SGD

May 22, 2020

0.1 Implement SGD for linear regression

To implement stochastic gradient descent to optimize a linear regression algorithm on Boston House Prices dataset which is already exists in sklearn as a sklearn.linear_model.SGDRegressor.here,SGD algorithm is defined manually and then comapring the both results.Linear regression is technique to predict on real values. ##### stochastic gradient descent technique , evaluates and updates the coefficients every iteration to minimize the error of a model on training data.

0.2 Objective:

To Implement stochastic gradient descent on Bostan House Prices dataset for linear Regression

- Implement SGD and deploy on Bostan House Prices dataset.
- Comapare the Results with sklearn.linear_model.SGDRegressor

[16]:
```
from sklearn.datasets import load_boston # to load  datasets from sklearn
import matplotlib.pyplot as plt
from sklearn.model_selection import cross_val_score

import sklearn.model_selection
from sklearn.model_selection import KFold
import numpy as np
import seaborn as sns
from sklearn.model_selection import train_test_split

from collections import Counter
from sklearn.metrics import accuracy_score

from sklearn.preprocessing import StandardScaler
import pandas as pd
import math

import pytablewriter
```

[6]:
```
boston = load_boston()
# Shape of Boston datasets
print(boston.data.shape)
```

```
(506, 13)
```

[7]:
```python
#  to understand datasets
print(boston.DESCR)
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value
(attribute 14) is usually the target.

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000
sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0
otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by
town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/


This dataset was taken from the StatLib library which is maintained at Carnegie
Mellon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
```

```
prices and the demand for clean air', J. Environ. Economics & Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics
…', Wiley, 1980.   N.B. Various transformations are used in the table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers that
address regression
problems.

.. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential
Data and Sources of Collinearity', Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In
Proceedings on the Tenth International Conference of Machine Learning, 236-243,
University of Massachusetts, Amherst. Morgan Kaufmann.
```

[8]:

```python
col= boston.feature_names
print(col)
```

```
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
 'B' 'LSTAT']
```

[9]:

```python
# real price values of bostan house datasets.
print(boston.target[:10])

# Output is real valued number
```

```
[24.  21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9]
```

[10]:

```python
# Boston datasets
bostan = pd.DataFrame(boston.data)
print(bostan.head())
# Boston dataset with columns names
bostan_col =pd.DataFrame(boston.data,columns=col)
print(bostan_col.head())
```

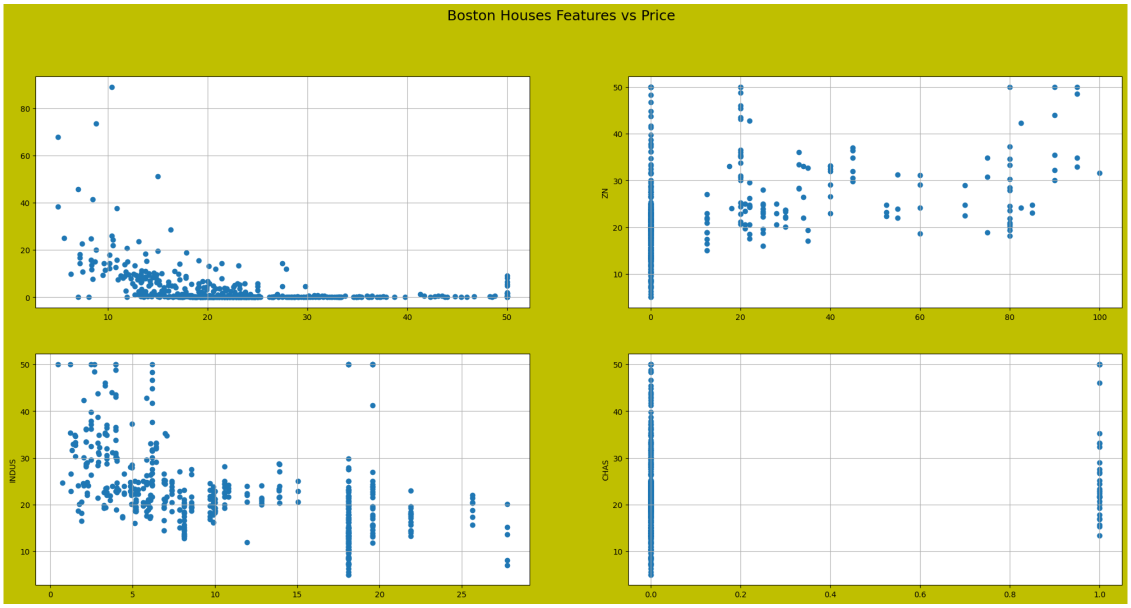
```
         0     1     2    3      4      5     6       7    8      9    10  \
0  0.00632  18.0  2.31  0.0  0.538  6.575  65.2  4.0900  1.0  296.0  15.3
1  0.02731   0.0  7.07  0.0  0.469  6.421  78.9  4.9671  2.0  242.0  17.8
2  0.02729   0.0  7.07  0.0  0.469  7.185  61.1  4.9671  2.0  242.0  17.8
3  0.03237   0.0  2.18  0.0  0.458  6.998  45.8  6.0622  3.0  222.0  18.7
4  0.06905   0.0  2.18  0.0  0.458  7.147  54.2  6.0622  3.0  222.0  18.7

       11    12
0  396.90  4.98
1  396.90  9.14
```

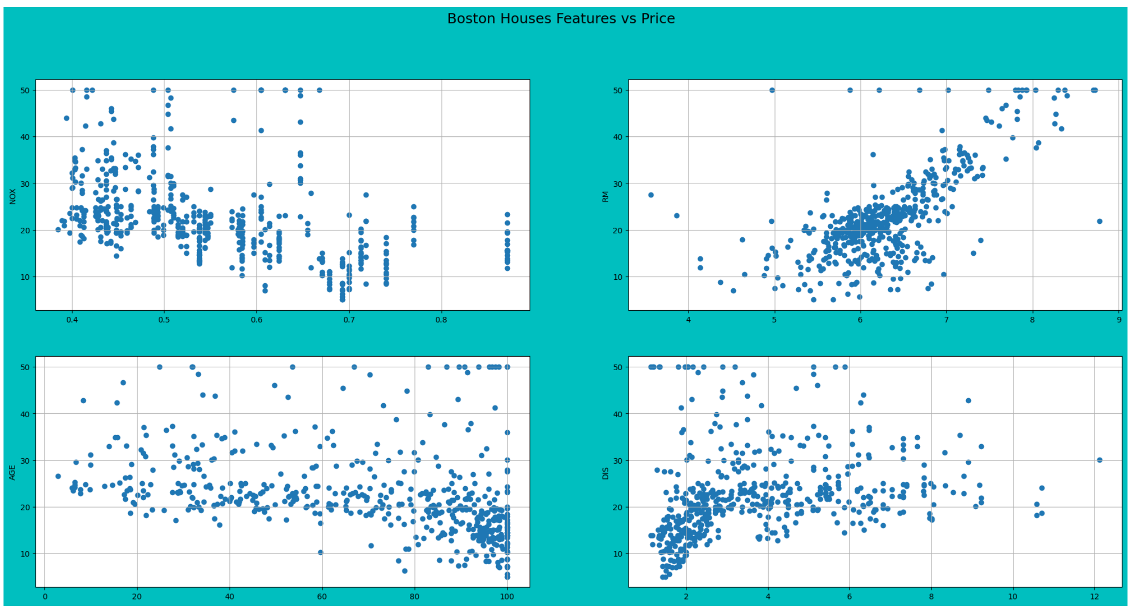
```
2  392.83  4.03
3  394.63  2.94
4  396.90  5.33
      CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  \
0  0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900  1.0  296.0
1  0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671  2.0  242.0
2  0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671  2.0  242.0
3  0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622  3.0  222.0
4  0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622  3.0  222.0

   PTRATIO       B  LSTAT
0     15.3  396.90   4.98
1     17.8  396.90   9.14
2     17.8  392.83   4.03
3     18.7  394.63   2.94
4     18.7  396.90   5.33
```

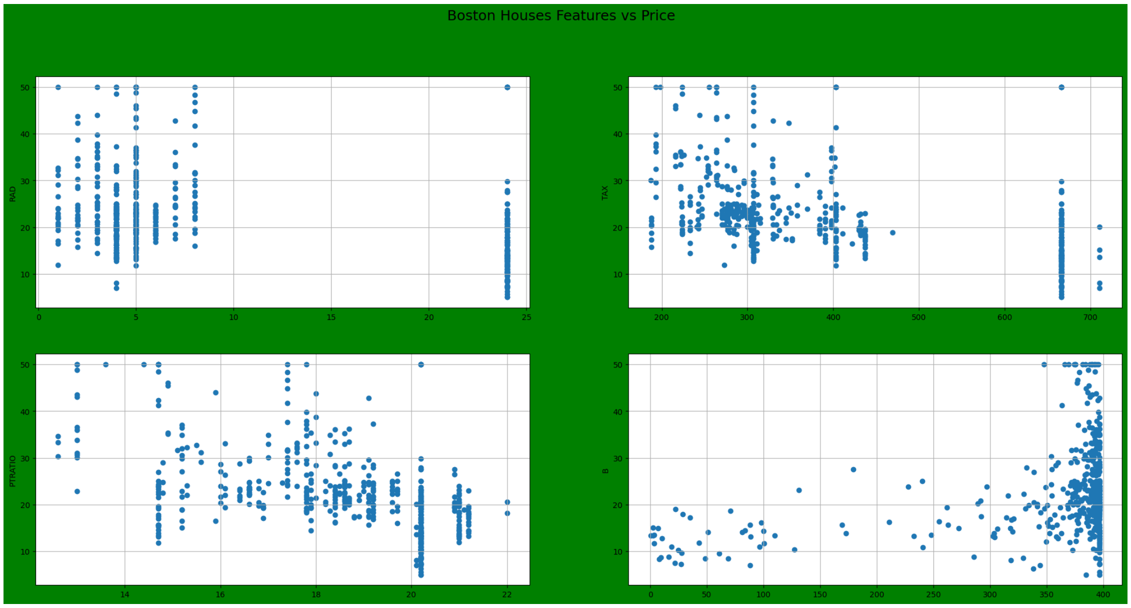
Boston Houses Features vs Price

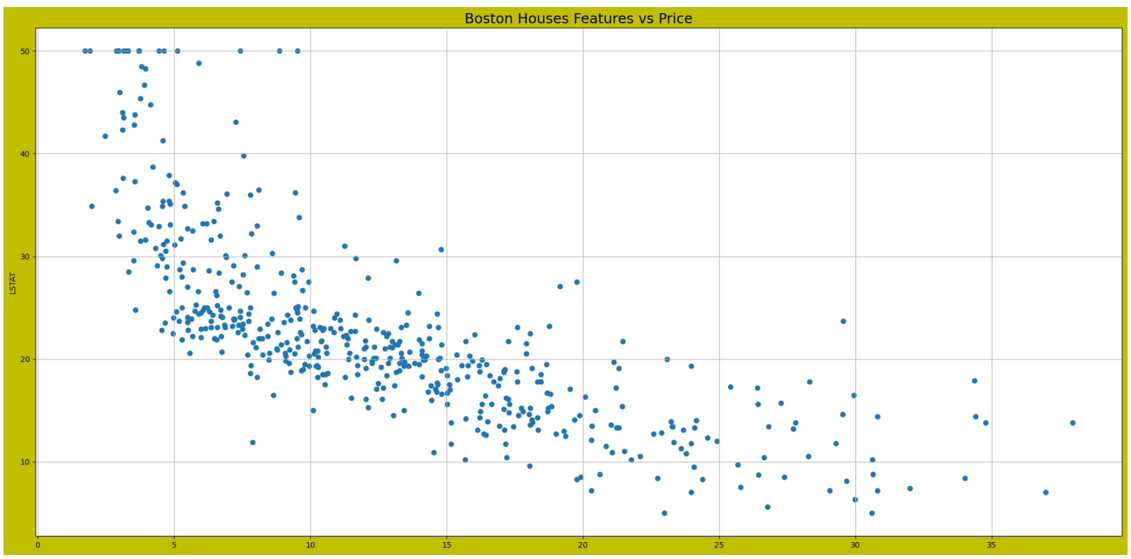
[11]:
```
#ax.title.set_text('Boston Houses Features vs Price')
fig = plt.figure(num=None, figsize=(25, 12), dpi=100, facecolor='y',␣
 ↪edgecolor='k')
fig.suptitle('Boston Houses Features vs Price', fontsize=18)
ax1 = fig.add_subplot(221)

ax1.scatter(boston.target,bostan_col.CRIM)
plt.grid()
ax2 = fig.add_subplot(222)
plt.ylabel('CRIM')
ax2.scatter(bostan_col.ZN,boston.target)
plt.ylabel('ZN')
plt.grid()
ax3 = fig.add_subplot(223)

ax3.scatter(bostan_col.INDUS,boston.target)
plt.ylabel('INDUS')
plt.grid()
ax4 = fig.add_subplot(224)
ax4.scatter(bostan_col.CHAS,boston.target)
plt.ylabel('CHAS')
plt.grid()
plt.show()
fig1 = plt.figure(num=None, figsize=(25, 12), dpi=100, facecolor='c',␣
 ↪edgecolor='k')
fig1.suptitle('Boston Houses Features vs Price', fontsize=18)
ax5 = fig1.add_subplot(221)
ax5.scatter(bostan_col.NOX,boston.target)
```

```
plt.ylabel('NOX')
plt.grid()
ax6 = fig1.add_subplot(222)
ax6.scatter(bostan_col.RM,boston.target)
plt.ylabel('RM')
plt.grid()
ax7 = fig1.add_subplot(223)
ax7.scatter(bostan_col.AGE,boston.target)
plt.ylabel('AGE')
plt.grid()
ax8 = fig1.add_subplot(224)
ax8.scatter(bostan_col.DIS,boston.target)
plt.ylabel('DIS')
plt.grid()
plt.show()
fig2 = plt.figure(num=None, figsize=(25, 12), dpi=100, facecolor='g',␣
 ↪edgecolor='k')
fig2.suptitle('Boston Houses Features vs Price', fontsize=18)
ax9 = fig2.add_subplot(221)
ax9.scatter(bostan_col.RAD,boston.target)
plt.ylabel('RAD')
plt.grid()
ax10 = fig2.add_subplot(222)
ax10.scatter(bostan_col.TAX,boston.target)
plt.ylabel('TAX')
plt.grid()
ax11 = fig2.add_subplot(223)
ax11.scatter(bostan_col.PTRATIO,boston.target)
plt.ylabel('PTRATIO')
plt.grid()
ax12 = fig2.add_subplot(224)
ax12.scatter(bostan_col.B,boston.target)
plt.ylabel('B')
plt.grid()
fig3 = plt.figure(num=None, figsize=(25, 12), dpi=100, facecolor='y',␣
 ↪edgecolor='k')

plt.scatter(bostan_col.LSTAT,boston.target)
plt.title('Boston Houses Features vs Price', fontsize=18)
plt.ylabel('LSTAT')
plt.grid()
plt.show()
```


Boston Houses Features vs Price
ZN
INDUS
CHAS


Boston Houses Features vs Price
NOX
RM
AGE
DIS





[12]:

```
bostan['PRICE'] = boston.target
# Boston datasets with 13 feautures  label as X
X = bostan.drop('PRICE', axis = 1)
#Boston dataset's price for 13 features lanel as Y
Y = bostan['PRICE']

print(X.head())
print(Y.shape)
```

```
         0     1     2    3      4      5     6       7    8      9    10  \
0  0.00632  18.0  2.31  0.0  0.538  6.575  65.2  4.0900  1.0  296.0  15.3
1  0.02731   0.0  7.07  0.0  0.469  6.421  78.9  4.9671  2.0  242.0  17.8
2  0.02729   0.0  7.07  0.0  0.469  7.185  61.1  4.9671  2.0  242.0  17.8
3  0.03237   0.0  2.18  0.0  0.458  6.998  45.8  6.0622  3.0  222.0  18.7
4  0.06905   0.0  2.18  0.0  0.458  7.147  54.2  6.0622  3.0  222.0  18.7

       11    12
0  396.90  4.98
1  396.90  9.14
2  392.83  4.03
3  394.63  2.94
4  396.90  5.33
(506,)
```

0.2.1 Training and testing datasets splitting with cross_validation

[13]:
```python
from sklearn import preprocessing
min_max_scaler = preprocessing.MinMaxScaler()
X_df = pd.DataFrame(min_max_scaler.fit_transform(pd.DataFrame(X)))

Y_df=Y
```

