E-ISSN 2581 – 7957
P-ISSN 2277 – 3916

CVR Journal of Science and Technology, Volume 24, June 2023
DOI: 10.32377/cvrjst2411

Implementing a Linear Regression Gradient Descent Model for Admission Process Framework

Ajeet K. Jain[1] and K. Venkatesh Sharma[2]
[1]Asst. Prof., Keshav Memorial Institute of Technology/ CSE Department, Hyderabad, India
Email: jainajeet123@gmail.com
[2]Professor, CVR College of Engineering/CSE Department, Hyderabad, India
Email: venkateshsharma.cse@gmail.com

Abstract: **Machine Learning (ML) is the fundamental learning paradigm in the scientific community having a wide range of applications in vivid domains. The hidden underlying patterns in data can be easily identified with use of popular ML algorithms. The meaningful pattern provides insight information extracted from the data. In so doing, human incapability hinders the process of recognising meaningful patterns in the given data sets. Such fine exemplary thoughts are given to machines with suitable algorithms and it can detect not only the finer patterns, but also provides meaningfulness of data spread in the domain. The area of ML is a blend of mathematics, probability, statistics and allied sciences in an articulated way and thus endows the ability to "learn *and adapt*". A generalized Gradient Descent (GD) based model is proposed which can be implemented on any dataset. The model is tested to forecast a student's admission on his (her) GRE score. Proposed model reflects good accuracy and the Pearson Correlation coefficient suggests the pertinent relationship among different attributes. The model also focuses the underlying mathematical derivation to a minimum to comprehend.**

Index Terms: **Linear Regression, Gradient Descent, RMS Error, MSE, ERM, Pearson Correlation**

I. Introduction

Linear regression finds a relationship between a dependent variable for a given set of independent variables—also known as a relationship involving *explanatory variables* and some real estimated *outcome*. The traditional straight-line equation **y = m x + b** ; is used to estimate '**y**' for a given set of '**x**' with bias matrix **'b'** and the domain X consists of $\mathbb{R}^{\mathbf{d}}$ and the label set **y** ϵ **Y** is the set of real numbers for a given **d** [1]. A linear function $\boldsymbol{h} : \mathbb{R}^{d} \mapsto \mathbb{R}$ that suitably estimates the relationship between given variables—for instance, predicting probability of getting the admission as a function of GRE score. [2] This, when combined with standard GD mechanism, can provide a fairly good idea how the blending works in ML for a given application domain and the generalized model is shown in Figure 1.

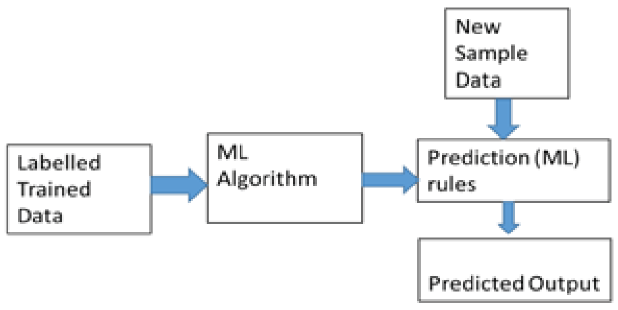


Figure 1. A Generic ML GD model framework

A. Linear Regression

A linear regressor is basically the set of linear functions [2]:

$$H_{reg} = L_d = \{x \mapsto \langle w, x \rangle + b : w \in R^d, b \in R \tag{1}$$

Here $\boldsymbol{H_{reg}}$ is a regression function; **x** is input space, and b as bias. Intuitively, to lessen the difference between actual and expected values, [3] we define a loss function calculating the discrepancy of values while using a regressor, in Figure 2. The most generally used squared-loss function is given by:

$$l\ (h, (x, y)) = (h(x) - y)^{2} \tag{2}$$

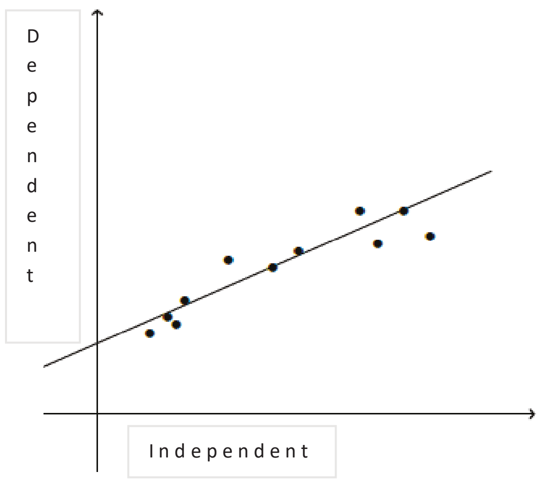


Figure 2. Linear regression showing variables.

