SGD - Stochastic Gradient Descent

Link :-

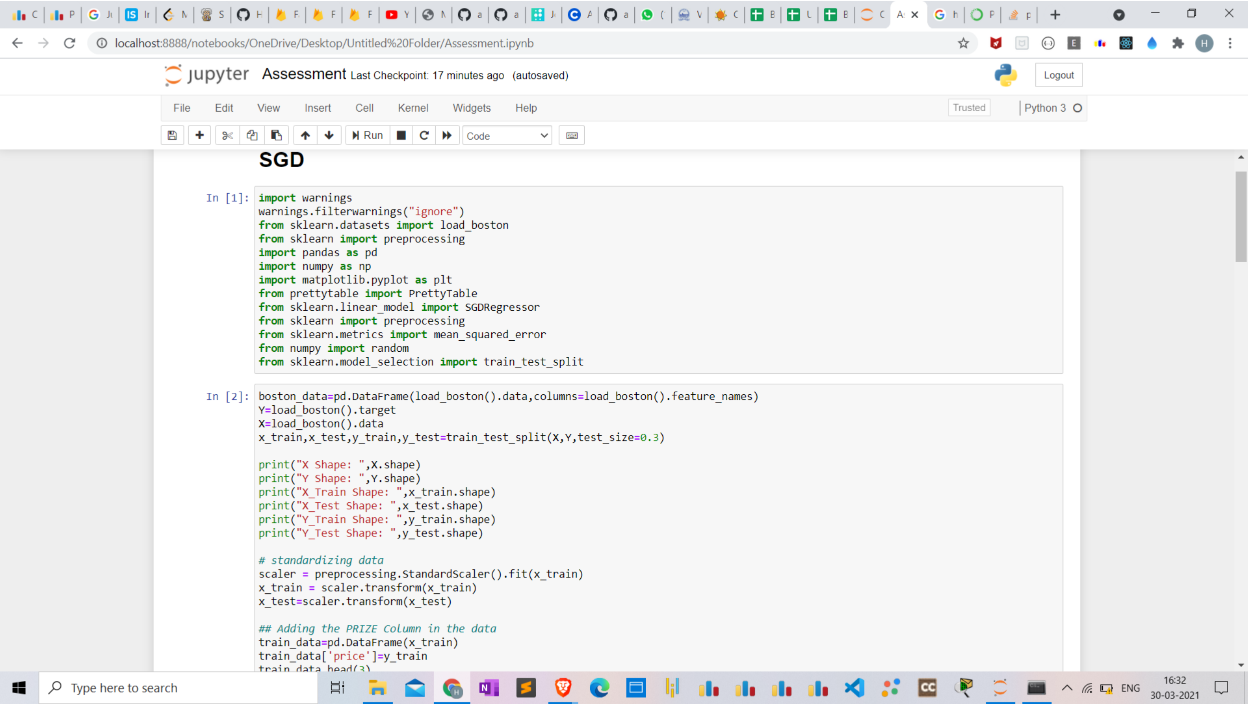
https://docs.google.com/spreadsheets/d/1uL7t2QqQwdBq0KLugIu0m2Iuaj8EIZTF3z7CgCrRvJk/edit?usp=sharing