[18]:
```python
# Training and testing datasets splitting with cross_validation
# Training and testing splitting data with 70% and 30%
# randomserach cross_validation is used
X_train, X_test, Y_train, Y_test = train_test_split(X_df,
                                                    Y_df,
                                                    test_size = 0.40,
                                                    random_state = 5)
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
print(type(X_train))
```

```
(303, 13)
(203, 13)
(303,)
(203,)
<class 'pandas.core.frame.DataFrame'>
```

0.2.2 linear Regression on Bostan House Dataset

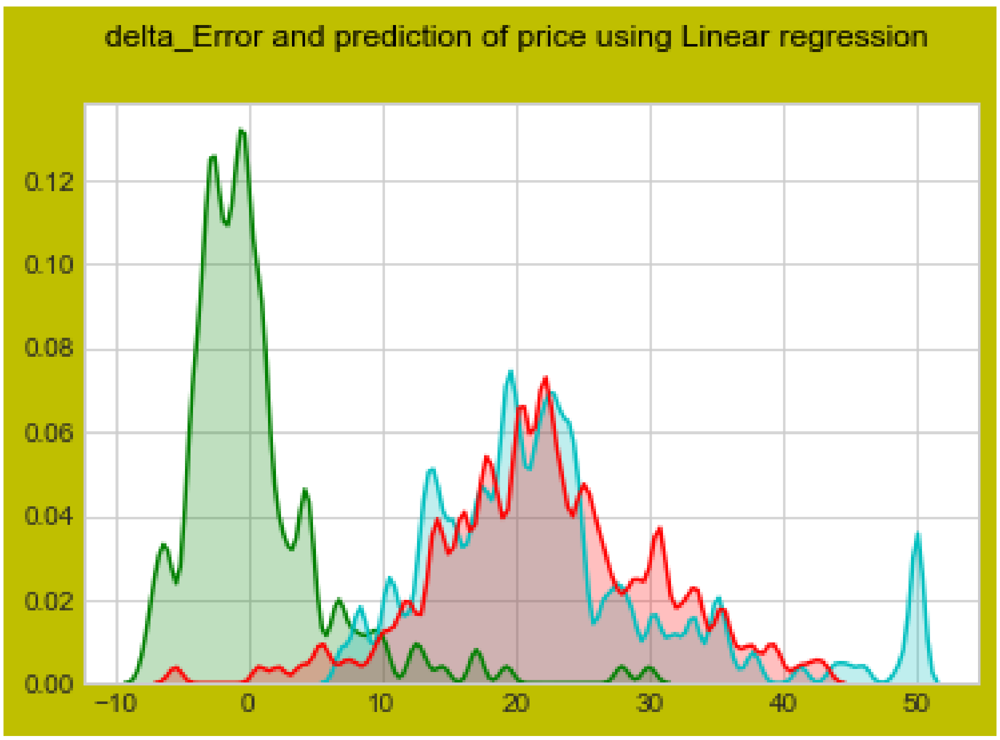
[19]:
```
# code source:https://medium.com/@haydar_ai/
↪learning-data-science-day-9-linear-regression-on-boston-housing-dataset-cd62a80775ef
from sklearn.linear_model import LinearRegression
lm = LinearRegression()
lm.fit(X_train, Y_train)
Y_pred = lm.predict(X_test)
error=abs(Y_test-Y_pred)
total_error = np.dot(error,error)
# Compute RMSE
rmse_lr= np.sqrt(total_error/len(error))
print('RMSE=',rmse_lr)
#plt.show()
plt.plot(Y_test, Y_pred,'g*')
plt.plot([0,50],[0,50], 'r-')
plt.title("Prices vs Predicted prices : $Y_i$ vs $\hat{Y}_i$")
plt.xlabel('Prices')
plt.ylabel('Predicted prices')
plt.show()
```

```
RMSE= 5.388131255020149
```

Delta_Error and Prediction of price using Linear regression

[12]:
```
delta_y = Y_test - Y_pred
import seaborn as sns
fig3 = plt.figure( facecolor='y', edgecolor='k')
fig3.suptitle('delta_Error and prediction of price using Linear regression',␣
 ↪fontsize=12)

sns.set_style('whitegrid')
sns.kdeplot(np.array(delta_y),shade=True, color="g", bw=0.5)
sns.kdeplot(np.array(Y_test),shade=True, color="c", bw=0.5)
sns.kdeplot(np.array(Y_pred),shade=True, color="r", bw=0.5)
```

[12]: <matplotlib.axes._subplots.AxesSubplot at 0x875f6d0>



- Red region is predicted price for bostan house datsets
- Blue Region is for y_test
- Green Region is difference between actual one and Predicted one.

1 sklearn.linear_model.SGDRegressor

alpha is as learning rate

n_iter is as batch size

[67]:
```python
models_performence1 = {
    'Model':[],
    'Batch_Size':[],
    'RMSE': [],
    'MSE':[],
    'Iteration':[],
    'Optimal learning Rate':[],


}
columns = ["Model","Batch_Size","RMSE","MSE", "Iteration", "Optimal learning␣
↪Rate"]
pd.DataFrame(models_performence1, columns=columns)
```

[67]:
```
Empty DataFrame
Columns: [Model, Batch_Size, RMSE, MSE, Iteration, Optimal learning Rate]
Index: []
```

[68]:
```python
 def square(list):
        return [(i ** 2) for i in list]
```

[69]:
```python
from sklearn import linear_model
import warnings
warnings.filterwarnings("ignore")
#Here, alpha is as learning rate

def sgdreg_function(x,initial_batch_size):
    #initial_batch_size=100
    batch=[]

    for l in range(x):
        batch_size_value= initial_batch_size + initial_batch_size * l
        batch.append(batch_size_value)
        z=0
        scale_max=np.max(Y_test[0:batch_size_value])

        Learning_rate=1 # initial learning rate=1
        score=[]
        LR=[] # storing value for learning rate
        Total_score=[]
        epoch1=[]
        global delta_error
        delta_error=[]
        Y_Test=[]
        global Y_hat_Predicted
        Y_hat_Predicted=[]
```

```
test_cost=[]
train_cost=[]
n_iter=100
for k in range(1,batch_size_value+1):
    # Appending learning rate
    LR.append(Learning_rate)

    # SGDRegressor
    sgdreg = linear_model.SGDRegressor(penalty='none',
                                       alpha=Learning_rate
                                       , n_iter=100)



    yii=Y_train[0:batch_size_value]
    xii=X_train[0:batch_size_value]
    xtt=X_test[0:batch_size_value]
    ytt=Y_test[0:batch_size_value]
    Y_Test.append(ytt)

    clf=sgdreg.fit(xii,yii)
    Traing_score=clf.score(xii,yii)
    train_cost.append(Traing_score)
    training_error=1-Traing_score

    # p predicting on x_test

    y_hat = sgdreg.predict(xtt)
    #testing_score=clf.score()
    clf1=sgdreg.fit(xtt,ytt)
    Testing_score=clf1.score(xtt,ytt)
    test_cost.append(Testing_score)
    Testing_error=1-Testing_score
    Y_hat_Predicted.append(y_hat)
    # error = Y_test - y_prediction
    err = abs(ytt - y_hat)
    delta_error.append(err)

    score.append(Testing_score)
    # print(rmse)

    # Iteration
    iteration_no=sgdreg.n_iter_
    epoch1.append(iteration_no)
    #print('Epoch=',iteration_no)
    #print('Learning_rate',Learning_rate)

    Learning_rate=Learning_rate/2
```

```
            z+=1
        print("Training Error=",training_error)
        print("Testing_error",Testing_error)

        models_performence1['Model'].append('sklearn.linear_model.SGDRegressor')
        # graph (Y_test) Prices Vs  (Y_prediction) Predicted prices
        fig4 = plt.figure( facecolor='c', edgecolor='k')
        fig4.suptitle('(Y_test) Prices Vs  (Y_prediction) Predicted prices:␣
↪$Y_i$ vs $\hat{Y}_i$ with batch size='+str(batch[l]), fontsize=12)
        plt.plot(Y_Test,Y_hat_Predicted,'g*')
        plt.plot([0,batch_size_value],[0,batch_size_value], 'r-')

        plt.xlabel('Y_test')
        plt.ylabel('Y_predicted')
        plt.show()


        # Plot delta_Error and prediction of price
        fig3 = plt.figure( facecolor='y', edgecolor='k')
        fig3.suptitle('delta_Error and prediction of price with batch␣
↪size='+str(batch[l]), fontsize=12)
        sns.set_style('darkgrid')
        Y_sklearn=np.array(sum(delta_error)/len(delta_error))
        sns.distplot(Y_sklearn,kde_kws={"color": "g", "lw": 3, "label":␣
↪"Delta_error_sklearn"} )
        sns.kdeplot(np.array(y_hat),shade=True, color="r", bw=0.5)
        plt.show()

        # Plot epoch Vs  RMSE
        fig = plt.figure( facecolor='y', edgecolor='k')
        fig.suptitle('epoch Vs  RMSE with batch size='+str(batch[l]),␣
↪fontsize=12)
        ax1 = fig.add_subplot(111)
        plt.plot(epoch1,score,'m*',linestyle='dashed')
        plt.grid()
        plt.xlabel('epoch')
        plt.ylabel('RMSE with batch size=')


        models_performence1['Iteration'].append(sum(epoch1)/len(epoch1))

        # plot Iterations Vs Train Cost & Test cost
        fig4 = plt.figure( facecolor='c', edgecolor='k')
        fig4.suptitle('Iterations Vs Train Cost & Test cost with batch␣
↪size='+str(batch[l]), fontsize=12)
        plt.plot(epoch1,train_cost,'m*',linestyle='dashed', label='Train cost')
        plt.plot(epoch1,test_cost,'r*', linestyle='dashed',label='Test cost')
```

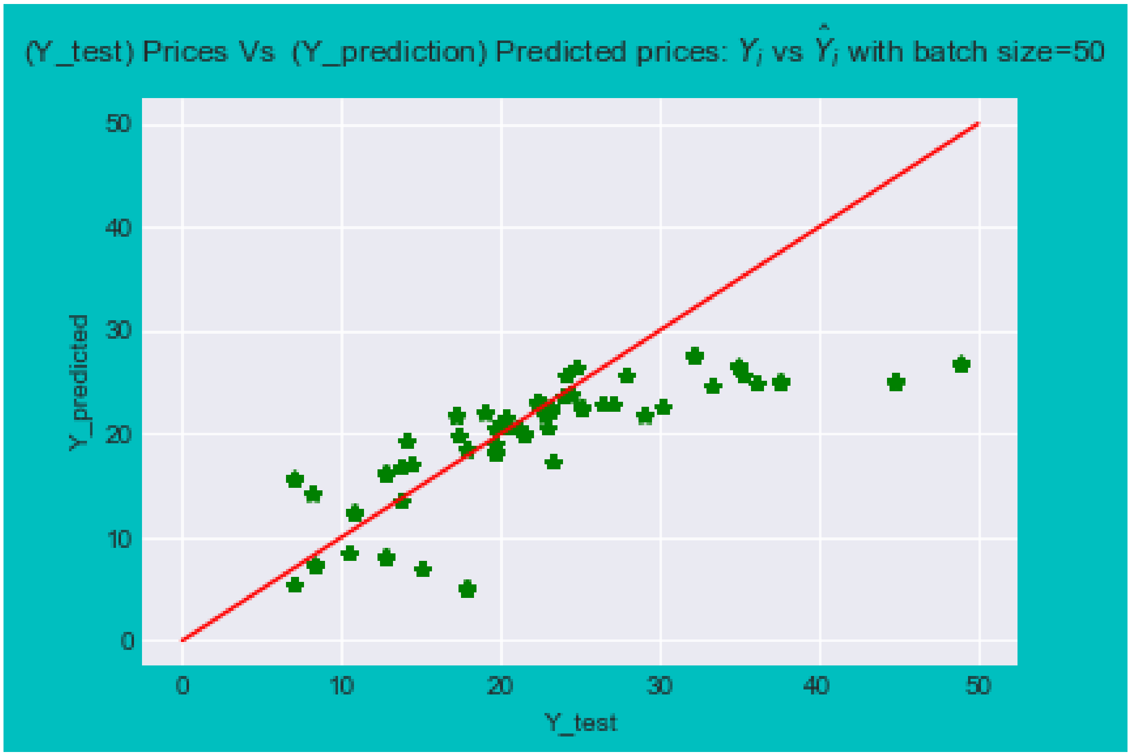
```
        plt.legend(loc='lower left')
        plt.grid()
        plt.xlabel('Iterations ')
        plt.ylabel('Performance Cost  ')
        plt.show()

        # Plot Learning rate Vs  RMSE
        fig2 = plt.figure( facecolor='y', edgecolor='k')
        fig2.suptitle('Learning rate Vs  RMSE with batch size='+str(batch[l]),␣
↪fontsize=12)
        ax2 = fig2.add_subplot(111)
        #ax2.set_title("Learning rate Vs  RMSE")
        plt.plot(LR,score,'m*',linestyle='dashed')
        plt.grid()
        plt.xlabel('Learning rate')
        plt.ylabel('RMSE')
        plt.show()


        global best_Learning_rate
        best_Learning_rate=LR[score.index(min(score))]
        models_performence1['Optimal learning Rate'].append(best_Learning_rate)
        print('\nThe best value of best_Learning_rate is %d.' %␣
↪(best_Learning_rate),7)
        MSEscore=scale_max*sum(score)/len(score)
        score_value=np.sqrt(MSEscore)
        print('Batch Size',batch[l])

        models_performence1['Batch_Size'].append(batch[l])
        print("RMSE with batch size="+str(batch[l]),score_value)
        models_performence1['RMSE'].append(score_value)
        print("MSE with batch size="+str(batch[l]),MSEscore)
        models_performence1['MSE'].append(MSEscore)
```

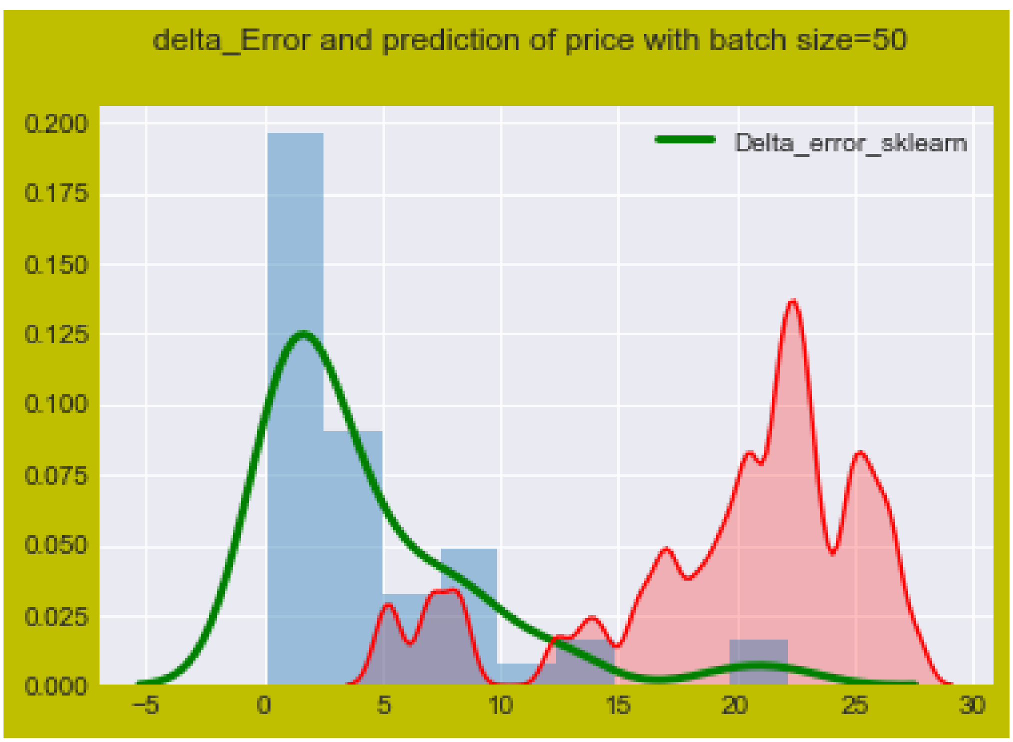
- sgdreg_function is function for stochastic gradient descen for linear regression using linear_model.SGDRegressor in sklearn.
- In this function different batch size (50,100,150,200) is applied on linear_model.SGDRegressor to get best learning rate,epoch value,error rate.
- here,delta_Error and prediction of price with batch size graph is shown.
- RMSE vs epoch graph is shown
- Also,RMSE vs learning rate graph is shown for different batch value.

