DL Assignment 1

Shah Kaushal Kamleshkumar

Enrollment no - 211370680003

DEEP LEARNING - 1628006

Post Graduate Diploma in Data Science

Graduate School of Engineering and Technology

Semester 2

1 Implement Perceptron on Iris dataset and visualize it

```python
import numpy as np
from matplotlib import pyplot as plt
import pandas as pd
from matplotlib.colors import ListedColormap

# Creation of the main perceptron object.
class Perceptron(object):
    #Initiating the learning rate and number of iterations.
    def __init__(self, Learn_Rate=0.5, Iterations=10):
        self.learn_rate = Learn_Rate
        self.Iterations = Iterations
        self.errors = []
        self.weights = np.zeros(1 + x.shape[1])

    # Defining fit method for model training.
    def fit(self, x, y):
        self.weights = np.zeros(1 + x.shape[1])
        for i in range(self.Iterations):
            error = 0
            for xi, target in zip(x, y):
                update = self.learn_rate * (target - self.predict(xi))
                self.weights[1:] += update*xi
                self.weights[0] += update
                error += int(update != 0)
            self.errors.append(error)
        return self

    # Net Input method for summing the given matrix inputs and their corresponding weights.
    def net_input(self, x):
        return np.dot(x, self.weights[1:]) + self.weights[0]

    # Predict method for predicting the classification of data inputs.
    def predict(self, x):
        return np.where(self.net_input(x) >= 0.0, 1, -1)
```

Figure 1: Perceptron implementation code

```python

y = pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data", header=None)
x = y.iloc[0:100, [0, 2]].values
plt.scatter(x[:50, 0], x[:50, 1], color='red',label = "versiocolor")
plt.scatter(x[50:100, 0], x[50:100, 1], color='blue',label = "setosa")
plt.scatter(x[100:150, 0], x[100:150, 1], color='yellow')
plt.xlabel('petal length(cm)')
plt.ylabel('sepal length(cm)')
plt.legend(loc='upper left')
plt.title("Results")
plt.title("Data points visualization")
plt.show()
y = y.iloc[0:100, 4].values
y = np.where(y == 'Iris-setosa', -1, 1)
```