Received on 19-01-2023, Revised on 07-03-2023, Accepted on 11-05-2023.

For the observed values, the loss is usually calculated as Mean Squared Error (MSE):

$$Ls(h) = \frac{1}{m}\sum_{1}^{m}(h(xi) - yi)^{2} \qquad (3)$$

[4] where the meaning of all the symbols used are intuitive. Such an algorithm solves the expected risk minimization (ERM) problem for linear regression predictors.

[5] The solution to this equation is to find the derivative, i.e., gradient of the objective function and compare it to zero:

$$\frac{2}{m}\sum_{i=1}^{m}(\langle \mathbf{w}, \mathbf{x}_i\rangle - y_i)\mathbf{x}_i = 0 \qquad (4)$$

This can be simplified in matrix notation form as

Aw = **b** where

$$A = \left(\sum_{i=1}^{m} \mathrm{x}_i \mathrm{x}_i^{\top}\right) \quad \text{and} \quad \mathrm{b} = \sum_{i=1}^{m} y_i \mathrm{x}_i. \qquad (5)$$

In other words, in the matrix notation form as:

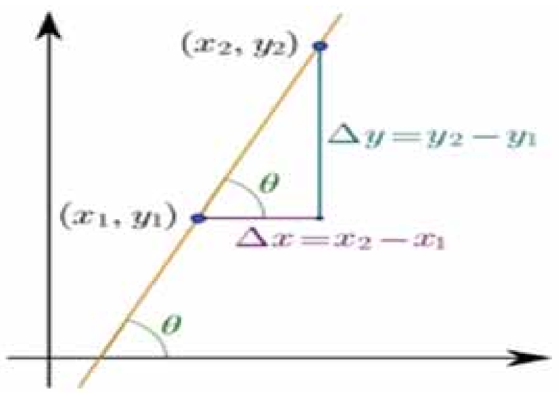


Figure 3. A Gradient vector

The solution to the ERM problem is w = $\mathbf{A}^{-1}$ **b** when A is invertible**.**

II. Gradient

[6] A gradient is a vector generalization of the derivative and we need to calculate its minimum value and moreover, a derivative is a scalar valued. . To be precise, a derivative is defined for functions having a single variable; on the other hand, a function with a number of variables, the gradient definition is a better and intuitive option, as depicted in Figure 3.

[7] More formally, for a given training set **S** and using homogenous description for $\mathbf{L_d}$ the class of problem towards ERM calculations is to find

$$\text{argmin} L_s(h_w) = \text{argmin } 1/m\sum((w,x_i) - y_i)^2$$

[8] The slope represents the gradient of a graph and directs towards increase or decrease in that direction. To optimize for minimum value, the first-order derivative helps us achieve these using iterations. In order to locate a local minimum using GD optimization, we step proportionally towards negative of the gradient of the function at the current point. [9] On the other hand, when we undertake the steps that are proportional to the estimated gradient, it is termed as gradient ascent [10]. These prevailing methods are robust in use and have found many applications in various domains – as the next section highlights those aspects.

III. Gradient Descent Model

The gradient based method has always been attractive due to its simplicity and robustness in optimization scenarios [1]. For a given machine learning algorithm, a MSE cost (loss) function can evaluate the parameters of the learning model with weights updates. The main focus is to find the set of parameters, i.e., weights which minimize the loss function. [11] This can provide a clue towards reaching local optima. We repeat this process (known as an epoch) until we reach near a valley point, as depicted in Figure 4 where J(w) is a loss unction for the parameter's 'w'. For more on this, one can refer to [1].

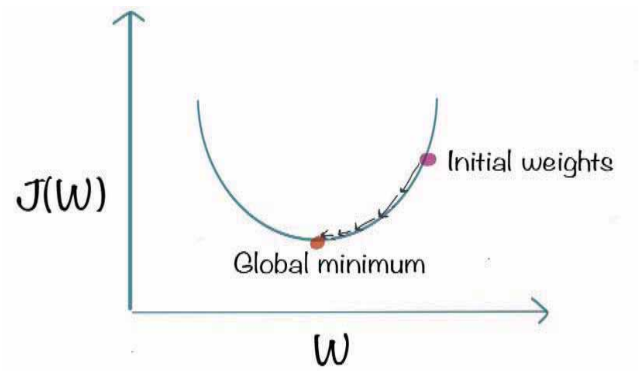


Figure 4. Cost (Loss) function

IV. Pearson Correlation Coefficient 'R' (PCC)[1]

In order to find the relationship between the variables, we use the metric of measure as Pearson coefficient – which suggests the strength of the relationship being either '*strong*' or '*weak*' or '*none* ' —we can infer relationships depending upon the value as depicted in Figure 5.

