MACHINE LEARNING

(Prediction of acceptance of admission in getting into university)

Summer Internship Report Submitted in partial fulfillment

of the requirement for undergraduate degree of

Bachelor of Technology

In

Computer Science Engineering

By

Kommera Shreya

221710313025

Under the Guidance of

Assistant Professor

GITAM
(DEEMED TO BE UNIVERSITY)
VISAKHAPATNAM HYDERABAD BENGALURU

Department Of Electronics and Communication Engineering
GITAM School of Technology
GITAM (Deemed to be University)
Hyderabad-502329
June 2019

DECLARATION

I submit this industrial training work entitled "Prediction of acceptance of admission in getting into university" to GITAM (Deemed To Be University), Hyderabad in partial fulfillment of the requirements for the award of the degree of "**Bachelor of Technology**" in "**Computer Science Engineering**". I declare that it was carried out independently by me under the guidance of **Mr.** Asst. Professor, GITAM (Deemed To Be University), Hyderabad, India.

The results embodied in this report have not been submitted to any other University or Institute for the award of any degree or diploma.

Place: HYDERABAD   Kommera Shreya

Date: 221710313025

GITAM (DEEMED TO BE UNIVERSITY)

Hyderabad-502329, India

Dated:

CERTIFICATE

This is to certify that the Industrial Training Report entitled **“Prediction of acceptance of admission in getting into university”** is being submitted by Kommera Shreya (221710313025) in partial fulfillment of the requirement for the award of **Bachelor of Technology in Computer Science Engineering** at GITAM (Deemed To Be University), Hyderabad during the academic year 2018-19

It is faithful record work carried out by her at the **Computer Science Engineering Department**, GITAM University Hyderabad Campus under my guidance and supervision.

Mr.	**Dr.**
Assistant Professor	Professor and HOD
Department of CSE	Department of CSE

ACKNOWLEDGEMENT

Apart from my effort, the success of this internship largely depends on the encouragement and guidance of many others. I take this opportunity to express my gratitude to the people who have helped me in the successful competition of this internship.

I would like to thank respected **Dr. N. Siva Prasad,** Pro Vice Chancellor, GITAM Hyderabad and **Dr. CH. Sanjay,** Principal, GITAM Hyderabad

I would like to thank respected **Dr.,** Head of the Department of Computer science Engineering for giving me such a wonderful opportunity to expand my knowledge for my own branch and giving me guidelines to present a internship report. It helped me a lot to realize of what we study for.

I would like to thank the respected faculties **Mr.** who helped me to make this internship a successful accomplishment.

I would also like to thank my friends who helped me to make my work more organized and well-stacked till the end.

Kommera Shreya

221710313025

ABSTRACT

Machine learning algorithms are used to predict the values from the data set by splitting the data set in to train and test and building Machine learning algorithms models of higher accuracy to predict the values is the primary task to be performed on acceptance data set My perception of understanding the given data set has been in the view of undertaking a Universities requirement for taking students.

Machine learning can be used for more than just evaluating whether to admit an applicant. It can also be used to estimate the probability that an applicant will accept if admitted. Predicting acceptance is arguably a much easier prediction problem and is just as useful because then you can take calculated risks on how many students to admit so that the correct expected number of students enroll each year.

Table of Contents:

TABLE OF FIGURES

CHAPTER 1
MACHINE LEARNING

1.1 INTRODUCTION:

Machine Learning (ML) is the scientific study of algorithms and statistical models that computer systems use in order to perform a specific task effectively without using explicit instructions, relying on patterns and inference instead. It is seen as a subset of Artificial Intelligence (AI).

1.2 IMPORTANCE OF MACHINE LEARNING:

Consider some of the instances where machine learning is applied: the self-driving Google car, cyber fraud detection, online recommendation engines—like friend suggestions on Facebook, Netflix showcasing the movies and shows you might like, and “more items to consider” and “get yourself a little something” on Amazon—are all examples of applied machine learning. All these examples echo the vital role machine learning has begun to take in today’s data-rich world.

Machines can aid in filtering useful pieces of information that help in major advancements, and we are already seeing how this technology is being implemented in a wide variety of industries.

With the constant evolution of the field, there has been a subsequent rise in the uses, demands, and importance of machine learning. Big data has become quite a buzzword in the last few years; that’s in part due to increased sophistication of machine learning, which helps analyze those big chunks of big data. Machine learning has also changed the way data extraction, and interpretation is done by involving automatic sets of generic methods that have replaced traditional statistical techniques.

The process flow depicted here represents how machine learning works

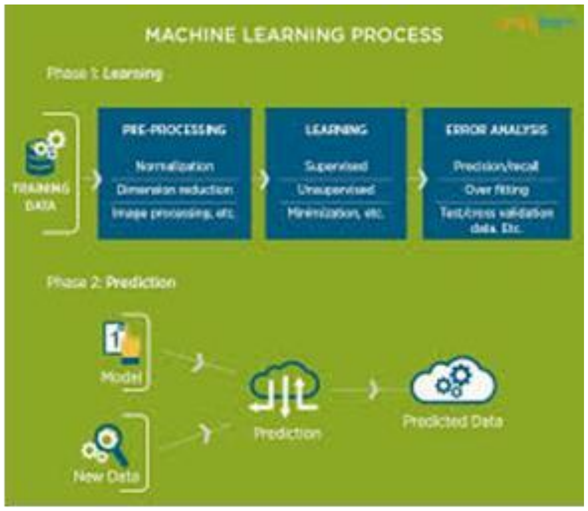


Figure 1: The Process Flow

1.3 USES OF MACHINE LEARNING:

Earlier in this article, we mentioned some applications of machine learning. To understand the concept of machine learning better, let's consider some more examples: web search results, real-time ads on web pages and mobile devices, email spam filtering, network intrusion detection, and pattern and image recognition. All these are by-products of applying machine learning to analyze huge volumes of data

Traditionally, data analysis was always being characterized by trial and error, an approach that becomes impossible when data sets are large and heterogeneous. Machine learning comes as the solution to all this chaos by proposing clever alternatives to analyzing huge volumes of data.

By developing fast and efficient algorithms and data-driven models for real-time processing of data, machine learning can produce accurate results and analysis.

By developing fast and efficient algorithms and data-driven models for real-time processing of data, machine learning can produce accurate results and analysis.

1.4 TYPES OF LEARNING ALGORITHMS:

The types of machine learning algorithms differ in their approach, the type of data they input and output, and the type of task or problem that they are intended to solve.

1.4.1 Supervised Learning:

When an algorithm learns from example data and associated target responses that can consist of numeric values or string labels, such as classes or tags, in order to later predict the correct response when posed with new examples comes under the category of supervised learning.

Supervised machine learning algorithms uncover insights, patterns, and relationships from a labeled training dataset – that is, a dataset that already contains a known value for the target variable for each record. Because you provide the machine learning algorithm with the correct answers for a problem during training, it is able to "learn" how the rest of the features relate to the target, enabling you to uncover insights and make predictions about future outcomes based on historical data.

Examples of Supervised Machine Learning Techniques are Regression, in which the algorithm returns a numerical target for each example, such as how much revenue will be generated from a new marketing campaign.

Classification, in which the algorithm attempts to label each example by choosing between two or more different classes. Choosing between two classes is called binary classification, such as determining whether or not someone will default on a loan. Choosing between more than two classes is referred to as multiclass classification.

1.4.2 Unsupervised Learning:

When an algorithm learns from plain examples without any associated response, leaving to the algorithm to determine the data patterns on its own. This type of algorithm tends to restructure the data into something else, such as new features that may represent a class or a new series of uncorrelated values. They are quite useful in providing humans with insights into the meaning of data and new useful inputs to supervised machine learning algorithms.

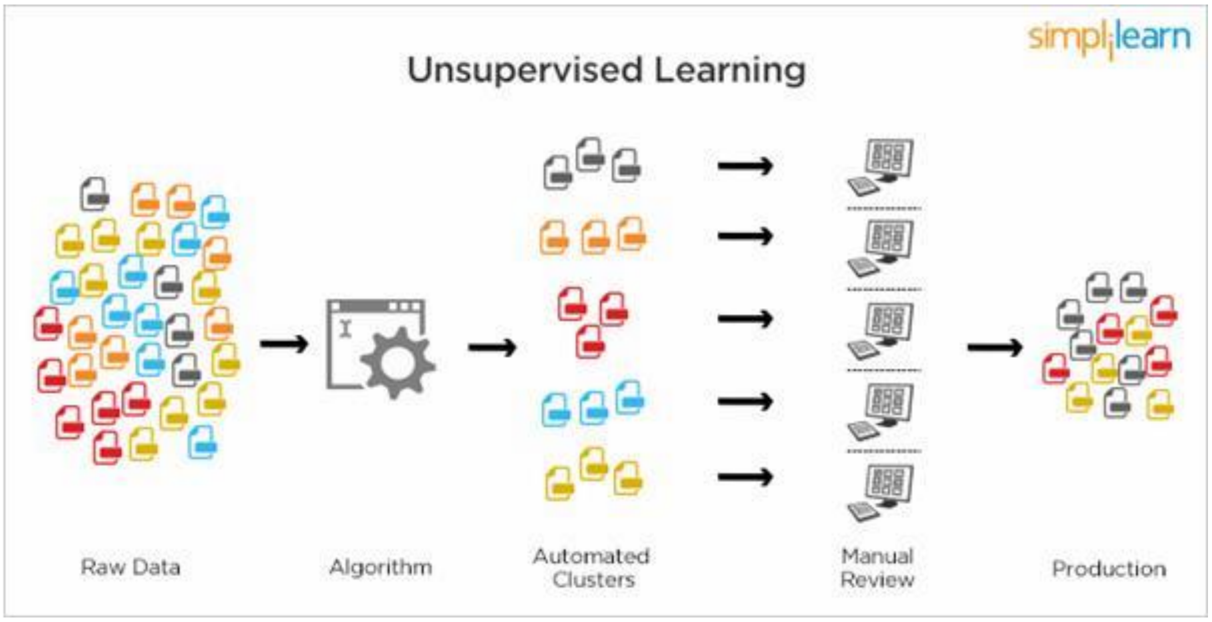
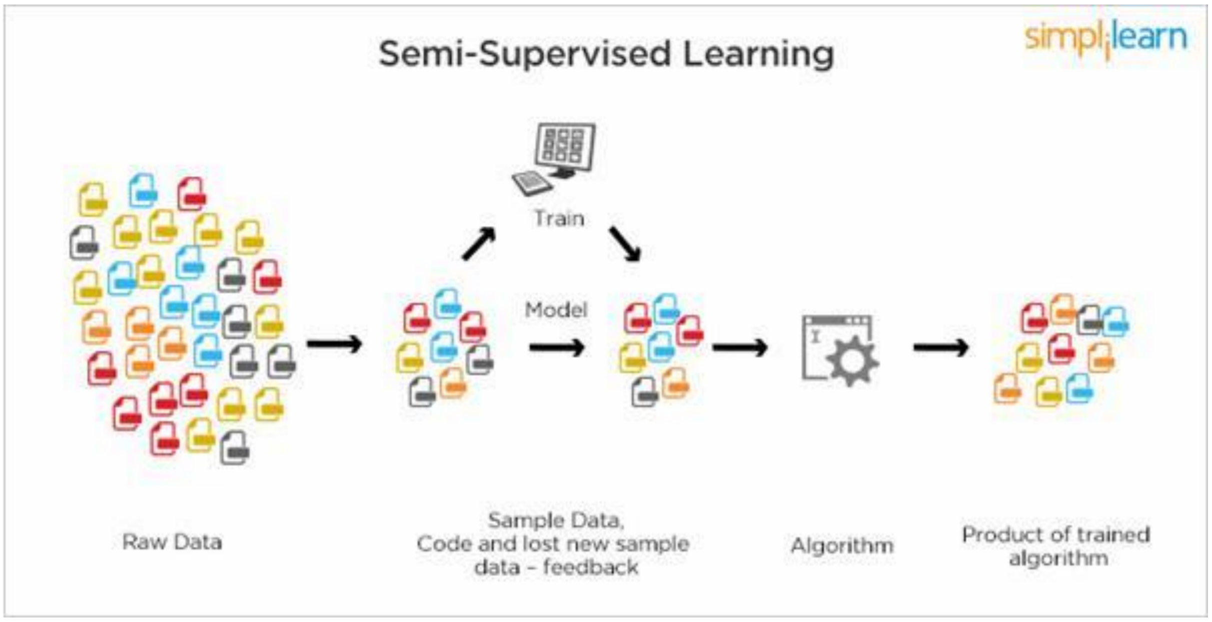


Figure 2: Unsupervised Learning

Popular techniques where unsupervised learning is used also include self-organizing maps, nearest neighbor mapping, singular value decomposition, and k-means clustering. Basically, online recommendations, identification of data outliers, and segment text topics are all examples of unsupervised learning.