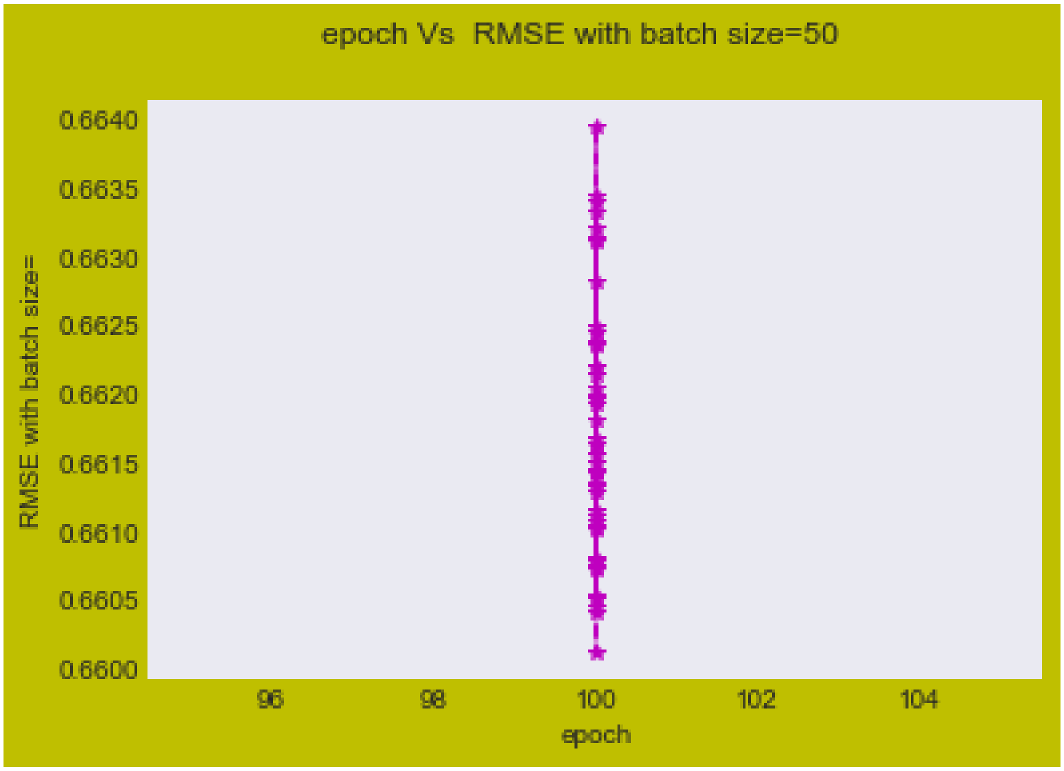
linear_model.SGDRegressor in sklearn for different batch size

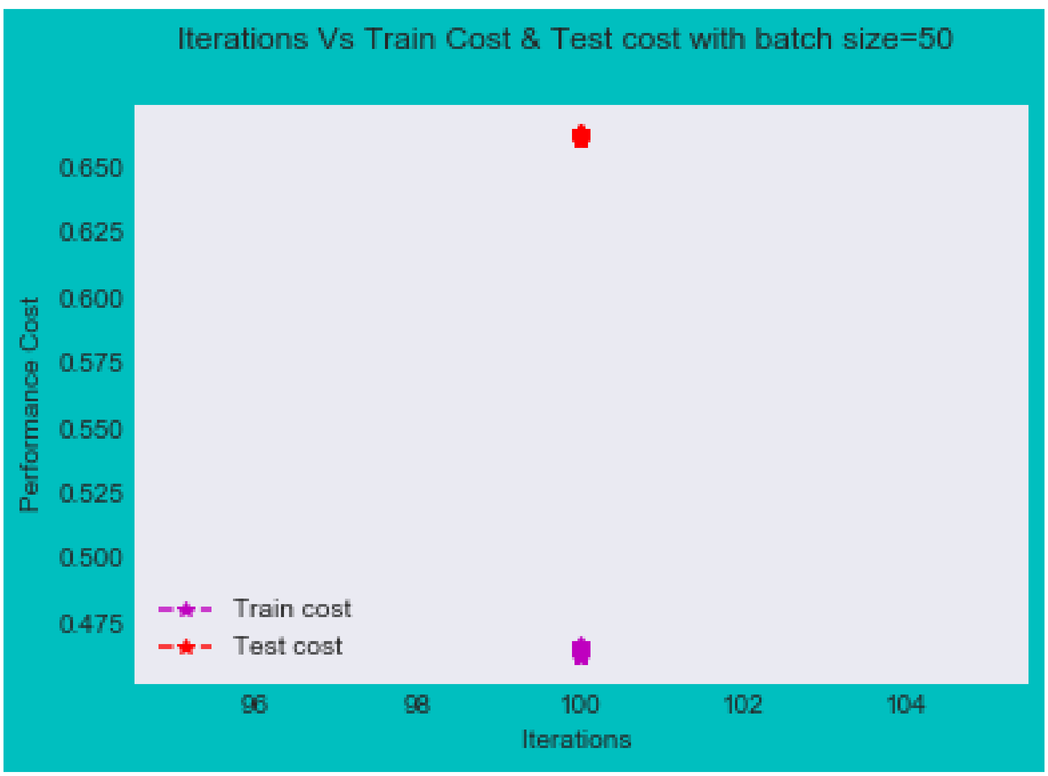
[70]:
```
sgdreg_function(4,50)
```

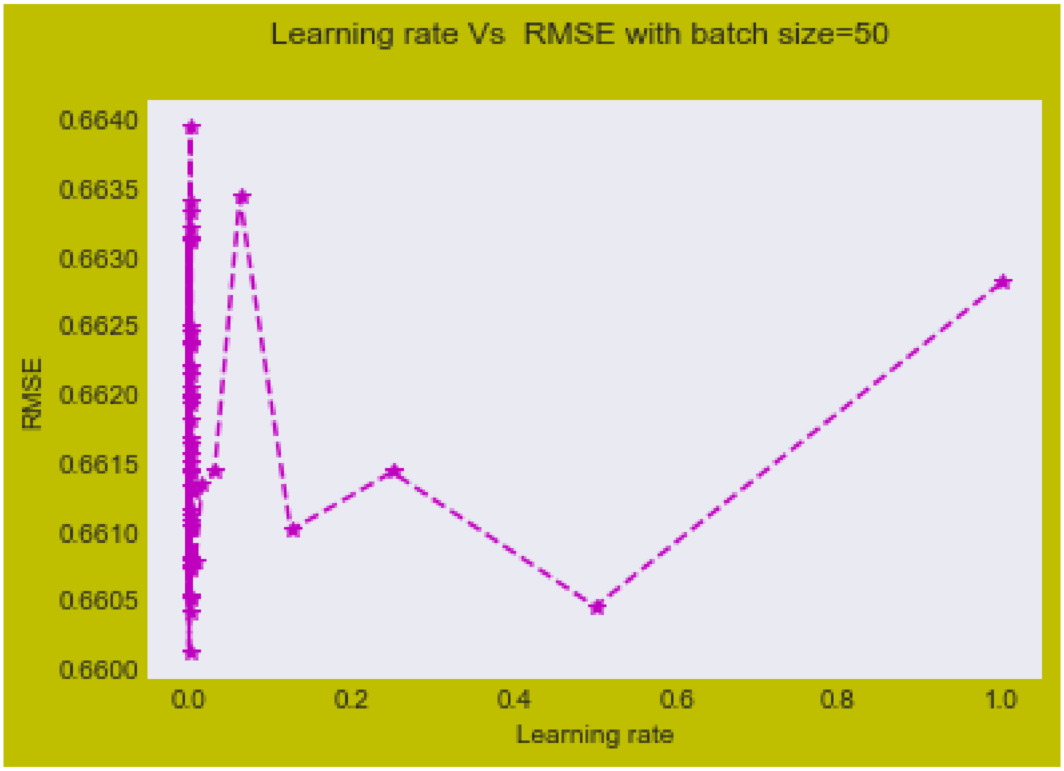
```
Training Error= 0.537075361718
Testing_error 0.337953285285
```



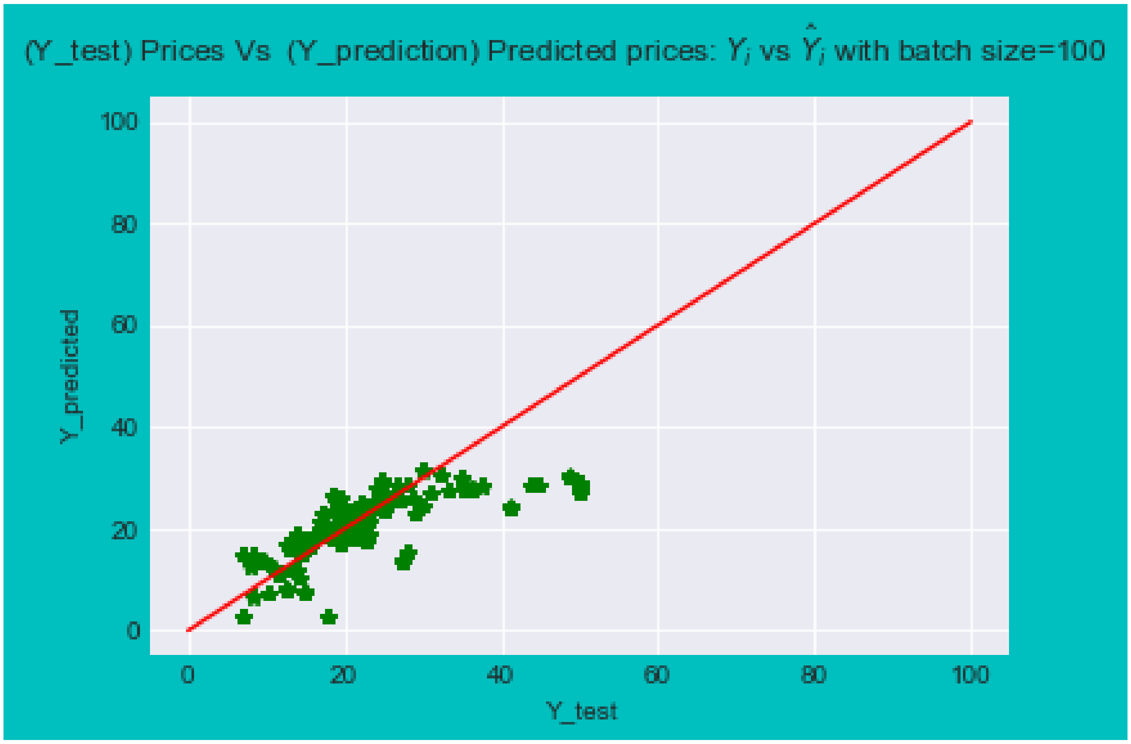


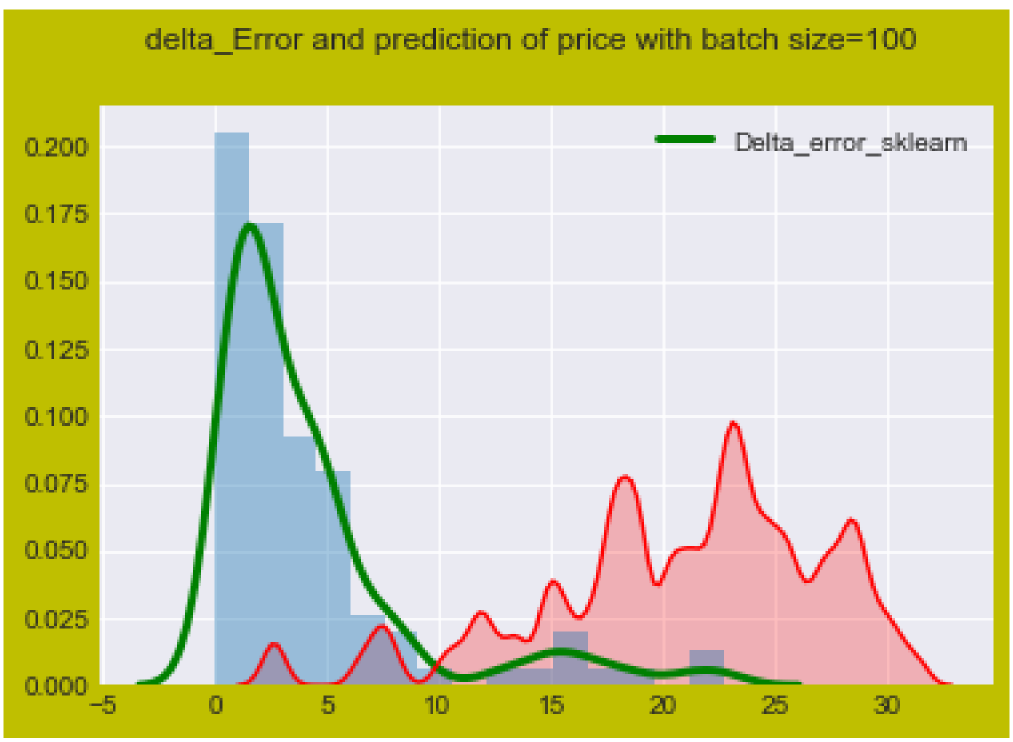

epoch Vs RMSE with batch size=50
RMSE with batch size=
0.6640
0.6635
0.6630
0.6625
0.6620
0.6615
0.6610
0.6605
0.6600
96
98
100
102
104
epoch

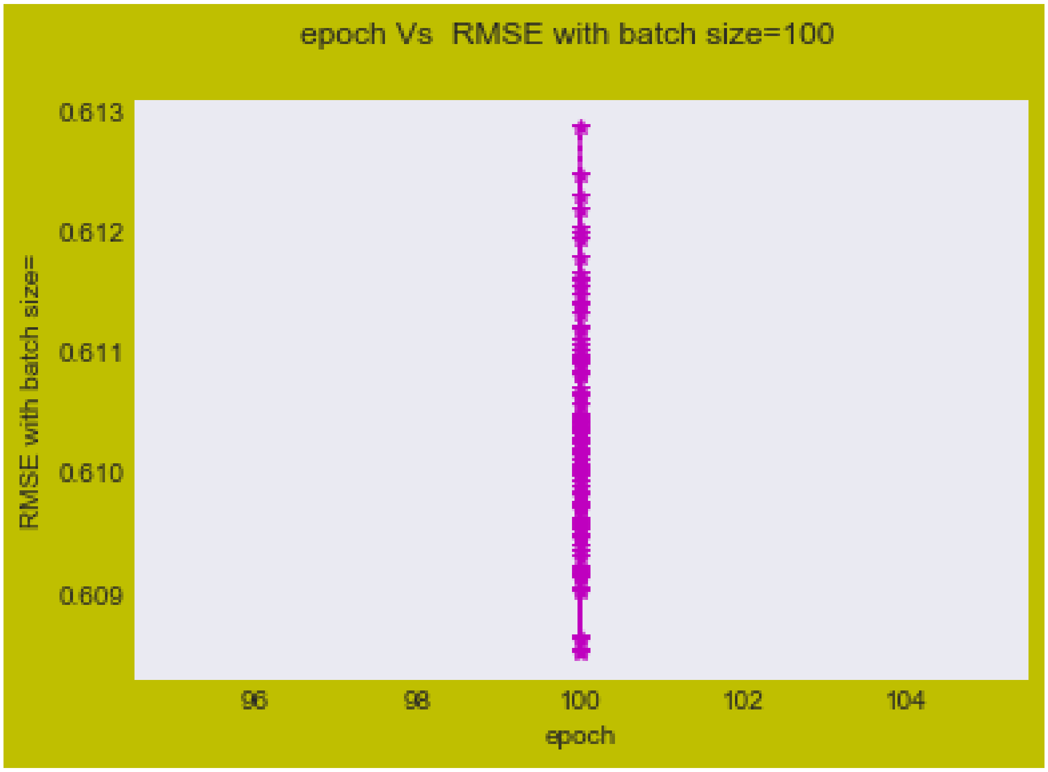

Iterations Vs Train Cost & Test cost with batch size=50
Performance Cost
0.650
0.625
0.600
0.575
0.550
0.525
0.500
0.475
Train cost
Test cost
96
98
100
102
104
Iterations

