

Final Year B. Tech (EE)

Trimester: X

Subject: AIML

Name: Shreerang Mhatre

Class: TY

Roll No: 52

Batch: A3

Experiment No: 05

Name of the Experiment: Linear Regression Using Python

Marks	Teacher's Signature with date

Performed on: 21/09/2023

Submitted on: 29/09/2023

Aim: Implement Linear Regression Using Gradient Descent Algorithm.

Prerequisite: Knowledge of Supervised Learning method, MLP, Activation function.

Objective:

To create and study the Neural Network by varying parameters.

1. Define Input and Output Variable
2. Define and custom Neural Network
3. Configure the network
4. Train the network to find output

Components and Equipment required:

Python software

AMP/WPU/AIML/2020

Expt. 5- 1

Theory:

Linear Regression:

Linear regression is an approach for modeling relationship between a scalar dependent variable y and one or more independent variables. When there is single independent variable it is called simple linear regression and when there are more than one independent variables, it is called multiple linear regression. Linear regression is a statistical procedure for predicting the value of a dependent variable from an independent variable when the relationship between the variables can be described with linear model given by $y = m * x + c$.

Gradient Descent

In this method goal is to obtain parameters such that sum of squared error between target output and actual output is minimized.

Gradient descent is an iterative method. Algorithm starts with some set of values for our model parameters (weights and biases) and improves them slowly. Change in weight for each iteration is proportional to derivative of cost function w.r.t. the current weight value.

Thus algorithm moves in the direction of minima of cost function.

Gradient descent is an optimization algorithm that approaches a local minimum of a function by taking steps proportional to the negative of the gradient of the function as the current point.

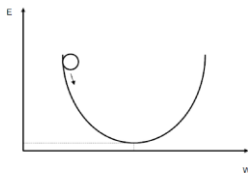


Figure 1: Error and global minima

To run gradient descent on error function, we first need to compute its gradient. The gradient will act like a compass and always point us downhill. To compute it, we will need to differentiate our error function. Since our function is defined by two parameters (w_1 and w_2), we will need to compute a partial derivative for each. These derivatives work out to be:

$$E(w_{ji}) = \frac{1}{2} (y_{tarj} - y_j)^2 \quad y_j = f(a_j) = \sum_i w_{ji} x_i$$

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial y_j} \cdot \frac{\partial y_j}{\partial w_{ji}} = -(y_{tarj} - y_j) x_i = -\delta x_i$$

We now have all the tools needed to run gradient descent. We can initialize our search to start at any pair of w_1 and w_2 values (i.e., any line) and let the gradient descent algorithm march downhill on our error function towards the best line. Each iteration will update w_1 and w_2 to a line that yields slightly lower error than the previous iteration.

The learning Rate variable controls how large of a step we take downhill during each iteration. If we take too large of a step, we may step over the minimum. However, if we take small steps, it will require much iteration to arrive at the minimum.

Below are some snapshots of gradient descent running for 2000 iterations for the example problem? We start out at point $w_1 = -1$ $w_2 = 0$. Each iteration w_1 and w_2 are updated to values that yield slightly lower error than the previous iteration. The left plot displays the current location of the gradient descent search (blue dot) and the path taken to get there (black line). The right plot displays the corresponding line for the current search location. Eventually we ended up with a pretty accurate fit.