Demonstration code :-

```
import warnings
warnings.filterwarnings("ignore")
from sklearn.datasets import load_boston
from sklearn import preprocessing
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from prettytable import PrettyTable
from sklearn.linear_model import SGDRegressor
from sklearn import preprocessing
from sklearn.metrics import mean_squared_error
from numpy import random
from sklearn.model_selection import train_test_split

boston_data=pd.DataFrame(load_boston().data,columns=load_boston().feature_names)
Y=load_boston().target
X=load_boston().data
x_train,x_test,y_train,y_test=train_test_split(X,Y,test_size=0.3)

print("X Shape: ",X.shape)
print("Y Shape: ",Y.shape)
print("X_Train Shape: ",x_train.shape)
print("X_Test Shape: ",x_test.shape)
print("Y_Train Shape: ",y_train.shape)
print("Y_Test Shape: ",y_test.shape)

# standardizing data
scaler = preprocessing.StandardScaler().fit(x_train)
x_train = scaler.transform(x_train)
x_test=scaler.transform(x_test)

## Adding the PRIZE Column in the data
train_data=pd.DataFrame(x_train)
train_data['price']=y_train
```

```
train_data.head(3)

x_test=np.array(x_test)
y_test=np.array(y_test)

n_iter=100
clf_ = SGDRegressor(max_iter=n_iter)
clf_.fit(x_train, y_train)
y_pred_sksgd=clf_.predict(x_test)
plt.scatter(y_test,y_pred_sksgd)
plt.grid()
plt.xlabel('Actual y')
plt.ylabel('Predicted y')
plt.title('Scatter plot from actual y and predicted y')
plt.show()

print('Mean Squared Error :',mean_squared_error(y_test, y_pred_sksgd))

def MyCustomSGD(train_data,learning_rate,n_iter,k,divideby):

    # Initially we will keep our W and B as 0 as per the Training Data
    w=np.zeros(shape=(1,train_data.shape[1]-1))
    b=0

    cur_iter=1
    while(cur_iter<=n_iter):

        # We will create a small training data set of size K
        temp=train_data.sample(k)

        # We create our X and Y from the above temp dataset
        y=np.array(temp['price'])
        x=np.array(temp.drop('price',axis=1))

        # We keep our initial gradients as 0
        w_gradient=np.zeros(shape=(1,train_data.shape[1]-1))
        b_gradient=0

        for i in range(k): # Calculating gradients for point in our K sized dataset
            prediction=np.dot(w,x[i])+b
            w_gradient=w_gradient+(-2)*x[i]*(y[i]-(prediction))
            b_gradient=b_gradient+(-2)*(y[i]-(prediction))

        #Updating the weights(W) and Bias(b) with the above calculated Gradients
        w=w-learning_rate*(w_gradient/k)
        b=b-learning_rate*(b_gradient/k)

        # Incrementing the iteration value
        cur_iter=cur_iter+1

        #Dividing the learning rate by the specified value
        learning_rate=learning_rate/divideby

    return w,b #Returning the weights and Bias

def predict(x,w,b):
    y_pred=[]
    for i in range(len(x)):
        y=np.asscalar(np.dot(w,x[i])+b)
        y_pred.append(y)
    return np.array(y_pred)

w,b=MyCustomSGD(train_data,learning_rate=1,n_iter=100,divideby=2,k=10)
y_pred_customsgd=predict(x_test,w,b)

plt.scatter(y_test,y_pred_customsgd)
plt.grid()
plt.xlabel('Actual y')
plt.ylabel('Predicted y')
plt.title('Scatter plot from actual y and predicted y')
plt.show()
print('Mean Squared Error :',mean_squared_error(y_test, y_pred_customsgd))

w,b=MyCustomSGD(train_data,learning_rate=0.001,n_iter=1000,divideby=1,k=10)
y_pred_customsgd_improved=predict(x_test,w,b)

plt.scatter(y_test,y_pred_customsgd_improved)
plt.grid()
plt.xlabel('Actual y')
plt.ylabel('Predicted y')
plt.title('Scatter plot from actual y and predicted y')
plt.show()
print('Mean Squared Error :',mean_squared_error(y_test, y_pred_customsgd_improved))
```