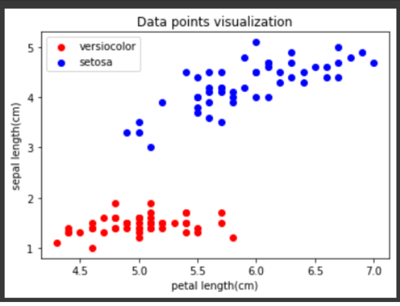


Figure 2: Iris Dataset data points visualization

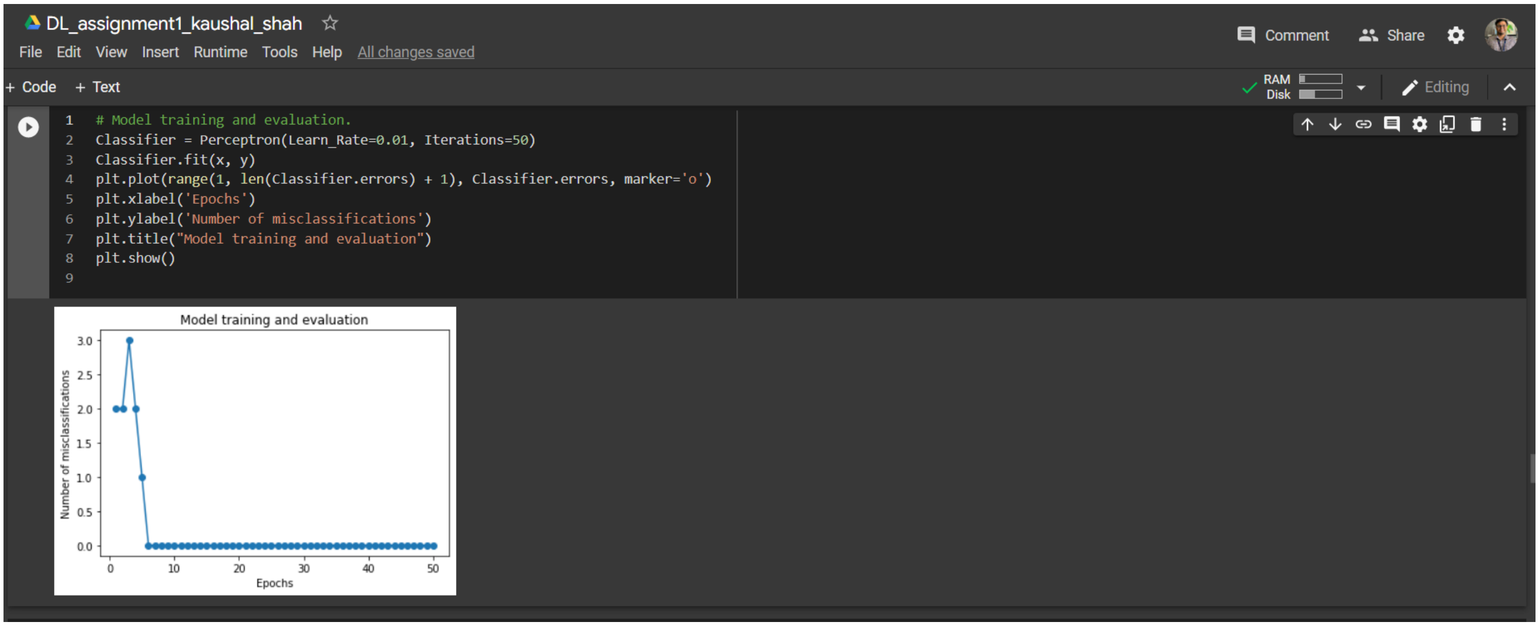


Figure 3: Perceptron model training and evaluation

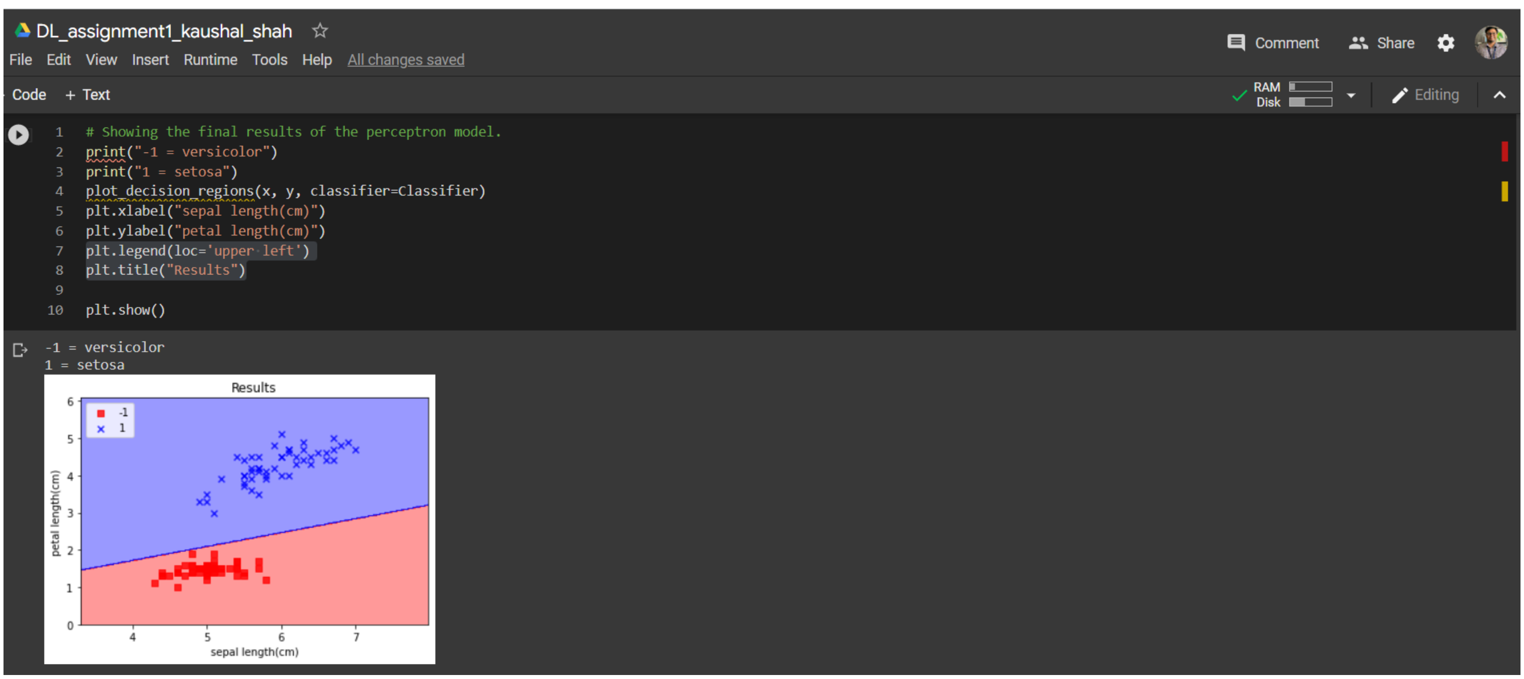


Figure 4: Results on perceptron model

2 Implement different gradient-based algorithms like stochastic, batch and mini-batch.

All three gradient descent algorithm is implemented on house price dataset