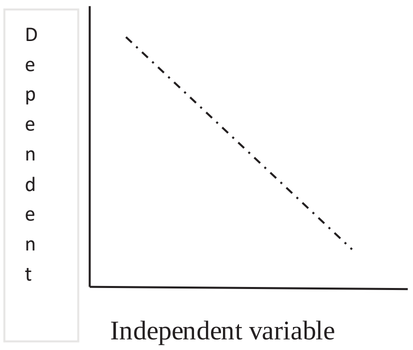


Figure 5(a) r = -1; A perfect negative relationship

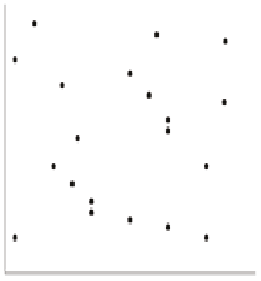


Figure 5 (b). r= 0
No coorelation

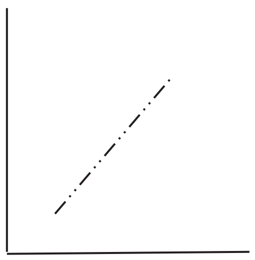


Figure 5 (c) . r = +1
A perfect positive relashion

The scatter plots for various values of 'r' suggest the relationships—meaning that in a negative correlation as one variable increases, the other variable decreases; and on other hand, a positive correlation shows that both the variables increase or decrease together.

V. Framework

The gradient descent method offers some interesting challenges with good convergence speed, as:

(i) Selecting an optimal value of learning rate is many times not obvious—a smaller value leads to slower convergence whereas a bigger value hinders convergence and causes the loss function to fluctuate around the minimum or even tends to diverge.

(ii) Following the Occam Razor principle in the proposed method, simply start from a lower value and keep judging the accuracy trend and a suggested way would be to keep trying with a learning rate schedule, so that reproducibility of the algorithm is guaranteed. The schedules are to be defined in advance and are sometimes unable to adapt to a dataset's characteristics. This is the preferred way implemented in the work.

(iii) Further, the same learning rate applies to all parameter updates. If the data is sparse with features having different frequencies, it would be wise not to update all of them to the same extent but perform a larger update for rarely occurring features. This adaptivity works well with sparsity of data.

(iv) Most challenging task is to avoid non-convex error functions getting trapped into their numerous local minima sub-optimally—as this difficulty arises not from local minima but from saddle points where one dimension slopes up and other slopes down. These saddle points' plateau makes the algorithm hard to escape as the gradient is becoming close to zero in all dimensions.

VI. Implementation

The test case implements GD in order to minimize a cost function J (w) parameterized by a model parameter. The gradient (derivative) shows the incline or slope of the cost function. Hence, to minimize the cost function, we move in the direction opposite to the gradient [12].

For a given dataset from Kaggle web site, predicting graduate admission process using GD technique.

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

The parameters included are:

- GRE Scores (out of 340)
- TOEFL Scores (out of 120)
- University Rating (out of 5)
- Statement of Purpose and Letter of Recommendation Strength (out of 5)
- Undergraduate GPA (out of 10)
- Research Experience (either 0 or 1)
- Chance of Admit (ranging from 0 to 1)

STEPS:

- Import the **csv** file from Kaggle ((https://www.kaggle.com/datasets/mohansacharya/graduate-admissions?select=Admission_Predict_Ver1.1.csv)
- Pre-process data and remove missing or null values.
- Apply gradient descent algorithm.
- Analyse algorithm performance using metrics.

The data set file used is available from Kaggle website and read into the Python environment and using appropriate commands, it shows all the columns under the heading. The corresponding box plot depicts the minimum, average and maximum GRE Score for the admission criterion along with a chance of admission.

Pseudo code for Gradient Descent Model

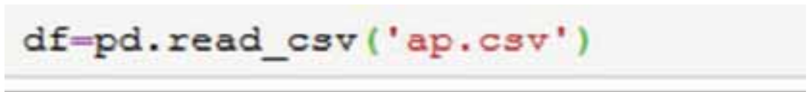
```
Steps:

Initialize weights ‘w’ at random
compute gradients $G = \nabla_{w} J(w)$ of loss function wrt parameters, i.e., $G_i = \partial J(w)\ \partial w_i$
Update weights proportional to G, i.e.
 $w = w - \eta \cdot G$
until J(w) stops reducing or other pre-defined termination criteria is met.
```

