
```



Plotting the graph :



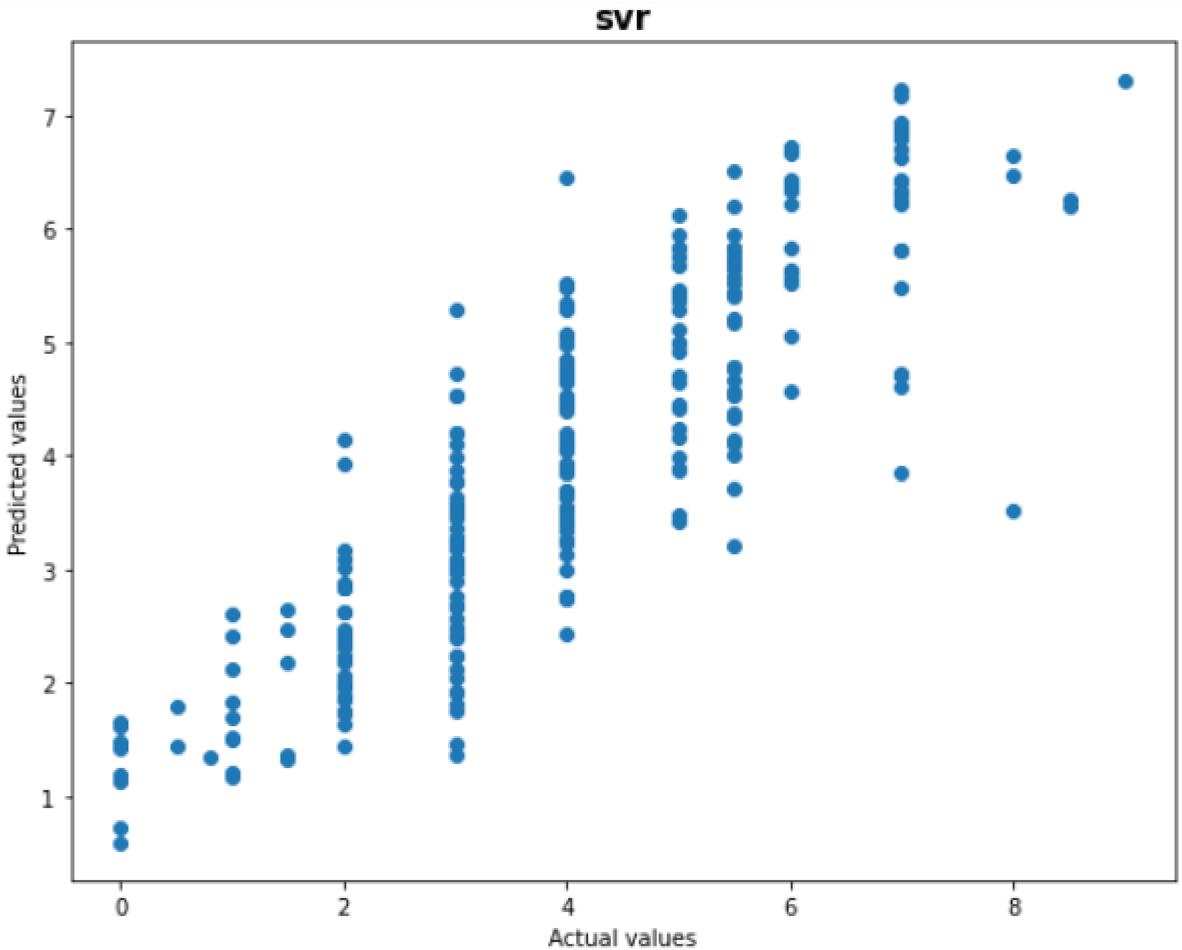

2. SVR

```
from sklearn.svm import SVR
regressor = SVR(kernel = 'rbf')
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)
```

```
·om sklearn.metrics import r2_score, mean_squared_error
nport numpy as np
·_r2 = r2_score(y_test, y_pred)
·_mse = mean_squared_error(y_test, y_pred)
·_rmse = np.sqrt(lr_mse)
·int('Linear Regression R2 Score: {0} \nLinear Regression MSE: {1}, \nLinear Regression RMSE:{2
```