1.4.3 Semi Supervised Learning:

As the name suggests, semi-supervised learning is a bit of both supervised and unsupervised learning and uses both labeled and unlabeled data for training. In a typical scenario, the algorithm would use a small amount of labeled data with a large amount of unlabeled data.



Semi-Supervised Learning
simplilearn
Train
Model
Raw Data
Sample Data, Code and lost new sample data - feedback
Algorithm
Product of trained algorithm

Figure 3: Semi Supervised Learning

1.5 RELATION BETWEEN DATA MINING, MACHINE LEARNING AND DEEP LEARNING:

Machine learning and data mining use the same algorithms and techniques as data mining, except the kinds of predictions vary. While data mining discovers previously unknown patterns and knowledge, machine learning reproduces known patterns and knowledge—and further automatically applies that information to data, decision-making, and actions.

Deep learning, on the other hand, uses advanced computing power and special

Types of neural networks and applies them to large amounts of data to learn, understand, and identify complicated patterns. Automatic language translation and medical diagnoses are examples of deep learning.

CHAPTER 2

PYTHON

Basic programming language used for machine learning is: PYTHON

2.1 INTRODUCTION TO PYHTON:

- Python is a high-level, interpreted, interactive and object-oriented scripting language.
- Python is a general purpose programming language that is often applied in scripting roles
- Python is interpreted: Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is like PERL and PHP.
- Python is Interactive: You can sit at a Python prompt and interact with the interpreter directly to write your programs.
- Python is Object-Oriented: Python supports the Object-Oriented style or technique of programming that encapsulates code within objects.

2.2 HISTORY OF PYTHON:

- Python was developed by GUIDO VAN ROSSUM in early 1990's
- Its latest version is 3.7 , it is generally called as python3

2.3 FEATURES OF PYTHON:

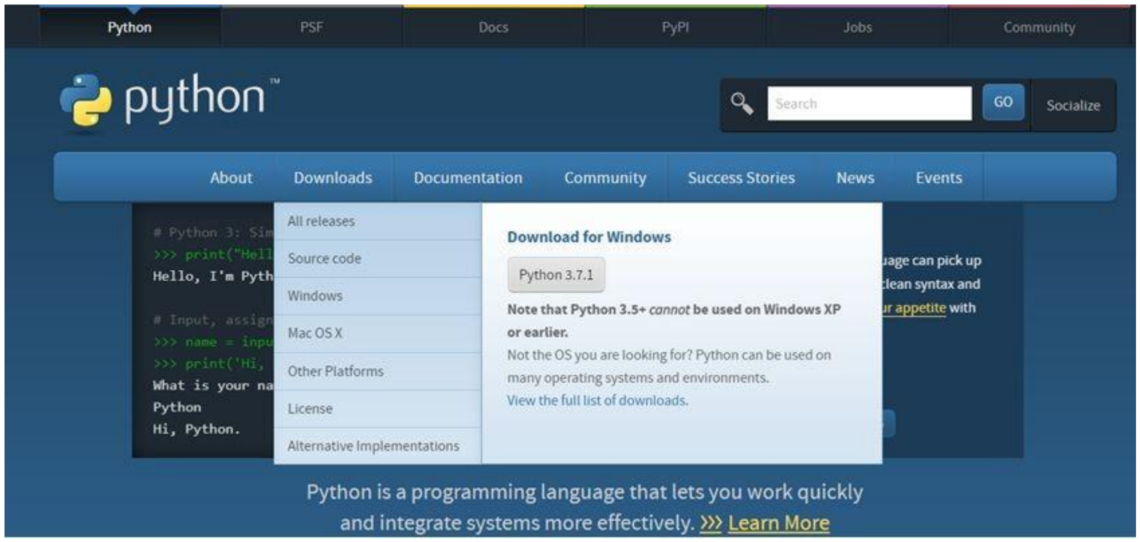
- Easy-to-learn: Python has few keywords, simple structure, and a clearly defined syntax, This allows the student to pick up the language quickly.
- Easy-to-read: Python code is more clearly defined and visible to the eyes.
- Easy-to-maintain: Python's source code is fairly easy-to-maintaining.
- A broad standard library: Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- Portable: Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- Extendable: You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- Databases: Python provides interfaces to all major commercial databases.
- GUI Programming: Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

2.4 HOW TO SETUP PYTHON:

- Python is available on a wide variety of platforms including Linux and Mac OS X. Let's understand how to set up our Python environment.
- The most up-to-date and current source code, binaries, documentation, news, etc., is available on the official website of Python.

2.4.1 Installation (using python IDLE):

- Installing python is generally easy, and nowadays many Linux and Mac OS distributions include a recent python.
- Download python from www.python.org
- When the download is completed, double click the file and follow the instructions to install it.
- When python is installed, a program called IDLE is also installed along with it. It provides a graphical user interface to work with python.



```
# Python 3: Sim
>>> print("Hell
Hello, I'm Pyth

# Input, assign
>>> name = inpu
>>> print('Hi,
What is your na
Python
Hi, Python.
```

Download for Windows

Note that Python 3.5+ *cannot* be used on Windows XP or earlier.

Not the OS you are looking for? Python can be used on many operating systems and environments.

View the full list of downloads.

uage can pick up
clean syntax and
ur appetite with

Python is a programming language that lets you work quickly and integrate systems more effectively. >>> Learn More

Figure 4: Python download

2.4.2 Installation (using Anaconda):

- Python programs are also executed using Anaconda.
- Anaconda is a free open source distribution of python for large scale data processing, predictive analytics and scientific computing.
- Conda is a package manager quickly installs and manages packages.

- In WINDOWS:
- In windows
  - Step 1: Open Anaconda.com/downloads in web browser.
  - Step 2: Download python 3.4 version for (32-bitgraphic installer/64 -bit graphic installer)
  - Step 3: select installation type(all users)
  - Step 4: Select path(i.e. add anaconda to path & register anaconda as default python 3.4) next click install and next click finish
  - Step 5: Open jupyter notebook (it opens in default browser)

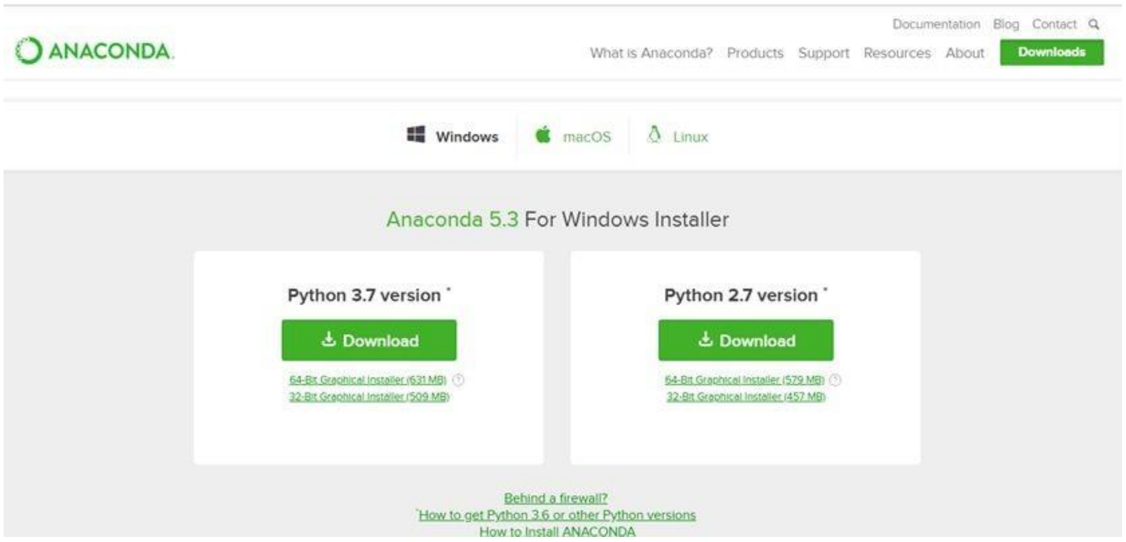

ANACONDA.
Documentation Blog Contact
What is Anaconda? Products Support Resources About Downloads
Windows macOS Linux
Anaconda 5.3 For Windows Installer
Python 3.7 version *
Download
64-Bit Graphical Installer (631 MB)
32-Bit Graphical Installer (509 MB)
Python 2.7 version *
Download
64-Bit Graphical Installer (579 MB)
32-Bit Graphical Installer (457 MB)
Behind a firewall?
*How to get Python 3.6 or other Python versions
How to Install ANACONDA

Figure 5 : Anaconda download

Figure 6 : Jupyter notebook

2.5 PYTHON VARIABLE TYPES:

- Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.
- Variables are nothing but reserved memory locations to store values.
- Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory.
- Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable.
- Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.
- Python has five standard data types –
    - Numbers
    - Strings
    - Lists

- Tuples
- Dictionary

2.5.1 Python Numbers:

- Number data types store numeric values. Number objects are created when you assign a value to them.
- Python supports four different numerical types − int (signed integers) long (long integers, they can also be represented in octal and hexadecimal) float (floating point real values) complex (complex numbers).

2.5.2 Python Strings:

- Strings in Python are identified as a contiguous set of characters represented in the quotation marks.
- Python allows for either pairs of single or double quotes.
- Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.
- The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator.

2.5.3 Python Lists:

- Lists are the most versatile of Python's compound data types.
- A list contains items separated by commas and enclosed within square brackets

([]).

- To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type.
- The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1.
- The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.

2.5.4 Python Tuples:

- A tuple is another sequence data type that is similar to the list.
- A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.
- The main differences between lists and tuples are: Lists are enclosed in brackets ([ ]) and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated.
- Tuples can be thought of as read-only lists.
- For example − Tuples are fixed size in nature whereas lists are dynamic. In other words, a tuple is immutable whereas a list is mutable. You can't add elements to a tuple. Tuples have no appended or extend method. You can't remove elements from a tuple. Tuples have no remove or pop method.

2.5.5 Python Dictionary:

- Python's dictionaries are kind of hash table type. They work like associative arrays

Or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

- Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).
- You can use numbers to "index" into a list, meaning you can use numbers to find out what's in lists. You should know this about lists by now, but make sure you understand that you can only use numbers to get items out of a list.
- What a dict does is let you use anything, not just numbers. Yes, a dict associates one thing to another, no matter what it is.

2.6 PYTHON FUNCTION:

2.6.1 Defining a Function:

You can define functions to provide the required functionality. Here are simple rules to define a function in Python. Function blocks begin with the keyword def followed by the function name and parentheses (i.e.()).

Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses

The code block within every function starts with a colon (:) and is indented. The statement returns [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

2.6.2 Calling a Function:

Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code. Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt.

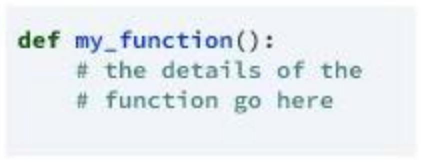
2.7 PYTHON USING OOP's CONCEPTS:

2.7.1 Class:

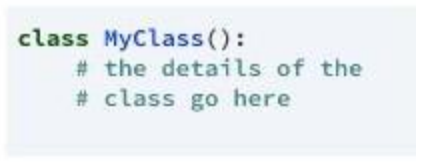
- Class: A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.
- Class variable: A variable that is shared by all instances of a class. Class variables are defined within a class but outside any of the class's methods. Class variables are not used as frequently as instance variables are.
- Data member: A class variable or instance variable that holds data associated with a class and its objects.
- Instance variable: A variable that is defined inside a method and belongs only to the current instance of a class.