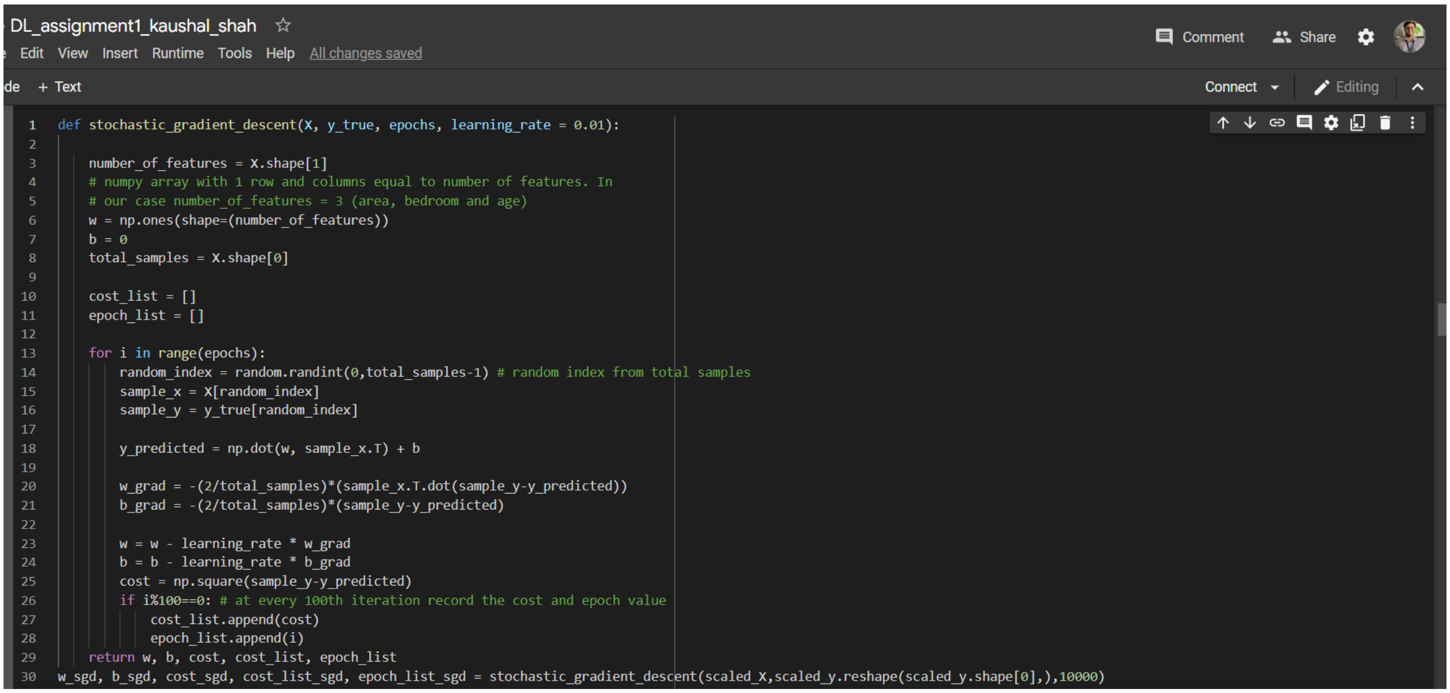


Figure 5: Stochastic gradient descent implementation code

Figure 6: Stochastic gradient descent output

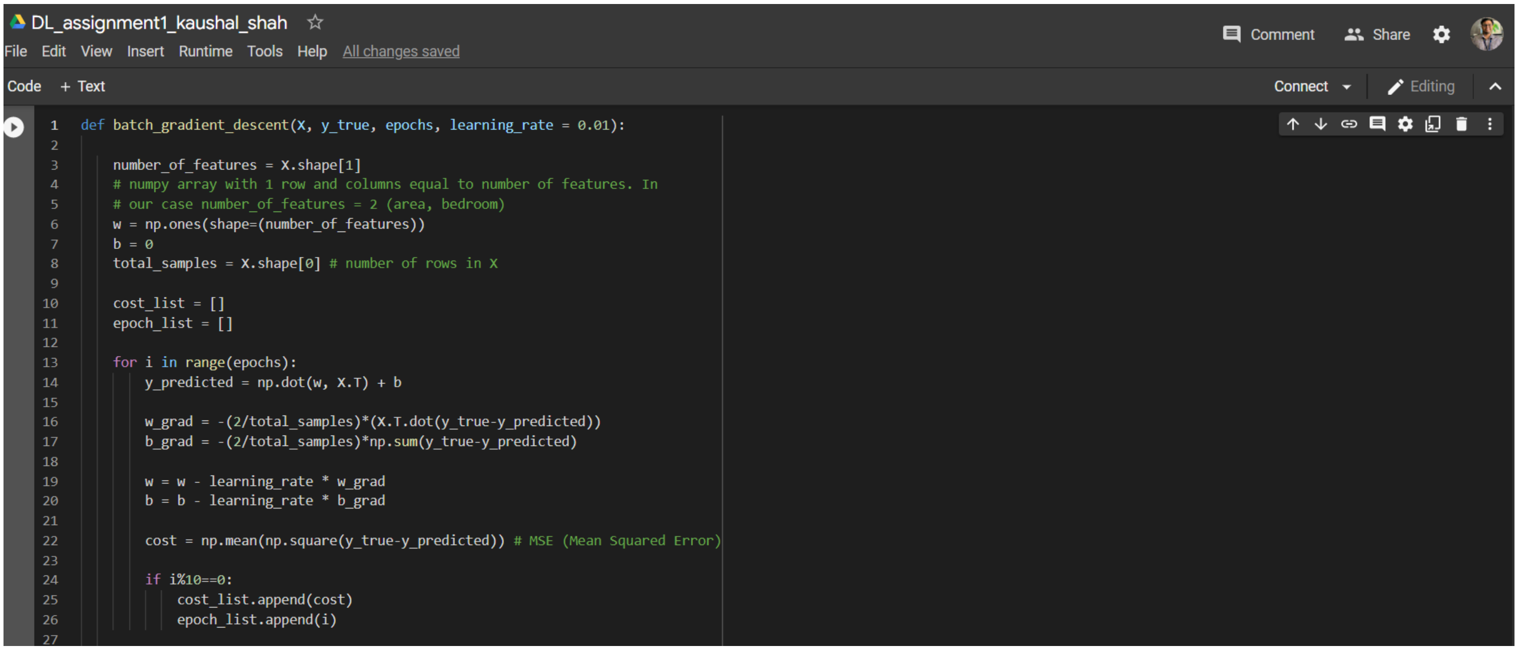


Figure 7: Batch gradient descent implementation code

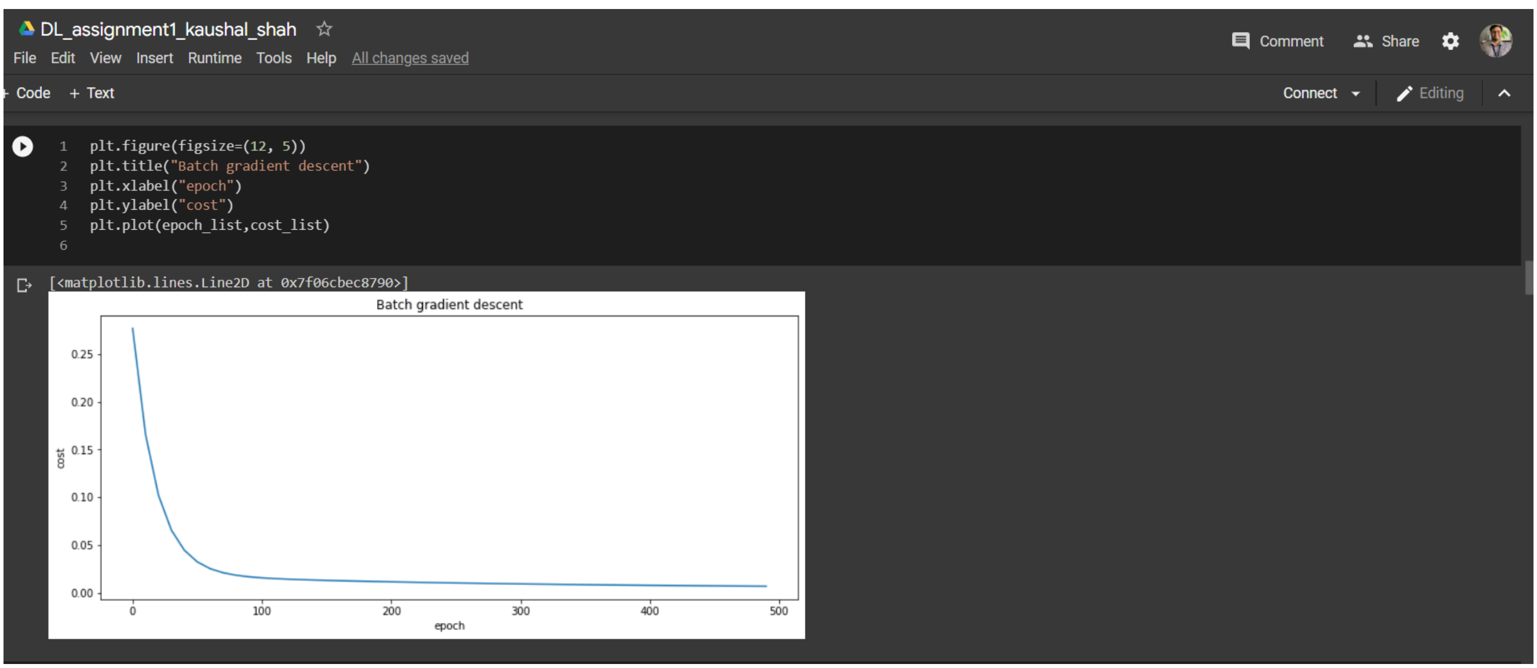
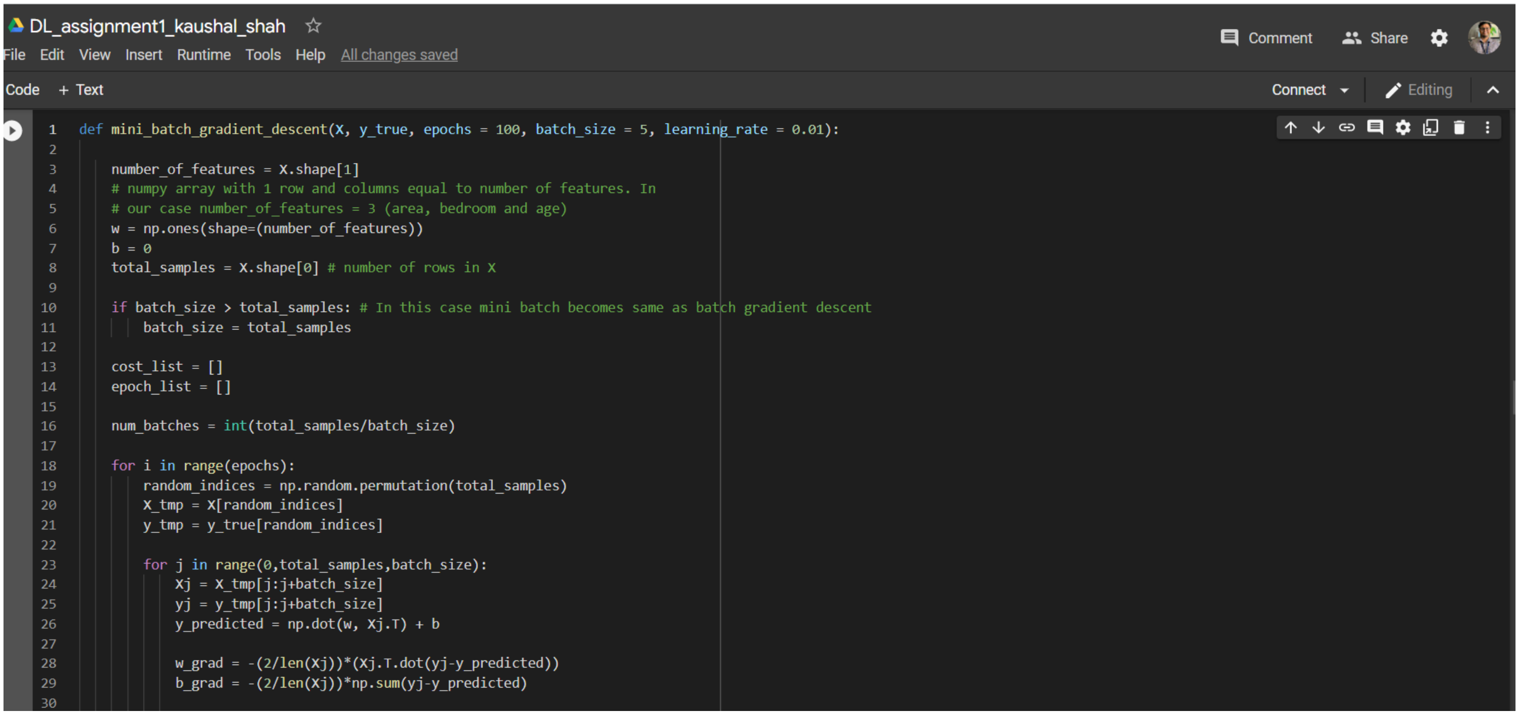
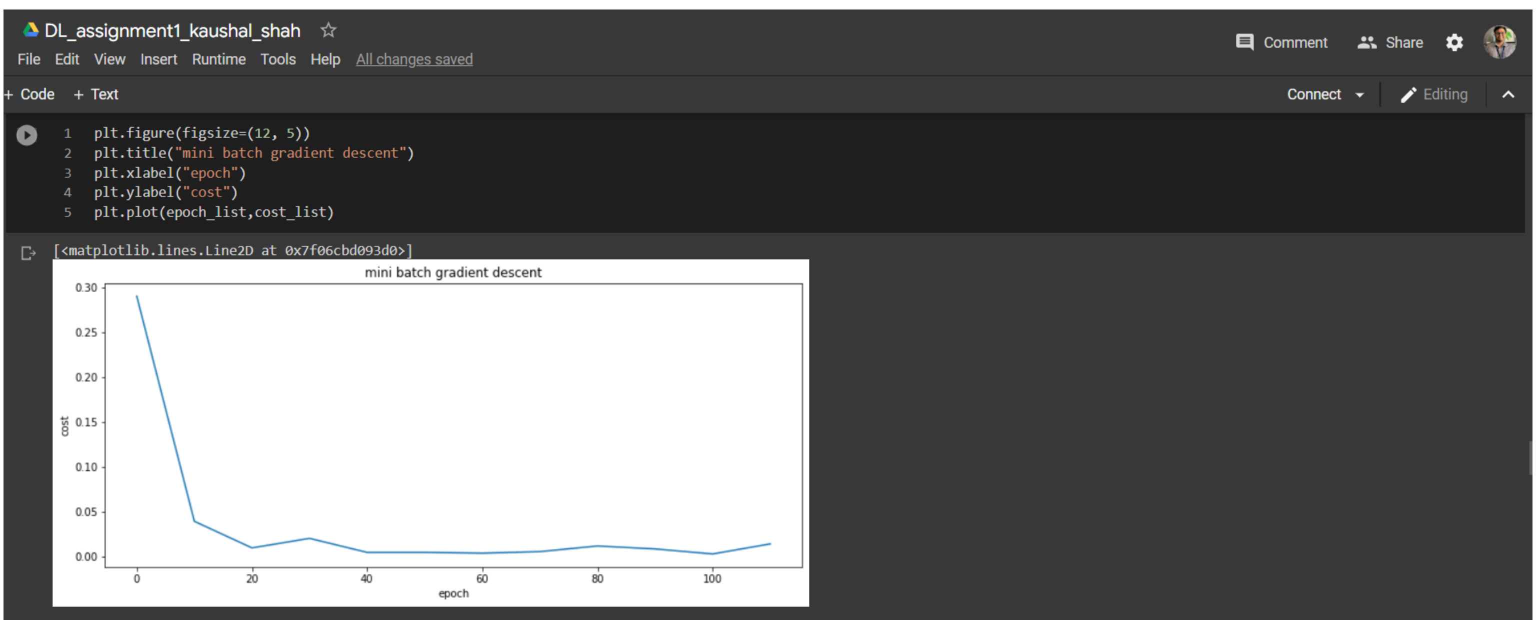