```
Linear Regression R2 Score: 0.745622520025346
Linear Regression MSE: 0.8895588379336169,
Linear Regression RMSE:0.9431642687960655
```

Plotting the graph:

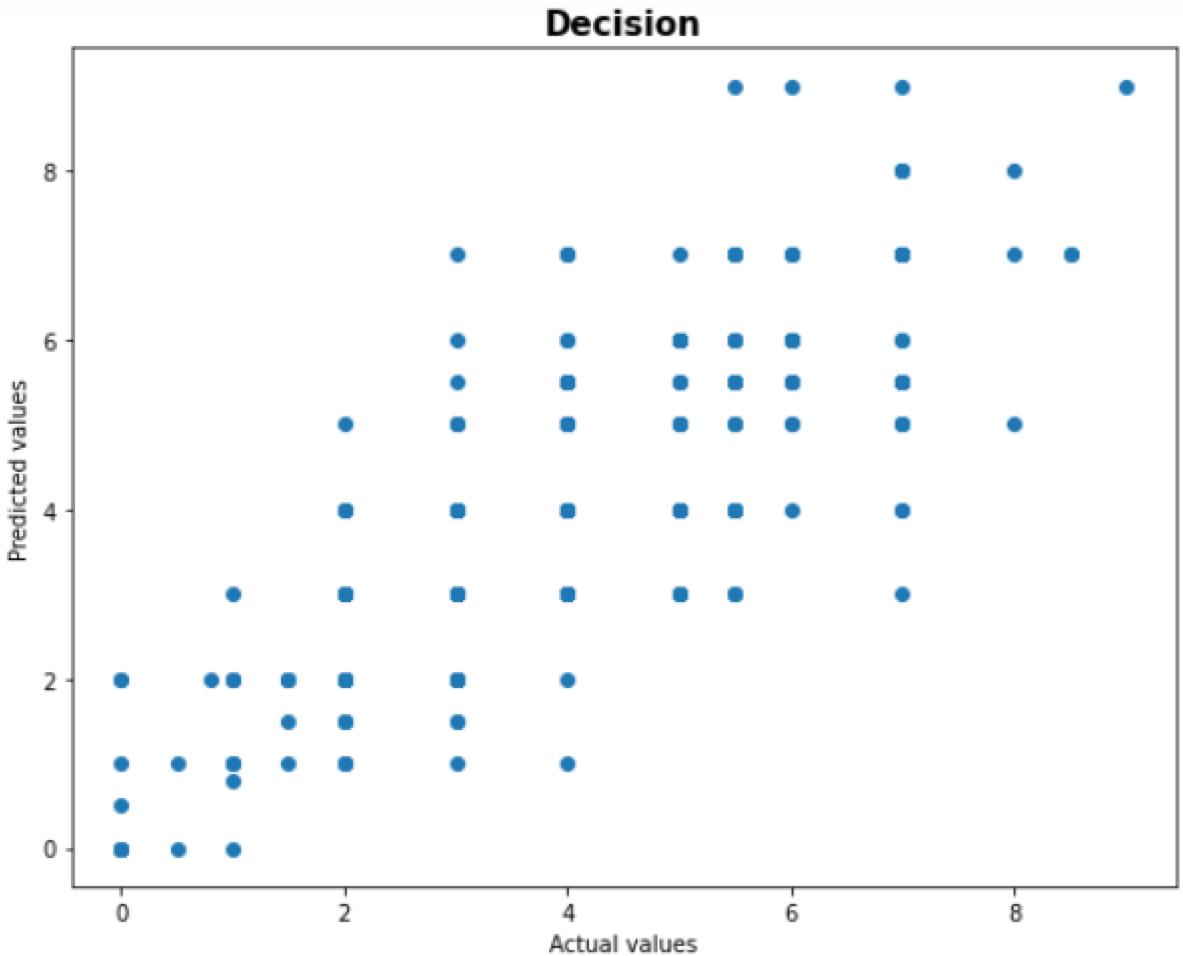



3. Decision Tree

```
from sklearn.tree import DecisionTreeRegressor
regressor = DecisionTreeRegressor(random_state = 0)
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)
from sklearn.metrics import r2_score, mean_squared_error
import numpy as np
lr_r2 = r2_score(y_test, y_pred)
lr_mse = mean_squared_error(y_test, y_pred)
lr_rmse = np.sqrt(lr_mse)
print('Linear Regression R2 Score: {0} \nLinear Regression MSE: {1}, \nLinear Regression RMSE
```

```
Linear Regression R2 Score: 0.5562642048430153
Linear Regression MSE: 1.5517454545454545,
Linear Regression RMSE:1.2456907539776694
```

Plotting the graph:




4. Random Forest Regression

```
from sklearn.ensemble import RandomForestRegressor
regressor = RandomForestRegressor(n_estimators = 500, random_state = 0)
regressor.fit(X_train, y_train)
```

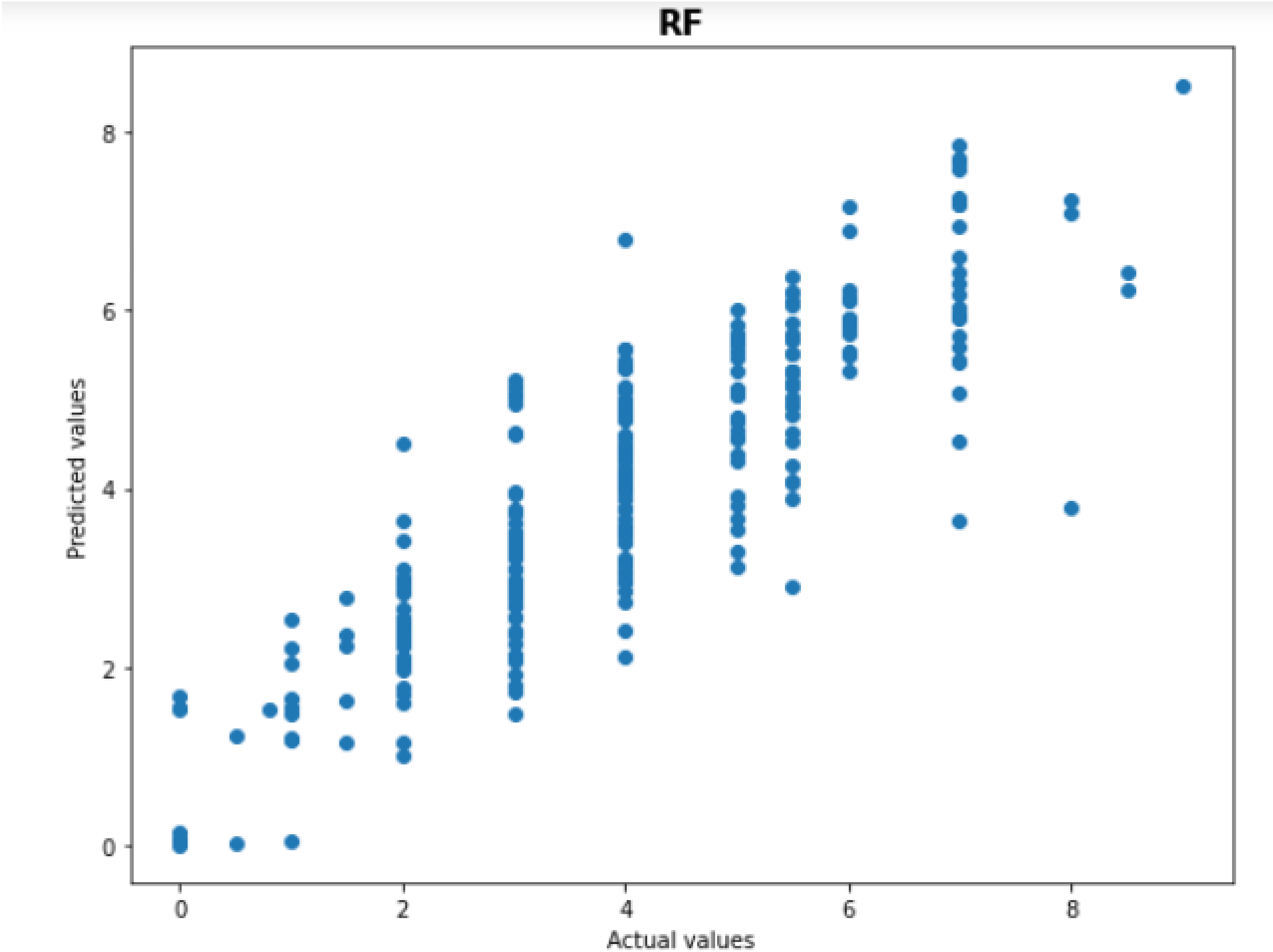
```
RandomForestRegressor(n_estimators=500, random_state=0)
```