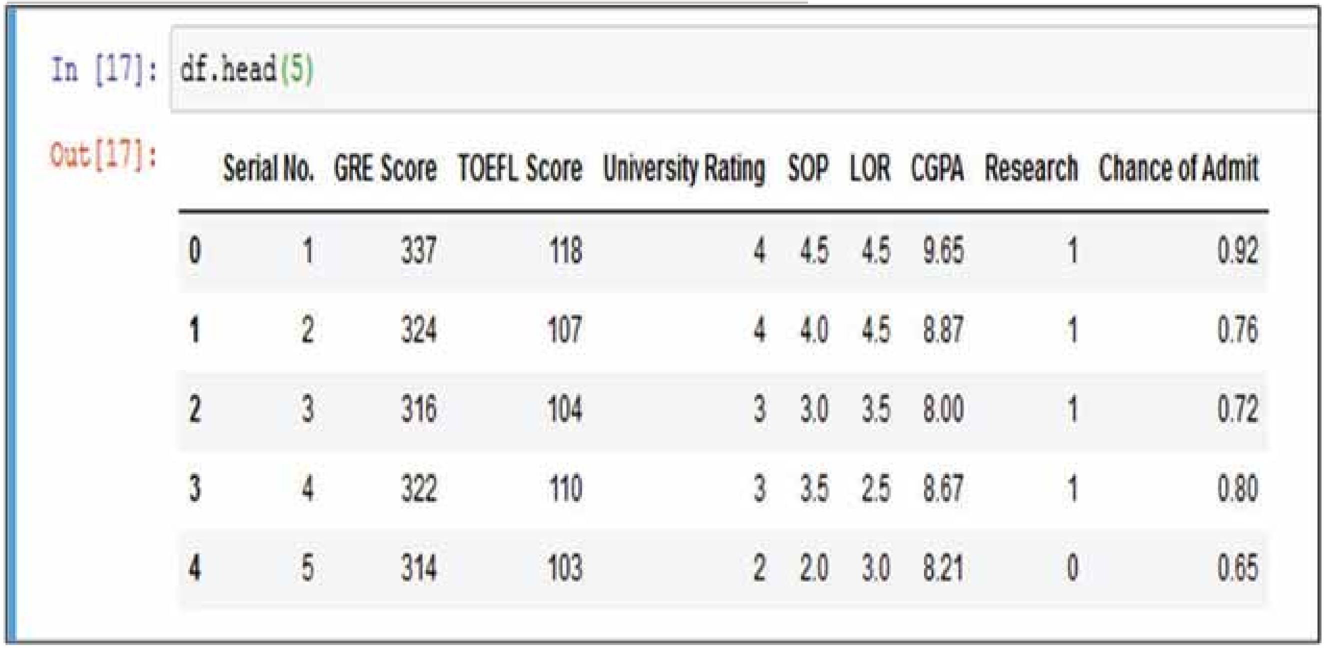
Towards this process, the Pearson coefficient method is being applied to know about those attributes having strong and weak relationship, thus profiling the user about the anticipated chances.

Finally, a min-max scaling is applied on the data to reveal the chances of admission.

This implementation is an extension of the standard gradient method making it quite obvious about the results to interpret and thus augmenting the process of knowing the chances. For instance, doing research and making a publication with a professor stands a higher chance, and so are the other factors too. The following screenshots depict various plots. The curve fitted emphasizing the admission process. The relationship plot shows various intriguing parameters and their relative values.



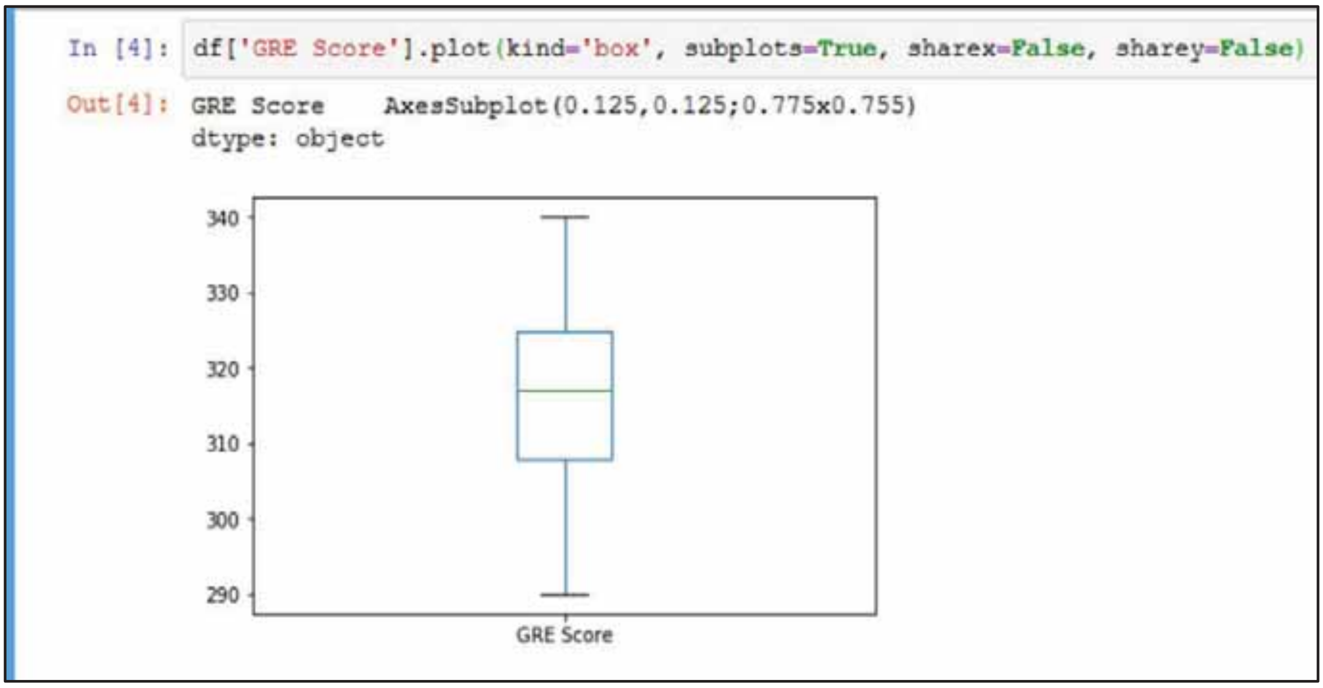
```
df=pd.read_csv('ap.csv')
```



```
In [17]: df.head(5)
```