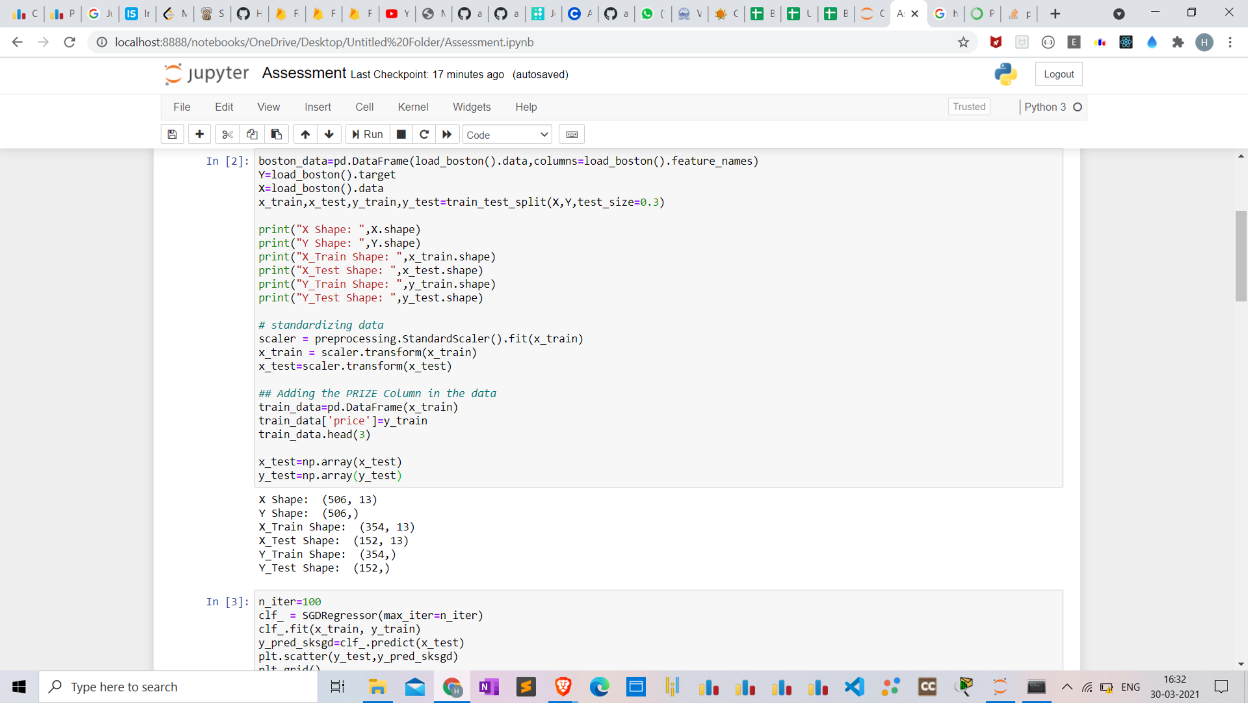
Screenshots :-
(Images may take a while to load)


localhost:8888/notebooks/OneDrive/Desktop/Untitled%20Folder/Assessment.ipynb
jupyter Assessment Last Checkpoint: 17 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
Type here to search
ENG
16:32
30-03-2021

SGD

In [1]:
```
import warnings
warnings.filterwarnings("ignore")
from sklearn.datasets import load_boston
from sklearn import preprocessing
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from prettytable import PrettyTable
from sklearn.linear_model import SGDRegressor
from sklearn import preprocessing
from sklearn.metrics import mean_squared_error
from numpy import random
from sklearn.model_selection import train_test_split
```