- Defining a Class:
  - We define a class in a very similar way how we define a function.
  - Just like a function, we use parentheses and a colon after the class name (i.e. () :) when we define a class. Similarly, the body of our class is

Indented like a functions body is.



```
def my_function():
    # the details of the
    # function go here
```



```
class MyClass():
    # the details of the
    # class go here
```

Figure 7: Defining a Class

2.7.2 __In it__method in Class:

- The in it method — also called a constructor — is a special method that runs when an instance is created so we can perform any tasks to set up the instance.
- The in it method has a special name that starts and ends with two underscores: in it ().

CHAPTER 3

CASE STUDY

3.1 PROBLEM STATEMENT:

Prediction of acceptance of admission in getting into university using Machine Learning.

3.2 DATA SET:

1. Serial No.
2. GRE Score
3. TOEFL Score
4. University Rating
5. SOP
6. LOR
7. CGPA
8. Research
9. Admit

3.3 OBJECTIVE OF THE CASE STUDY:

This dataset was built with the purpose of helping students in shortlisting universities with their profiles. The predicted output gives them a fair idea about their chances for a particular university.

CHAPTER 4

MODEL BUILDING

4.1 Data Collection:

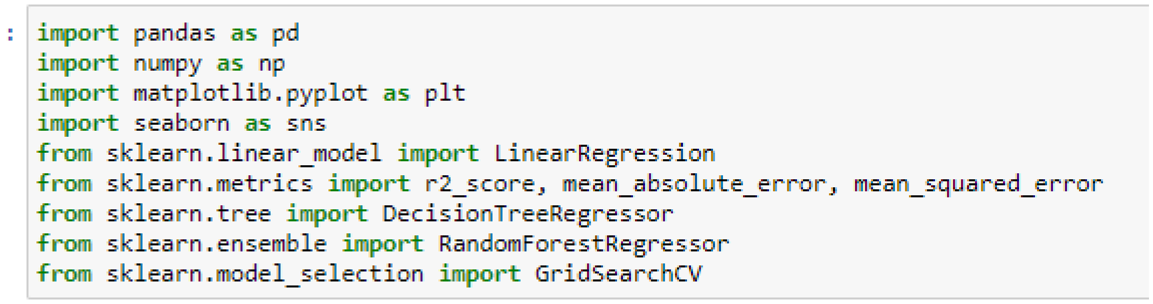
Preprocessing of the data actually involves the following steps:

4.1.1 Getting the dataset:

We can get the data set from the database.

4.1.2 Importing the libraries:

We have to import the libraries as per the requirement of the algorithm.



```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
```

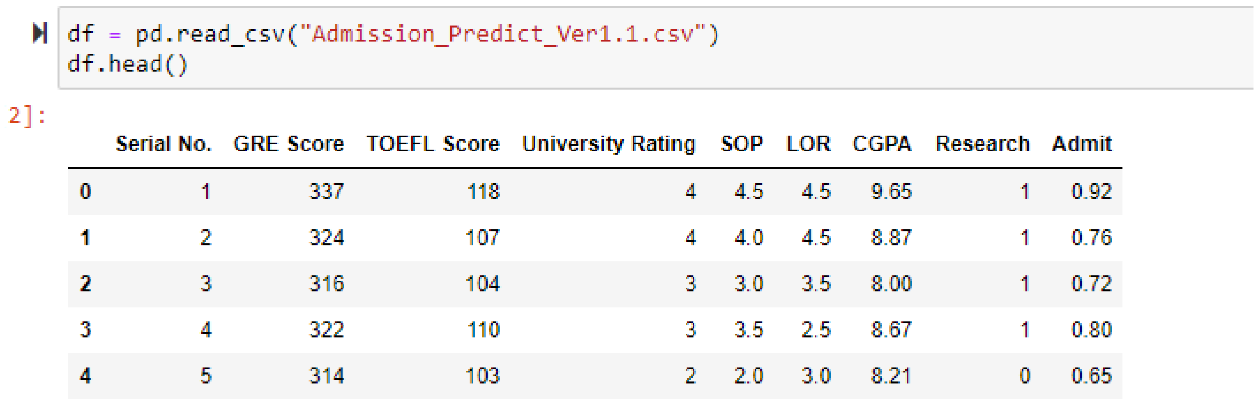
Fig 8:importing libraries

4.1.3 Importing data set:

We need to load data the data .The most common format for data loading is CSV(comma-seperated-values). Csv is a format to store data in tabular data such as spread sheet in plain text.

To load CSV data file is by *Pandas* and *pandas.read_csv()function*. This is the very flexible function that returns a pandas.DataFrame which can be used immediately for plotting.

Python script implemented by using head() function of Pandas DataFrame.



```
df = pd.read_csv("Admission_Predict_Ver1.1.csv")
df.head()
```

2]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Admit
0	1	337	118	4	4.5	4.5	9.65	1	0.92
1	2	324	107	4	4.0	4.5	8.87	1	0.76
2	3	316	104	3	3.0	3.5	8.00	1	0.72
3	4	322	110	3	3.5	2.5	8.67	1	0.80
4	5	314	103	2	2.0	3.0	8.21	0	0.65

Fig 9: Reading the data set

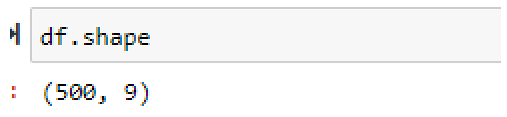
4.2 Exploratory Data Analysis:

4.2.1 Checking the dimensions of data:

To know how much data, in terms of rows and columns, we are having for our ML project. The reasons behind are −

- Suppose if we have too many rows and columns then it would take long time to run the algorithm and train the model.
- Suppose if we have too less rows and columns then it we would not have enough data to well train the model.

Following is a Python script implemented by printing the shape property on Pandas Data Frame. We are going to implement it on iris data set for getting the total number of rows and columns in it.



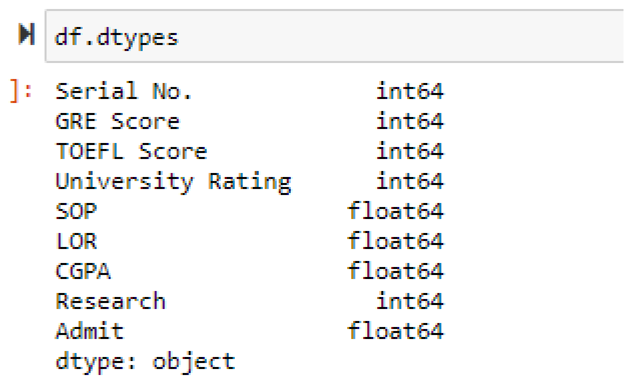
```
df.shape
```

```
: (500, 9)
```

Fig10: dimensions of data

4.2.2 Getting Each Attribute’s Data Type:

It is another good practice to know data type of each attribute. The reason behind is that, as per to the requirement, sometimes we may need to convert one data type to another. For example, we may need to convert string into floating point or in for representing categorical or ordinal values. We can have an idea about the attribute’s data type by looking at the raw data, but another way is to use *dtypes* property of Pandas DataFrame. With the help of *dtypes* property we can categorize each attributes data type.



```
df.dtypes
```

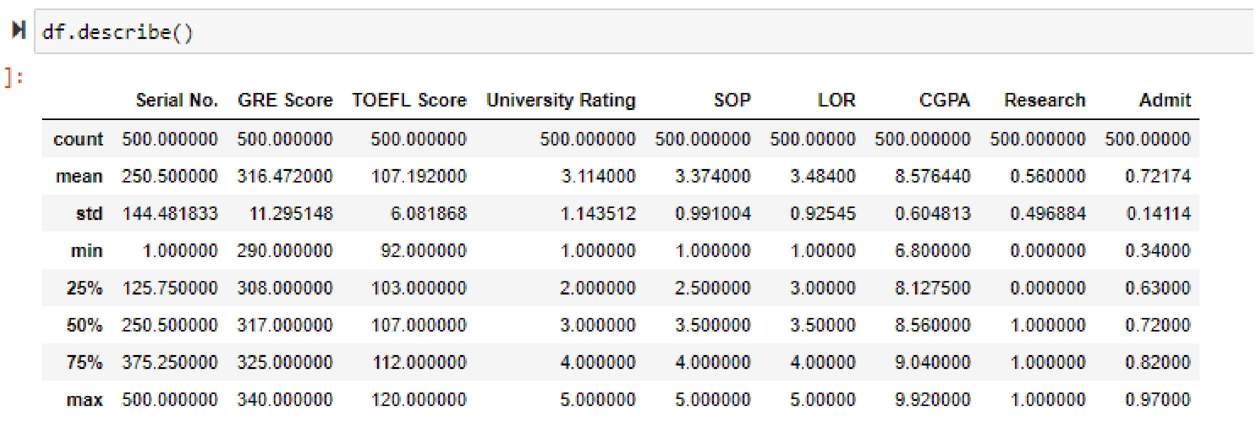
```
]: Serial No.             int64
   GRE Score              int64
   TOEFL Score            int64
   University Rating      int64
   SOP                  float64
   LOR                  float64
   CGPA                 float64
   Research               int64
   Admit                float64
   dtype: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

Fig 11: data types

4.2.3 Statistical summary data:

To get the shape i.e. number of rows and columns, of data but many times we need to review the summaries out of that shape of data. It can be done with the help of *describe()* function of Pandas DataFrame that further provide the following 8 statistical properties of each & every data attribute −

- Count
- Mean
- Standard Deviation
- Minimum Value
- Maximum value
- 25%
- Median i.e. 50%
- 75%



```
df.describe()
```

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	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

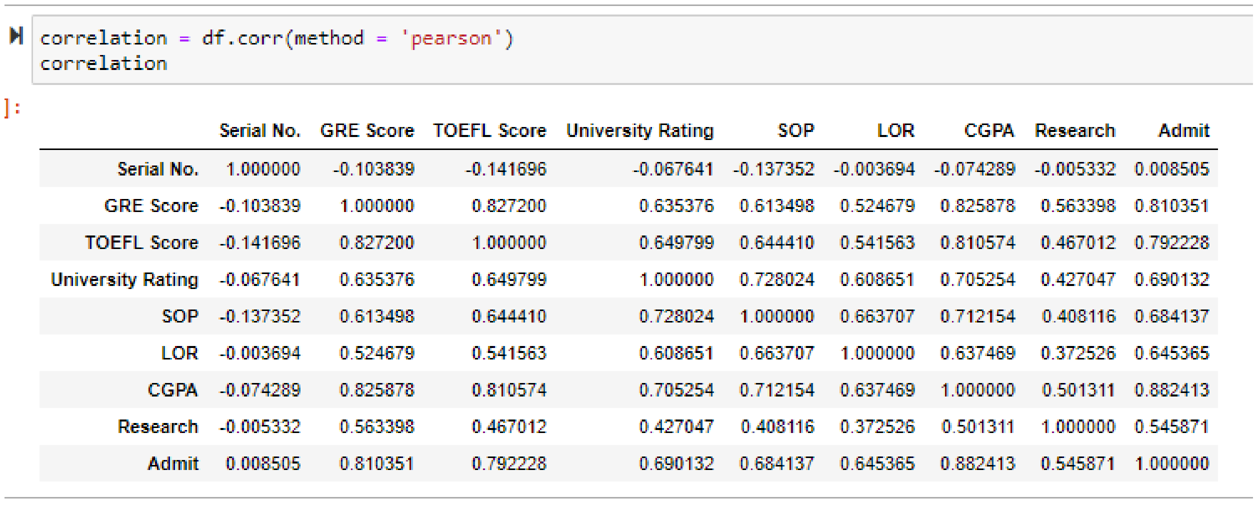
Fig 12: describe the data

4.2.4 **Correlation between attributes**:

The relationship between two variables is called correlation. In statistics, the most common method for calculating correlation is Pearson's Correlation Coefficient. It can have three values as follows −

- **Coefficient value = 1** − It represents full **positive** correlation between variables.
- **Coefficient value = -1** − It represents full **negative** correlation between variables.
- **Coefficient value = 0** − It represents **no** correlation at all between variables.

It is always good for us to review the pairwise correlations of the attributes in our dataset before using it into ML project because some machine learning algorithms such as linear regression and logistic regression will perform poorly if we have highly correlated attributes. In Python, we can easily calculate a correlation matrix of dataset attributes with the help of *corr()* function on Pandas DataFrame.