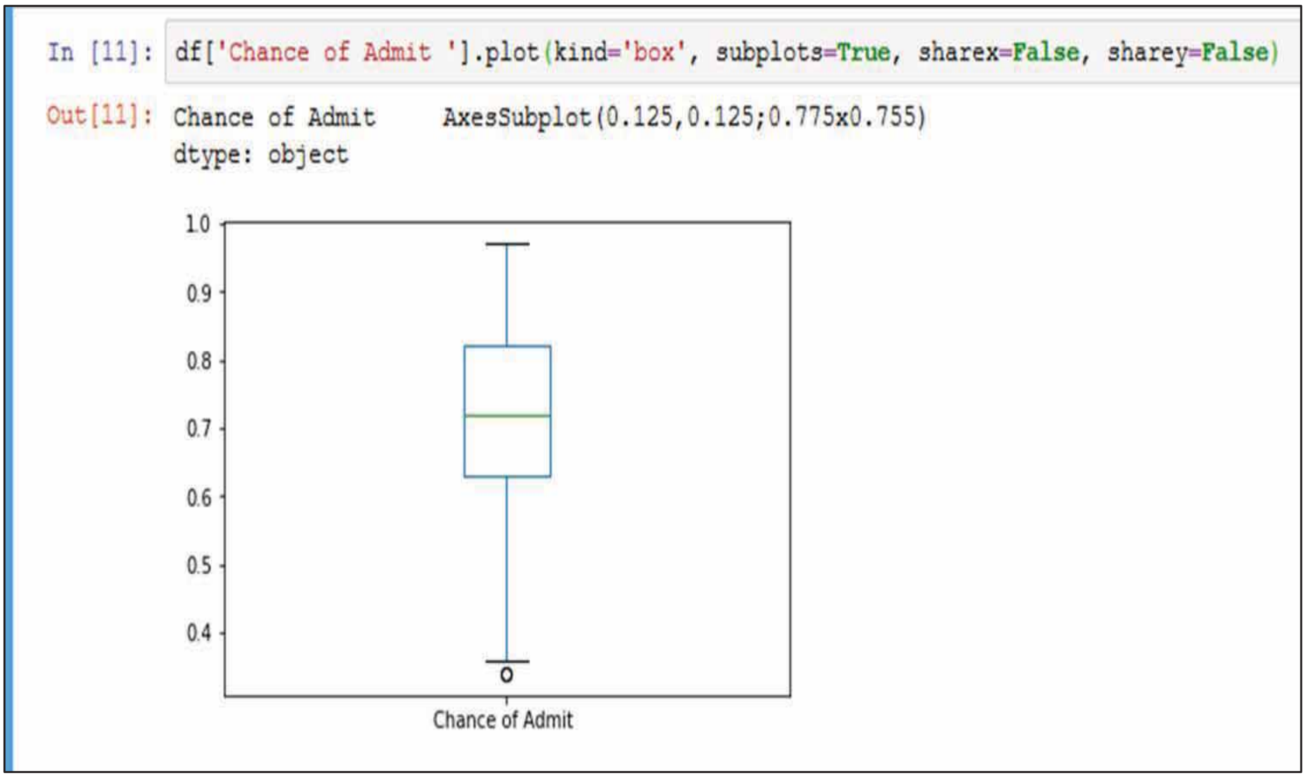
Out[17]:

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



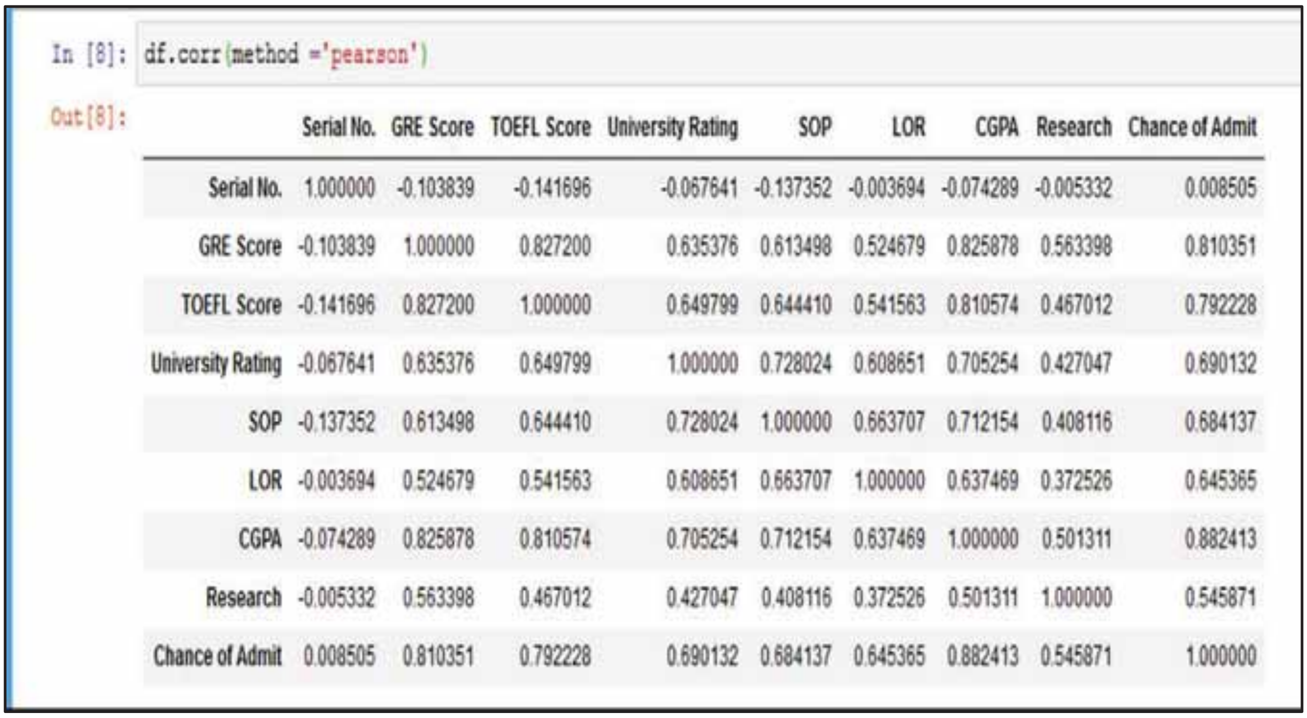
```
In [4]: df['GRE Score'].plot(kind='box', subplots=True, sharex=False, sharey=False)

Out[4]: GRE Score    AxesSubplot(0.125,0.125;0.775x0.755)
        dtype: object
```



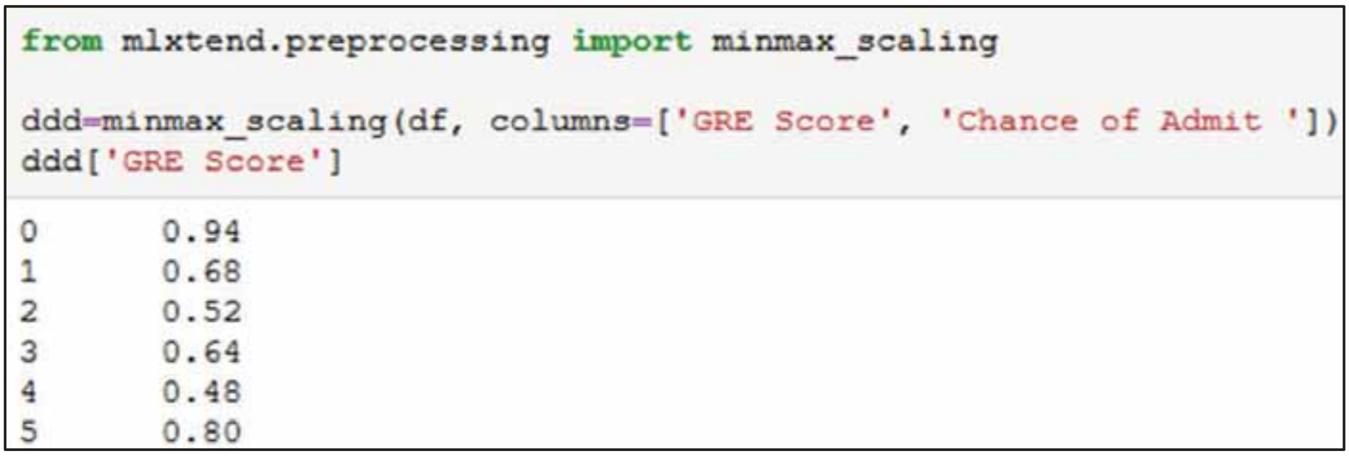
```
In [11]: df['Chance of Admit '].plot(kind='box', subplots=True, sharex=False, sharey=False)

Out[11]: Chance of Admit     AxesSubplot(0.125,0.125;0.775x0.755)
         dtype: object
```



```
In [8]: df.corr(method ='pearson')
Out[8]:
```