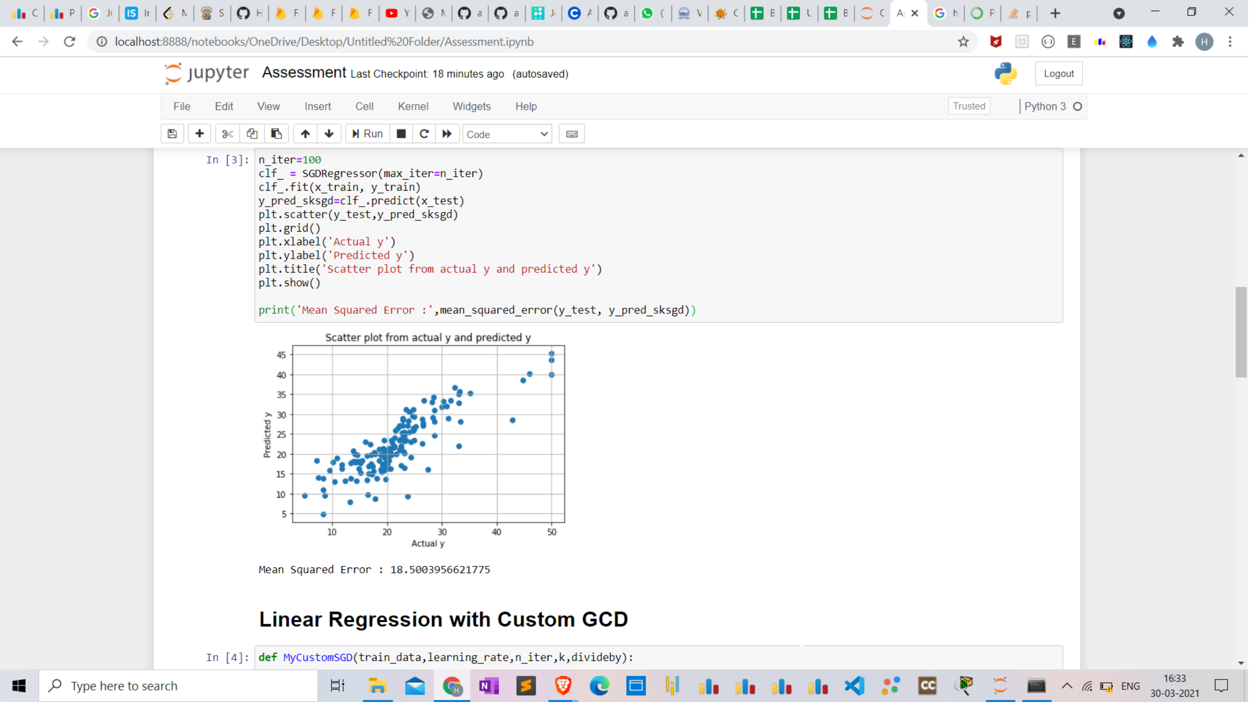
In [2]:
```
boston_data=pd.DataFrame(load_boston().data,columns=load_boston().feature_names)
Y=load_boston().target
X=load_boston().data
x_train,x_test,y_train,y_test=train_test_split(X,Y,test_size=0.3)

print("X Shape: ",X.shape)
print("Y Shape: ",Y.shape)
print("X_Train Shape: ",x_train.shape)
print("X_Test Shape: ",x_test.shape)
print("Y_Train Shape: ",y_train.shape)
print("Y_Test Shape: ",y_test.shape)

# standardizing data
scaler = preprocessing.StandardScaler().fit(x_train)
x_train = scaler.transform(x_train)
x_test=scaler.transform(x_test)

## Adding the PRIZE Column in the data
train_data=pd.DataFrame(x_train)
train_data['price']=y_train
train_data.head(3)
```


localhost:8888/notebooks/OneDrive/Desktop/Untitled%20Folder/Assessment.ipynb
jupyter Assessment Last Checkpoint: 17 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
Type here to search
ENG
16:32
30-03-2021

In [2]:
```
boston_data=pd.DataFrame(load_boston().data,columns=load_boston().feature_names)
Y=load_boston().target
X=load_boston().data
x_train,x_test,y_train,y_test=train_test_split(X,Y,test_size=0.3)

print("X Shape: ",X.shape)
print("Y Shape: ",Y.shape)
print("X_Train Shape: ",x_train.shape)
print("X_Test Shape: ",x_test.shape)
print("Y_Train Shape: ",y_train.shape)
print("Y_Test Shape: ",y_test.shape)

# standardizing data
scaler = preprocessing.StandardScaler().fit(x_train)
x_train = scaler.transform(x_train)
x_test=scaler.transform(x_test)

## Adding the PRIZE Column in the data
train_data=pd.DataFrame(x_train)
train_data['price']=y_train
train_data.head(3)

x_test=np.array(x_test)
y_test=np.array(y_test)
```

```
X Shape:  (506, 13)
Y Shape:  (506,)
X_Train Shape:  (354, 13)
X_Test Shape:  (152, 13)
Y_Train Shape:  (354,)
Y_Test Shape:  (152,)
```

In [3]:
```
n_iter=100
clf_ = SGDRegressor(max_iter=n_iter)
clf_.fit(x_train, y_train)
y_pred_sksgd=clf_.predict(x_test)
plt.scatter(y_test,y_pred_sksgd)
plt.grid()
```