```
correlation = df.corr(method = 'pearson')
correlation
```

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	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
Admit	0.008505	0.810351	0.792228	0.690132	0.684137	0.645365	0.882413	0.545871	1.000000

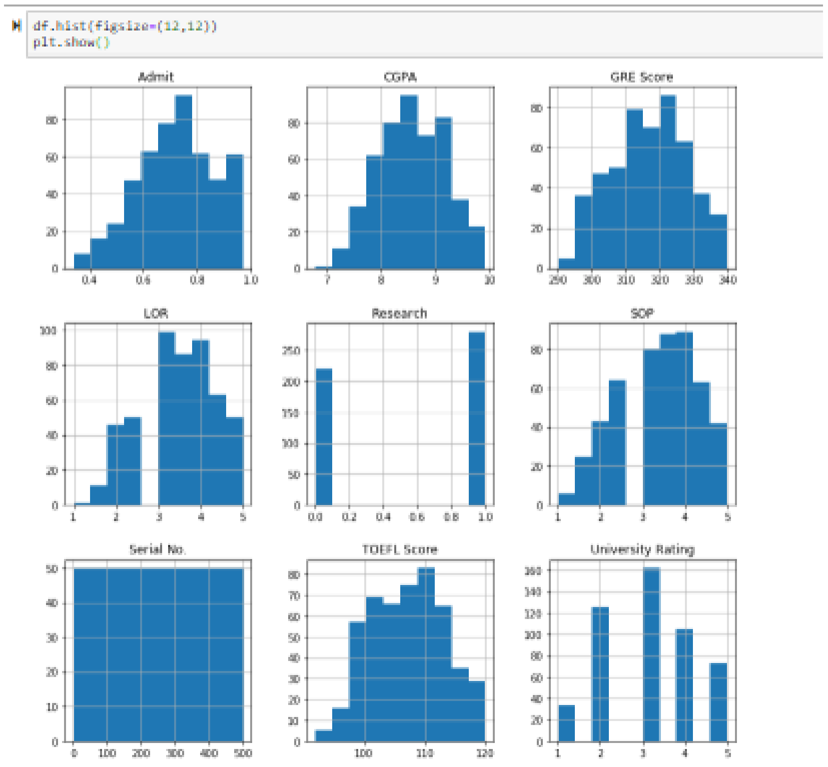
Fig13: correlation between attributes

4.3 Data Visualization:

4.3.1 Univariate plots:

4.3.1.1 Histograms:

Histograms group the data in bins and is the fastest way to get idea about the distribution of each attribute in dataset.

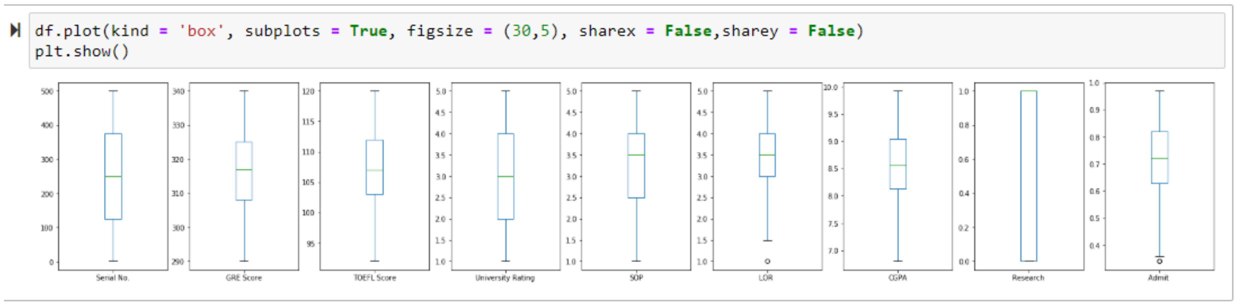


```
df.hist(figsize=(12,12))
plt.show()
```

Fig14: Histograms

4.3.1.2 Box plots:

Useful technique to review the distribution of each attribute's distribution.



```
df.plot(kind = 'box', subplots = True, figsize = (30,5), sharex = False,sharey = False)
plt.show()
```

Fig 15: Box plots

4.3.2 Multivariate plots:

4.3.2.1 Correlation Matrix plot:

Correlation is an indication about the changes between two variables.

The heatmap is a way of representing the data in a 2-dimensional form. The data values are represented as colors in the graph. The goal of the heatmap is to provide a colored visual summary of information.

```
corelation = df.corr()
```

```
sns.heatmap(corelation, xticklabels=corelation.columns, yticklabels=corelation.columns, annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2490d654208>
```

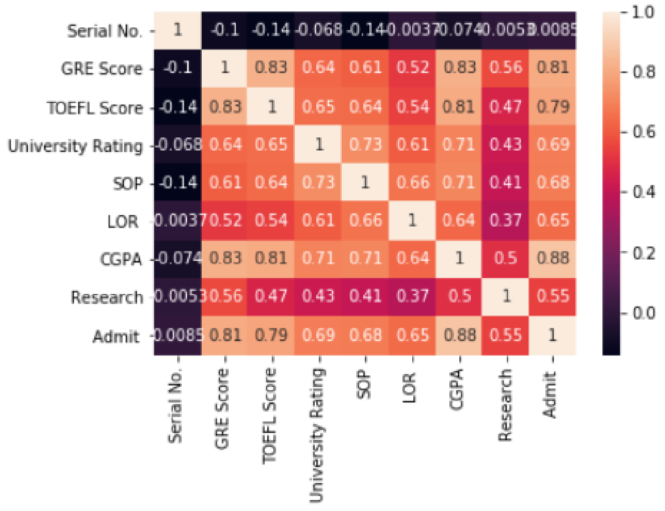


Fig 16: correlation matrix plot

4.3.2.2 Scatter Matrix plot:

Scatter plots shows how much one variable is affected by another or the relationship between them with the help of dots in two dimensions.

A pairs plot allows us to see both distribution of single variables and relationships between two variables. The pairs plot builds on two figures, the histogram and the scatter plot. The histogram allows us to see the distribution of a single variable while the scatter plots shows the relationship between two variables.

```
sns.pairplot(df)
```

```
<seaborn.axisgrid.PairGrid at 0x2490d2eb388>
```

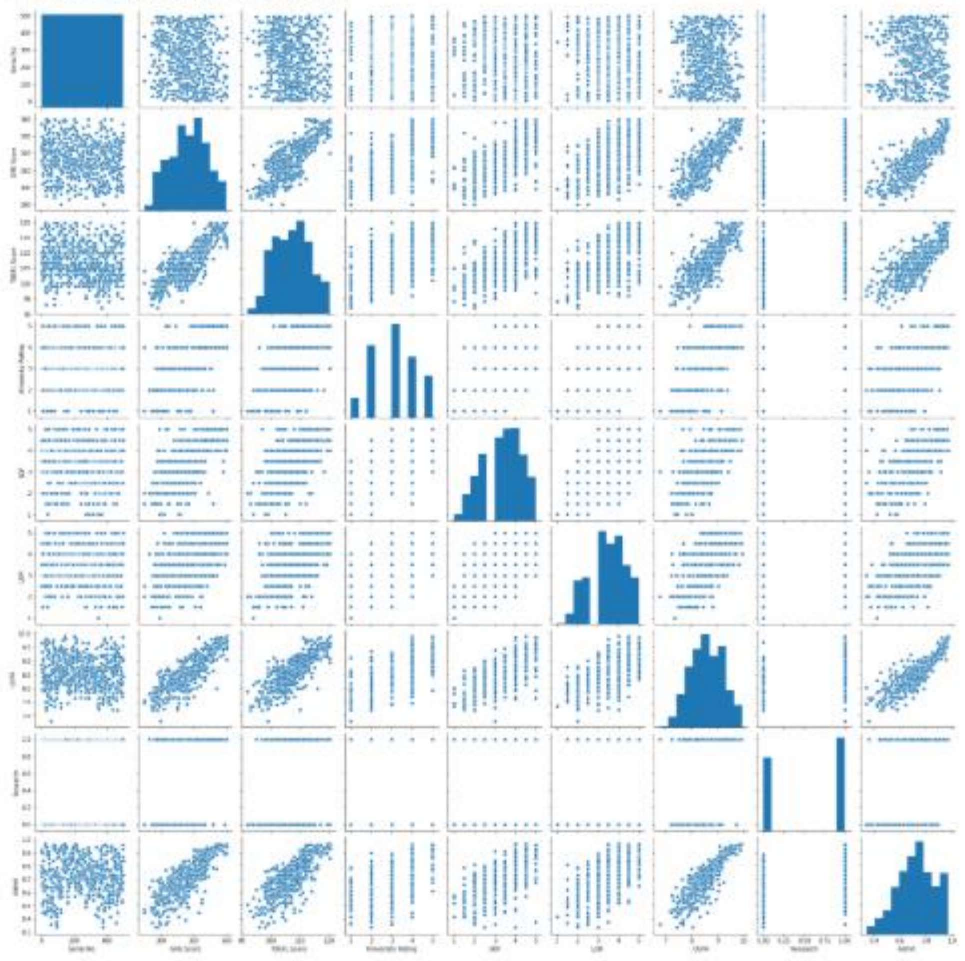
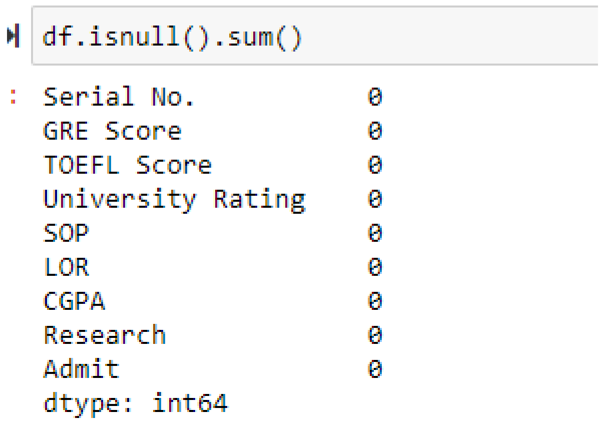


Fig 17: scatter matrix plot

4.4 Data Cleaning:

4.4.1 To check Null values:

In, this data set we have no missing values.



```
df.isnull().sum()
```

```
Serial No.           0
GRE Score            0
TOEFL Score          0
University Rating    0
SOP                  0
LOR                  0
CGPA                 0
Research             0
Admit                0
dtype: int64
```

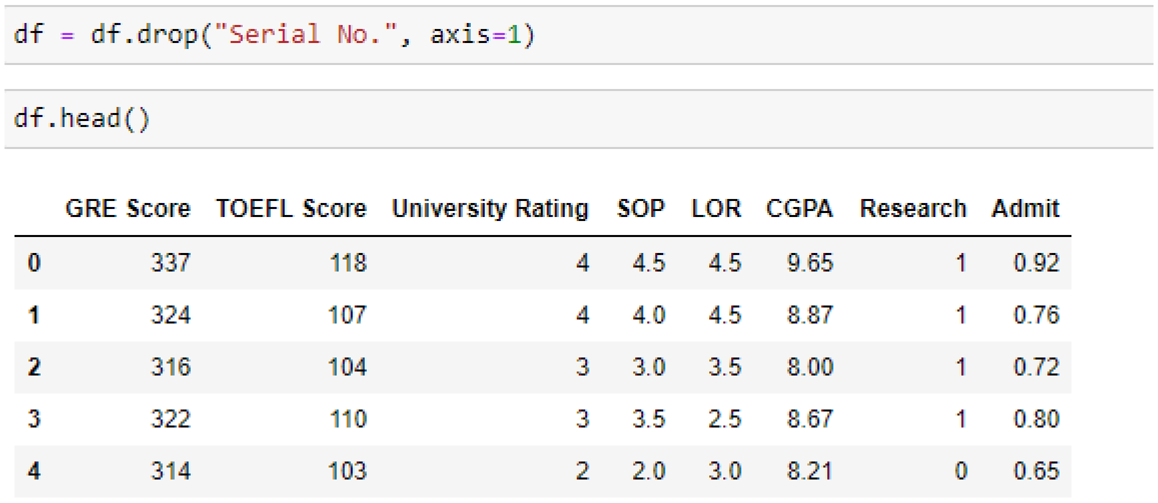
Fig18: finding out null values

4.5 Data selection:

4.5.1 Dropping irrelevant columns:

Removing the columns which are not required using drop method.

Serial No. is just an index for students, which we can drop it.