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

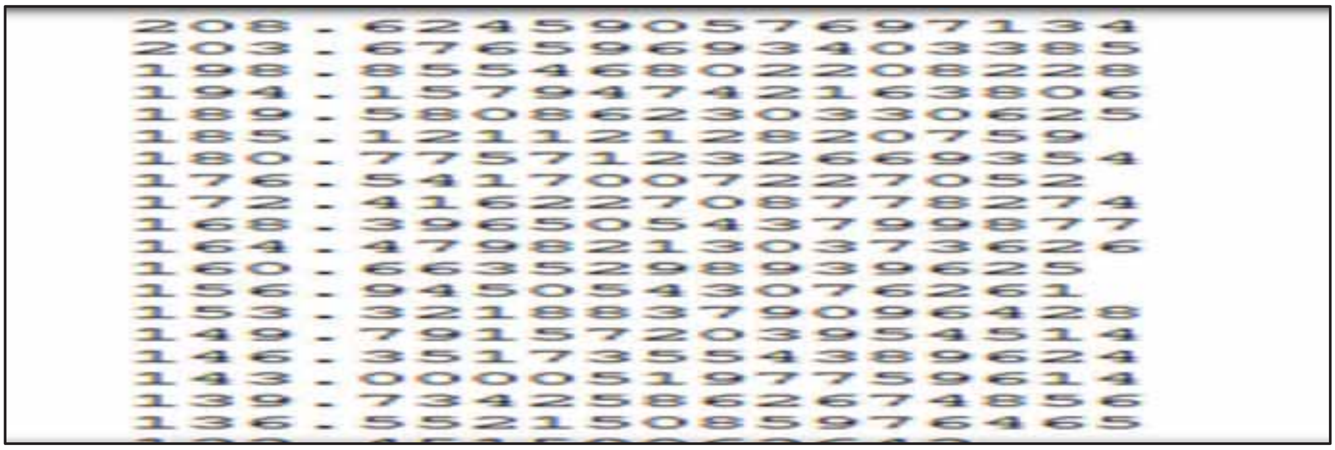


```
from mlxtend.preprocessing import minmax_scaling

ddd=minmax_scaling(df, columns=['GRE Score', 'Chance of Admit '])
ddd['GRE Score']
```

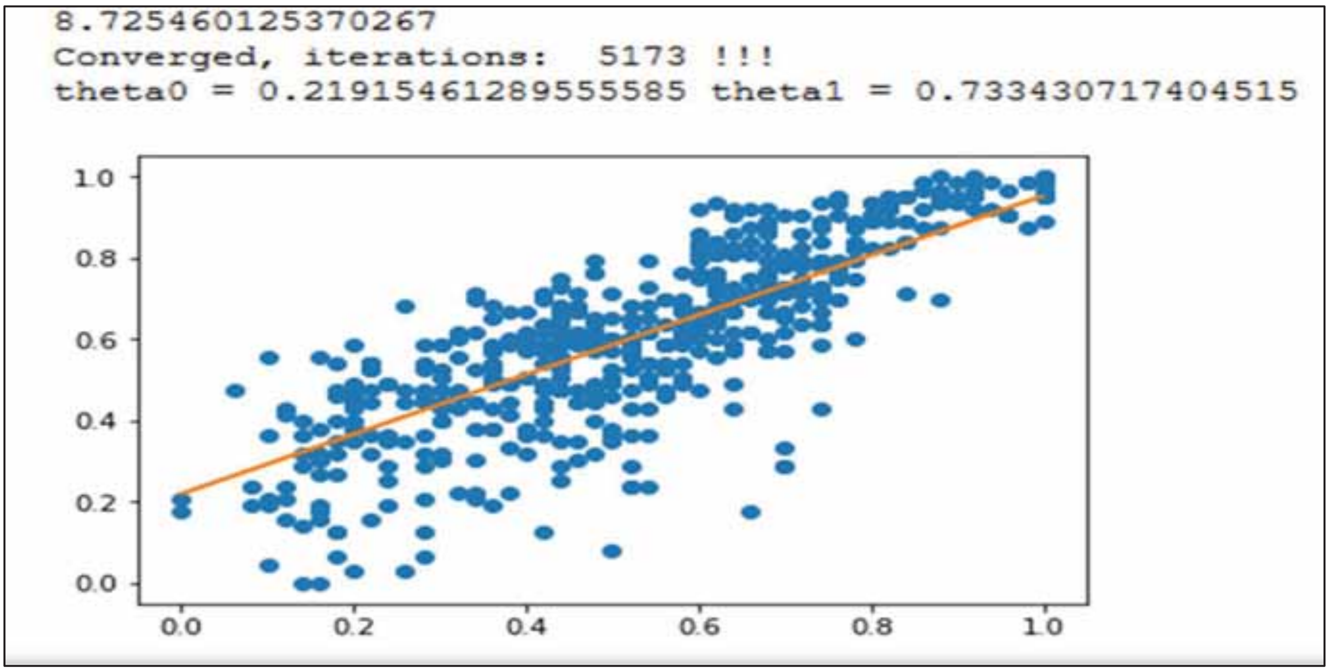
```
0       0.94
1       0.68
2       0.52
3       0.64
4       0.48
5       0.80
```