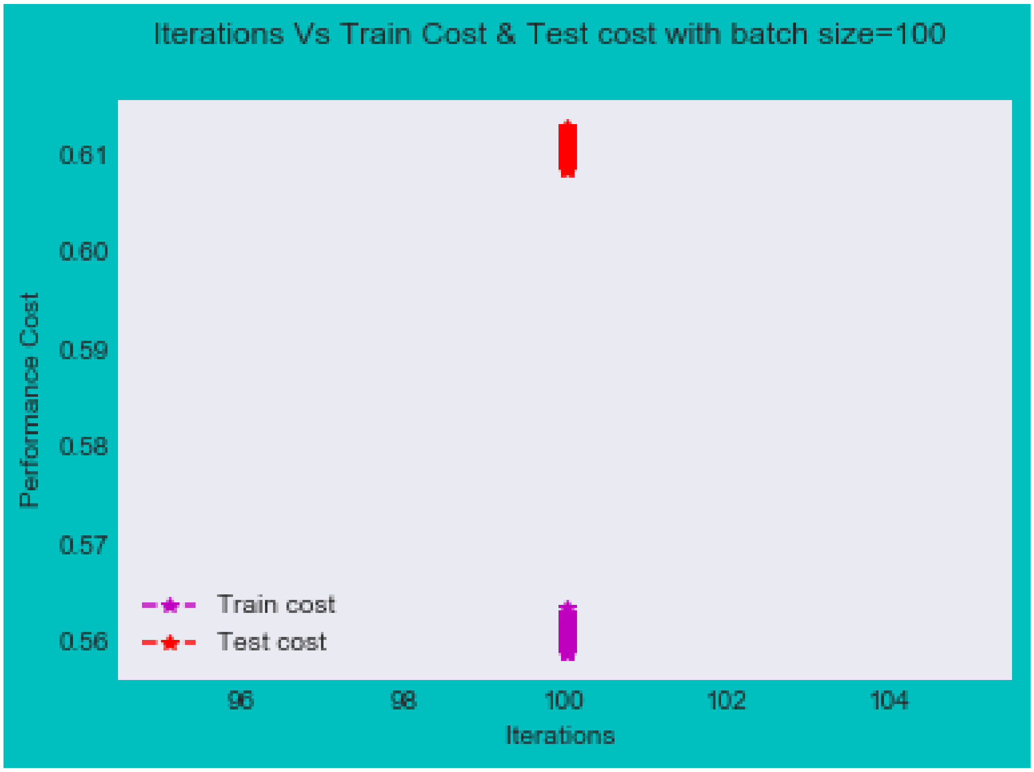


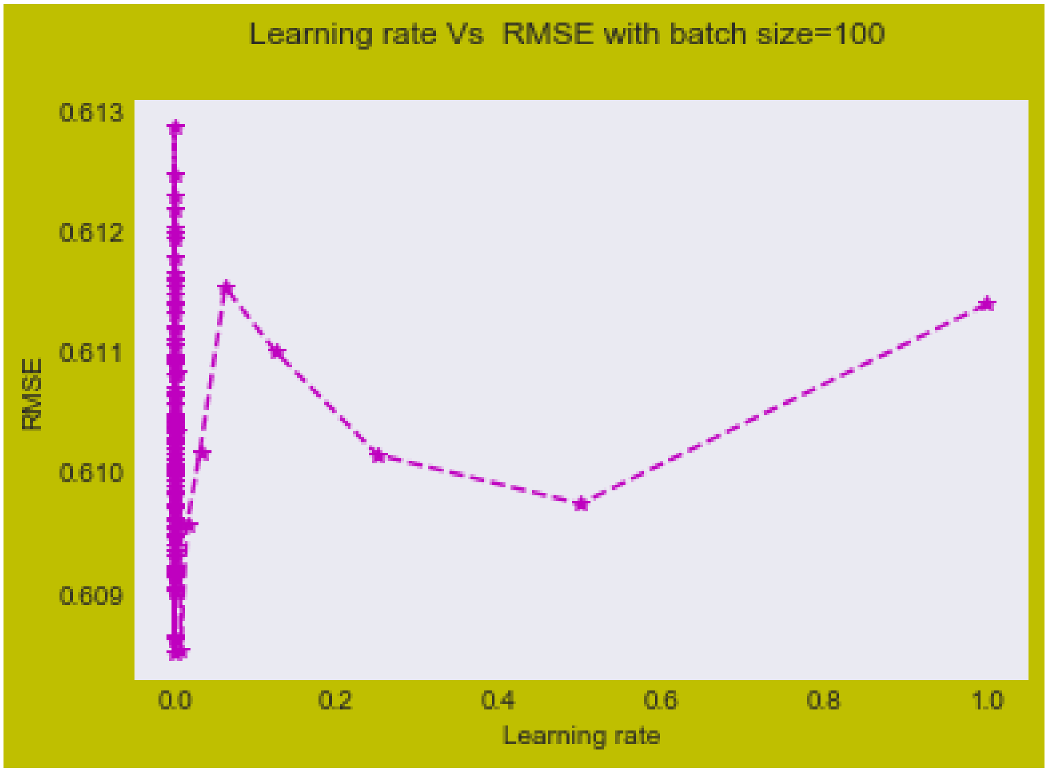

```
The best value of best_Learning_rate is 0. 7
Batch Size 50
RMSE with batch size=50 5.68273968837
MSE with batch size=50 32.2935303658
Training Error= 0.439011285215
Testing_error 0.3905243053
```


(Y_test) Prices Vs (Y_prediction) Predicted prices: Y_i vs $\hat{Y}_i$ with batch size=100
Y_predicted
Y_test
0
20
40
60
80
100

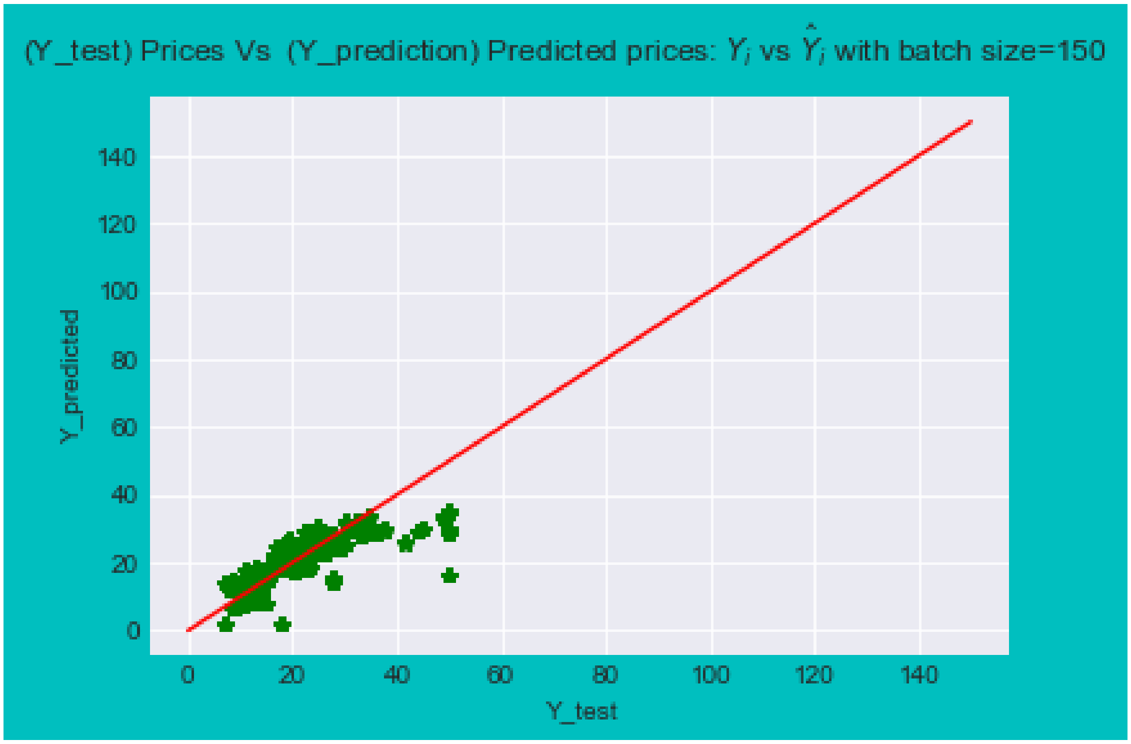

delta_Error and prediction of price with batch size=100
Delta_error_sklearn
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
−5
0
5
10
15
20
25
30

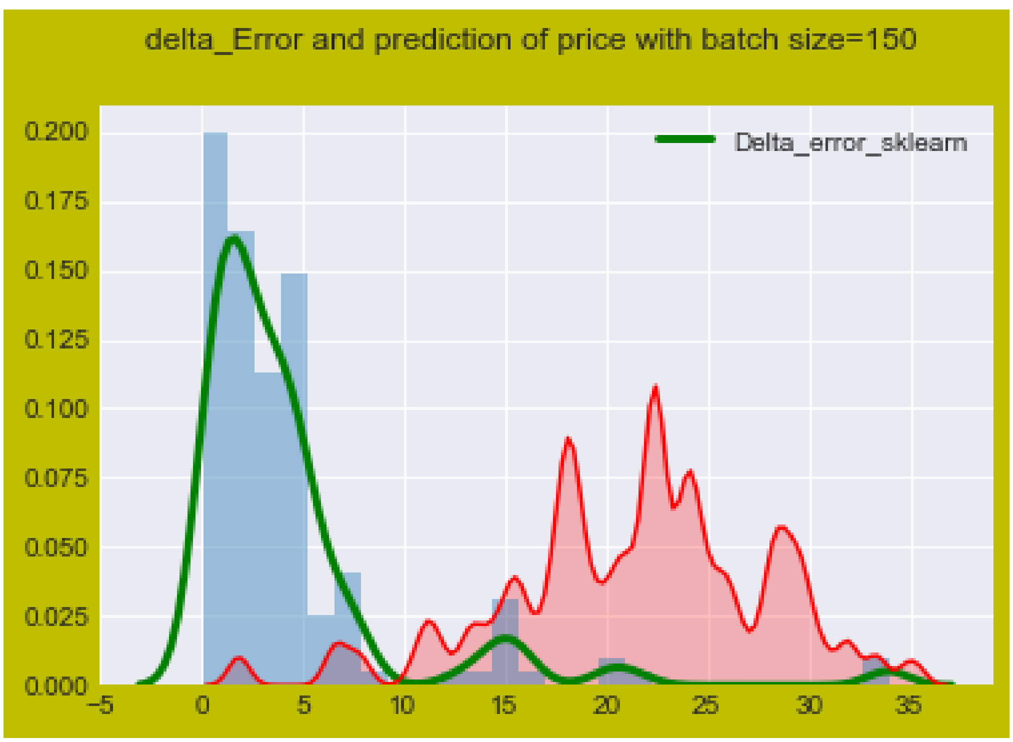

epoch Vs RMSE with batch size=100
RMSE with batch size=
0.613
0.612
0.611
0.610
0.609
96
98
100
102
104
epoch

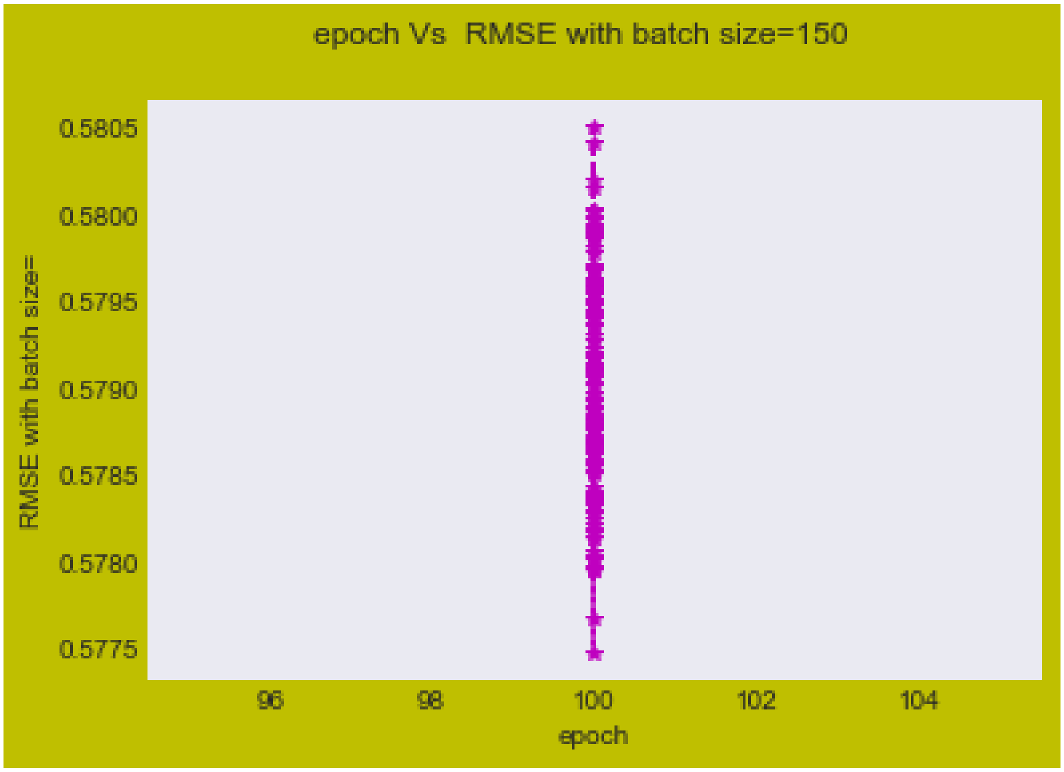

Iterations Vs Train Cost & Test cost with batch size=100
Performance Cost
0.61
0.60
0.59
0.58
0.57
0.56
Train cost
Test cost
96
98
100
102
104
Iterations

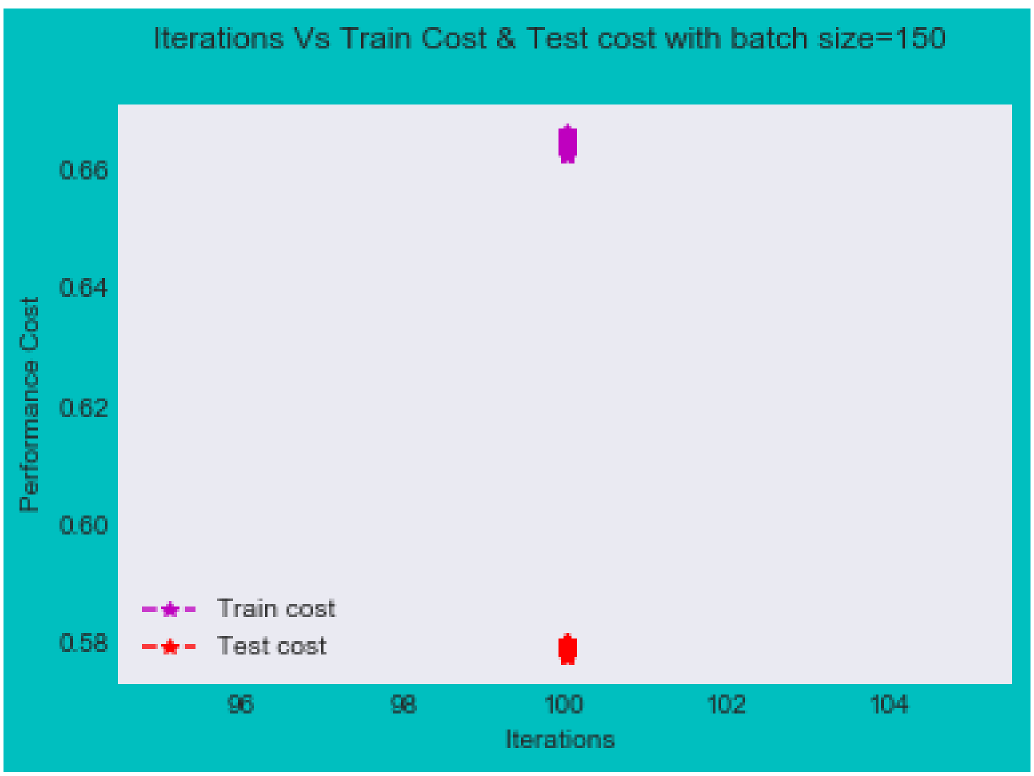


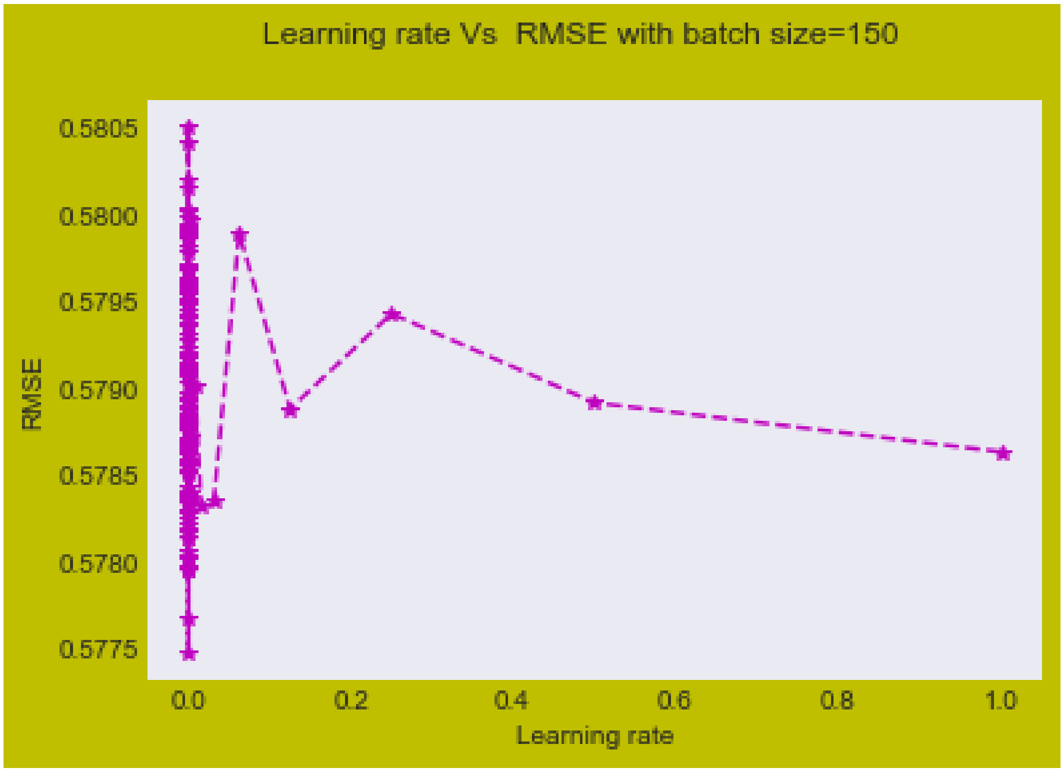

```
The best value of best_Learning_rate is 0. 7
Batch Size 100
RMSE with batch size=100 5.52418752057
MSE with batch size=100 30.5166477624
Training Error= 0.336803653429
Testing_error 0.420504857604
```