```
df = df.drop("Serial No.", axis=1)
```

```
df.head()
```

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Admit
0	337	118	4	4.5	4.5	9.65	1	0.92
1	324	107	4	4.0	4.5	8.87	1	0.76
2	316	104	3	3.0	3.5	8.00	1	0.72
3	322	110	3	3.5	2.5	8.67	1	0.80
4	314	103	2	2.0	3.0	8.21	0	0.65

Fig 19: dropping the columns

4.6 Pick an Algorithm:

4.6.1 Linear Regression:

Linear regression may be defined as the statistical model that analyzes the linear relationship between a dependent variable with given set of independent variables. Linear relationship between variables means that when the value of one or more independent variables will change (increase or decrease), the value of dependent variable will also change accordingly (increase or decrease).

4.6.2 Decision Tree Regression:

Decision tree builds regression or classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes

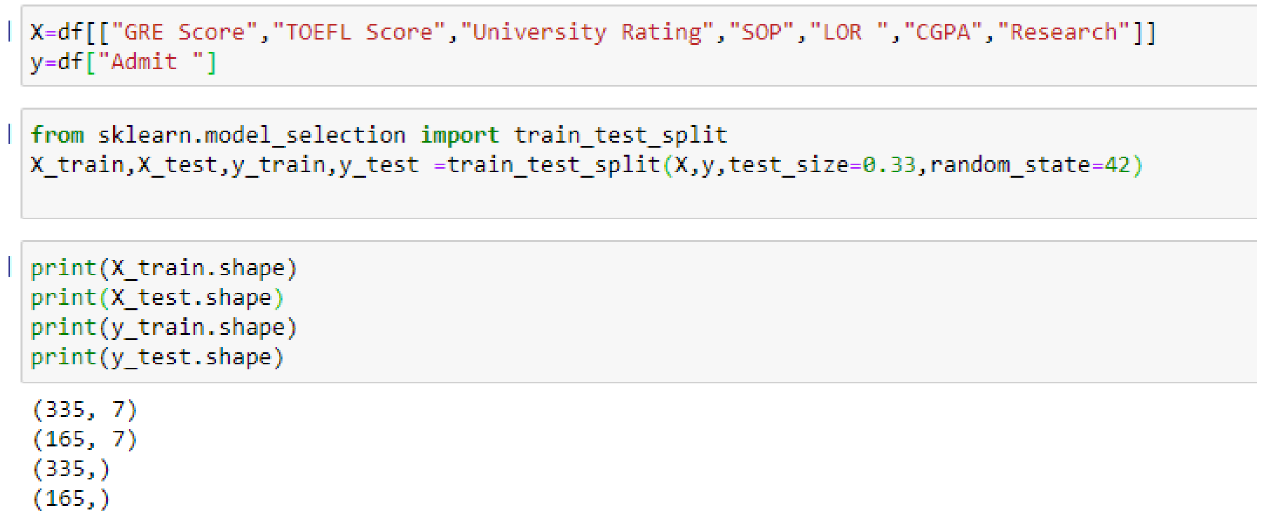
4.6.3 Random forest Regression:

A Random Forest is an ensemble technique capable of performing both regression and classification tasks with the use of multiple decision trees and a technique called Bootstrap and Aggregation, commonly known as bagging. The basic idea behind this is to combine multiple decision trees in determining the final output rather than relying on individual decision trees.

CHAPTER 5: Train the model:

- Import the train_test_split from model_selection package from scikitlearn library

- Then assigning the output to four different variables, before assigning we have to mention the train size or test size as a parameter to train_test_split.



```
X=df[["GRE Score","TOEFL Score","University Rating","SOP","LOR ","CGPA","Research"]]
y=df["Admit "]
```

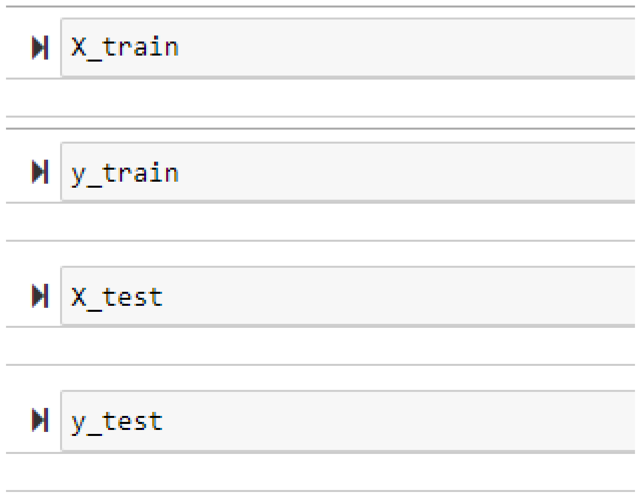
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test =train_test_split(X,y,test_size=0.33,random_state=42)
```

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(335, 7)
(165, 7)
(335,)
(165,)
```

Fig 20: Splitting the data

- Then this method will split according to the size and assigns it to four variables



```
X_train
```

```
y_train
```

```
X_test
```

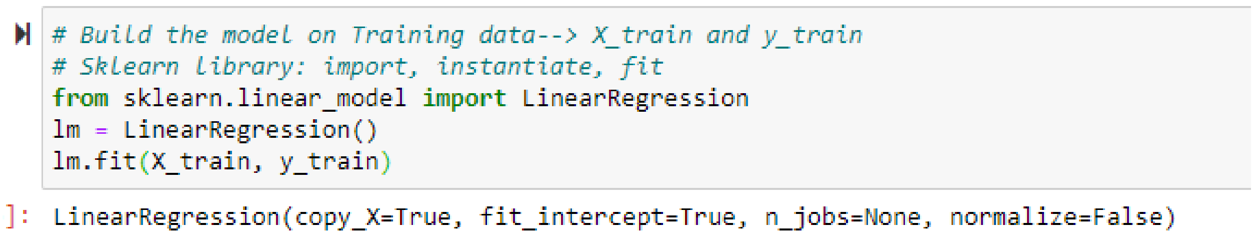
```
y_test
```

Fig 21: Training and testing data

CHAPTER 6 :Model building and evaluation:

6.1 Linear Regression:

- Import linear regression method which is available in linear_model package from scikit learn library

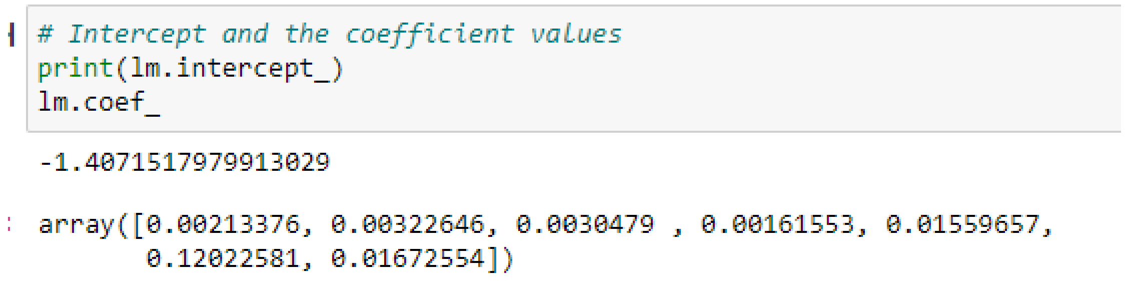


```
# Build the model on Training data--> X_train and y_train
# Sklearn library: import, instantiate, fit
from sklearn.linear_model import LinearRegression
lm = LinearRegression()
lm.fit(X_train, y_train)
```

```
]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

Fig22: importing linear regression

- Extracting the intercept and the coefficient.



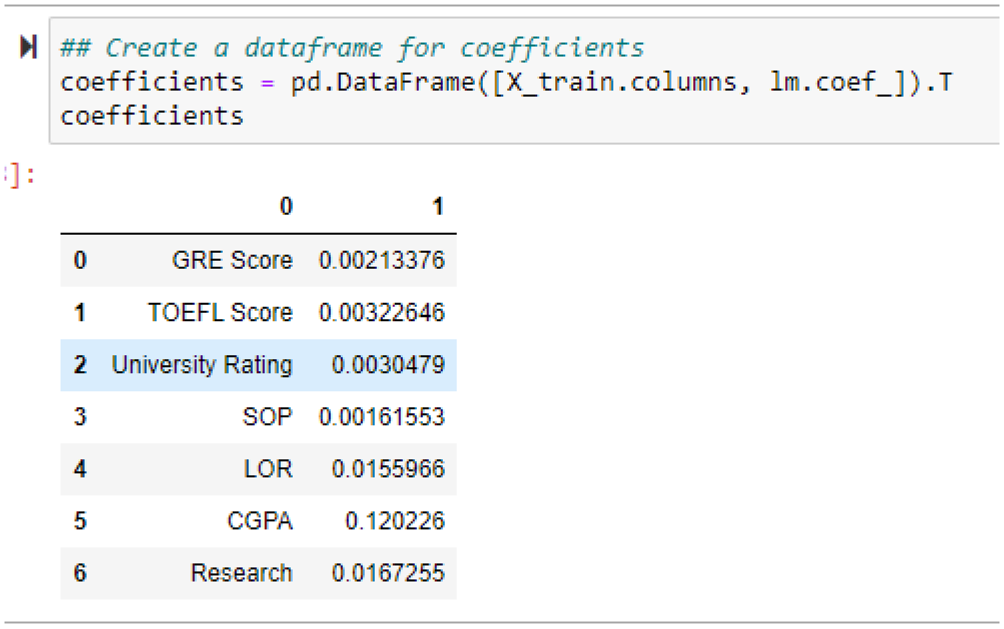
```
# Intercept and the coefficient values
print(lm.intercept_)
lm.coef_
```

```
-1.4071517979913029
```

```
array([0.00213376, 0.00322646, 0.0030479 , 0.00161553, 0.01559657,
       0.12022581, 0.01672554])
```

Fig23: Extracting the intercept and coefficient

- Creating a data frame for coefficient

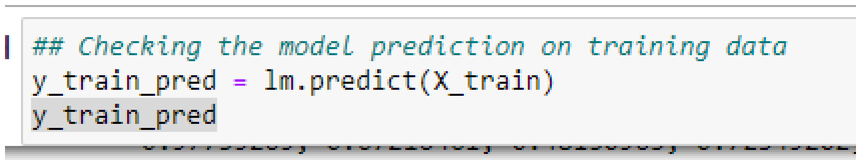


```
## Create a dataframe for coefficients
coefficients = pd.DataFrame([X_train.columns, lm.coef_]).T
coefficients
```

	0	1
0	GRE Score	0.00213376
1	TOEFL Score	0.00322646
2	University Rating	0.0030479
3	SOP	0.00161553
4	LOR	0.0155966
5	CGPA	0.120226
6	Research	0.0167255