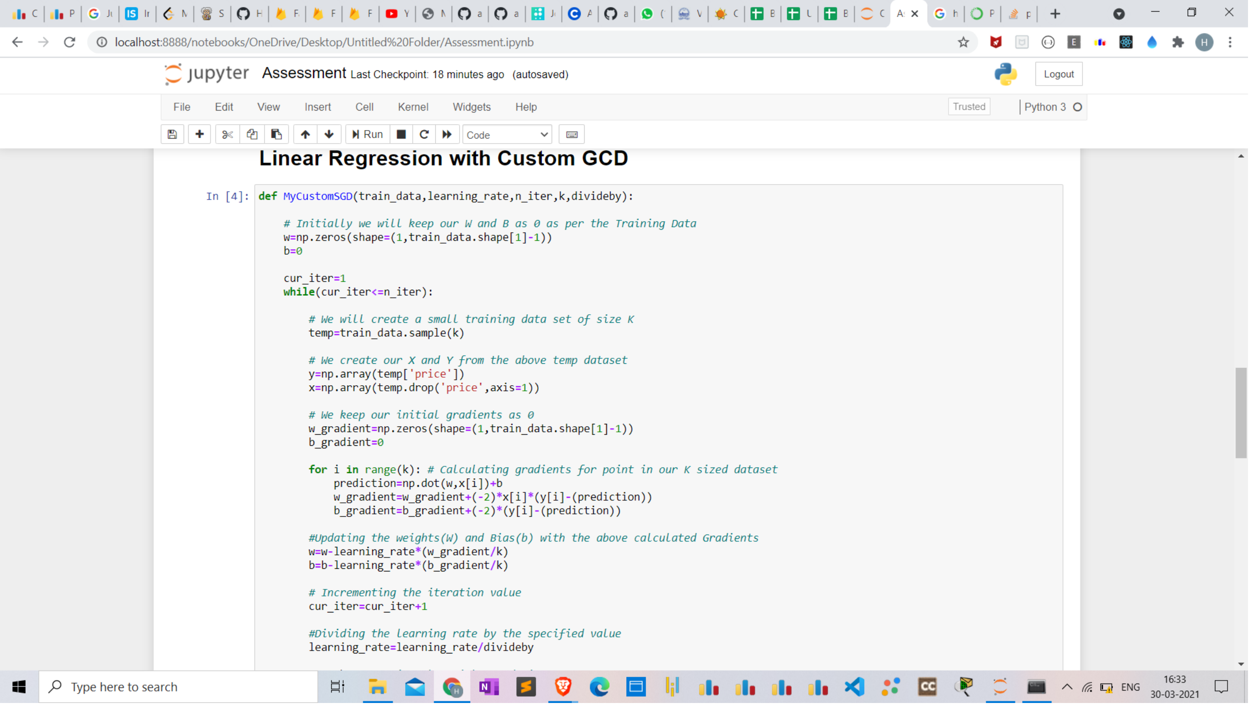

localhost:8888/notebooks/OneDrive/Desktop/Untitled%20Folder/Assessment.ipynb
jupyter Assessment Last Checkpoint: 18 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
Scatter plot from actual y and predicted y
Predicted y
Actual y
45
40
35
30
25
20
15
10
5
10
20
30
40
50
Type here to search
ENG
16:33
30-03-2021

```
In [3]: n_iter=100
        clf_ = SGDRegressor(max_iter=n_iter)
        clf_.fit(x_train, y_train)
        y_pred_sksgd=clf_.predict(x_test)
        plt.scatter(y_test,y_pred_sksgd)
        plt.grid()
        plt.xlabel('Actual y')
        plt.ylabel('Predicted y')
        plt.title('Scatter plot from actual y and predicted y')
        plt.show()

        print('Mean Squared Error :',mean_squared_error(y_test, y_pred_sksgd))
```

```
Mean Squared Error : 18.5003956621775
```