```
y_pred = regressor.predict(X_test)
```

```
rom sklearn.metrics import r2_score, mean_squared_error
nport numpy as np
_r2 = r2_score(y_test, y_pred)
_mse = mean_squared_error(y_test, y_pred)
_rmse = np.sqrt(lr_mse)
int('Linear Regression R2 Score: {0} \nLinear Regression MSE: {1}, \nLinear Regression RMSE:{2}'.forma
```

```
Linear Regression R2 Score: 0.7428964569049179
Linear Regression MSE: 0.8990918891368262,
Linear Regression RMSE:0.9482045608078598
```

Plotting the graph:




# Algorithm Evaluation (Student Admission Prediction)

1. MultipleLinear Regression:

```
from sklearn.model_selection import train_test_split
X_train10, X_test10, y_train10, y_test10 = train_test_split(X10, y10, test_size = 0.2 , random_state = 50)
```

```
from sklearn.linear_model import LinearRegression
lin_reg10 = LinearRegression()
lin_reg10.fit(X_train10, y_train10)
```

```
LinearRegression()
```

```
In [27]: pd.DataFrame({"Actual": y_test10, "Predict": y_pred10})
```

Out[27]:

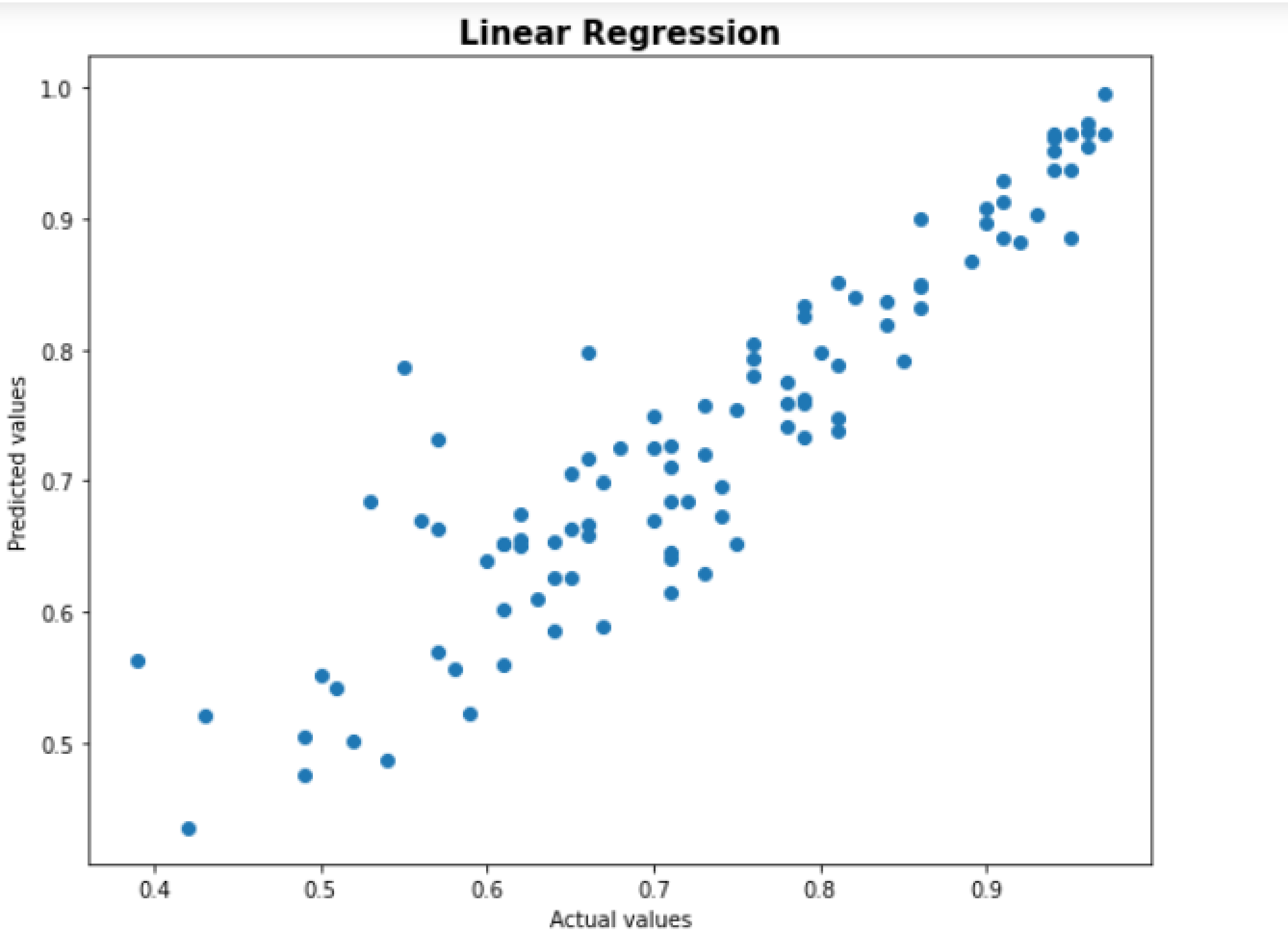
| | Actual | Predict |
|---|---|---|
| **0** | 0.73 | 0.628585 |
| **1** | 0.39 | 0.563045 |
| **2** | 0.64 | 0.626108 |
| **3** | 0.59 | 0.522782 |
| **4** | 0.49 | 0.504517 |
| **...** | ... | ... |
| **95** | 0.66 | 0.798018 |
| **96** | 0.62 | 0.654737 |
| **97** | 0.55 | 0.787248 |
| **98** | 0.66 | 0.658405 |
| **99** | 0.80 | 0.797785 |

100 rows × 2 columns