OUTPUT



```
208.6245905769 7134
```

```
208.62459057697134
203.67659693403385
198.85546802208228
194.15794742163806
189.58086230330625
185.1211212820759
180.77571232669354
176.5417007222705 2
172.41622270877 8274
168.39650543799877
164.47982130373626
160.6635298939625
156.9450543076261
153.32188379096428
149.79157203954514
146.35173555438962 4
143.00005197759614
139.73425862674856
136.55215085976465
```



```
8.725460125370267
Converged, iterations:  5173 !!!
theta0 = 0.21915461289555585 theta1 = 0.733430717404515
```

VII. CONCLUSIONS

The GD model has been implemented in the work on the principle of Occam Razor: "*simplest things which work first, are better* ". The model takes gradient based approach to fit the regressor on Kaggle^2 dataset and it can be best suited on any linearly predictable scenario. It captures the parameters which play the critical role for a student's chances for admission process as it reveals the internal relationship amongst parameters. Further, the model is able to predict the outcome as a percentage of the chance of a GRE score. The Pearson correlation coefficient 'r' has a strong bearing on the underlying parameters- as it is evident from the program run and various plots. As the linear regressor is one of the most suitable models fit for such type of data, there are improvements also possible-like using Stochastic GD for better performance and efficiency with a niche fit. Moreover, the model can further be extended to undertake features of multiple dependencies as well by way of looking at the extracted features of the data set and fitting it intuitively.

REFERENCES

[1] Ajeet K. Jain, Dr. PVRD Prasad Rao and Dr. K Venkatesh Sharma; "Deep Learning with Recursive Neural Network for Temporal Logic Implementation", International Journal of Advanced Trends in Computer Science and Engineering, Volume 9, No.4, July – August 2020 https://doi.org/10.30534/ijatcse/2020/383942020

[2] Ajeet K. Jain, Dr. PVRD Prasad Rao and Dr. K Venkatesh Sharma;"A Perspective Analysis of Regularization and Optimization Techniques in Machine Learning", Chapter submitted to Computational Analysis and Understanding of Deep Learning for Medical Care: Principles, Methods, and Applications", CUDLMC 2020.

[3] Haykin Simon 2016, Neural Network and Machine Learning, 3rd Ed. Pearson Edn. 2010

[4] CholletFrancois 2018, Deep Learning with Python, Manning Pub., 1st Ed. 2018

[5] Frassord Davi, Linear regression with numpy 2016 (GitHub); www.github /Frossard.htm

[6] Lachlan Miller, Machine Learning: Cost Function, Gradient Descent and Univariate Linear Regression, www.github/lachlanmiller.htm

[7] Ravidran B 2019, NPTEL Course, Introduction to Machine Learning

[8] Karl Gustav Jensson 2108, Introduction to Machine Learning.ML, NPTEL

[9] Gupta A and Biswas G P 2020, Python Programming, Problem solving, packages and libraries, 1st Ed.TMH

[10] Potter Merle C, Potter, J.L. Goldberg and Edward F. Aboufadel 2010 Advanced Engg. Mathematics, Oxford University Press, 3rd Ed.

[11] Charu C. Agrawal, Neural Networks and Deep Learning, 1st Ed. Springer 2018

[12] Bishop, C.M., Neural Network for Pattern Recognition, 1995.