Fig 24: data frame for coefficients

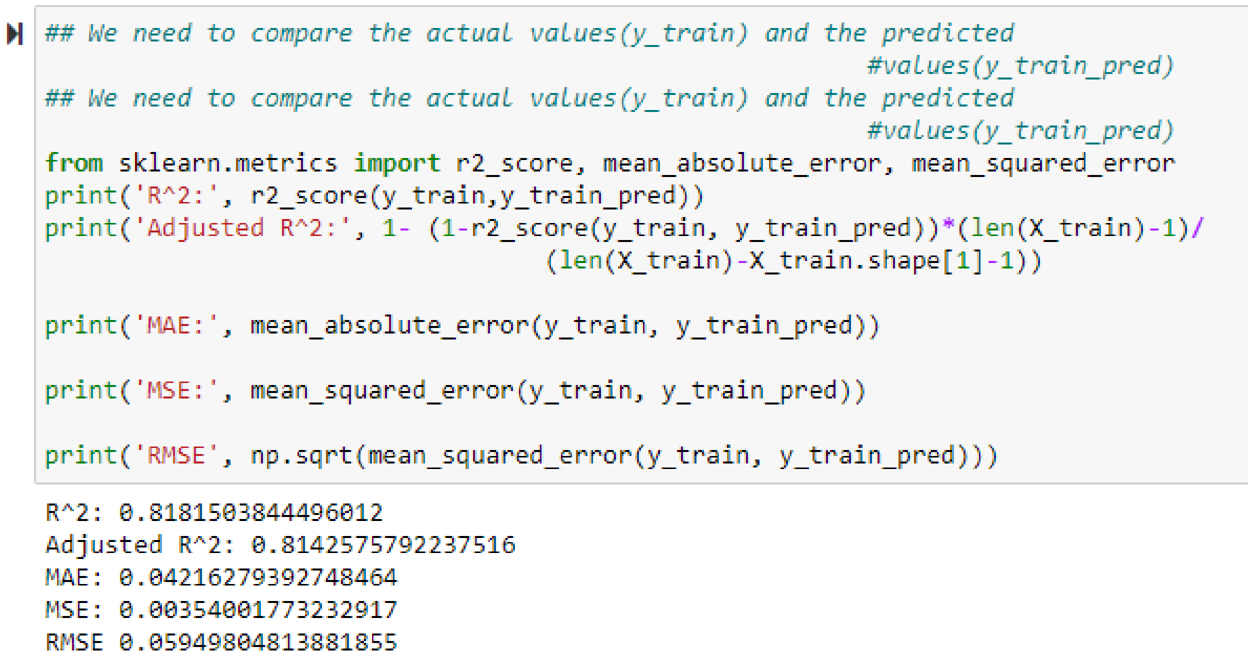
- Checking the model prediction on training data



```
## Checking the model prediction on training data
y_train_pred = lm.predict(X_train)
y_train_pred
```

Fig 25: Predicting the output for training data

- We need to compare the actual values(y_train) and the predicted values(y_train_pred)
- We need to compare the actual values(y_train) and the predicted values(y_train_pred)



```
## We need to compare the actual values(y_train) and the predicted
                                                       #values(y_train_pred)
## We need to compare the actual values(y_train) and the predicted
                                                       #values(y_train_pred)
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
print('R^2:', r2_score(y_train,y_train_pred))
print('Adjusted R^2:', 1- (1-r2_score(y_train, y_train_pred))*(len(X_train)-1)/
                                  (len(X_train)-X_train.shape[1]-1))

print('MAE:', mean_absolute_error(y_train, y_train_pred))

print('MSE:', mean_squared_error(y_train, y_train_pred))

print('RMSE', np.sqrt(mean_squared_error(y_train, y_train_pred)))
```

```
R^2: 0.8181503844496012
Adjusted R^2: 0.8142575792237516
MAE: 0.04216279392748464
MSE: 0.00354001773232917
RMSE 0.05949804813881855
```

Fig 26: Metrics for training the data

- Relation plot for chance of admit and predicted values.

```
## regplot(y_train--> actual values and y_train_pred--> predicted values)
sns.regplot(y_train, y_train_pred)
```

```
]: <matplotlib.axes._subplots.AxesSubplot at 0x134f7286ec8>
```

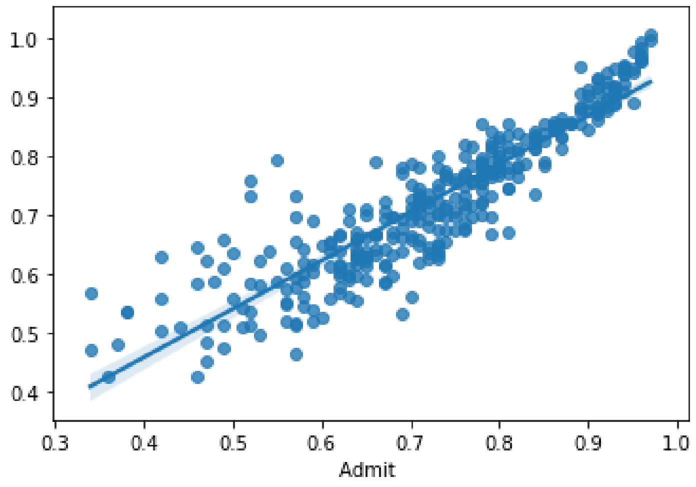
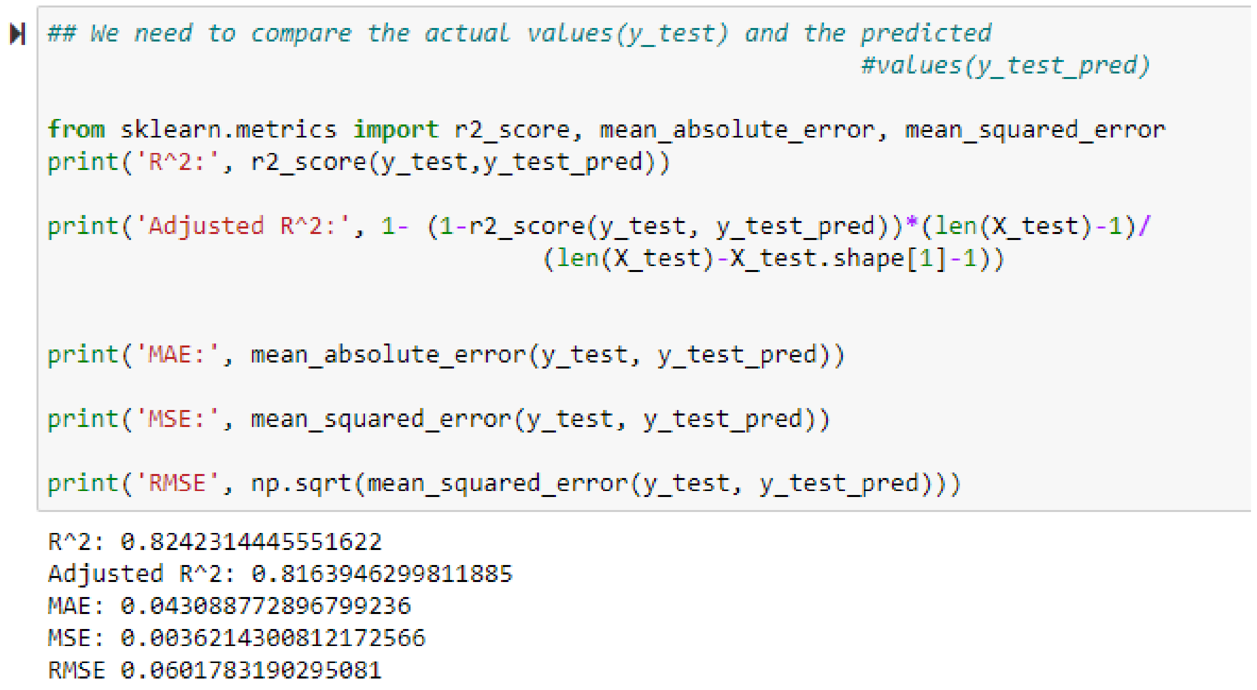


Fig 27: relation plot for chance of admit and predicted values

```
## Test the model on testing data
y_test_pred = lm.predict(X_test)  # test data--> unseen data
y_test_pred
```

Fig 28: predicting output for testing data

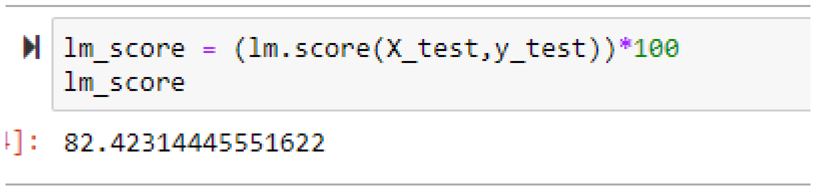


```
## We need to compare the actual values(y_test) and the predicted
                                                        #values(y_test_pred)

from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
print('R^2:', r2_score(y_test,y_test_pred))

print('Adjusted R^2:', 1- (1-r2_score(y_test, y_test_pred))*(len(X_test)-1)/
                                  (len(X_test)-X_test.shape[1]-1))


print('MAE:', mean_absolute_error(y_test, y_test_pred))

print('MSE:', mean_squared_error(y_test, y_test_pred))

print('RMSE', np.sqrt(mean_squared_error(y_test, y_test_pred)))
```

```
R^2: 0.8242314445551622
Adjusted R^2: 0.8163946299811885
MAE: 0.043088772896799236
MSE: 0.0036214300812172566
RMSE 0.0601783190295081
```

Fig 29: Metrics for testing data

- Accuracy score for linear regression



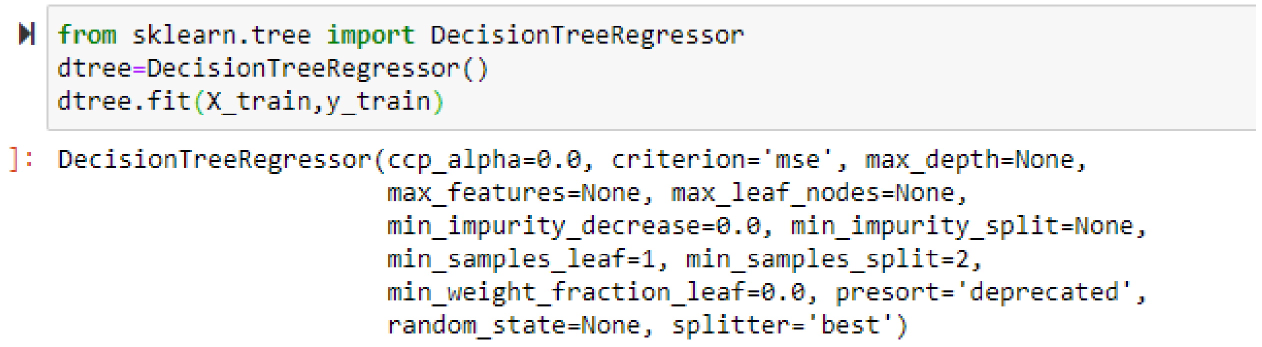
```
lm_score = (lm.score(X_test,y_test))*100
lm_score
```

```
]: 82.42314445551622
```

Fig30: Accuracy core for linear regression

6.2 Decision Tree Regression:

- Import decision tree regressor method which is available in tree package from scikit learn library

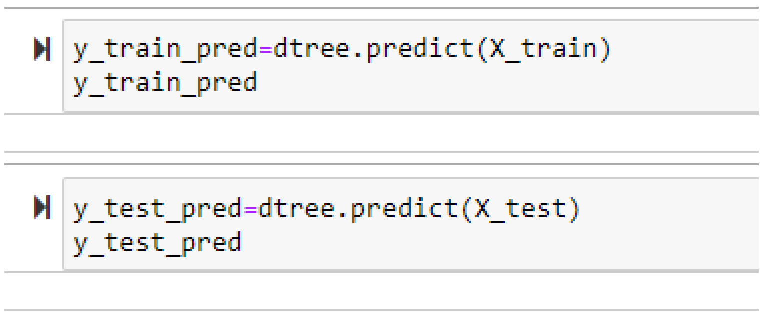


```
from sklearn.tree import DecisionTreeRegressor
dtree=DecisionTreeRegressor()
dtree.fit(X_train,y_train)
```

```
]: DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=None,
                         max_features=None, max_leaf_nodes=None,
                         min_impurity_decrease=0.0, min_impurity_split=None,
                         min_samples_leaf=1, min_samples_split=2,
                         min_weight_fraction_leaf=0.0, presort='deprecated',
                         random_state=None, splitter='best')
```