```
from sklearn.metrics import r2_score, mean_squared_error
import numpy as np
lr_r2 = r2_score(y_test10, y_pred10)
lr_mse = mean_squared_error(y_test10, y_pred10)
lr_rmse = np.sqrt(lr_mse)
print('Linear Regression R2 Score: {0} \nLinear Regression MSE: {1}, \nLinear Regression RMSE:{2}'.format(lr_r2, lr_mse, lr_rmse)
```

```
Linear Regression R2 Score: 0.8413731950456493
Linear Regression MSE: 0.0032409296323111297,
Linear Regression RMSE:0.05692916328483258
```

Plotting the Result:




## 2. Random Forest Regression:

```
from sklearn.ensemble import RandomForestRegressor
regressor = RandomForestRegressor(n_estimators = 500, random_state = 0)
regressor.fit(X_train10, y_train10)
```

```
RandomForestRegressor(n_estimators=500, random_state=0)
```

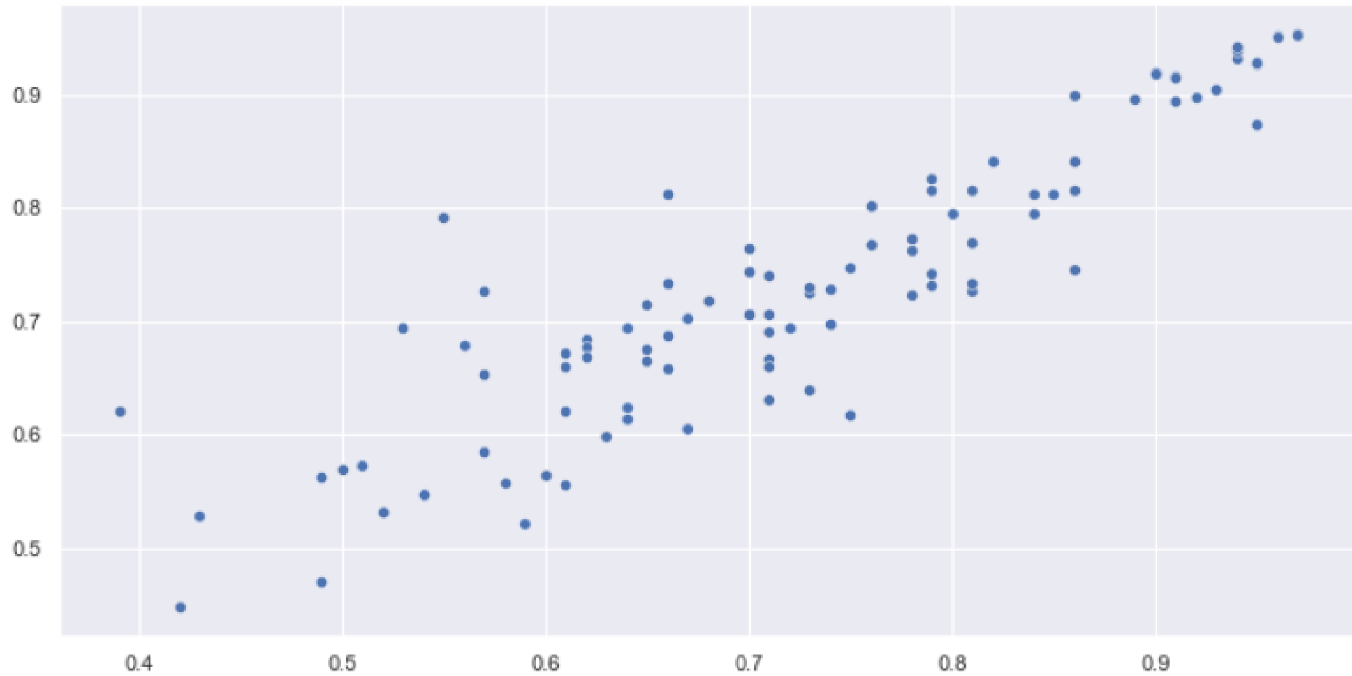
```
y_pred10 = regressor.predict(X_test10)
```