Figure 8: Batch gradient descent output



```python
def mini_batch_gradient_descent(X, y_true, epochs = 100, batch_size = 5, learning_rate = 0.01):

    number_of_features = X.shape[1]
    # numpy array with 1 row and columns equal to number of features. In
    # our case number_of_features = 3 (area, bedroom and age)
    w = np.ones(shape=(number_of_features))
    b = 0
    total_samples = X.shape[0] # number of rows in X

    if batch_size > total_samples: # In this case mini batch becomes same as batch gradient descent
        batch_size = total_samples

    cost_list = []
    epoch_list = []

    num_batches = int(total_samples/batch_size)

    for i in range(epochs):
        random_indices = np.random.permutation(total_samples)
        X_tmp = X[random_indices]
        y_tmp = y_true[random_indices]

        for j in range(0,total_samples,batch_size):
            Xj = X_tmp[j:j+batch_size]
            yj = y_tmp[j:j+batch_size]
            y_predicted = np.dot(w, Xj.T) + b

            w_grad = -(2/len(Xj))*(Xj.T.dot(yj-y_predicted))
            b_grad = -(2/len(Xj))*np.sum(yj-y_predicted)
```


Figure 9: Mini batch gradient descent implementation code



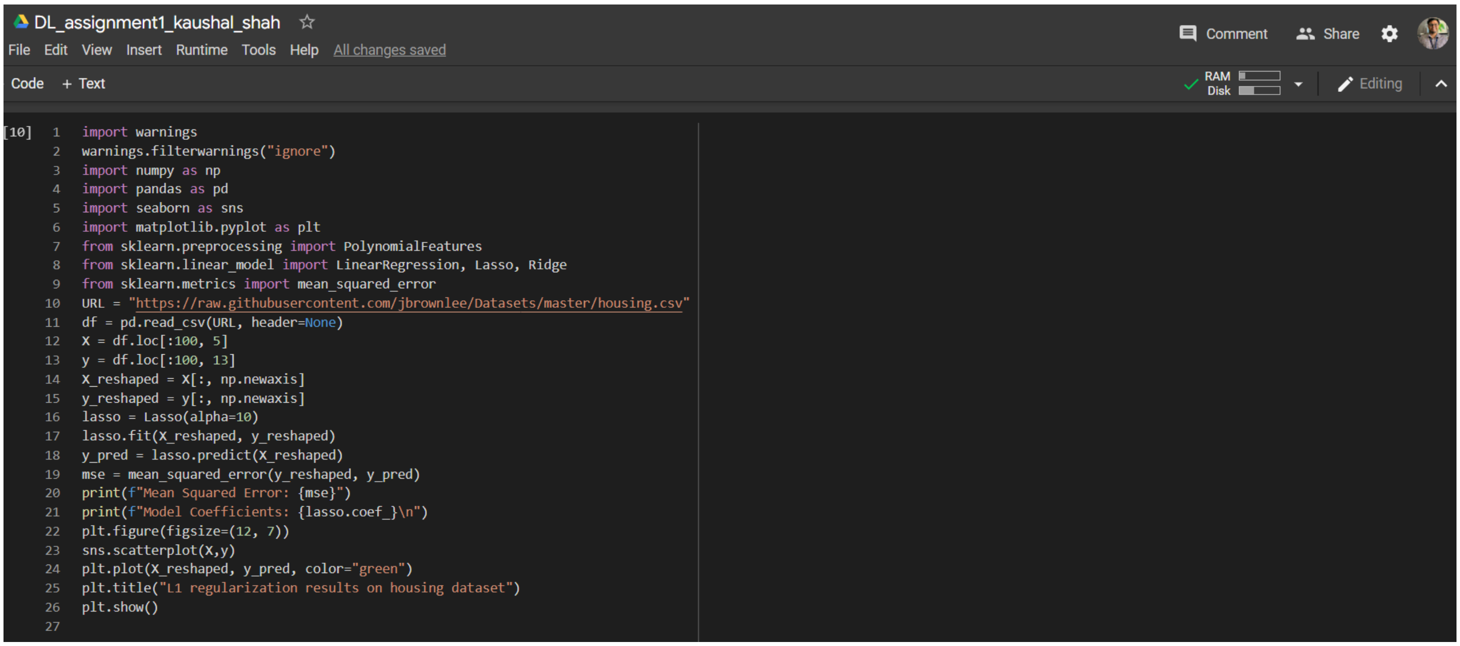
```python
plt.figure(figsize=(12, 5))
plt.title("mini batch gradient descent")
plt.xlabel("epoch")
plt.ylabel("cost")
plt.plot(epoch_list,cost_list)
```

```
[<matplotlib.lines.Line2D at 0x7f06cbd093d0>]
```


Figure 10: Mini batch gradient descent output

3 Implement L1 and L2 regularization.

L1 and L2 regulization implemented on housing dataset




```python
import warnings
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.metrics import mean_squared_error
URL = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/housing.csv"
df = pd.read_csv(URL, header=None)
X = df.loc[:100, 5]
y = df.loc[:100, 13]
X_reshaped = X[:, np.newaxis]
y_reshaped = y[:, np.newaxis]
lasso = Lasso(alpha=10)
lasso.fit(X_reshaped, y_reshaped)
y_pred = lasso.predict(X_reshaped)
mse = mean_squared_error(y_reshaped, y_pred)
print(f"Mean Squared Error: {mse}")
print(f"Model Coefficients: {lasso.coef_}\n")
plt.figure(figsize=(12, 7))
sns.scatterplot(X,y)
plt.plot(X_reshaped, y_pred, color="green")
plt.title("L1 regularization results on housing dataset")
plt.show()
```