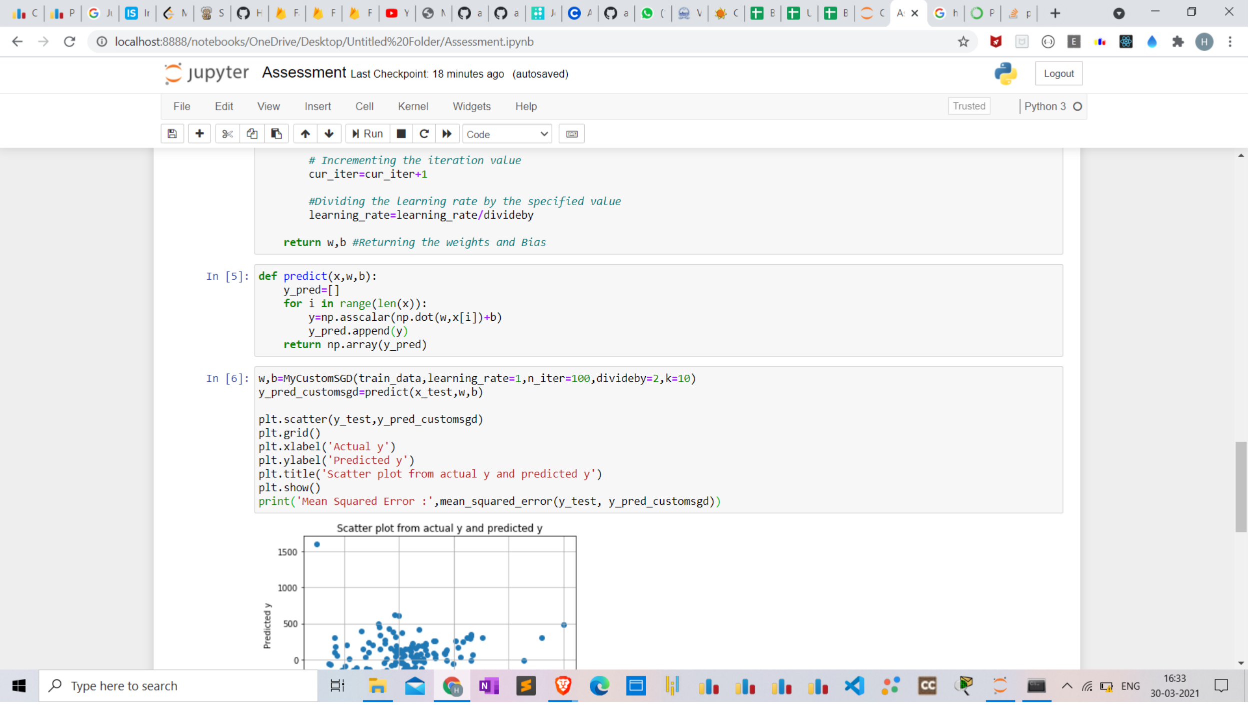
Linear Regression with Custom GCD

```
In [4]: def MyCustomSGD(train_data,learning_rate,n_iter,k,divideby):
```


localhost:8888/notebooks/OneDrive/Desktop/Untitled%20Folder/Assessment.ipynb
jupyter Assessment Last Checkpoint: 18 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
Type here to search
ENG
16:33
30-03-2021