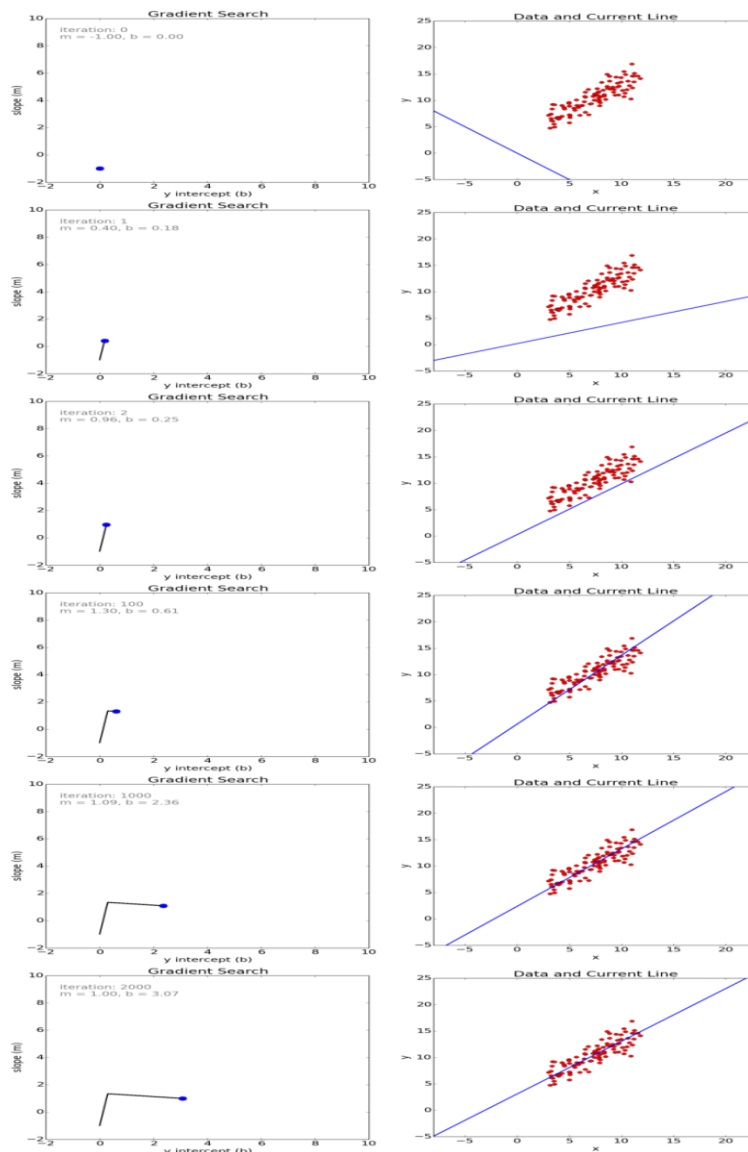




Figure 2: Different iterations of gradient descent

We can also observe how the error changes as we move toward the minimum. A good way to ensure that gradient descent is working correctly is to make sure that the error decreases for each iteration. Below is a plot of error values for the first 100 iterations of the above gradient search.

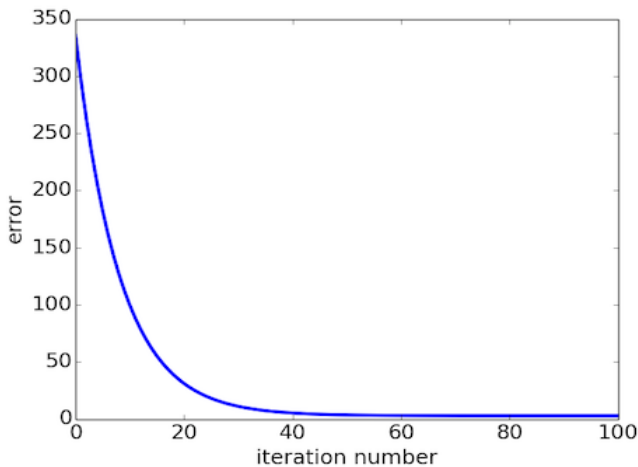


Figure 3: MSE Vs Epochs

While the model in above example was a line, the concept of minimizing a cost function to tune parameters also applies to regression problems that use higher order polynomials and other problems found around the machine learning world.

Procedure

Algorithm:

Step 0: Initialize weights w_0, w_1 . (Set to small random values) and α (learning rate).

Step 1: While stopping condition is false, do steps 2-8.

Step 2: For each training pair with (x_i, y_i) , do steps 3-6.

Step 3: Each input unit ($X_i, i = 1, \dots, n$) receives input signal x_i .

Step 4: Estimate $E(w)$, where $E(w) = 1/(2M) * \sum ((w_0 + w_1 * x_i) - y_i)^2$

Step 5: Estimate new values of $w_{0(new)}, w_{1(new)}$ using the following equations and store it in a temporary variable.