(Y_test) Prices Vs (Y_prediction) Predicted prices: Y_i vs $\hat{Y}_i$ with batch size=150
Y_predicted
Y_test
0
20
40
60
80
100
120
140

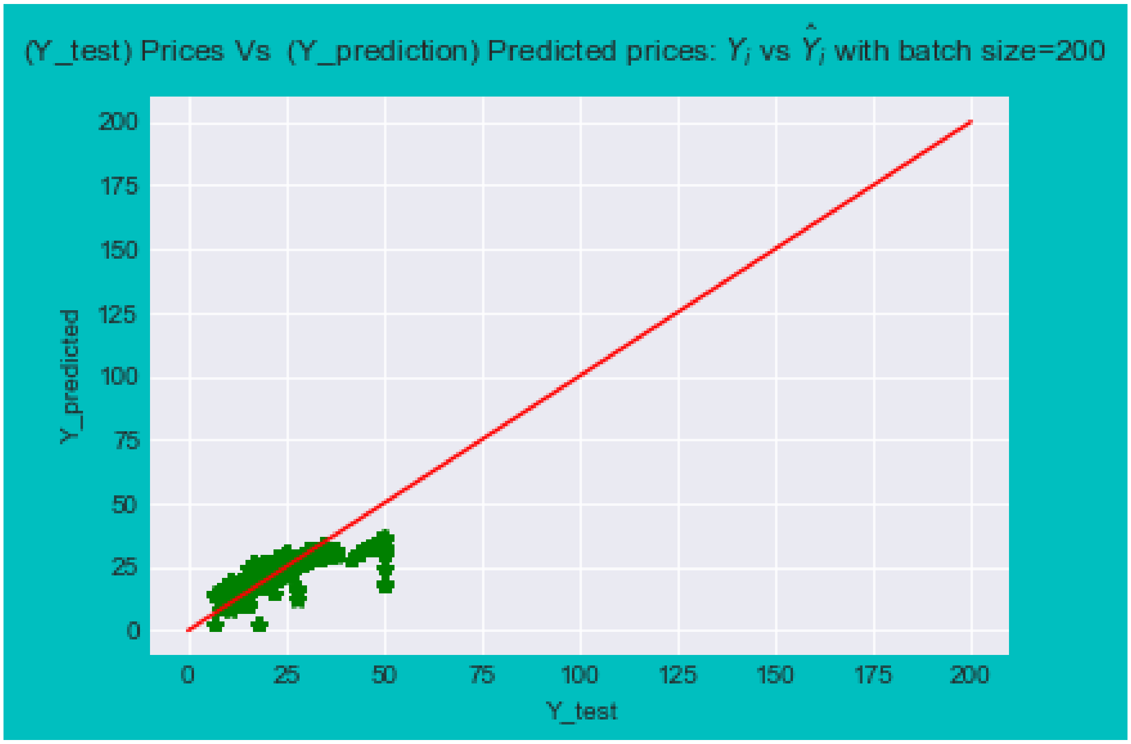

delta_Error and prediction of price with batch size=150
Delta_error_sklearn
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
−5
0
5
10
15
20
25
30
35

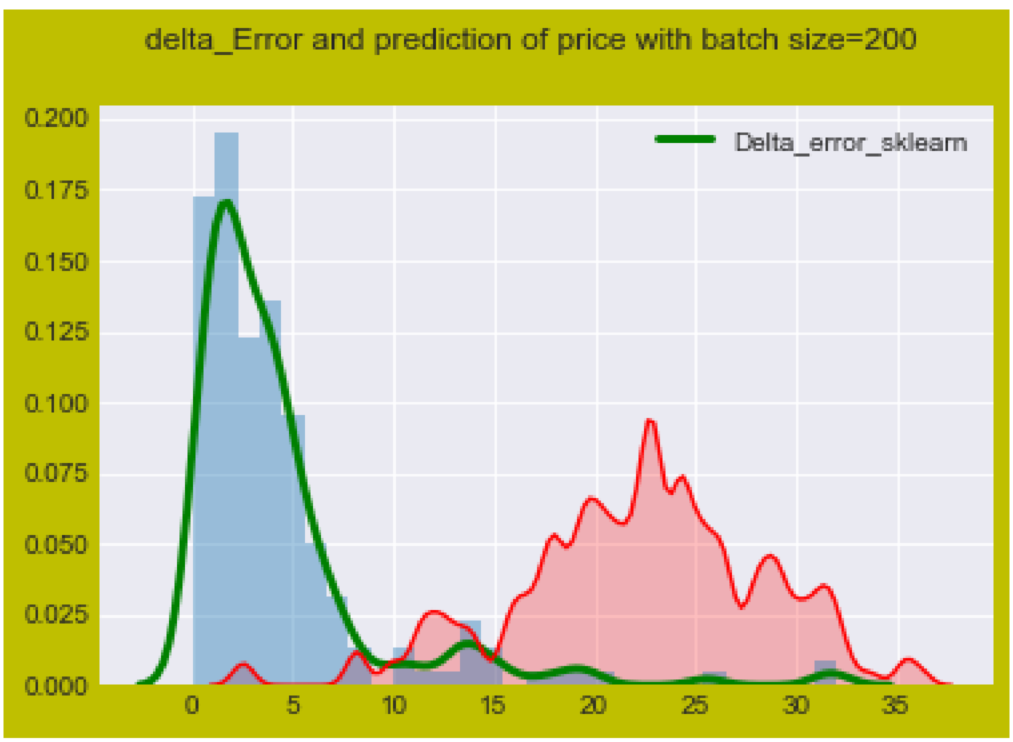

epoch Vs RMSE with batch size=150
RMSE with batch size=
0.5805
0.5800
0.5795
0.5790
0.5785
0.5780
0.5775
96
98
100
102
104
epoch

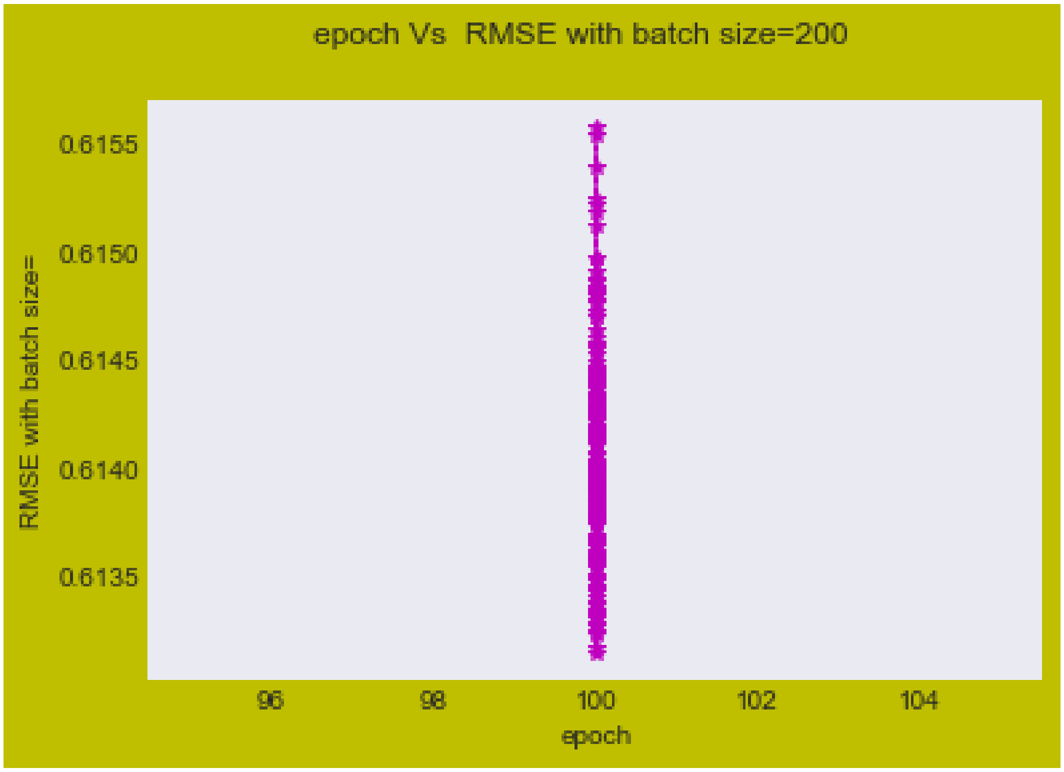

Iterations Vs Train Cost & Test cost with batch size=150
Performance Cost
0.66
0.64
0.62
0.60
0.58
Train cost
Test cost
96
98
100
102
104
Iterations

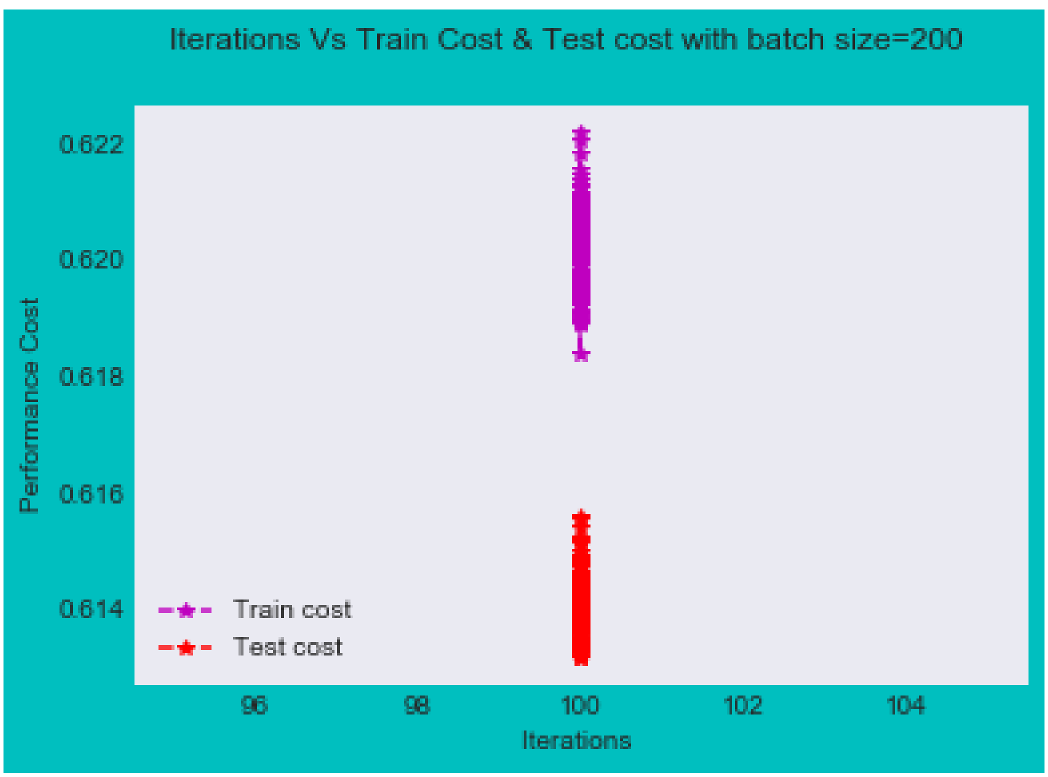


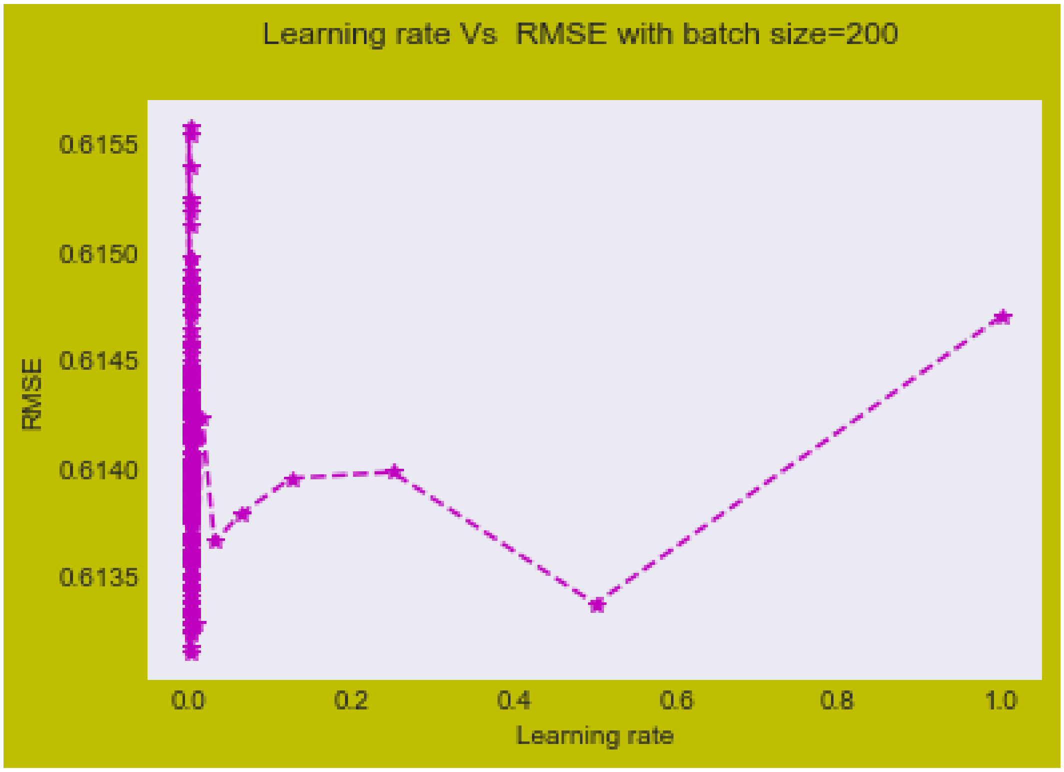

```
The best value of best_Learning_rate is 0. 7
Batch Size 150
RMSE with batch size=150 5.38058107219
MSE with batch size=150 28.9506526744
Training Error= 0.378789692009
Testing_error 0.385421539925
```


(Y_test) Prices Vs (Y_prediction) Predicted prices: Y_i vs $\hat{Y}_i$ with batch size=200
Y_predicted
Y_test
0
25
50
75
100
125
150
175
200


delta_Error and prediction of price with batch size=200
Delta_error_sklearn
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200
0
5
10
15
20
25
30
35


epoch Vs RMSE with batch size=200
RMSE with batch size=
0.6155
0.6150
0.6145
0.6140
0.6135
96
98
100
102
104
epoch


Iterations Vs Train Cost & Test cost with batch size=200
Performance Cost
0.622
0.620
0.618
0.616
0.614
Train cost
Test cost
96
98
100
102
104
Iterations




```
The best value of best_Learning_rate is 0. 7
Batch Size 200
RMSE with batch size=200 5.54121163907
MSE with batch size=200 30.705026429
```

[73]:
```python
columns = ["Model","Batch_Size","RMSE","MSE", "Iteration", "Optimal learning␣
 ↪Rate"]
pd.DataFrame(models_performence1, columns=columns)
```

[73]:
```
                              Model  Batch_Size      RMSE        MSE  \
0  sklearn.linear_model.SGDRegressor          50  5.682740  32.293530
1  sklearn.linear_model.SGDRegressor         100  5.524188  30.516648
2  sklearn.linear_model.SGDRegressor         150  5.380581  28.950653
3  sklearn.linear_model.SGDRegressor         200  5.541212  30.705026

   Iteration  Optimal learning Rate
0      100.0           1.525879e-05
1      100.0           4.882812e-04
2      100.0           7.450581e-09
3      100.0           6.462349e-27
```

Observation:

- In sklearn SGDRegressor,It is observed that as batch size increases optimal learning rate decreses.
- RMSE value is around 5 and MSE value is around 30
- RMSE value for batch size 100 is high comparatively with others batch size.
- For Batch size=200, RMSE & learning Rate is lowest.