Fig 31: Importing Decision Tree

- Checking the model prediction on training and testing data

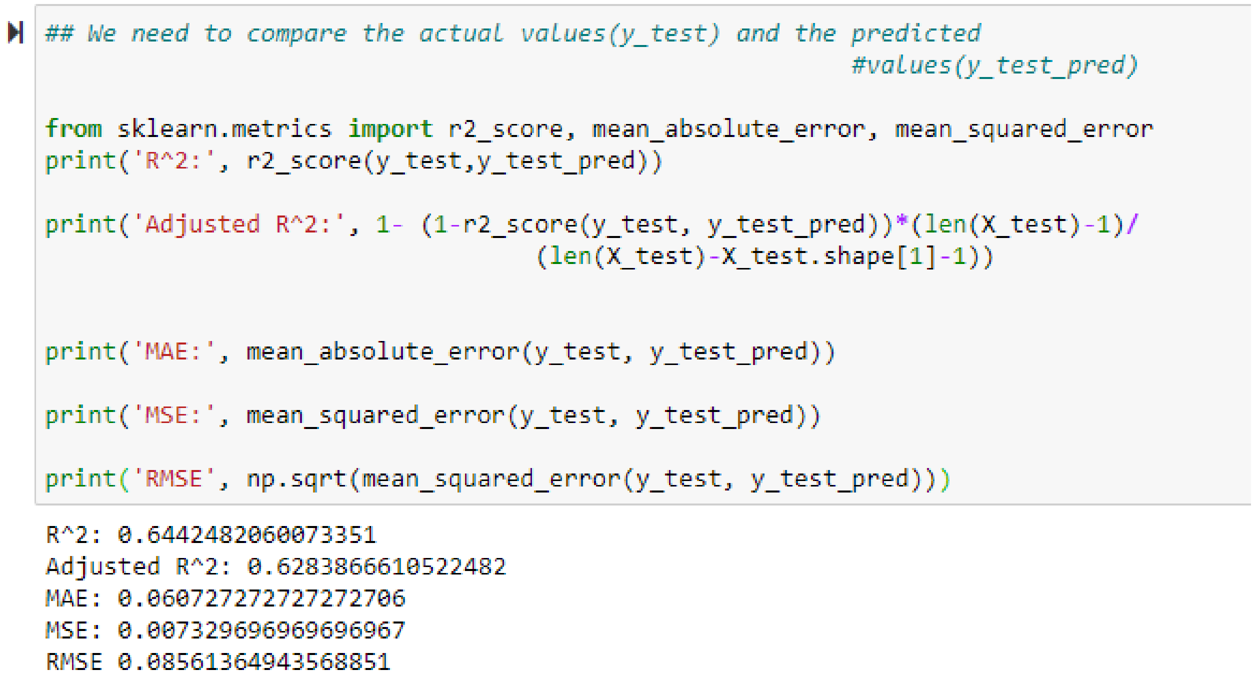


```
y_train_pred=dtree.predict(X_train)
y_train_pred
```

```
y_test_pred=dtree.predict(X_test)
y_test_pred
```

Fig 32 predicted output for training and testing data

- We need to compare the actual values(y_train) and the predicted values(y_train_pred)
- We need to compare the actual values(y_train) and the predicted values(y_train_pred)

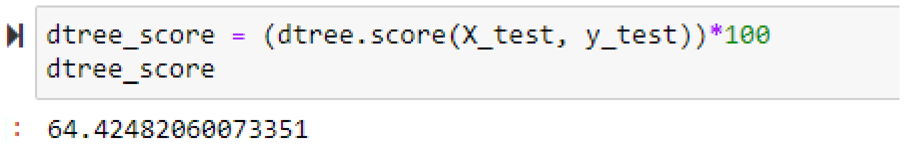


```
## We need to compare the actual values(y_test) and the predicted
                                                    #values(y_test_pred)

from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
print('R^2:', r2_score(y_test,y_test_pred))

print('Adjusted R^2:', 1- (1-r2_score(y_test, y_test_pred))*(len(X_test)-1)/
                              (len(X_test)-X_test.shape[1]-1))


print('MAE:', mean_absolute_error(y_test, y_test_pred))

print('MSE:', mean_squared_error(y_test, y_test_pred))

print('RMSE', np.sqrt(mean_squared_error(y_test, y_test_pred)))
```

```
R^2: 0.6442482060073351
Adjusted R^2: 0.6283866610522482
MAE: 0.060727272727272706
MSE: 0.007329696969696967
RMSE 0.08561364943568851
```

Fig33: Metrics for testing data

- Accuracy score for decision tree



```
dtree_score = (dtree.score(X_test, y_test))*100
dtree_score
```

```
: 64.42482060073351
```

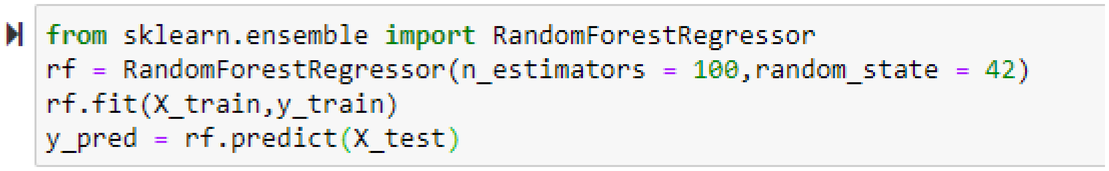
Fig34: Accuracy score for Decision tree

6.3 Random Forest Regression:

Import random forest regressor method which is available in ensemble package from scikit learn library

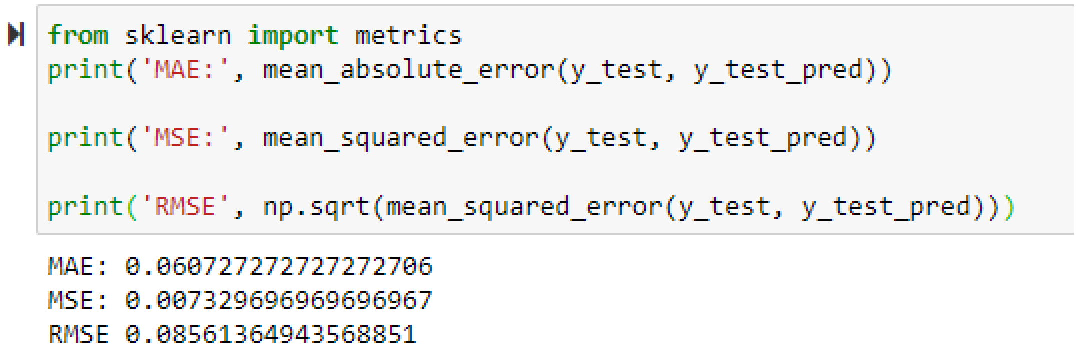
RFs train each tree independently, using a random sample of the data. This randomness helps to make the model more robust than a single decision tree, and less likely to over fit on the training data.

n_estimators is the number of trees to be used in the forest. Since Random Forest is an ensemble method comprising of creating multiple decision trees, this parameter is used to control the number of trees to be used in the process.



```
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(n_estimators = 100,random_state = 42)
rf.fit(X_train,y_train)
y_pred = rf.predict(X_test)
```

Fig 35: Importing random forest regression

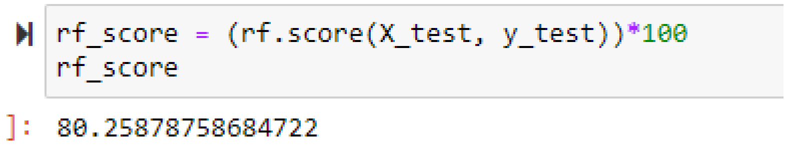


```
from sklearn import metrics
print('MAE:', mean_absolute_error(y_test, y_test_pred))

print('MSE:', mean_squared_error(y_test, y_test_pred))

print('RMSE', np.sqrt(mean_squared_error(y_test, y_test_pred)))
```

```
MAE: 0.060727272727272706
MSE: 0.007329696969696967
RMSE 0.08561364943568851
```

Fig36: metrics for Random Forest Regression

- Accuracy score for random forest

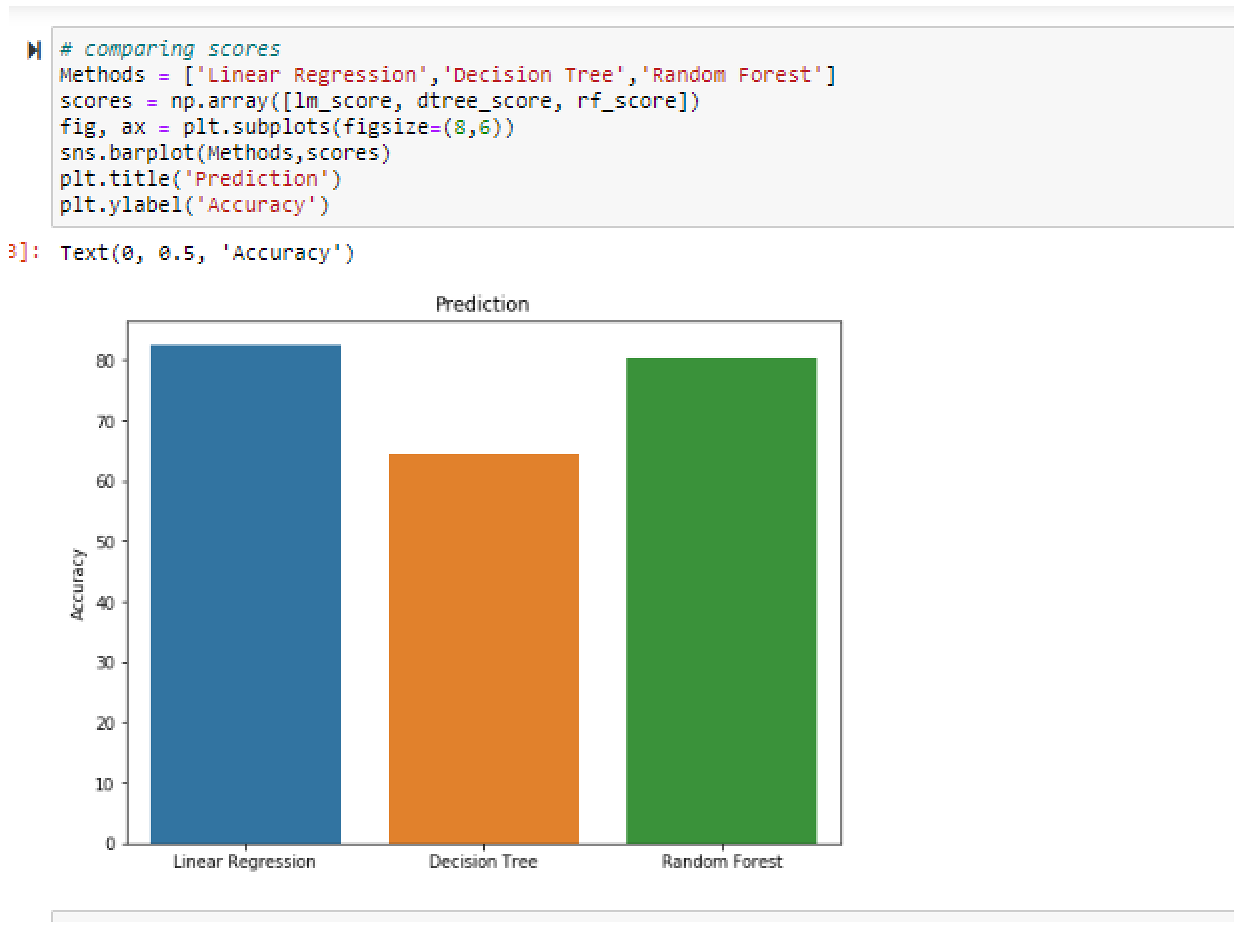


```
rf_score = (rf.score(X_test, y_test))*100
rf_score
```

```
]:  80.25878758684722
```

Fig 37:Accuracy score for Random forest

6.4 Selecting best algorithm



```
# comparing scores
Methods = ['Linear Regression','Decision Tree','Random Forest']
scores = np.array([lm_score, dtree_score, rf_score])
fig, ax = plt.subplots(figsize=(8,6))
sns.barplot(Methods,scores)
plt.title('Prediction')
plt.ylabel('Accuracy')
```

```
3]: Text(0, 0.5, 'Accuracy')
```

Fig38: comparing accuracy scores

1. Linear Regression – 82.42%
2. Random Forests – 80.25%
3. Decision Trees – 64.42%

It seems that Linear Regression is the most accurate of the 3 methods and will be used to predict the future applicant's chances of admission.