```
from sklearn.metrics import r2_score, mean_squared_error
import numpy as np
lr_r2 = r2_score(y_test10, y_pred10)
lr_mse = mean_squared_error(y_test10, y_pred10)
lr_rmse = np.sqrt(lr_mse)
print('Linear Regression R2 Score: {0} \nLinear Regression MSE: {1}, \nLinear Regression RMSE:{2}'.format(lr_r2, lr_mse, lr_rmse)
```

```
Linear Regression R2 Score: 0.808413484696906
Linear Regression MSE: 0.003914334747999962,
Linear Regression RMSE:0.06256464455265419
```



Plotting the Results:




## 3. SVR

```
from sklearn.svm import SVR
regressor = SVR(kernel = 'rbf')
regressor.fit(X_train10, y_train10)
```

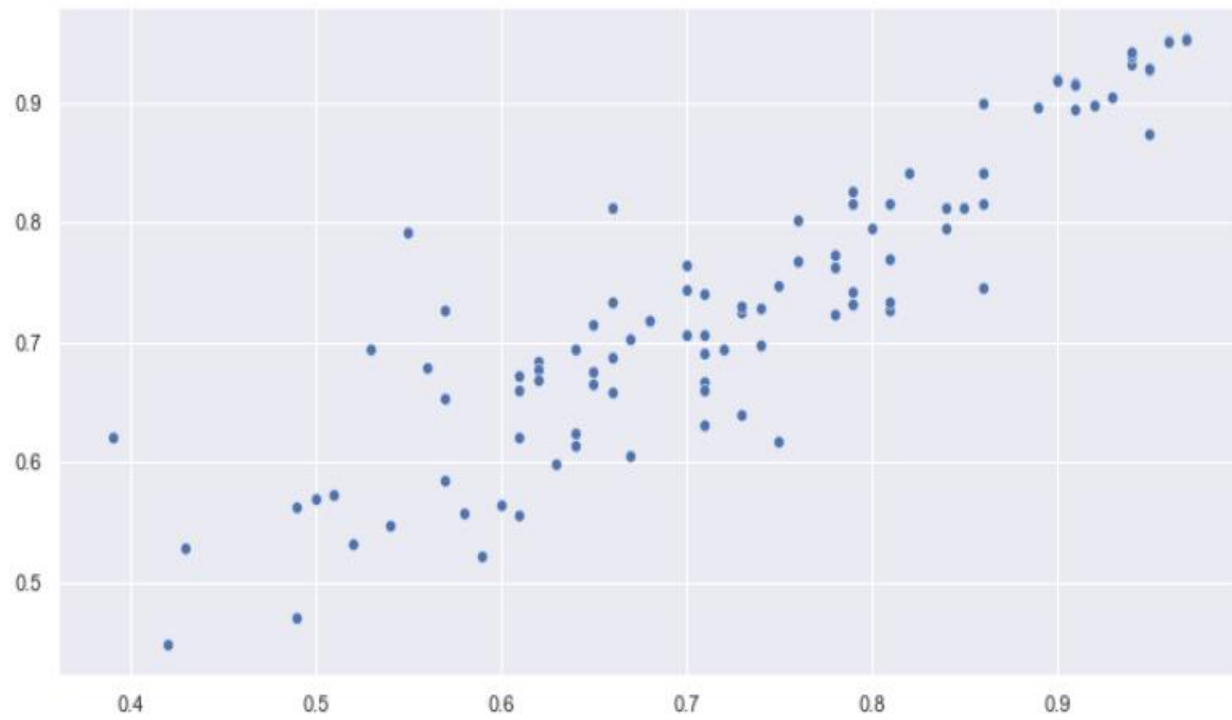
```
SVR()
```

```
y_pred10 = regressor.predict(X_test10)
```

```
from sklearn.metrics import r2_score, mean_squared_error
import numpy as np
lr_r2 = r2_score(y_test10, y_pred10)
lr_mse = mean_squared_error(y_test10, y_pred10)
lr_rmse = np.sqrt(lr_mse)
print('Linear Regression R2 Score: {0} \nLinear Regression MSE: {1}, \nLinear Regression RMSE:{2}'.format(lr_r2, lr_mse,
```

```
Linear Regression R2 Score: 0.7010500829742108
Linear Regression MSE: 0.006107893586740624,
Linear Regression RMSE:0.07815301393254533
```

Plotting the results:



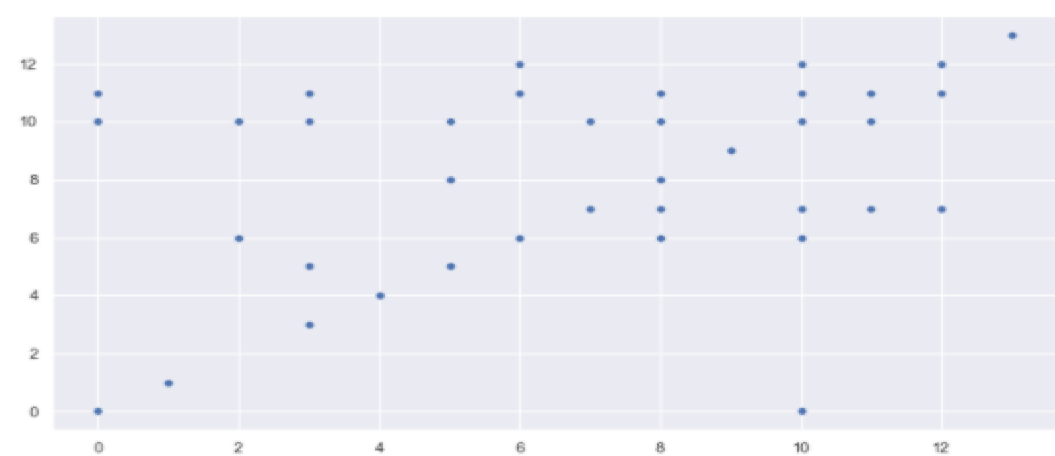
## Algorithm Evaluation (Job Recommendation):

### 1. Gaussian Naïve Byes:

```
from sklearn.mixture import GaussianMixture
from sklearn.cluster import KMeans
from sklearn.metrics import accuracy_score, auc, roc_curve, roc_auc_score


from sklearn.naive_bayes import GaussianNB, MultinomialNB
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
gnb = GaussianNB()
train_preds = gnb.fit(X_train, y_train).predict(X_train)
test_preds = gnb.predict(X_test)

print('Train acc: {0}'.format(accuracy_score(y_train, train_preds)))
print('Test acc: {0}'.format(accuracy_score(y_test, test_preds)))
```

```
Train acc: 0.9231987331749802
Test acc: 0.7008928571428571
```

Plotting the results:

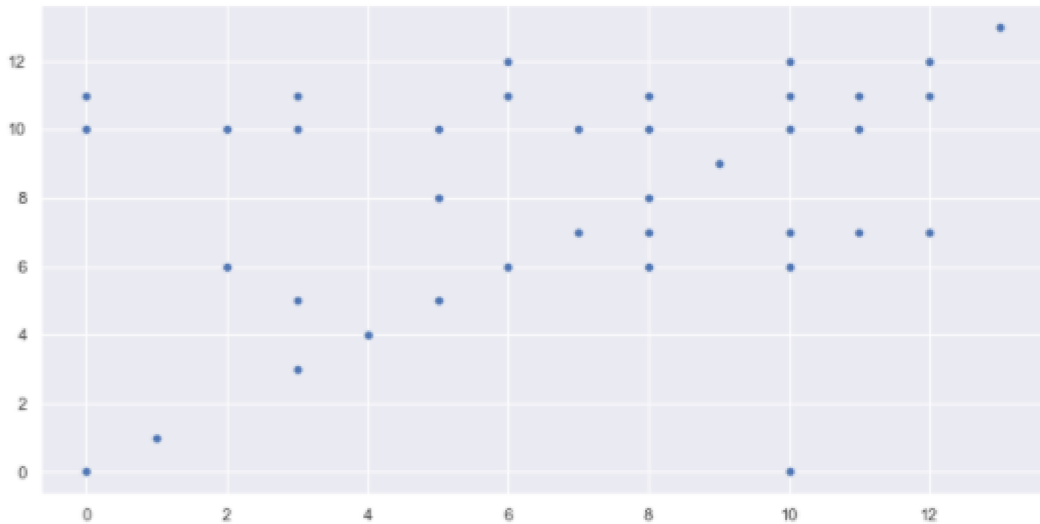


## 2. Logistic Regression:

```
from sklearn.linear_model import LogisticRegression

logistic = LogisticRegression()

train_preds = logistic.fit(X_train, y_train).predict(X_train)
test_preds = logistic.predict(X_test)

print('Train acc: {0}'.format(accuracy_score(y_train, train_preds)))
print('Test acc: {0}'.format(accuracy_score(y_test, test_preds)))
```

```
Train acc: 0.880443388756928
Test acc: 0.7991071428571429
```

Plotting the results:



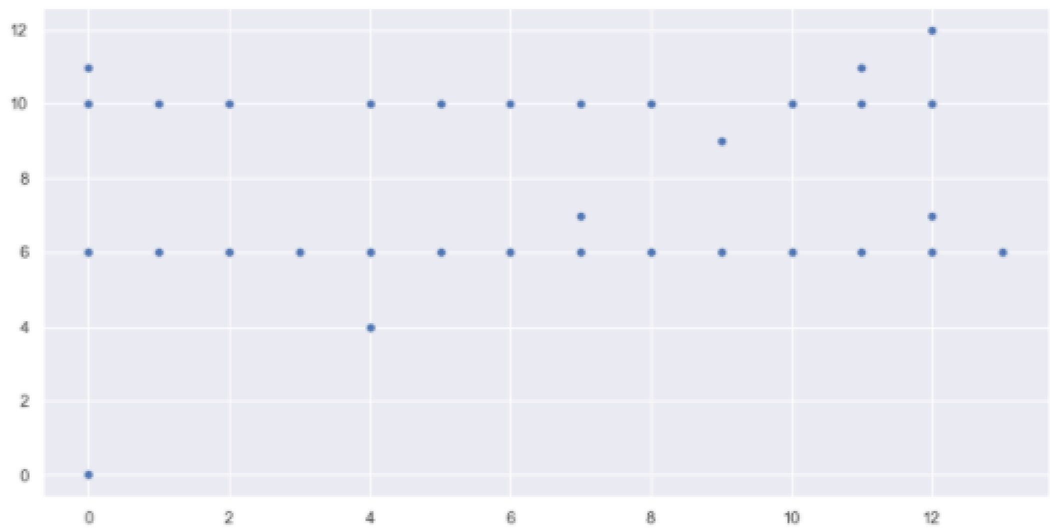

## 3. Random Forest Classification:

```
]: from sklearn.ensemble import RandomForestClassifier
   classifier = RandomForestClassifier(max_depth = 10, max_features = 5, min_samples_leaf = 4,min_samples_split = 10,n_estimators =
   train_preds = classifier.fit(X_train, y_train).predict(X_train)
   test_preds = classifier.predict(X_test)

   from sklearn.metrics import accuracy_score
   accuracy_score(y_test, test_preds)
```