Step 6: Perform simultaneous update of $w_{0(new)} = \text{Temp}_0$

$$w_{1(new)} = \text{Temp}_1$$

Step 7: Update Mean square error, until the Error function $E(w)$ is minimized

Step 8: Stop.

Python Code

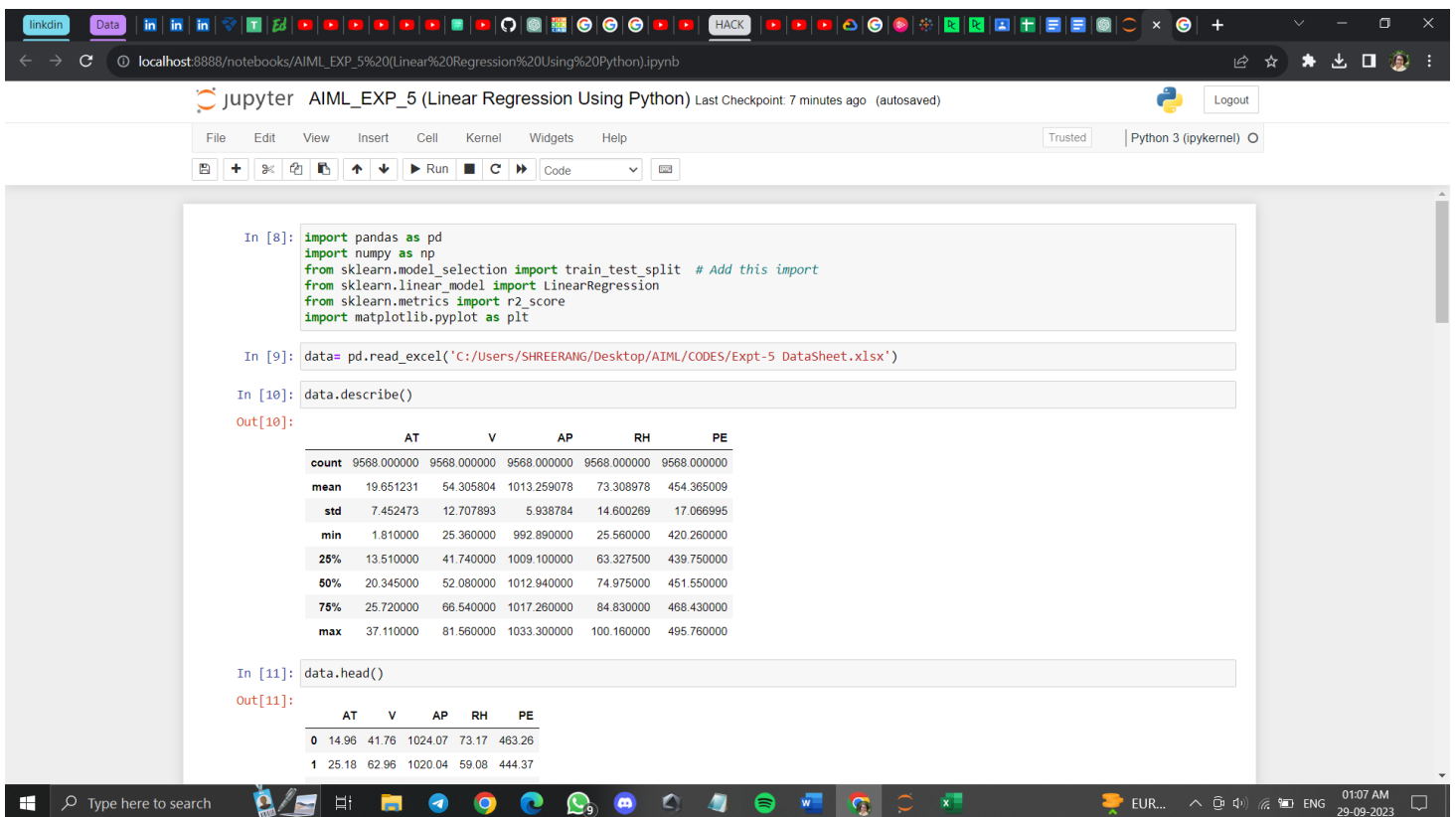
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
data = pd.read_csv('age1.csv')
data.head(5)
data.describe()
plt.scatter(data['AgeGroup'],data['TotalCases'])
plt.xlabel("Age Group of people")
plt.ylabel("Total Confirmed cases")
plt.title("Corona with age Prediction")
data_n=data.values
m=data_n[:,0].size
x=data_n[:,0].reshape(m,1)
y=data_n[:,2].reshape(m,1)
xTrain,xTest,yTrain,yTest=train_test_split(x,y,test_size=0.1,random_state=0)
yTrain
linearRegressor=LinearRegression()
linearRegressor.fit(xTrain,yTrain)
yprediction=linearRegressor.predict(xTest)
xTest,yprediction
import matplotlib.pyplot as plot
plot.scatter(xTest,yTest,color='red')
plot.plot(xTrain,linearRegressor.predict(xTrain))
plot.title('Age vs Corona Confirmed cases')
plot.xlabel('Age Group No')