Figure 11: L1 regularization implementation code

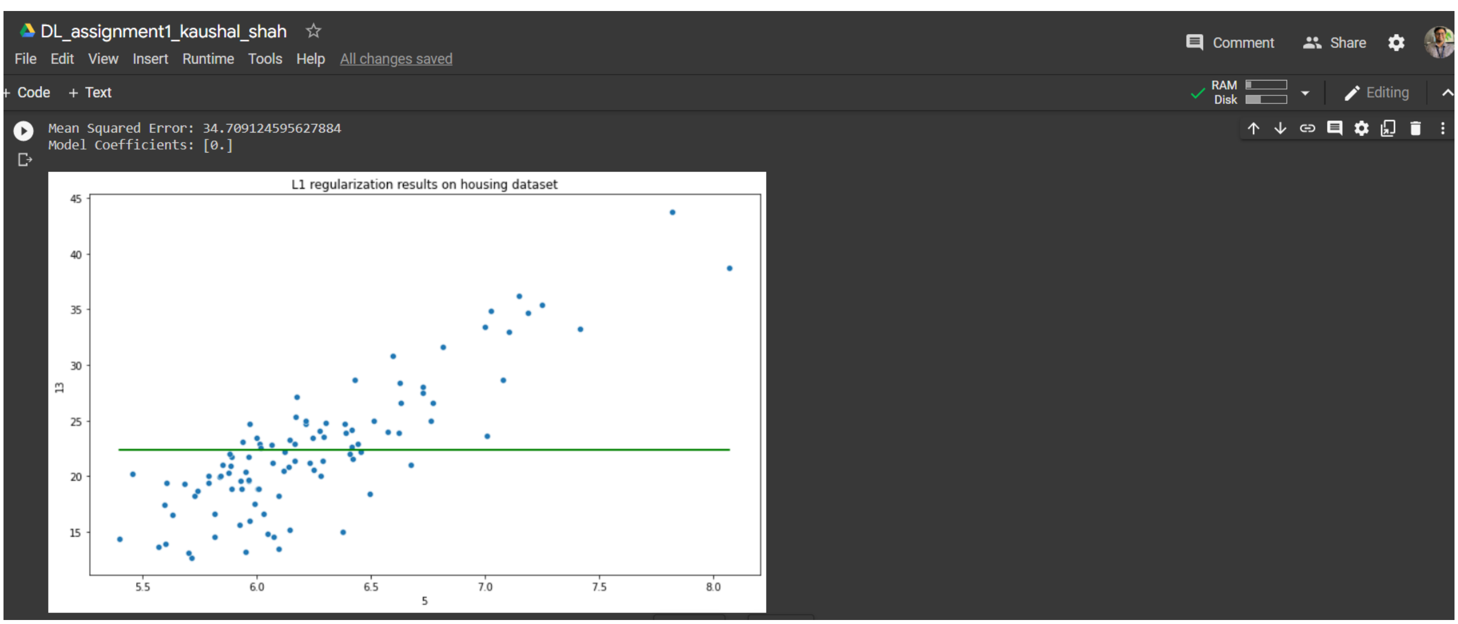


Figure 12: L1 regularization output

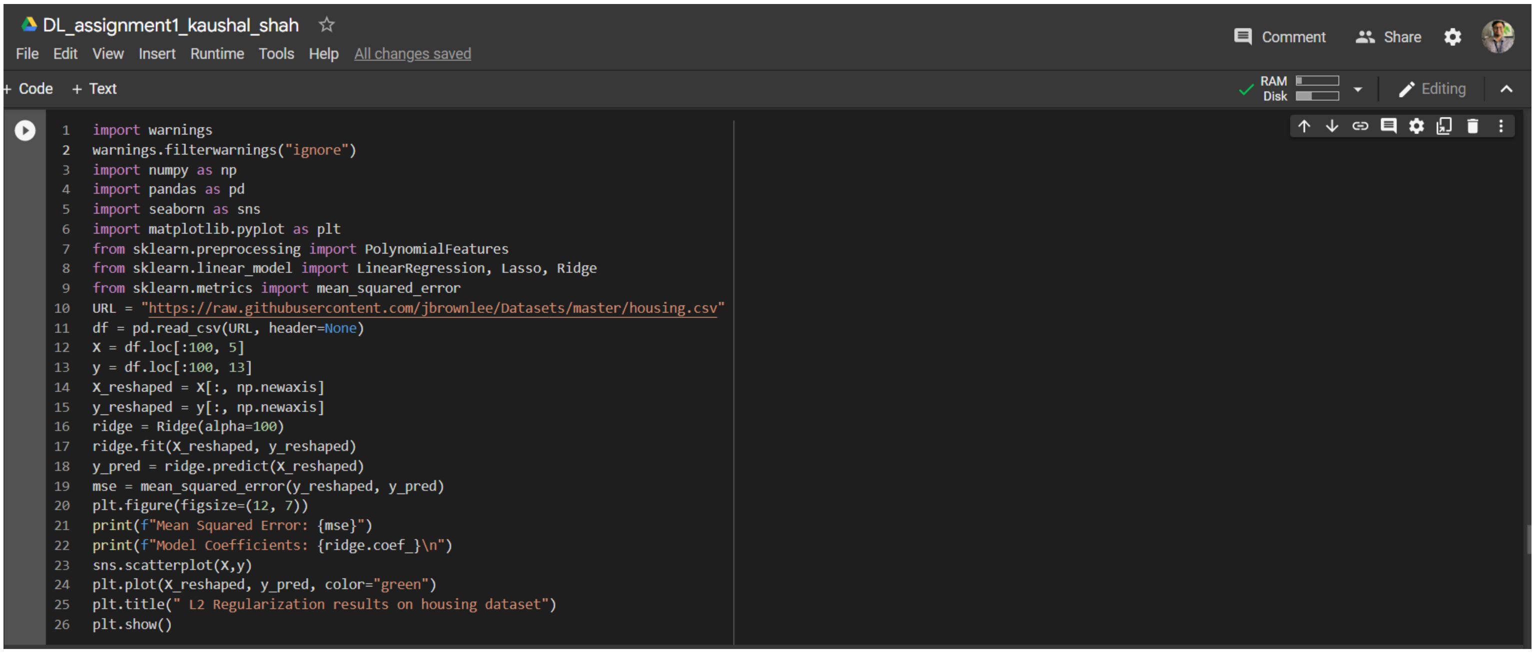


Figure 13: L2 regularization implementation code

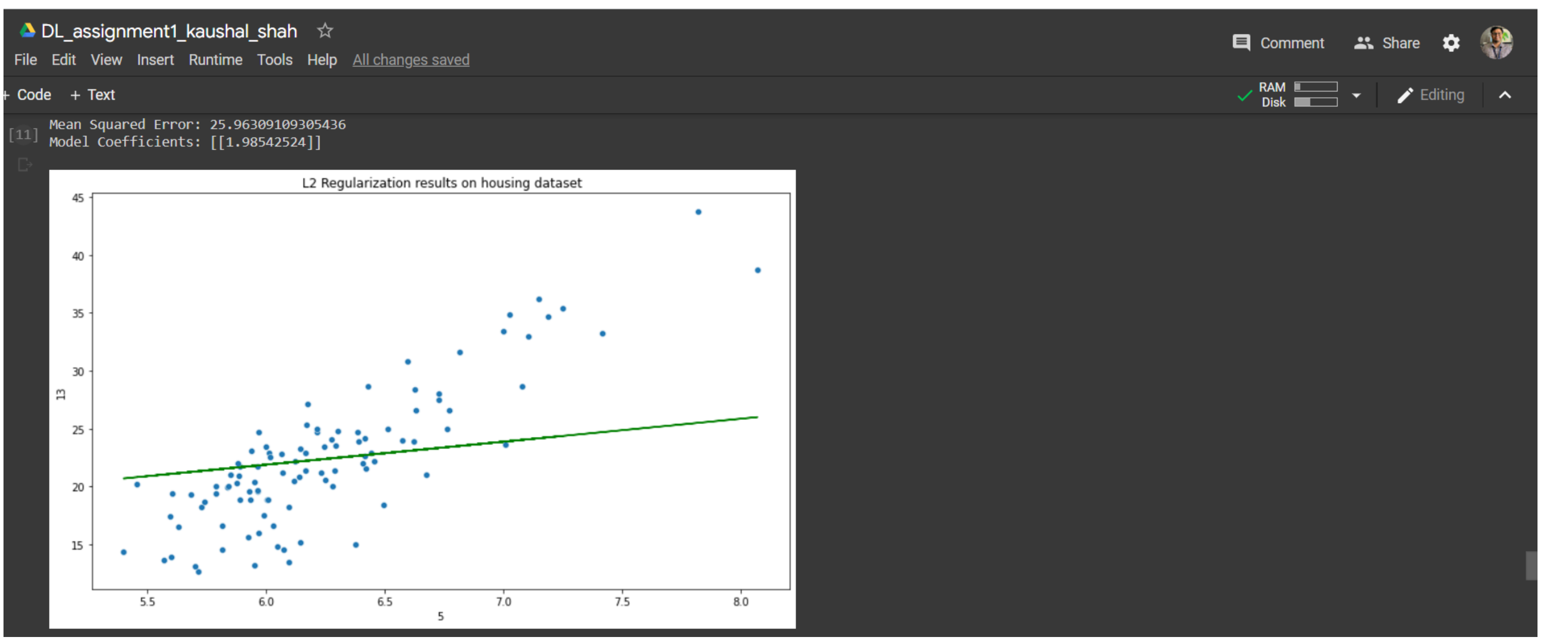


Figure 14: L2 regularization output

4 Implement various data augmentation techniques on image of your choice such as

Image augmentation operation implemented on MRI brain image dataset

a. Rotation

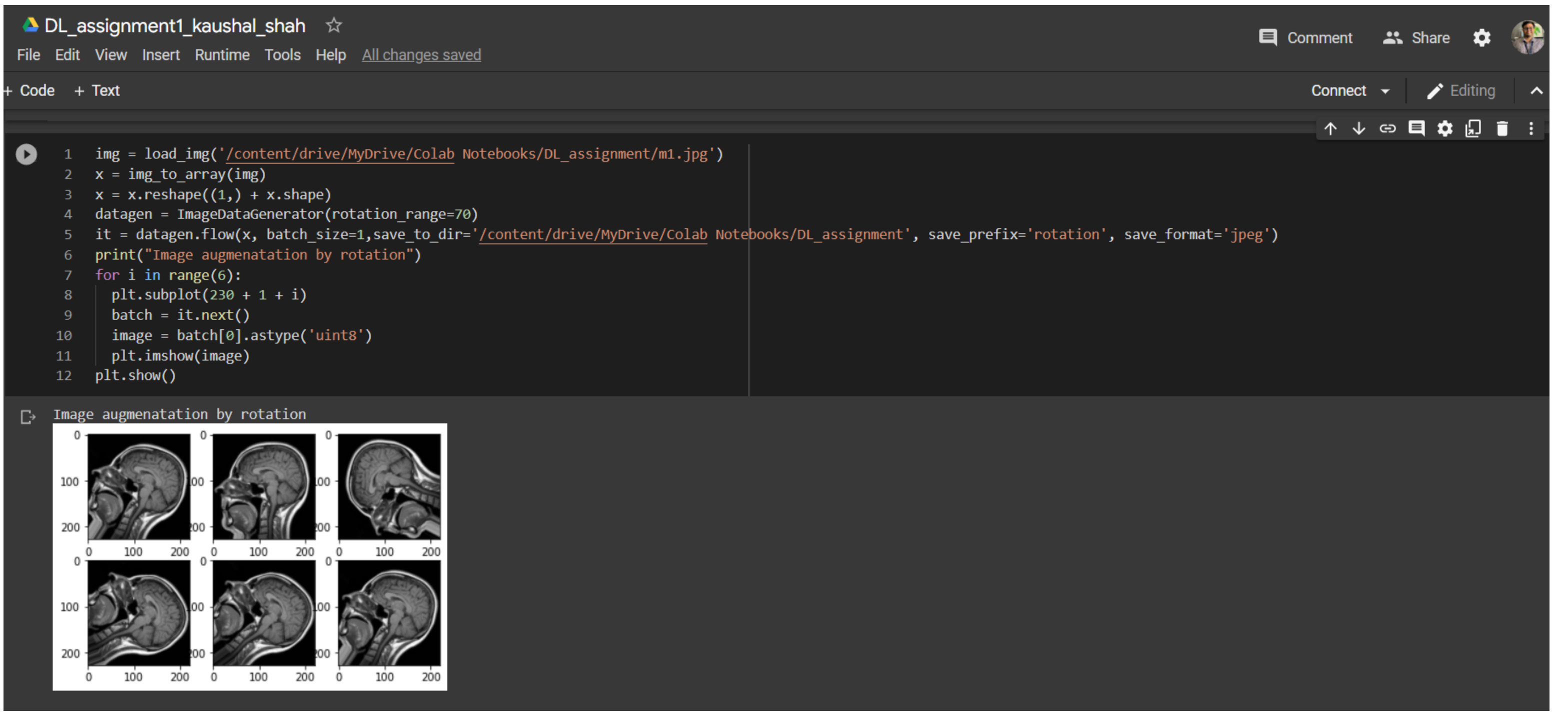


Figure 15: Image augmentation by Rotation of image

b. Shearing

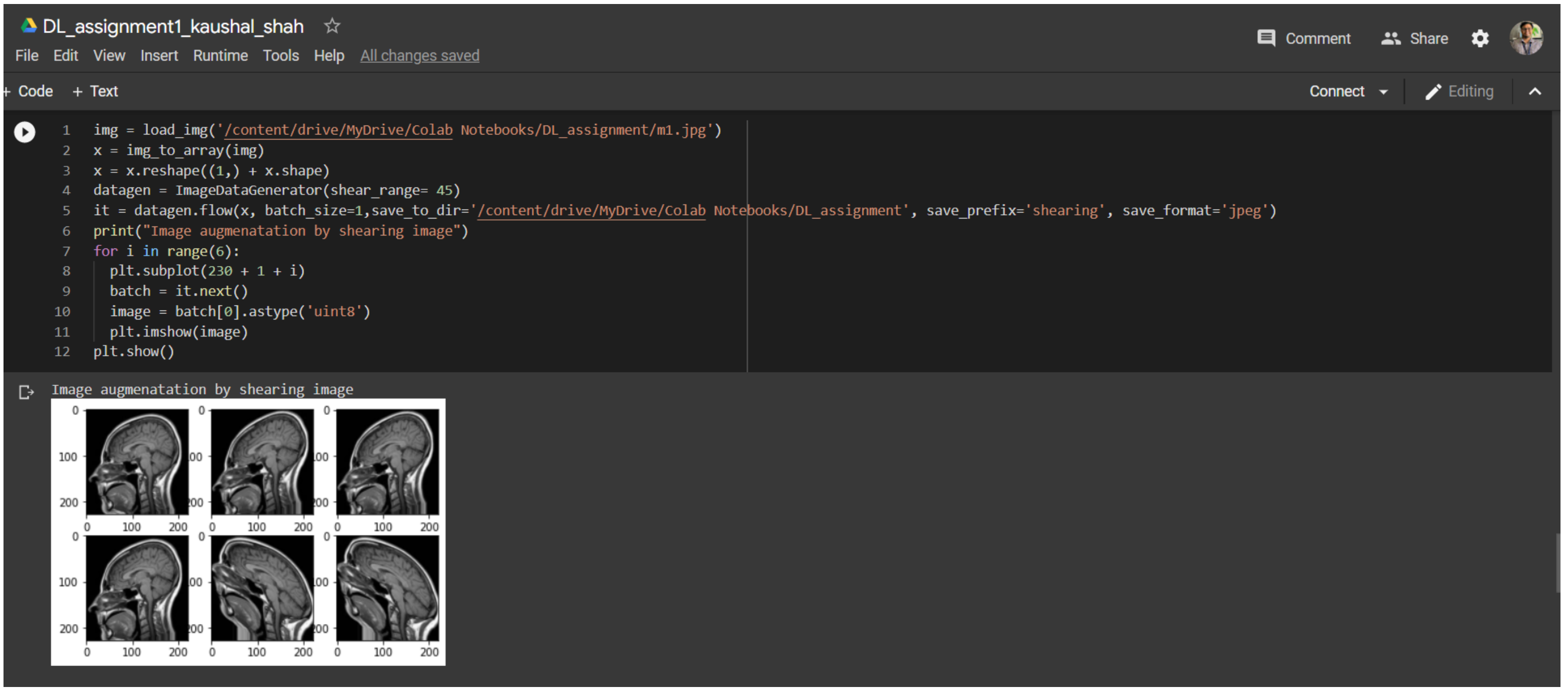


Figure 16: Image augmentation by Shearing of image

c. Zooming

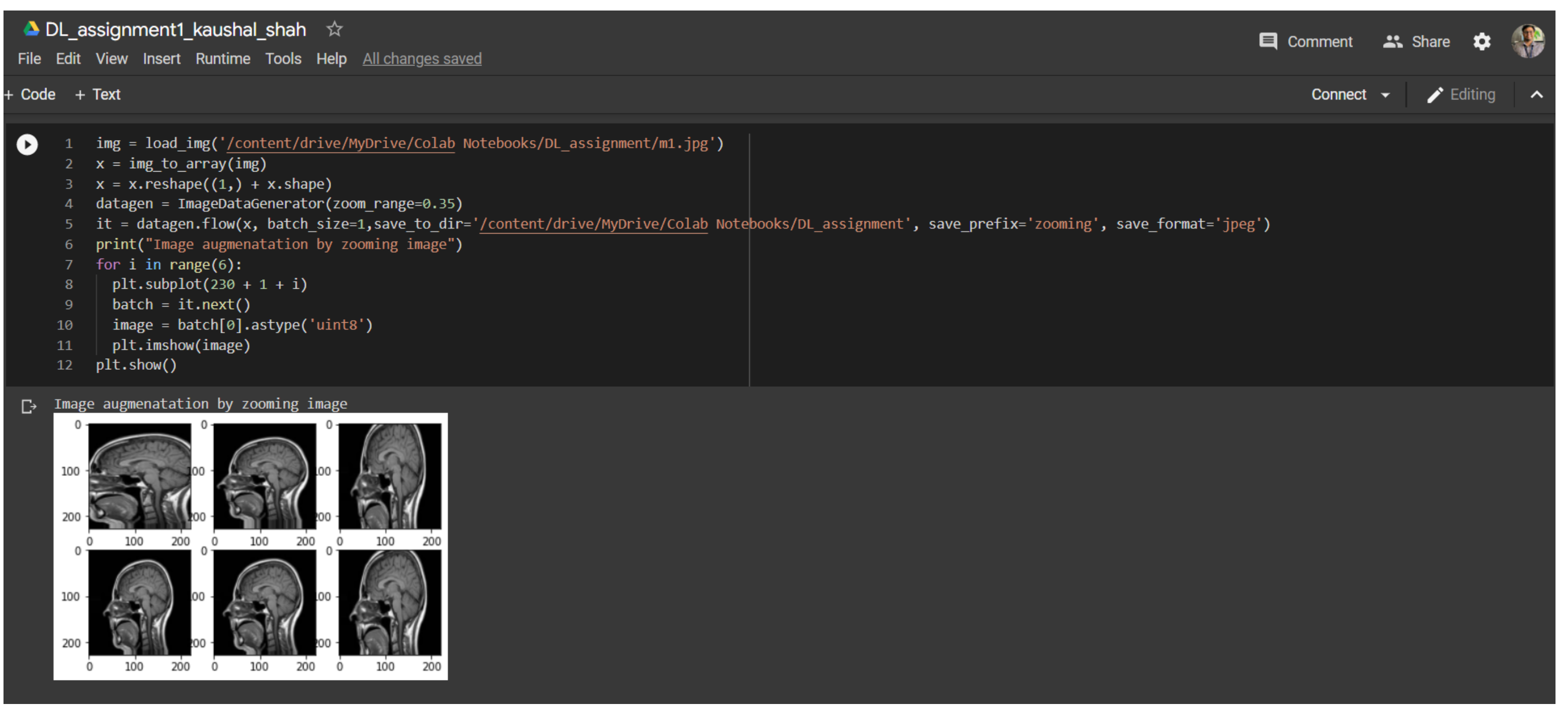