1.1 Standardization training and testing data accourding to batch size

2 Manual SGD function

L(w,b)=min w,b{sum(square{yi-wTxi-b})}

Derivative of Lw w.r.t w ==>

Lw= sum({-2*xi}{yi-wT.xi-b})

Derivative of Lb w.r.t b==>

lb=sum(-2*{yi-wTxi-b})

[30]:
```
models_performence1 = {
    'Model':[],
    'Batch_Size':[],
    'RMSE': [],
    'MSE':[],
    'Iteration':[],
    'Optimal learning Rate':[],


}
columns = ["Model","Batch_Size","RMSE","MSE", "Iteration", "Optimal learning␣
 ↪Rate"]
pd.DataFrame(models_performence1, columns=columns)
```

[30]:
```
Empty DataFrame
Columns: [Model, Batch_Size, RMSE, MSE, Iteration, Optimal learning Rate]
Index: []
```

[31]:
```
def denorm(scale,list):
    return [(scale*i) for i in list]

# scale
scale=np.max(Y_test)
```

```python
print(scale)
```

50.0

[32]:
```python
# SGD function
#L(w,b)=min w,b{sum(square{yi-wTxi-b})}
def SGD(batch_size):
    X_batch_size =X_train[:batch_size]
    price_batch_size =Y_train[:batch_size]
    X_test_batch=X_test[:batch_size]
    ytt_batch_size= Y_test[:batch_size]




    N = len(X_batch_size)

    xi_1=[]
    yprice=[]
    xtt=[]
    ytt=[]
    ytt1=[]
    for j in range(N):
        # standardization of datasets
        scaler = StandardScaler()
        scaler.fit(X_batch_size)
        X_scaled_batch_size = scaler.transform(X_batch_size)
        X_scaled_batch_size=preprocessing.normalize(X_scaled_batch_size)
        xi_1.append(X_scaled_batch_size)

        X_test_batch_size=scaler.transform(X_test_batch)
        X_test_batch_size=preprocessing.normalize(X_test_batch_size)
        xtt.append(X_test_batch_size)
        Y_scaled_batch_size=np.asmatrix(price_batch_size)
        #Y_scaled_batch_size=preprocessing.normalize(Y_scaled_batch_size)
        yprice.append(Y_scaled_batch_size)
        Ytt_scaled_batch_size1=np.asmatrix(Y_test[:batch_size])
        Ytt_scaled_batch_size=preprocessing.normalize(Ytt_scaled_batch_size1)
        ytt1.append(Ytt_scaled_batch_size1)
        ytt.append(Ytt_scaled_batch_size)

    xi=xi_1
    price=yprice

    Lw = 0
    Lb = 0
```

```
learning_rate = 1
iteration = 1
w0_random = np.random.rand(13)
w0 = np.asmatrix(w0_random).T
b = np.random.rand()
b0 = np.random.rand()
global learning_rate1
learning_rate1=[]
global epoch
epoch=[]
global rmse1
rmse1=[]
global y_hat_manual_SGD
y_hat_manual_SGD=[]
global delta_Error
delta_Error=[]


while True:
    learning_rate1.append(learning_rate)
    epoch.append(iteration)

    for i in range(N):
        wj=w0
        bj=b0
        #derivative of Lw w.r.t w
        #Lw= sum({-2*xi}{yi-wT.xi-b})
        #print(price[i] .shape)
        Lw = (1/N)*np.dot((-2*xi[i].T ), (price[i] - np.dot( xi[i],wj) -
↪bj))
        #derivative of Lb w.r.t b
        #lb=sum(-2*{yi-wTxi-b})
        Lb = (-2/N)*(price[i] - np.dot( xi[i],wj ) - bj)
        #print('yi',Lw.shape)
        y_new=(1/N)*(xtt[i].dot(Lw))+Lb
        #print(y_new[i])
        y_pred=np.absolute(np.array(y_new[i]))
        y_hat_manual_SGD.append( y_pred)

        delta_error = np.absolute(np.array(ytt[i] ) - np.array(y_new[i]))
        delta_Error.append(delta_error.mean())
        #delta_error=price[i]  - y_new[i]

        error=np.sum(np.dot(delta_error ,delta_error.T))

    rmse1.append(error)
```

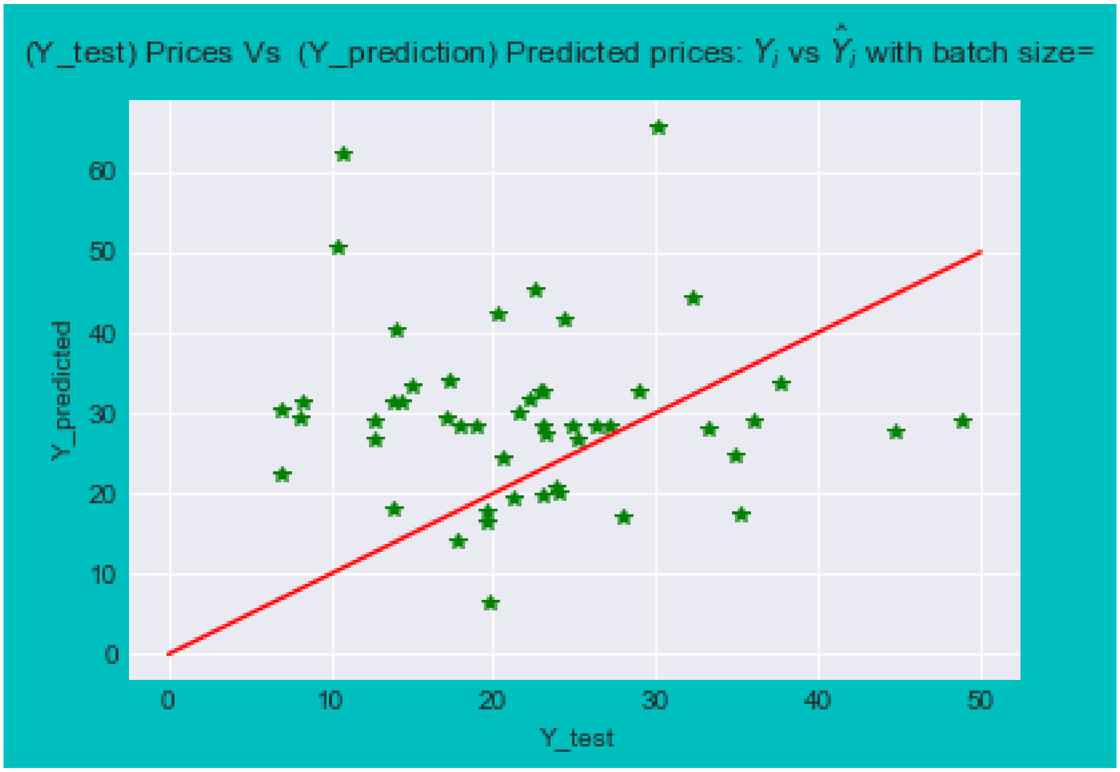
```
        w0_new = Lw * learning_rate
        b0_new = Lb * learning_rate
        wj = w0 - w0_new
        bj = b0 - b0_new
        iteration += 1
        if (w0==wj).all():
            break
        else:
            w0 = wj
            b0 = bj
            learning_rate = learning_rate/2

    print('For batch size'+str(batch_size))


    RMSE=(scale*np.asarray(rmse1))

    # Y_test function
    vvv=denorm(1,ytt1)
    cv=vvv[0]
    # Y_hat_test function after normationzation
    cvv=denorm(scale,y_hat_manual_SGD[batch_size])
    #print(sum(delta_error)/len(delta_error))
    fig4 = plt.figure( facecolor='c', edgecolor='k')
    fig4.suptitle('(Y_test) Prices Vs  (Y_prediction) Predicted prices: $Y_i$␣
↪vs $\hat{Y}_i$ with batch size=', fontsize=12)
    plt.plot(cv,cvv,'g*')
    plt.plot([0,batch_size],[0,batch_size], 'r-')

    plt.xlabel('Y_test')
    plt.ylabel('Y_predicted')
    plt.show()

     # Plot delta_Error and prediction of price
    fig3 = plt.figure( facecolor='y', edgecolor='k')
    fig3.suptitle('delta_Error  with batch size='+str(batch_size), fontsize=12)
    sns.set_style('darkgrid')
    sns.distplot(np.array(delta_Error),kde_kws={"color": "r", "lw": 3, "label":␣
↪"Delta_error_manual"} )
    #sns.kdeplot(np.array(ghy),shade=True, color="r", bw=0.5)
    plt.show()

    #For plotting epoch vs RMSE
    models_performence1['Model'].append('SGD Manual Function')
    models_performence1['Batch_Size'].append(batch_size)
    fig = plt.figure( facecolor='c', edgecolor='k')
    fig.suptitle('epoch Vs RMSE with batch size='+str(batch_size), fontsize=12)
```

```
    ax1 = fig.add_subplot(111)
    plt.plot(epoch,RMSE,'r*',linestyle='dashed')
    plt.xlabel('epoch')
    plt.ylabel('RMSE with batch size='+str(batch_size))
    plt.plot(epoch,RMSE,'y',linestyle='dashed')
    plt.show()

    #Best learning rate
    global best_Learning_rate1
    best_Learning_rate1=learning_rate1[rmse1.index(min(rmse1))]
    print('\nThe best value of best_Learning_rate is %d.' %␣
↪(best_Learning_rate1))
    models_performence1['Optimal learning Rate'].append(best_Learning_rate1)
    fig1 = plt.figure( facecolor='y', edgecolor='k')
    fig1.suptitle('Learning rate Vs RMSE with batch size='+str(batch_size),␣
↪fontsize=12)
    ax1 = fig1.add_subplot(111)
    plt.plot(learning_rate1,rmse1,'m*')
    plt.xlabel('Learning rate')
    plt.ylabel('RMSE')


    global RMSE_value
    MSE_value = sum(rmse1)/len(rmse1)
    print("MSE_value=",MSE_value  )
    models_performence1['MSE'].append(MSE_value)
    RMSE_value =np.sqrt(MSE_value)
    models_performence1['RMSE'].append(RMSE_value)


    models_performence1['Iteration'].append(iteration)

    print("RMSE = ",RMSE_value)
    print('For batch size'+str(batch_size))


    print('iteration =',iteration)

    print('Total number of learning_rate=',len(learning_rate1))
    plt.plot(learning_rate1,rmse1,'y',linestyle='dashed')
    plt.show()
```

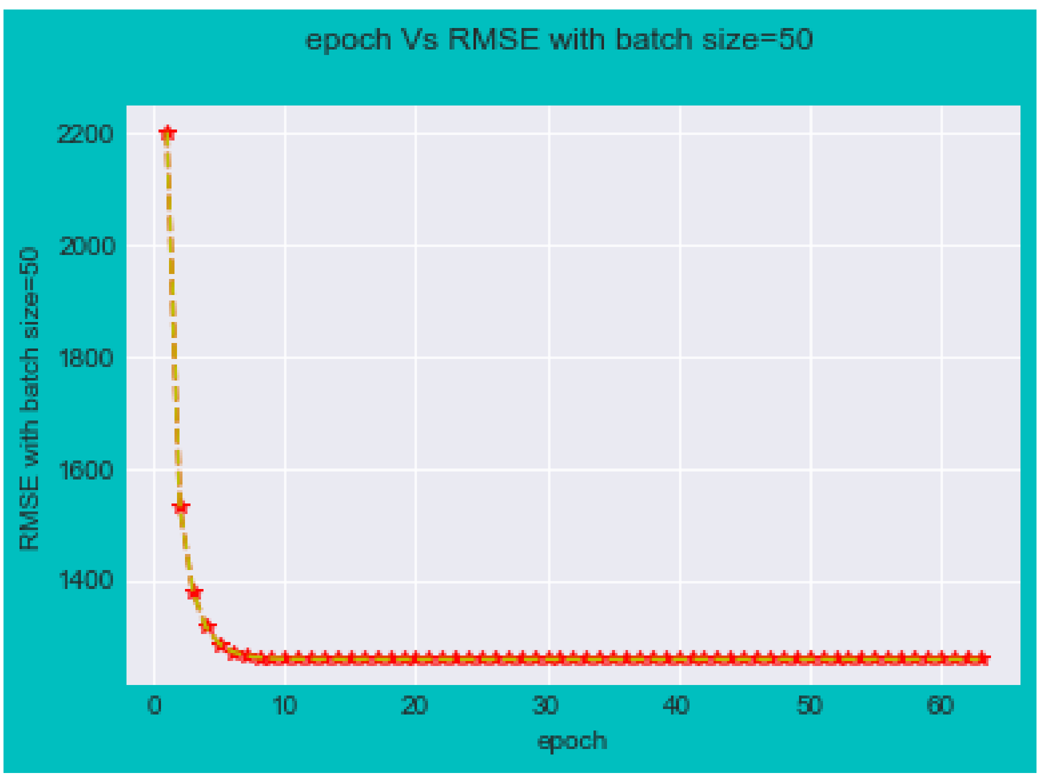
[33]:
```
initial_batch_size=50

for l in range(4):
    batch_size_value= initial_batch_size + initial_batch_size * l
```

```
print(batch_size_value)
SGD(batch_size_value)
```