plot.ylabel('Confirmed cases')
plot.show()
```

Conclusions:

Post Lab Questions:

1. Explain the Gradient Descent algorithm.
2. Explain significance of learning rate in case of gradient descent algorithm?
3. What is global minima and local minima?



localhost:8888/notebooks/AIML_EXP_5(Linear%20Regression%20Using%20Python).ipynb

jupyter AIML_EXP_5 (Linear Regression Using Python) Last Checkpoint: 7 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

```
In [8]: import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split # Add this import
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
import matplotlib.pyplot as plt

In [9]: data= pd.read_excel('C:/Users/SHREERANG/Desktop/AIML/CODES/Expt-5 DataSheet.xlsx')

In [10]: data.describe()

Out[10]:
```

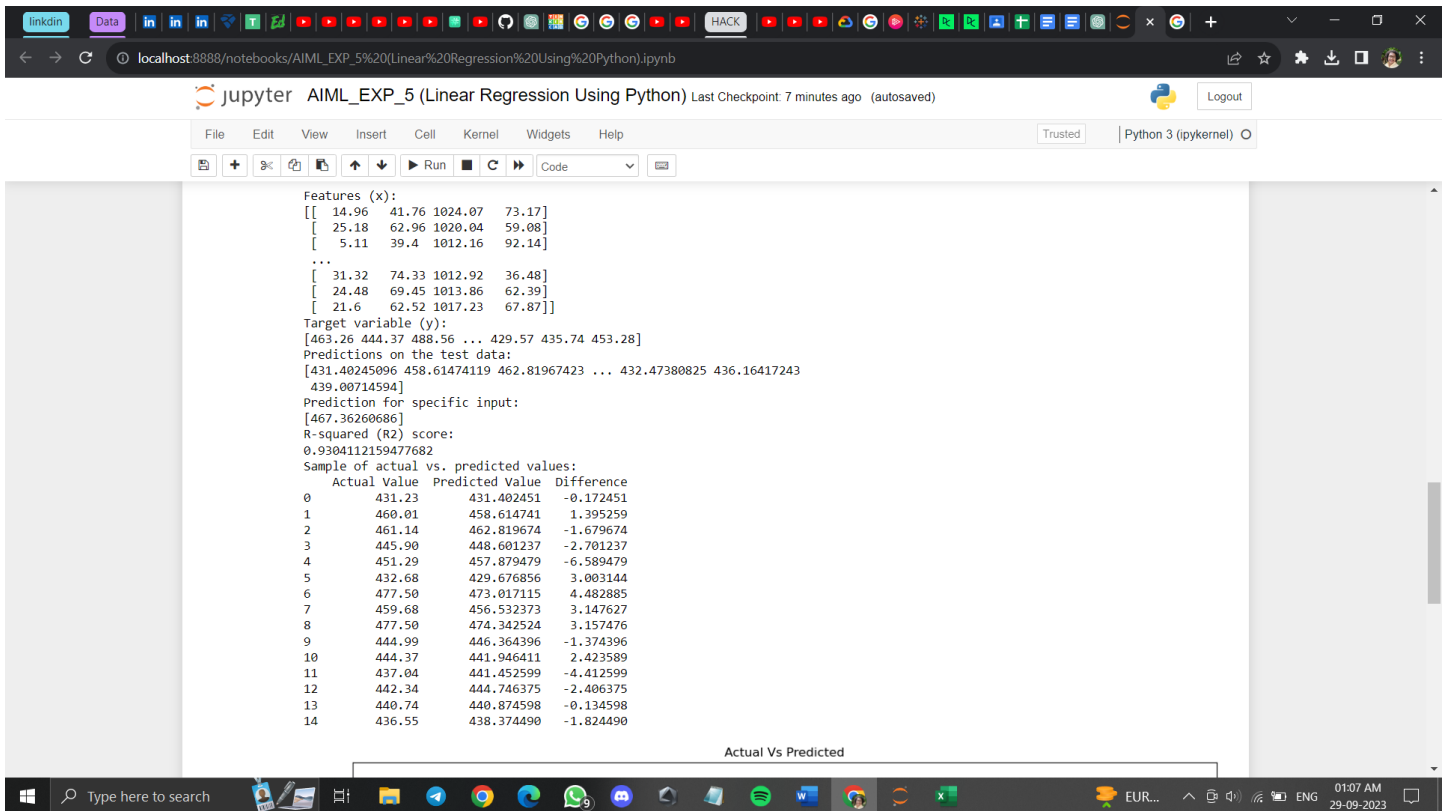
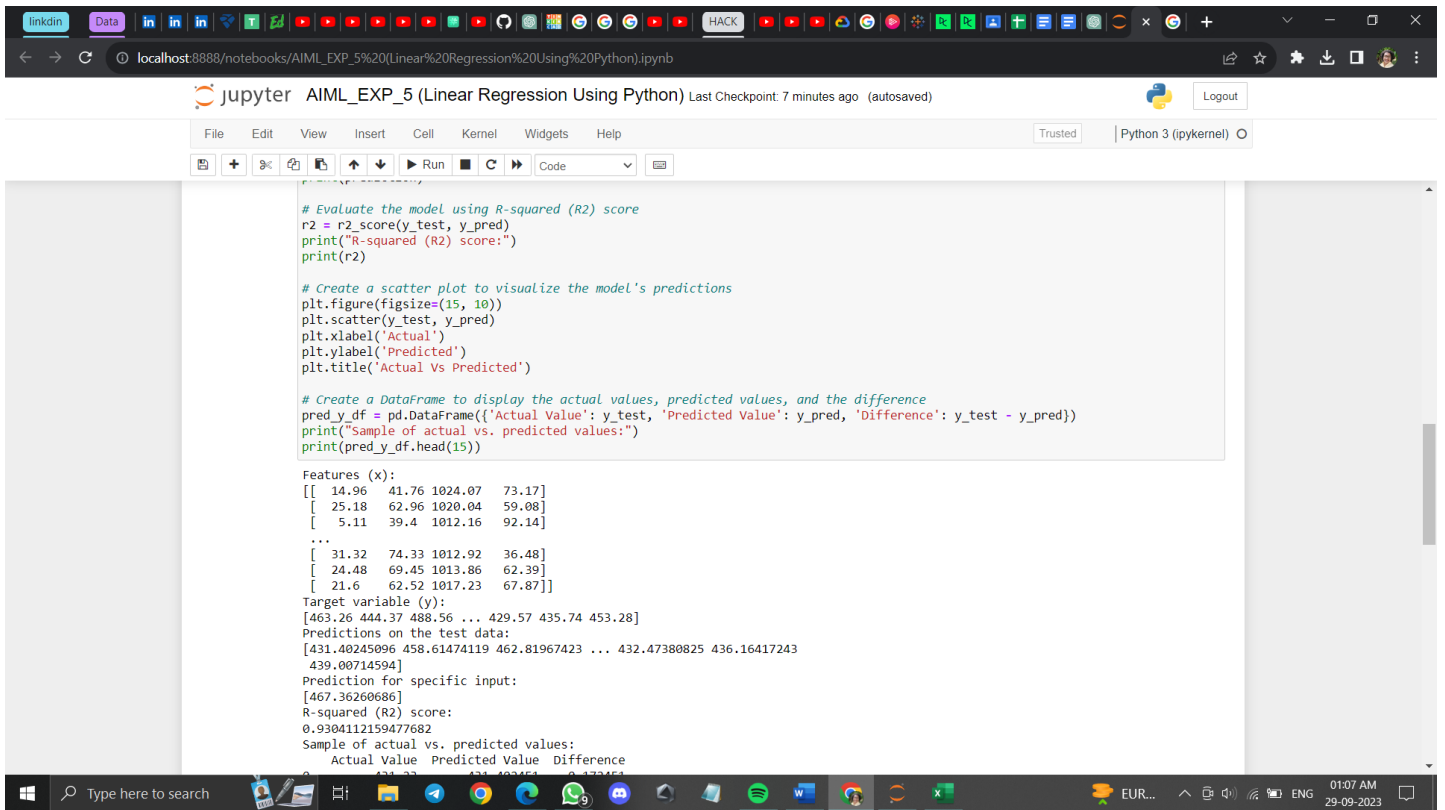
	AT	V	AP	RH	PE
count	9568.000000	9568.000000	9568.000000	9568.000000	9568.000000