Linear Regression with Custom GCD

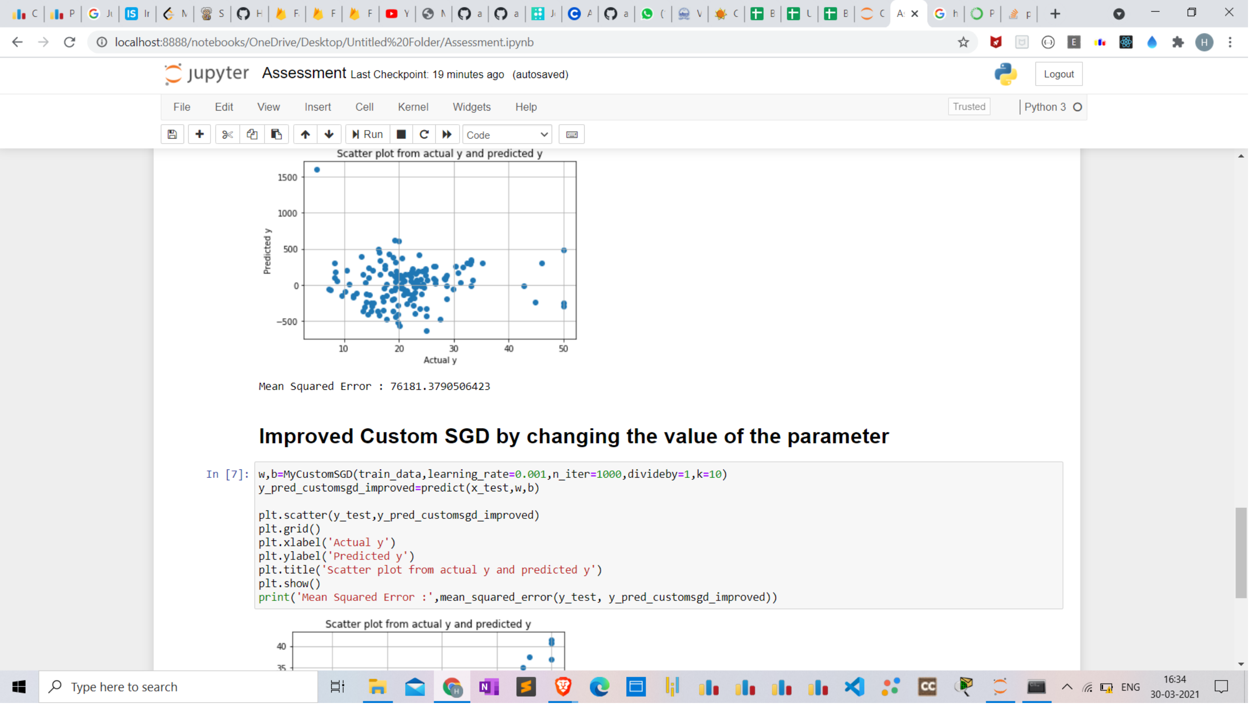
```
In [4]: def MyCustomSGD(train_data,learning_rate,n_iter,k,divideby):

            # Initially we will keep our W and B as 0 as per the Training Data
            w=np.zeros(shape=(1,train_data.shape[1]-1))
            b=0

            cur_iter=1
            while(cur_iter<=n_iter):

                # We will create a small training data set of size K
                temp=train_data.sample(k)

                # We create our X and Y from the above temp dataset
                y=np.array(temp['price'])
                x=np.array(temp.drop('price',axis=1))

                # We keep our initial gradients as 0
                w_gradient=np.zeros(shape=(1,train_data.shape[1]-1))
                b_gradient=0

                for i in range(k): # Calculating gradients for point in our K sized dataset
                    prediction=np.dot(w,x[i])+b
                    w_gradient=w_gradient+(-2)*x[i]*(y[i]-(prediction))
                    b_gradient=b_gradient+(-2)*(y[i]-(prediction))

                #Updating the weights(W) and Bias(b) with the above calculated Gradients
                w=w-learning_rate*(w_gradient/k)
                b=b-learning_rate*(b_gradient/k)

                # Incrementing the iteration value
                cur_iter=cur_iter+1

                #Dividing the learning rate by the specified value
                learning_rate=learning_rate/divideby
```


localhost:8888/notebooks/OneDrive/Desktop/Untitled%20Folder/Assessment.ipynb
jupyter
Assessment Last Checkpoint: 18 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
Scatter plot from actual y and predicted y
1500
1000
500
0
Predicted y
Type here to search
ENG
16:33
30-03-2021

```
        # Incrementing the iteration value
        cur_iter=cur_iter+1

        #Dividing the learning rate by the specified value
        learning_rate=learning_rate/divideby

    return w,b #Returning the weights and Bias
```

```
In [5]: def predict(x,w,b):
            y_pred=[]
            for i in range(len(x)):
                y=np.asscalar(np.dot(w,x[i])+b)
                y_pred.append(y)
            return np.array(y_pred)
```