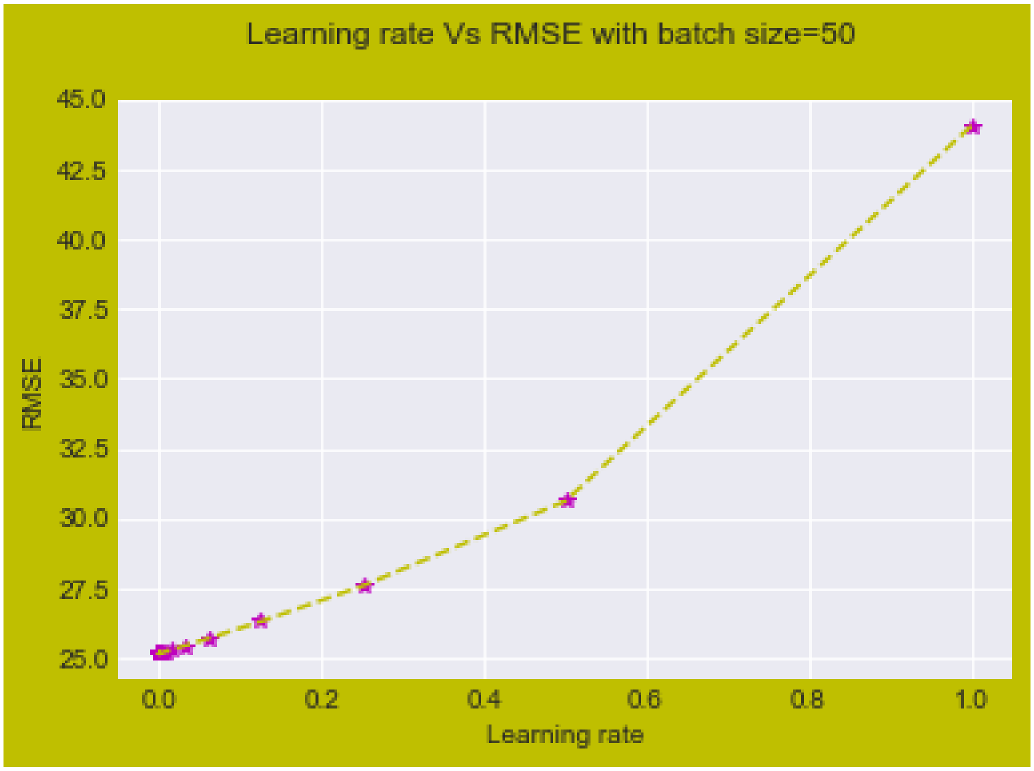
```
50
For batch size50
```



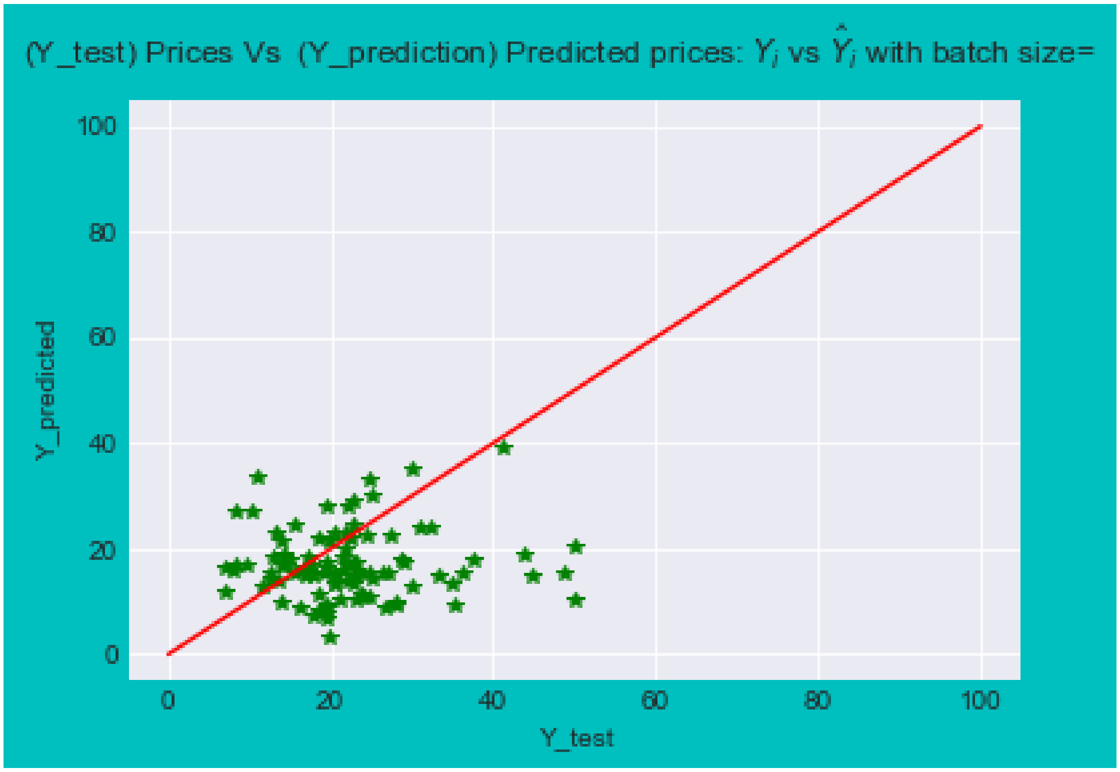

delta_Error with batch size=50
Delta_error_manual
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.5
0.6
0.7
0.8
0.9
1.0
1.1
1.2
1.3

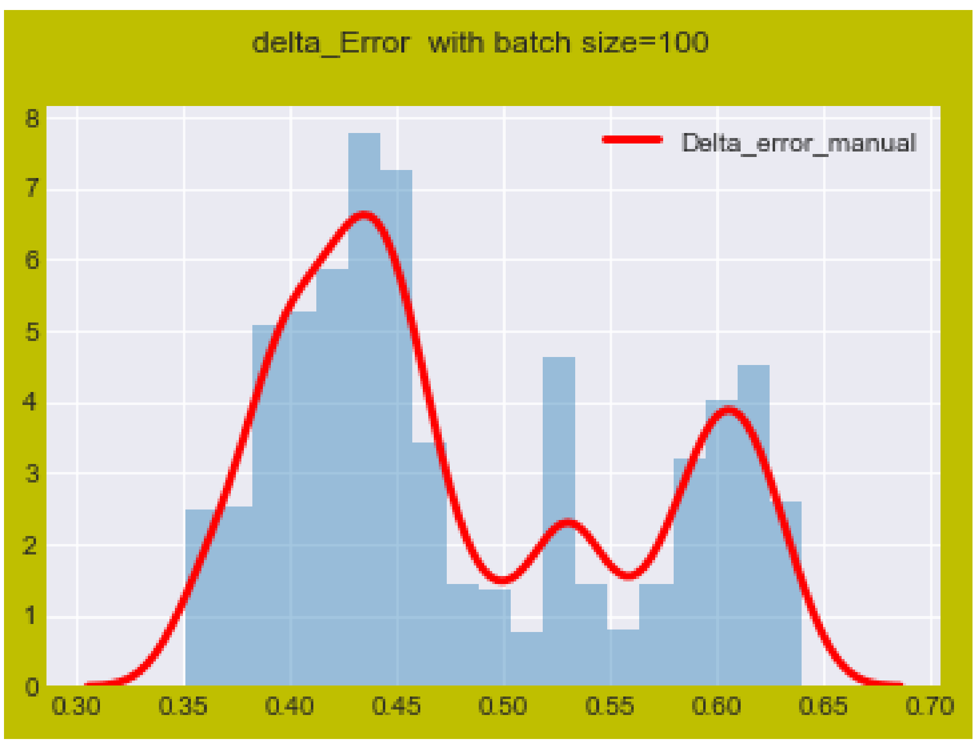

epoch Vs RMSE with batch size=50
RMSE with batch size=50
2200
2000
1800
1600
1400
0
10
20
30
40
50
60
epoch

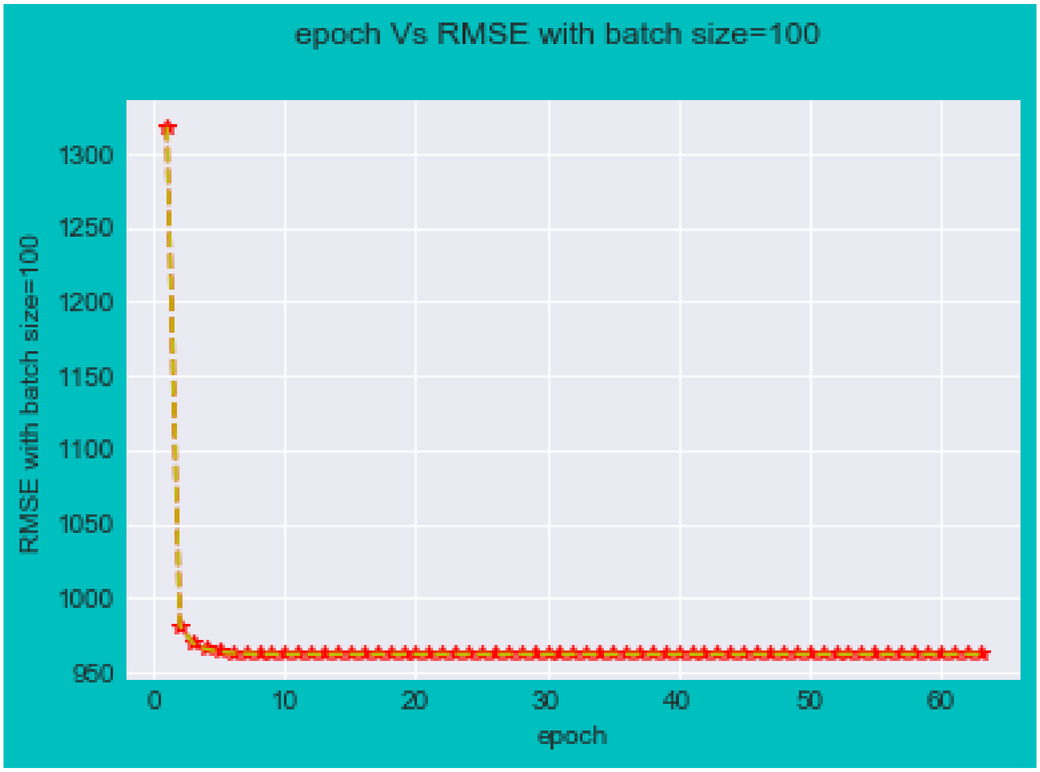
```
The best value of best_Learning_rate is 0.
MSE_value= 25.6298267037
RMSE =  5.06259090819
For batch size50
iteration = 64
Total number of learning_rate= 63
```



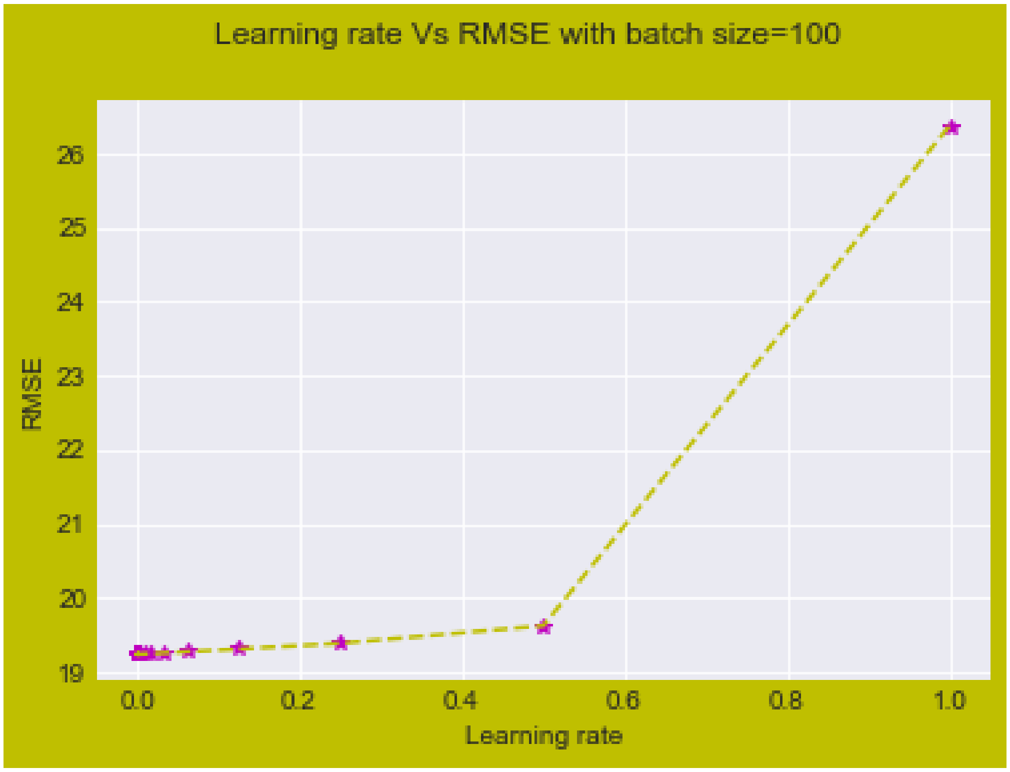

```
100
For batch size100
```


(Y_test) Prices Vs (Y_prediction) Predicted prices: Y_i vs $\hat{Y}_i$ with batch size=
Y_predicted
Y_test
0
20
40
60
80
100

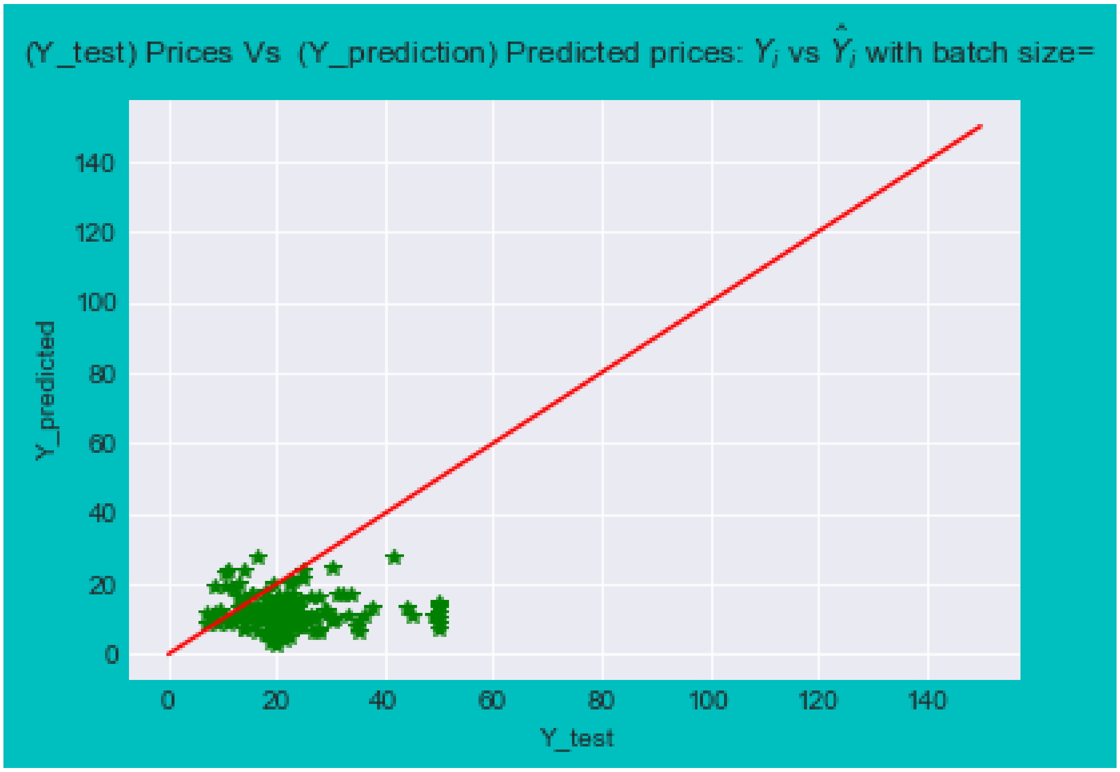

delta_Error with batch size=100
Delta_error_manual
0.30
0.35
0.40
0.45
0.50
0.55
0.60
0.65
0.70
0
1
2
3
4
5
6
7
8

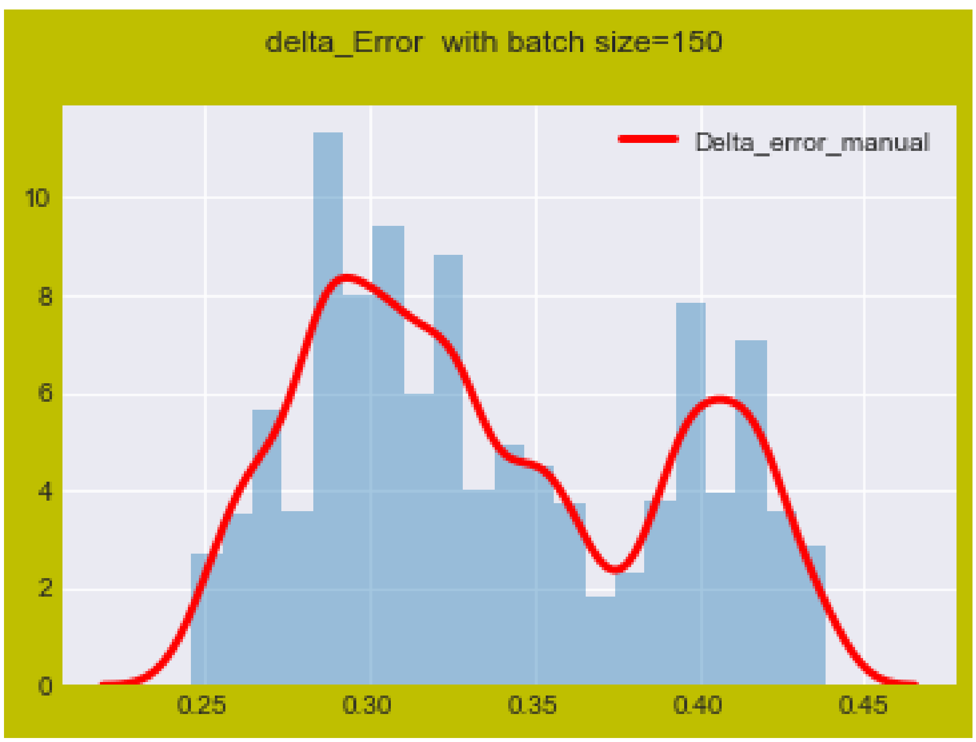


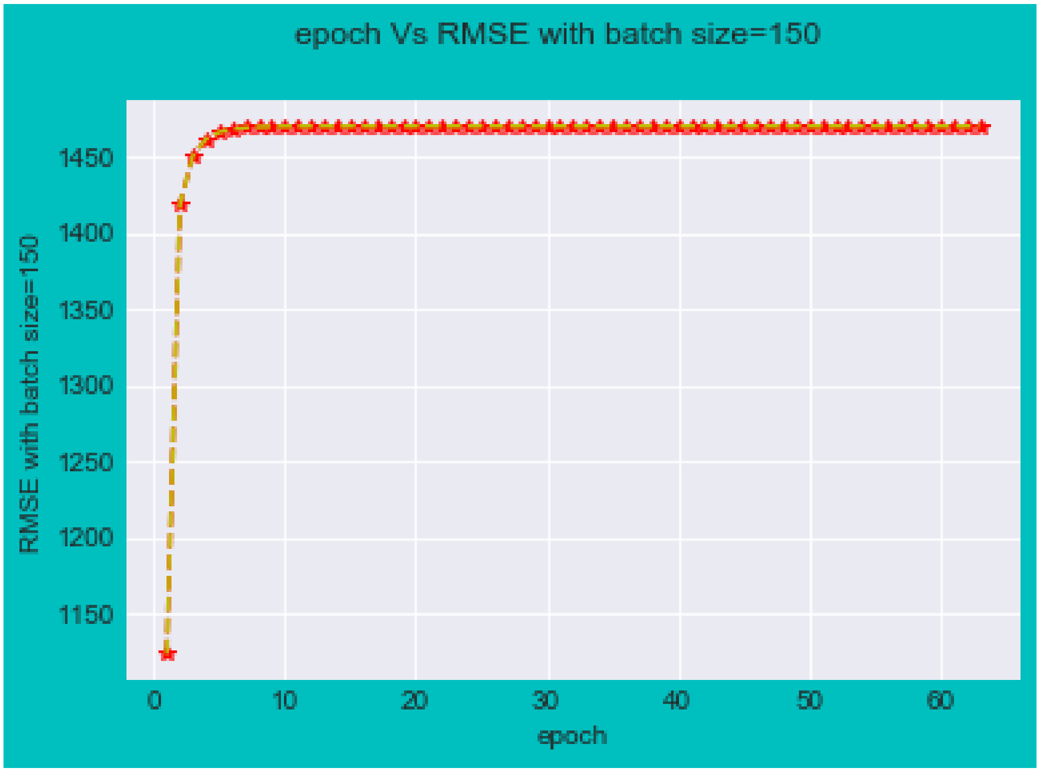

```
The best value of best_Learning_rate is 0.
MSE_value= 19.360248669
RMSE =  4.40002825775
For batch size100
iteration = 64
Total number of learning_rate= 63
```