mean	19.651231	54.305804	1013.259078	73.308978	454.365009
std	7.452473	12.707893	5.938784	14.600269	17.066995
min	1.810000	25.360000	992.890000	25.560000	420.260000
25%	13.510000	41.740000	1009.100000	63.327500	439.750000
50%	20.345000	52.080000	1012.940000	74.975000	451.550000
75%	25.720000	66.540000	1017.260000	84.830000	468.430000
max	37.110000	81.560000	1033.300000	100.160000	495.760000

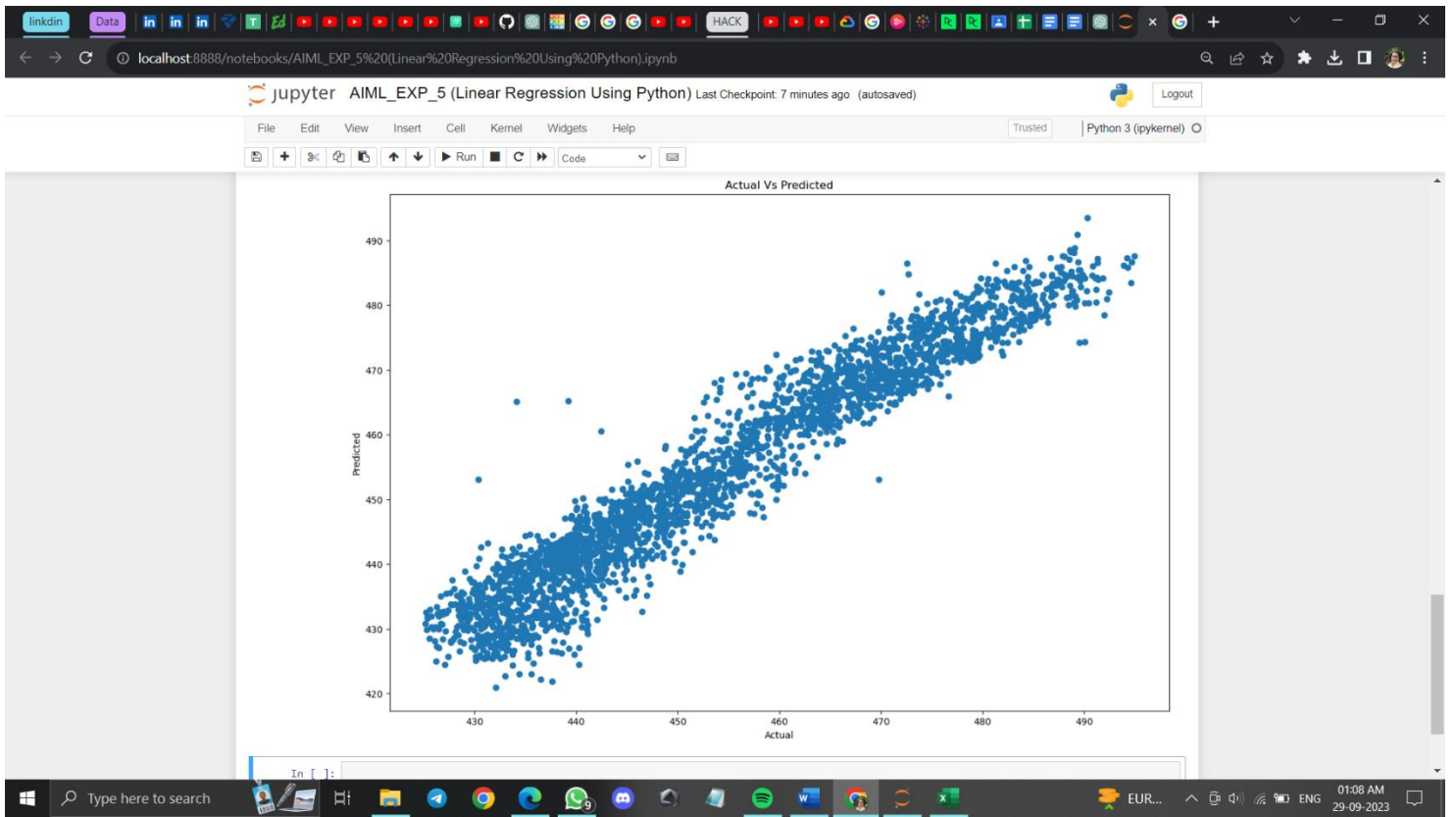
```
In [11]: data.head()