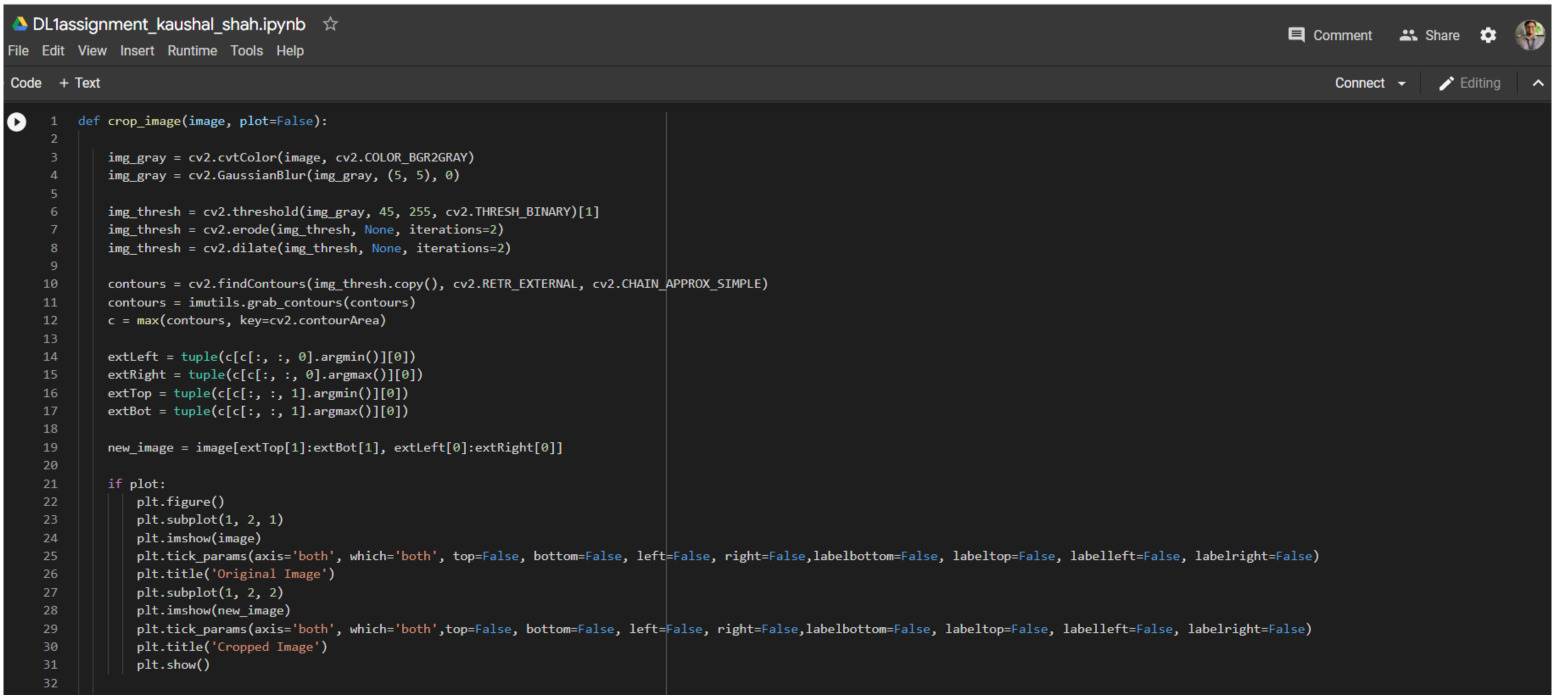


Figure 17: Image augmentation by Zooming of image

d. Cropping



```python
def crop_image(image, plot=False):

    img_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    img_gray = cv2.GaussianBlur(img_gray, (5, 5), 0)

    img_thresh = cv2.threshold(img_gray, 45, 255, cv2.THRESH_BINARY)[1]
    img_thresh = cv2.erode(img_thresh, None, iterations=2)
    img_thresh = cv2.dilate(img_thresh, None, iterations=2)

    contours = cv2.findContours(img_thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    contours = imutils.grab_contours(contours)
    c = max(contours, key=cv2.contourArea)

    extLeft = tuple(c[c[:, :, 0].argmin()][0])
    extRight = tuple(c[c[:, :, 0].argmax()][0])
    extTop = tuple(c[c[:, :, 1].argmin()][0])
    extBot = tuple(c[c[:, :, 1].argmax()][0])

    new_image = image[extTop[1]:extBot[1], extLeft[0]:extRight[0]]

    if plot:
        plt.figure()
        plt.subplot(1, 2, 1)
        plt.imshow(image)
        plt.tick_params(axis='both', which='both', top=False, bottom=False, left=False, right=False,labelbottom=False, labeltop=False, labelleft=False, labelright=False)
        plt.title('Original Image')
        plt.subplot(1, 2, 2)
        plt.imshow(new_image)
        plt.tick_params(axis='both', which='both',top=False, bottom=False, left=False, right=False,labelbottom=False, labeltop=False, labelleft=False, labelright=False)
        plt.title('Cropped Image')
        plt.show()
```