```
]: 0.35267857142857145
```

```
]: import warnings # current version of seaborn generates a bunch of warnings that we'll ignore
   warnings.filterwarnings("ignore")
   import seaborn as sns
   import matplotlib.pyplot as plt
   sns.set(rc={'figure.figsize':(12,6)})
   # sns.(y_test, y_pred)
   sns.scatterplot(y_test, test_preds)
   plt.show()
```

Plotting the results:




## Hyper parameter tuning for Logistic Regression (Job Recommendation):

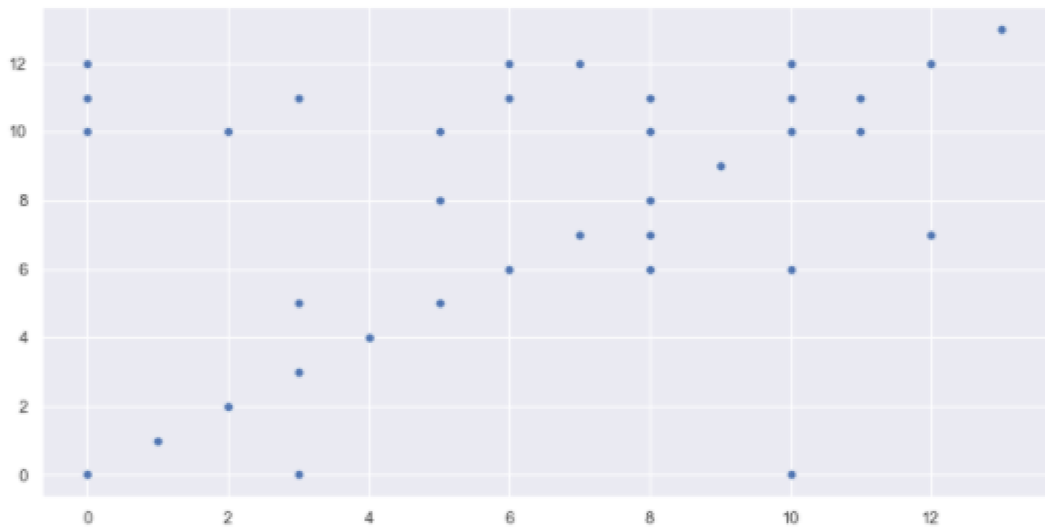
```
# Grid search cross validation
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
import numpy as np
grid={"C":np.logspace(-3,3,7), "penalty":["l1","l2"]}# l1 lasso l2 ridge
logreg=LogisticRegression()
logreg_cv=GridSearchCV(logreg,grid,cv=10)
logreg_cv.fit(X_train,y_train)

print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_)
print("accuracy :",logreg_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'C': 100.0, 'penalty': 'l2'}
accuracy : 0.8061492313460817
```

```
from sklearn.linear_model import LogisticRegression

logistic = LogisticRegression(C= 100.0, penalty= 'l2')

train_preds = logistic.fit(X_train, y_train).predict(X_train)
test_preds = logistic.predict(X_test)

print('Train acc: {0}'.format(accuracy_score(y_train, train_preds)))
print('Test acc: {0}'.format(accuracy_score(y_test, test_preds)))
```

```
Train acc: 0.9912905779889153
Test acc: 0.8348214285714286
```

## Plotting the results:



## Hyper parameter tuning for Random Forest Classifier (Job Recommendation):

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
param_grid = {
    'bootstrap': [True],
    'max_depth': [10,20,30,40],
    'max_features': [3,4,5],
    'min_samples_leaf': [3, 4, 5],
    'min_samples_split': [8, 10, 12],
    'n_estimators': [100, 200, 300, 500,1000]
}
rf = RandomForestClassifier()
grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
                          cv = 3, n_jobs = -1, verbose = 2)
```

```
grid_search.fit(X_train, y_train)
grid_search.best_params_
```

```
Fitting 3 folds for each of 540 candidates, totalling 1620 fits
```