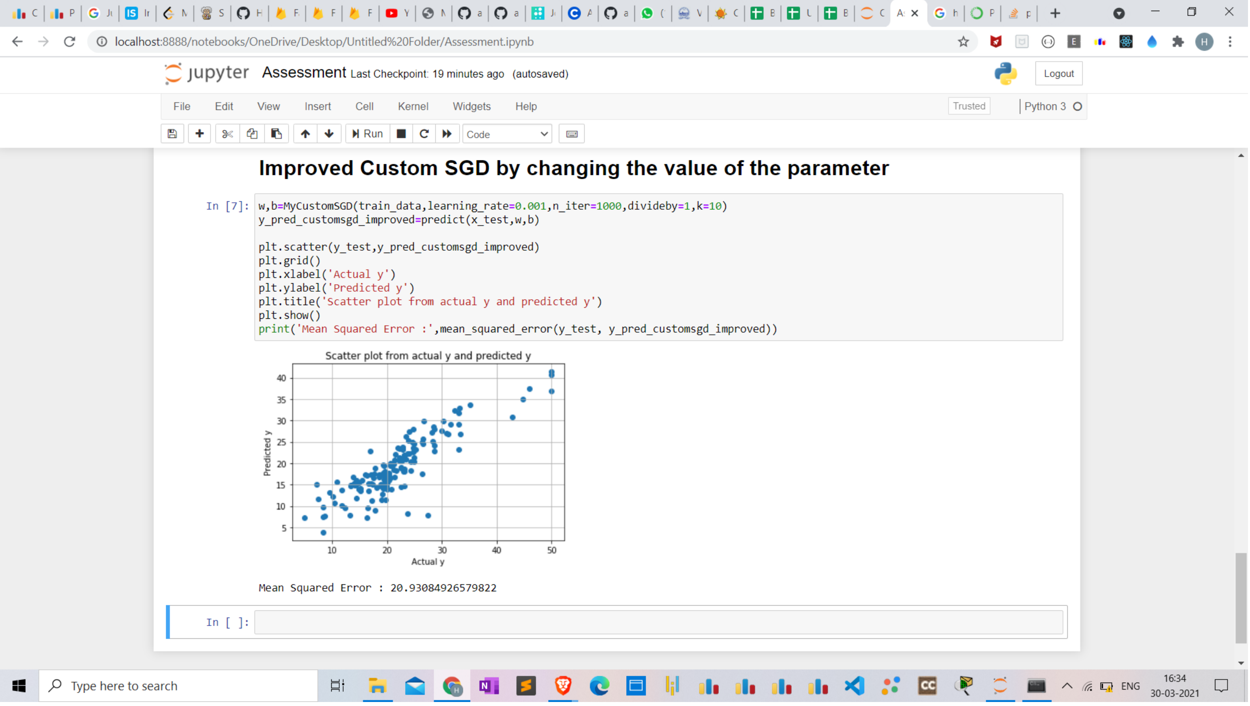
```
In [6]: w,b=MyCustomSGD(train_data,learning_rate=1,n_iter=100,divideby=2,k=10)
        y_pred_customsgd=predict(x_test,w,b)

        plt.scatter(y_test,y_pred_customsgd)
        plt.grid()
        plt.xlabel('Actual y')
        plt.ylabel('Predicted y')
        plt.title('Scatter plot from actual y and predicted y')
        plt.show()
        print('Mean Squared Error :',mean_squared_error(y_test, y_pred_customsgd))
```


localhost:8888/notebooks/OneDrive/Desktop/Untitled%20Folder/Assessment.ipynb
jupyter
Assessment Last Checkpoint: 19 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
Scatter plot from actual y and predicted y
1500
1000
500
0
−500
Predicted y
10
20
30
40
50
Actual y
Scatter plot from actual y and predicted y
40
35
Type here to search
ENG
16:34
30-03-2021

```
Mean Squared Error : 76181.3790506423
```

Improved Custom SGD by changing the value of the parameter

```
In [7]: w,b=MyCustomSGD(train_data,learning_rate=0.001,n_iter=1000,divideby=1,k=10)
        y_pred_customsgd_improved=predict(x_test,w,b)

        plt.scatter(y_test,y_pred_customsgd_improved)
        plt.grid()
        plt.xlabel('Actual y')
        plt.ylabel('Predicted y')
        plt.title('Scatter plot from actual y and predicted y')
        plt.show()
        print('Mean Squared Error :',mean_squared_error(y_test, y_pred_customsgd_improved))
```



localhost:8888/notebooks/OneDrive/Desktop/Untitled%20Folder/Assessment.ipynb
jupyter
Assessment Last Checkpoint: 19 minutes ago (autosaved)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3 ○
Run
Code
Type here to search
ENG
16:34
30-03-2021

Improved Custom SGD by changing the value of the parameter

```
In [7]: w,b=MyCustomSGD(train_data,learning_rate=0.001,n_iter=1000,divideby=1,k=10)
        y_pred_customsgd_improved=predict(x_test,w,b)

        plt.scatter(y_test,y_pred_customsgd_improved)
        plt.grid()
        plt.xlabel('Actual y')
        plt.ylabel('Predicted y')
        plt.title('Scatter plot from actual y and predicted y')
        plt.show()
        print('Mean Squared Error :',mean_squared_error(y_test, y_pred_customsgd_improved))
```

```
Mean Squared Error : 20.93084926579822
```

```
In [ ]:
```