Figure 18: Code for implementation of Cropping of image

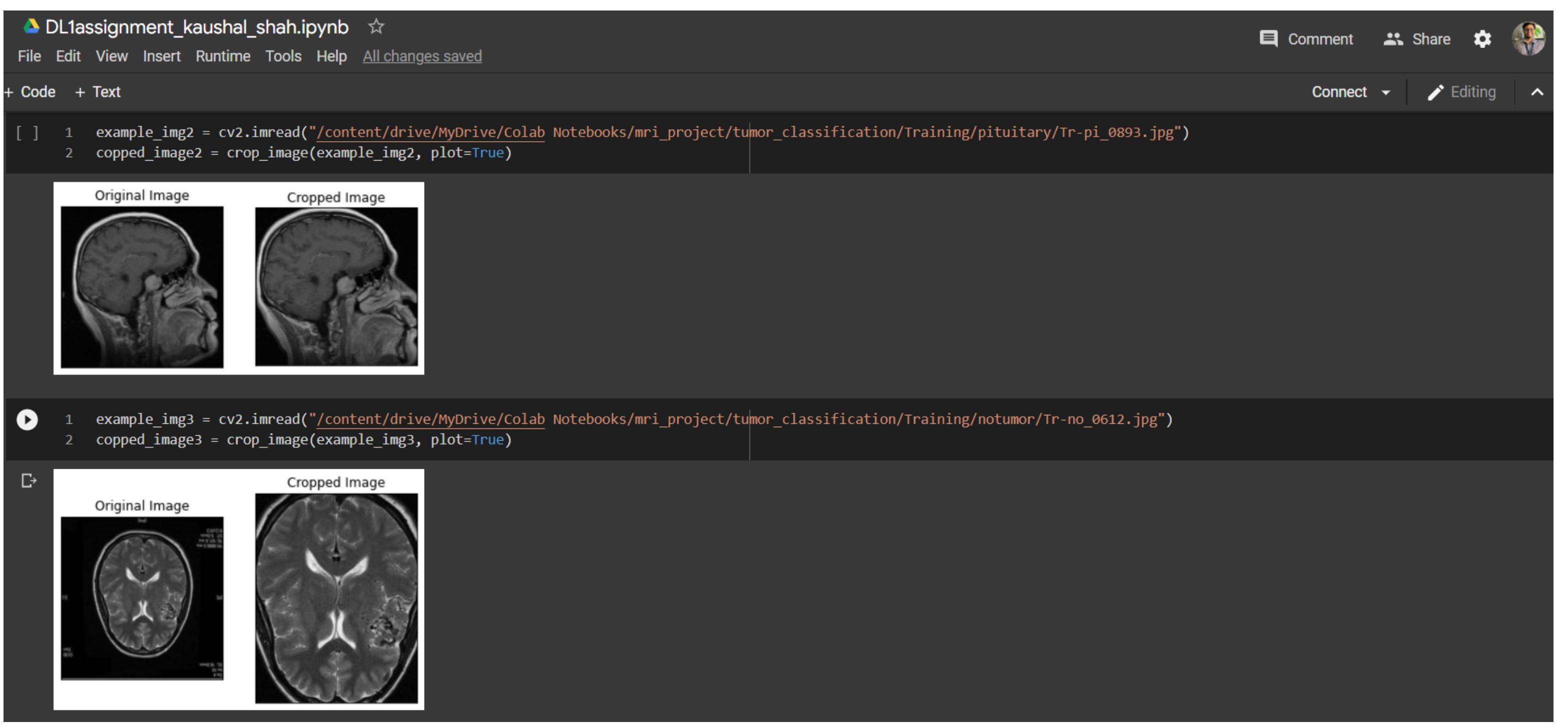


Figure 19: Image augmentation by Cropping of image

e. Flipping

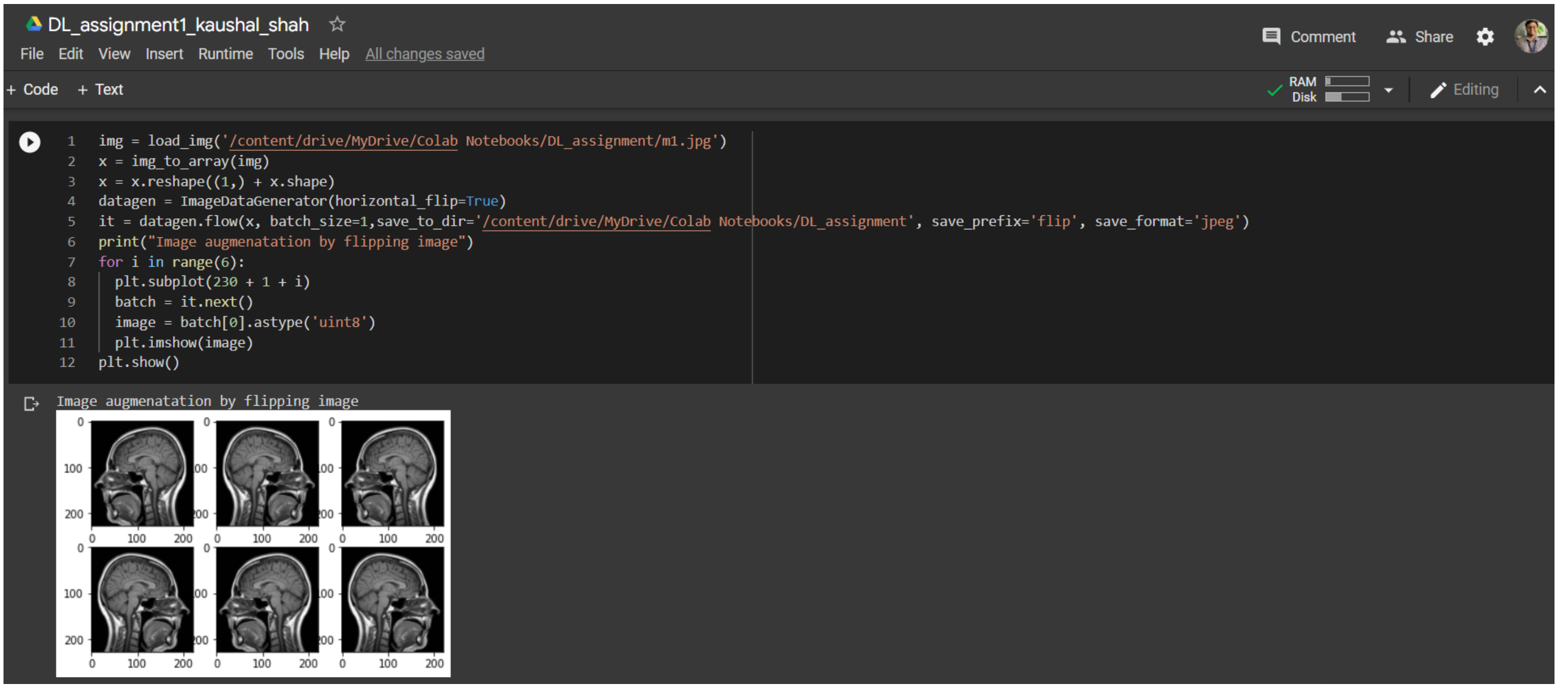


Figure 20: Image augmentation by Flipping of image

f. Changing the brightness level

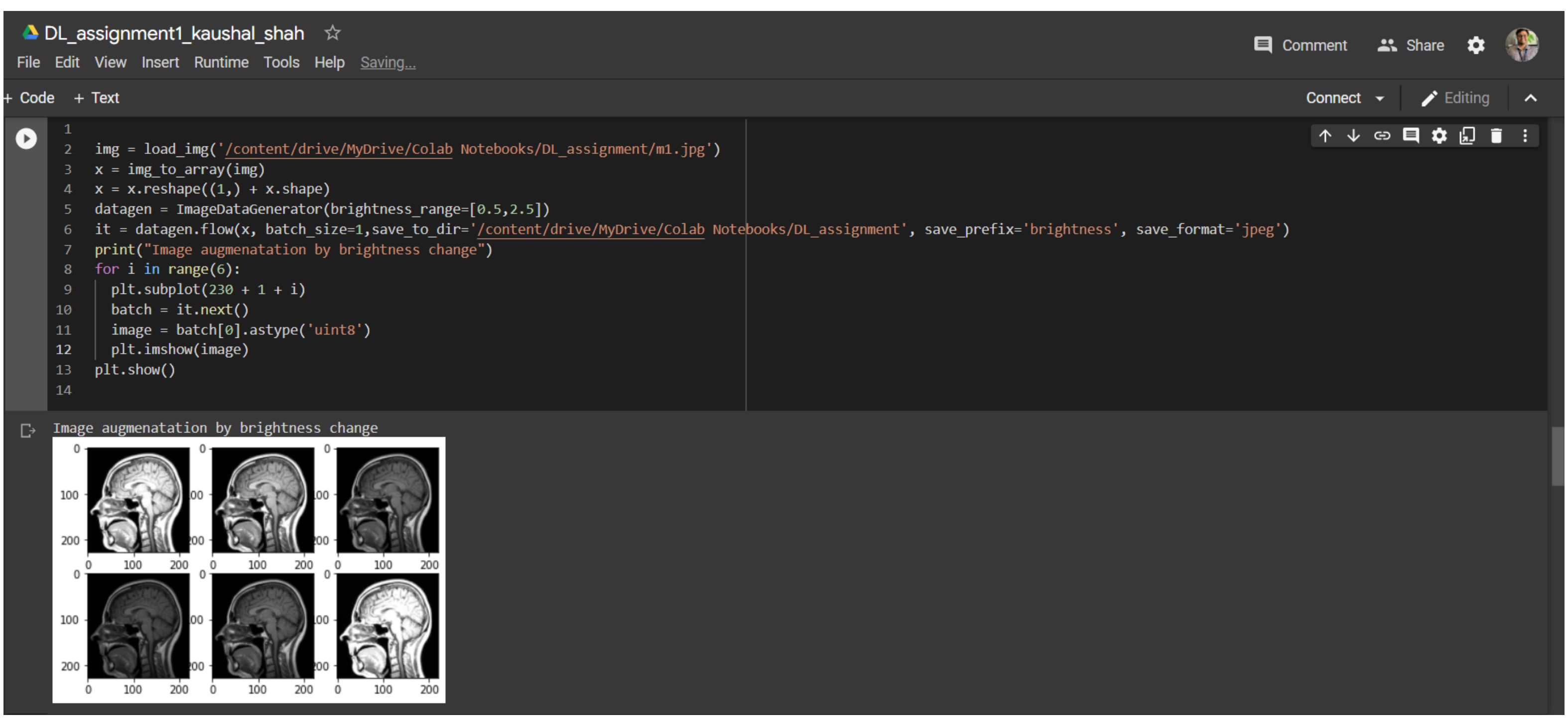


Figure 21: Image augmentation by changing the brightness level