Out[11]:
```

	AT	V	AP	RH	PE
0	14.96	41.76	1024.07	73.17	463.26
1	25.18	62.96	1020.04	59.08	444.37

01:07 AM
29-09-2023





Exp5 - Linear Regression Using Python

* Post Lab Questions -

(Q1) Explain the Gradient Descent algorithm.

→ Gradient Descent is an optimization algorithm used in machine learning to find the best parameters for a model by minimizing a cost function. Here's a concise explanation of the algorithm -

- ① Initialize - Start with initial parameter values.
- ② Calculate Gradient - Compute the gradient (slope) of the cost function with respect to each parameter.
- ③ Update Parameters - Adjust parameters in the opposite direction of the gradient by a small step.
- ④ Repeat - Continue steps 2 & 3 iteratively until convergence or a set number of iterations.
- ⑤ Convergence - Monitor cost function to check if it's minimized, indicating optimal parameters.

Q2) Explain significance of learning rate in case of gradient descent algorithm?

→ The learning rate in the Gradient Descent algorithm plays a pivotal role in the training process of machine learning models. It determines the size of the steps taken to update model parameters during optimization. A higher learning rate can lead to faster convergence but risks overshooting and divergence, while a lower rate provides stability by resulting in slow convergence. Therefore, selecting the right learning rate is essential; it serves as a critical hyperparameter that must be tuned for each specific problem and dataset. A well-chosen learning rate balances the trade-off between training speed and stability, ensuring the algorithm efficiently finds optimal parameter values and avoids getting stuck in local minima.

Q3) What is global minima & local minima?

→ ① Global minima -

The global minima is the lowest point in the entire function, representing the absolute lowest value of the function across its entire domain. Finding the global minimum is the ~~primary~~ primary objective in optimization problems.

② Local minima -

Local minima are points within a function where the value is lower than in the surrounding neighborhood but not necessarily the absolute lowest value in the entire function. They can be problematic in optimization as they may trap algorithms if the search starts from a point near a local minimum, preventing them from reaching the global minimum.