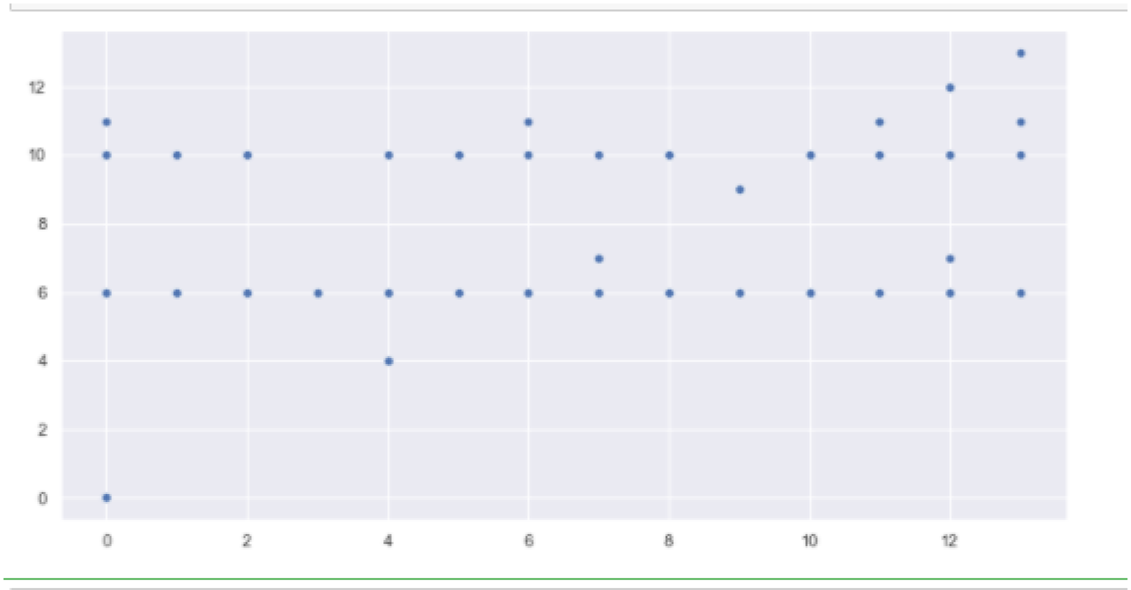
```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  33 tasks      | elapsed:   15.5s
[Parallel(n_jobs=-1)]: Done 154 tasks      | elapsed:  1.0min
[Parallel(n_jobs=-1)]: Done 357 tasks      | elapsed:  2.3min
[Parallel(n_jobs=-1)]: Done 640 tasks      | elapsed:  4.1min
[Parallel(n_jobs=-1)]: Done 1005 tasks      | elapsed:  6.7min
[Parallel(n_jobs=-1)]: Done 1450 tasks      | elapsed:  9.7min
[Parallel(n_jobs=-1)]: Done 1620 out of 1620 | elapsed: 11.0min finished
```

```
{'bootstrap': True,
 'max_depth': 40,
 'max_features': 5,
 'min_samples_leaf': 3,
 'min_samples_split': 12,
 'n_estimators': 100}
```

```
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(max_depth = 40, max_features = 5, min_samples_leaf = 3,min_samples_split = 12,n_estimators =
train_preds = classifier.fit(X_train, y_train).predict(X_train)
test_preds = classifier.predict(X_test)

from sklearn.metrics import accuracy_score
accuracy_score(y_test, test_preds)
```

```
0.42410714285714285
```

## Plotting the results:




## Candidate Algorithm Selection (Student Math Grade):

| Algorithm | Mean Absolute Error | Root Mean Square Error | R2_Score |
|---|---|---|---|
| Multiple Linear Regression | 0.708 | 0.84 | 0.806 |
| SVR | 0.826 | 0.909 | 0.773 |
| Random Forest Regression | 0.686 | 0.828 | 0.8122 |
| Decision Tree Regression | 1.31 | 1.144 | 0.641 |

Based on the above statistics, my top algorithm to solve this problem would be Linear Regression

## Candidate Algorithm Selection (Student English Grade):

| Algorithm | Mean Absolute Error | Root Mean Square Error | R2_Score |
|---|---|---|---|
| Multiple Linear Regression | 0.8405 | 0.9168 | 0.7596 |
| SVR | 0.889 | 0.9431 | 0.7456 |
| Random Forest Regression | 0.899 | 0.94 | 0.74 |
| Decision Tree Regression | 1.55 | 1.24 | 0.556 |

Based on the above statistics, my top algorithm to solve this problem would be Linear Regression

## Candidate Algorithm Selection (Student Admission Prediction):

| Algorithm | Mean Absolute Error | Root Mean Square Error | R2_Score |
|---|---|---|---|
| Multiple Linear Regression | 0.05 | 0.003 | 0.841 |
| SVR | 0.07 | 0.006 | 0.701 |
| Random Forest Regression | 0.06 | 0.0039 | 0.8084 |

Based on the above statistics, my top algorithm to solve this problem would be Linear Regression

## Candidate Algorithm Selection (Student Admission Prediction):

| Algorithm | Classification Accuracy | Accuracy after Hyperparameter tuning |
|---|---|---|
| Gaussian NB | 0.70089 | - |
| Logistic Regression | 0.79910 | 0.8348 |
| Random Forest Regression | 0.352 | 0.424 |

Based on the above statistics, my top algorithm to solve this problem would be Logistic Regression

## Rationale:

There were many classification and regression algorithms to try before coming to the final decision of which algorithm suits best to solve the given problem.

For the student grade prediction of mathematics and English, the model showed higher accuracy with linear regression when compared with very strong algorithms like random forest
That is why for student grade prediction, I am going ahead with linear regression

For the student admission prediction, the model showed higher accuracy with linear regression when compared with very strong algorithms like random forest
That is why for the student admission prediction, I am going ahead with linear regression

For the job recommendation model, I have used different set of classification algorithms like Gaussian Naïve Bayes and Logistic Regression along with the common Classification algorithms like Random Forest Classifier
The model that showed the highest accuracy after tuning the parameters was logistic Regression. I will be going ahead with logistic regression to solve this model

## Final Inference

Running a test case scenario can be called as just a step before deploying the application. Any errors or bugs detected in this step can result into working of remodelling of whole application. Hence, an intensive testing should be performed to ensure the expected output is obtained.

Student Admission Prediction

```
pop = [[338,112,5,5,5,9.5,1]]          #STUDENT WITH GOOD GRADES
y_pred101 = lin_reg10.predict(pop)
```

```
y_pred101
```

```
array([0.93670222])
```

```
pop = [[222,86,2,2,2,4.5,0]]           #STUDENT WITH BAD GRADES
y_pred101 = lin_reg10.predict(pop)
```

```
y_pred101
```

```
array([-0.05574587])
```

```
pop = [[222,86,2,2,2,7.5,0]]            #STUDENT WITH AVERAGE GRADES
y_pred101 = lin_reg10.predict(pop)
```

```
y_pred101
```

```
array([0.31773349])
```

```
pop = [[2,3,1.7,2.9,2.2]]
y_pred101 = regressor.predict(pop)
```

```
y_pred101
```

```
array([8.684])
```

These are the test runs which shows how the model can be used to solve the problem