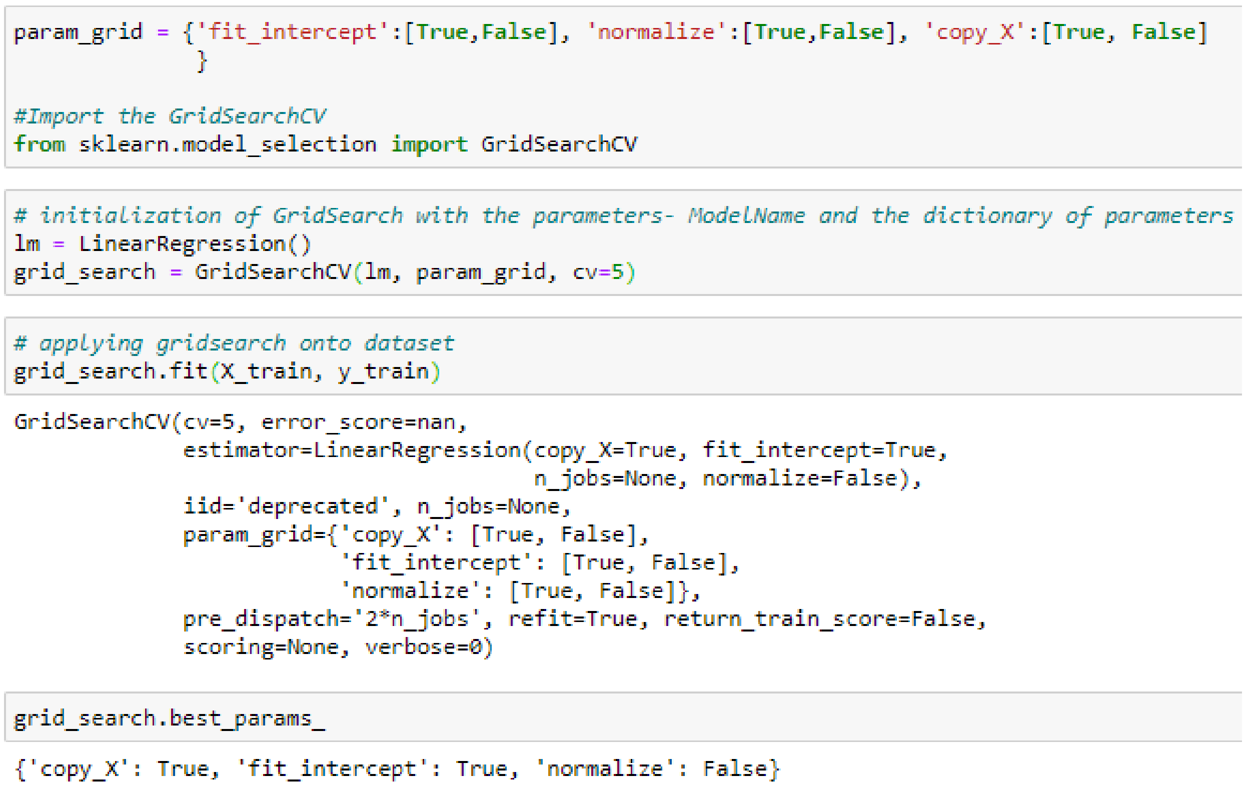
CHAPTER 7 HYPERPARAMETER TUNING:

A hyperparameter is a parameter whose value is set before the learning process begins.
Hyperparameter tuning is also tricky in the sense that there is no direct way to calculate how a change in the hyperparameter value will reduce the loss of your model, so we usually resort to experimentation. This starts with us specifying a range of possible values for all the hyperparameters. Now, this is where most get stuck, what values you are going to try, and to answer that question, you first need to understand what these hyperparameters mean and how changing a hyperparameter will affect your model architecture, thereby try to understand how your model performance might change.
The next step after you define the range of values is to use a hyperparameter tuning method, there's a bunch, the most common and expensive being Grid Search

Grid Search CV:

Grid search is a traditional way to perform hyperparameter optimization. It works by searching exhaustively through a specified subset of hyperparameters.
Grid search is the process of performing hyper parameter tuning in order to determine the optimal values for a given model. This is significant as the performance of the entire model is based on the hyper parameter values specified.
Using sklearn's GridSearchCV, we first define our grid of parameters to search over and then run the grid search.

- Hyperparameter Tuning for Linear Regression:



```
param_grid = {'fit_intercept':[True,False], 'normalize':[True,False], 'copy_X':[True, False]
              }

#Import the GridSearchCV
from sklearn.model_selection import GridSearchCV
```

```
# initialization of GridSearch with the parameters- ModelName and the dictionary of parameters
lm = LinearRegression()
grid_search = GridSearchCV(lm, param_grid, cv=5)
```

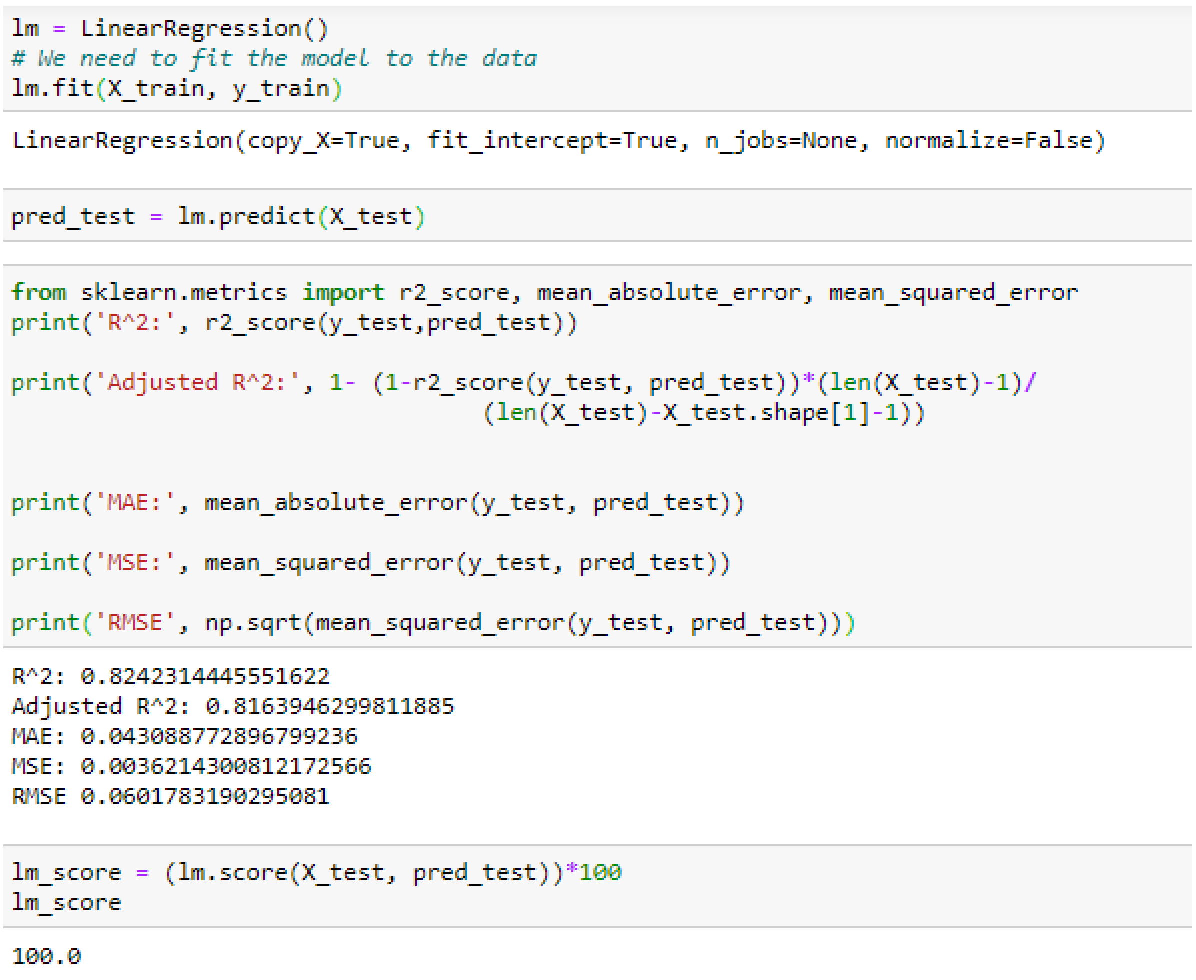
```
# applying gridsearch onto dataset
grid_search.fit(X_train, y_train)
```

```
GridSearchCV(cv=5, error_score=nan,
             estimator=LinearRegression(copy_X=True, fit_intercept=True,
                                        n_jobs=None, normalize=False),
             iid='deprecated', n_jobs=None,
             param_grid={'copy_X': [True, False],
                         'fit_intercept': [True, False],
                         'normalize': [True, False]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

```
grid_search.best_params_
```

```
{'copy_X': True, 'fit_intercept': True, 'normalize': False}
```

Fig39: Hyper parameter tuning for Linear Regression



```
lm = LinearRegression()
# We need to fit the model to the data
lm.fit(X_train, y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

```
pred_test = lm.predict(X_test)
```

```
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
print('R^2:', r2_score(y_test,pred_test))

print('Adjusted R^2:', 1- (1-r2_score(y_test, pred_test))*(len(X_test)-1)/
                                  (len(X_test)-X_test.shape[1]-1))


print('MAE:', mean_absolute_error(y_test, pred_test))

print('MSE:', mean_squared_error(y_test, pred_test))

print('RMSE', np.sqrt(mean_squared_error(y_test, pred_test)))
```

```
R^2: 0.8242314445551622
Adjusted R^2: 0.8163946299811885
MAE: 0.043088772896799236
MSE: 0.0036214300812172566
RMSE 0.0601783190295081
```

```
lm_score = (lm.score(X_test, pred_test))*100
lm_score
```

```
100.0
```

Fig 40: Accuracy score (after grid search)

```
# comparing scores
Methods = ['Linear Regression','Decision Tree','Random Forest']
scores = np.array([lm_score, dtree_score, rf_score])
fig, ax = plt.subplots(figsize=(8,6))
sns.barplot(Methods,scores)
plt.title('Prediction')
plt.ylabel('Accuracy')
```

```
Text(0, 0.5, 'Accuracy')
```

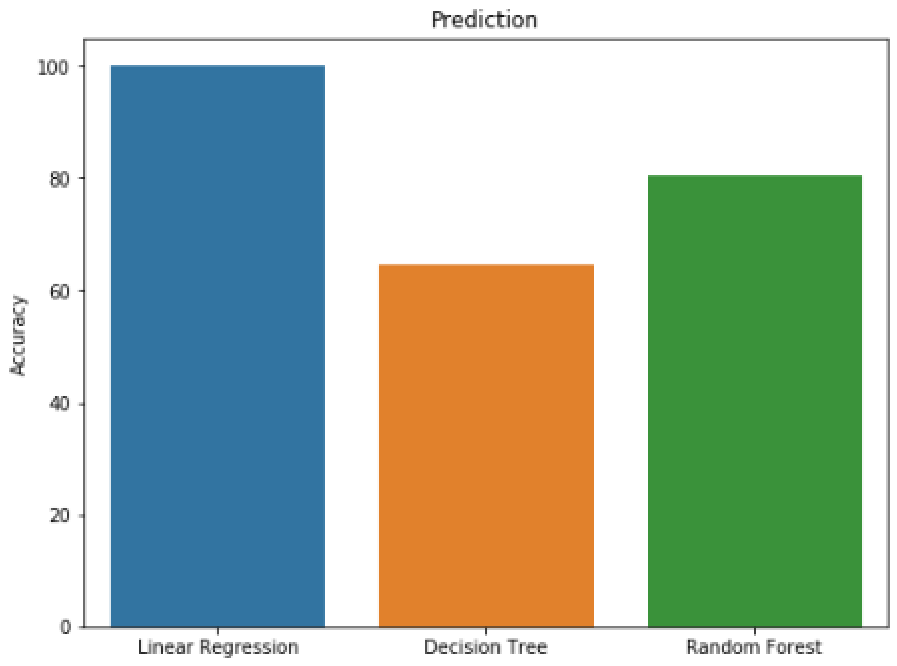


Fig 41: comparing scores (after grid search)

CONCLUSION:

It is concluded after performing thorough Exploratory Data analysis which includes Stats models which are computed to get accuracy and also Heat maps which are computed to get a clear understanding of the data set. All three methods and will be used to predict the future applicants chance of admission.

References:

https://www.tutorialspoint.com/machine_learning_with_python/index.htm

https://www.kaggle.com/adepvenugopal/predicting-graduate-admissions-using-ml