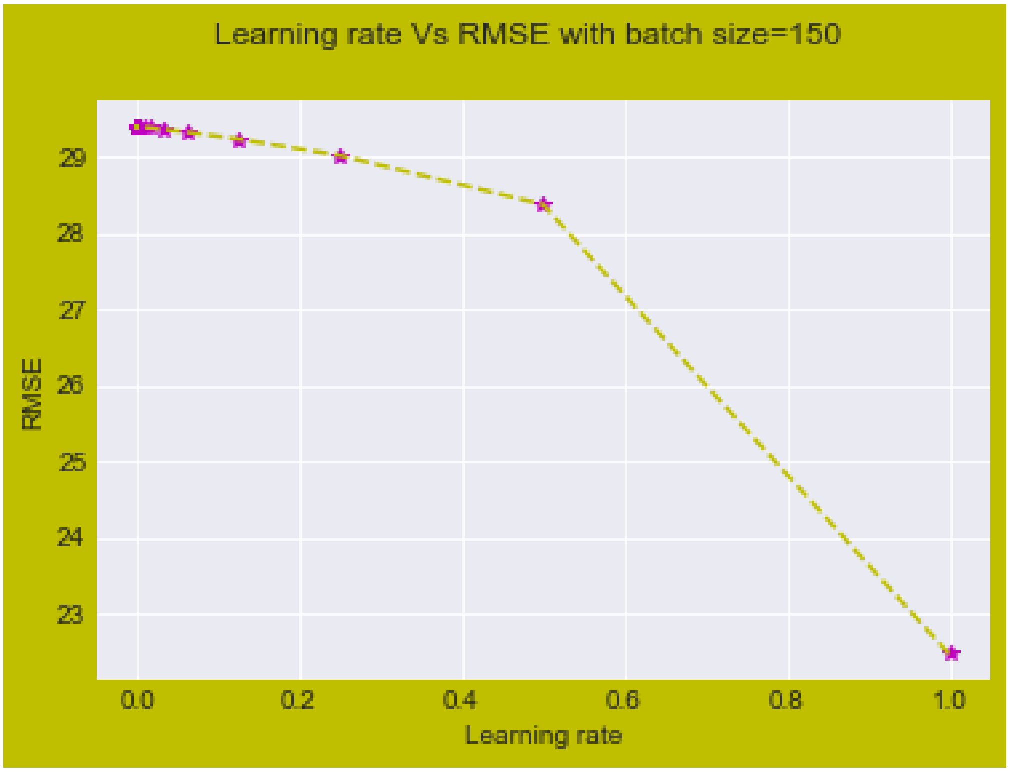

```
150
For batch size150
```


(Y_test) Prices Vs (Y_prediction) Predicted prices: Y_i vs $\hat{Y}_i$ with batch size=
Y_predicted
Y_test
0
20
40
60
80
100
120
140

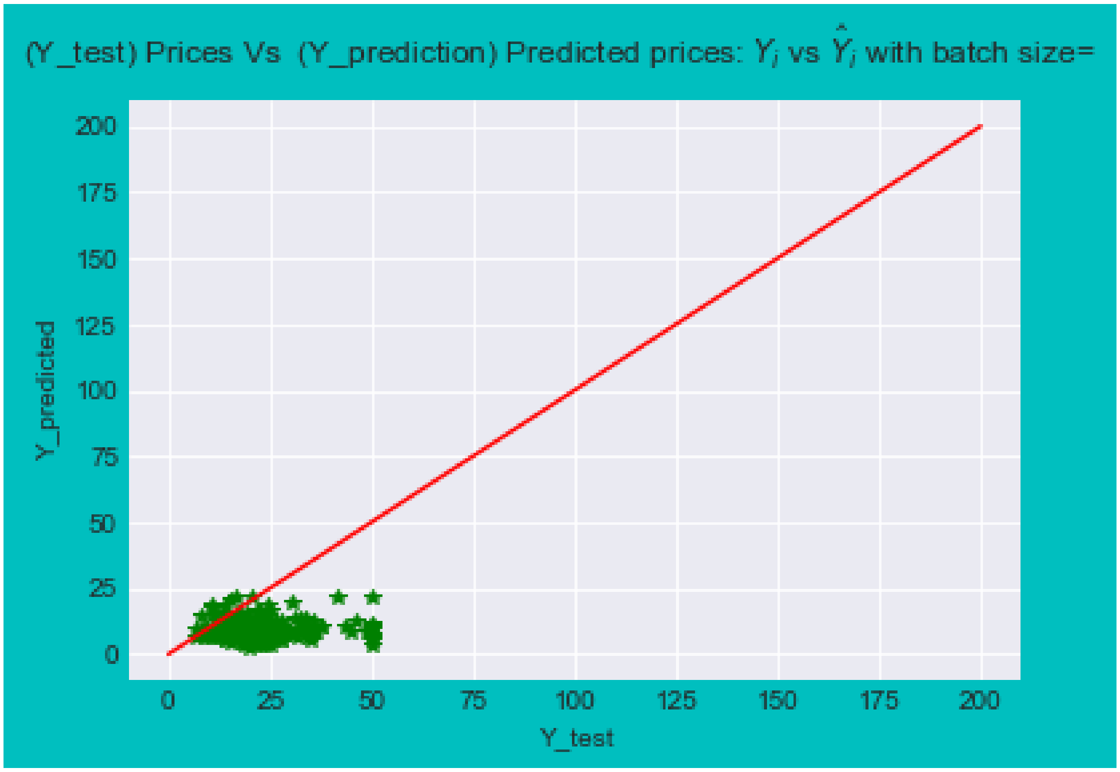

delta_Error with batch size=150
Delta_error_manual
0
2
4
6
8
10
0.25
0.30
0.35
0.40
0.45

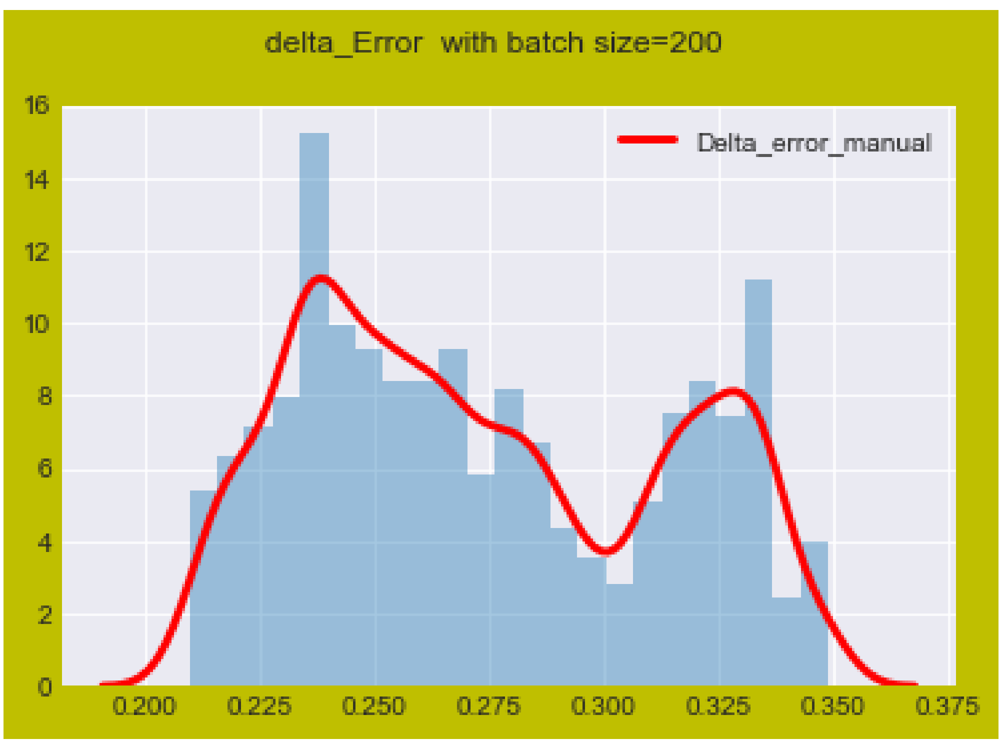


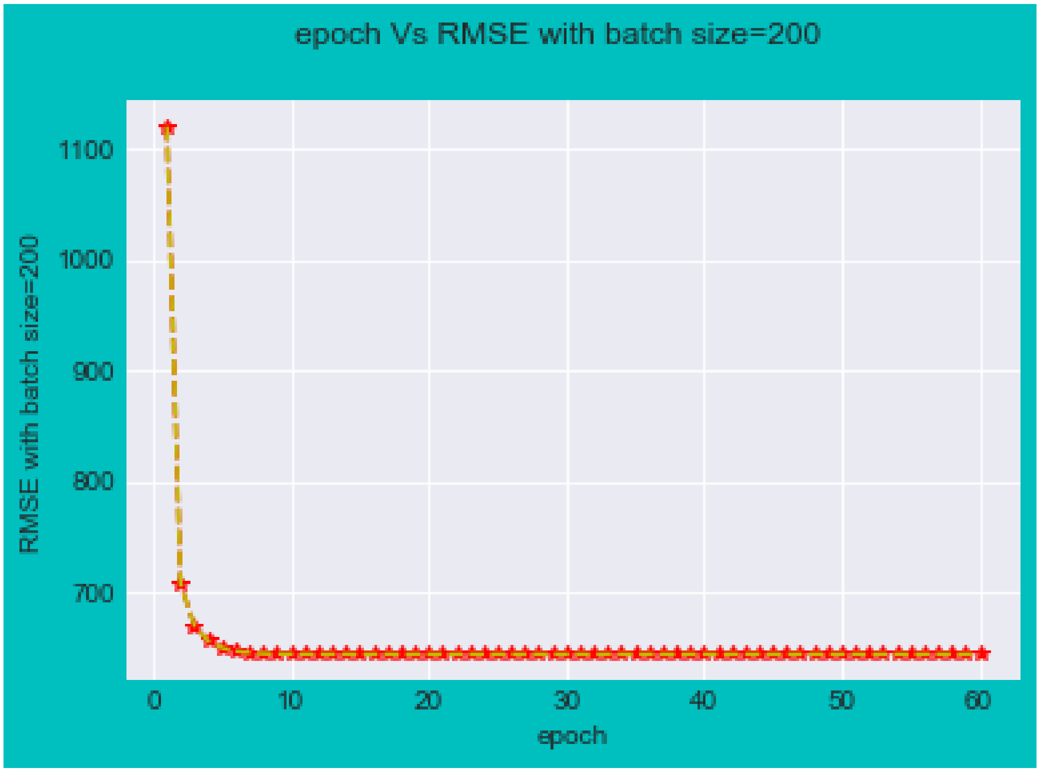

```
The best value of best_Learning_rate is 1.
MSE_value= 29.2543117223
RMSE =  5.40872551737
For batch size150
iteration = 64
Total number of learning_rate= 63
```



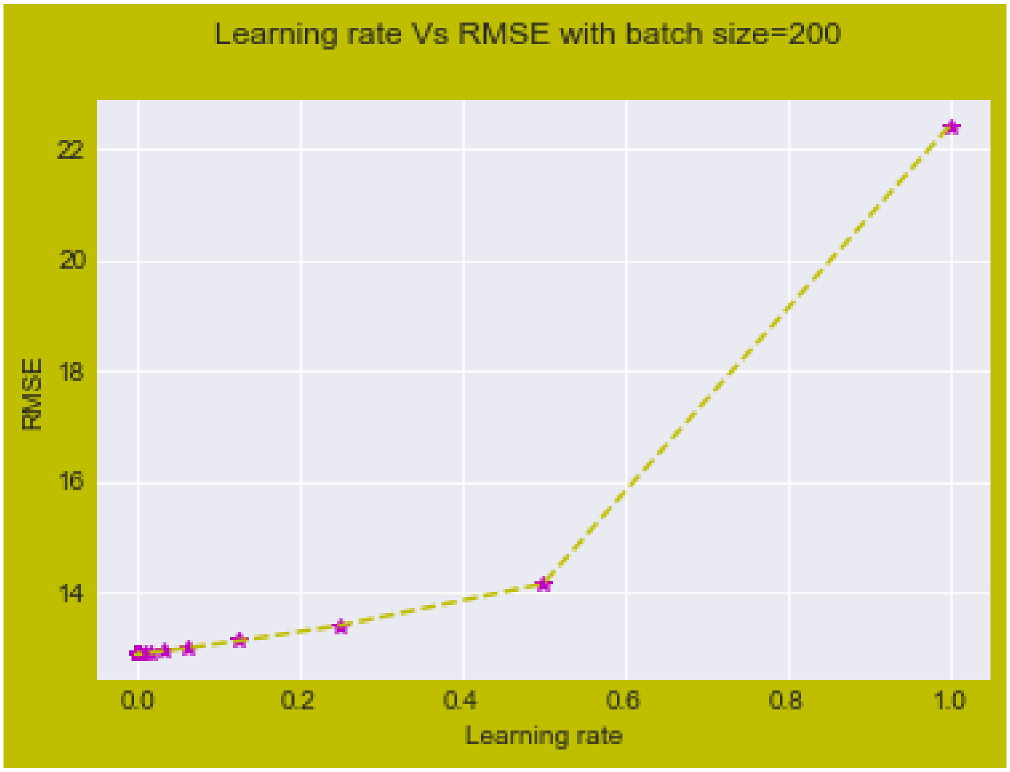

```
200
For batch size200
```


(Y_test) Prices Vs (Y_prediction) Predicted prices: Y_i vs $\hat{Y}_i$ with batch size=
Y_predicted
Y_test
0
25
50
75
100
125
150
175
200


delta_Error with batch size=200
Delta_error_manual
0
2
4
6
8
10
12
14
16
0.200
0.225
0.250
0.275
0.300
0.325
0.350
0.375




```
The best value of best_Learning_rate is 0.
MSE_value= 13.069762568
RMSE =  3.61521265875
For batch size200
iteration = 61
Total number of learning_rate= 60
```



[34]:
```
columns = ["Model","Batch_Size","RMSE","MSE", "Iteration", "Optimal learning␣
↪Rate"]
pd.DataFrame(models_performence1, columns=columns)
```

[34]:
```
                  Model  Batch_Size      RMSE        MSE  Iteration  \
0  SGD Manual Function          50  5.062591  25.629827         64
1  SGD Manual Function         100  4.400028  19.360249         64
2  SGD Manual Function         150  5.408726  29.254312         64
3  SGD Manual Function         200  3.615213  13.069763         61

   Optimal learning Rate
0           2.220446e-16
1           3.552714e-15
2           1.000000e+00
3           4.440892e-16
```

3 SGD_Manual Vs SGD_sklearn

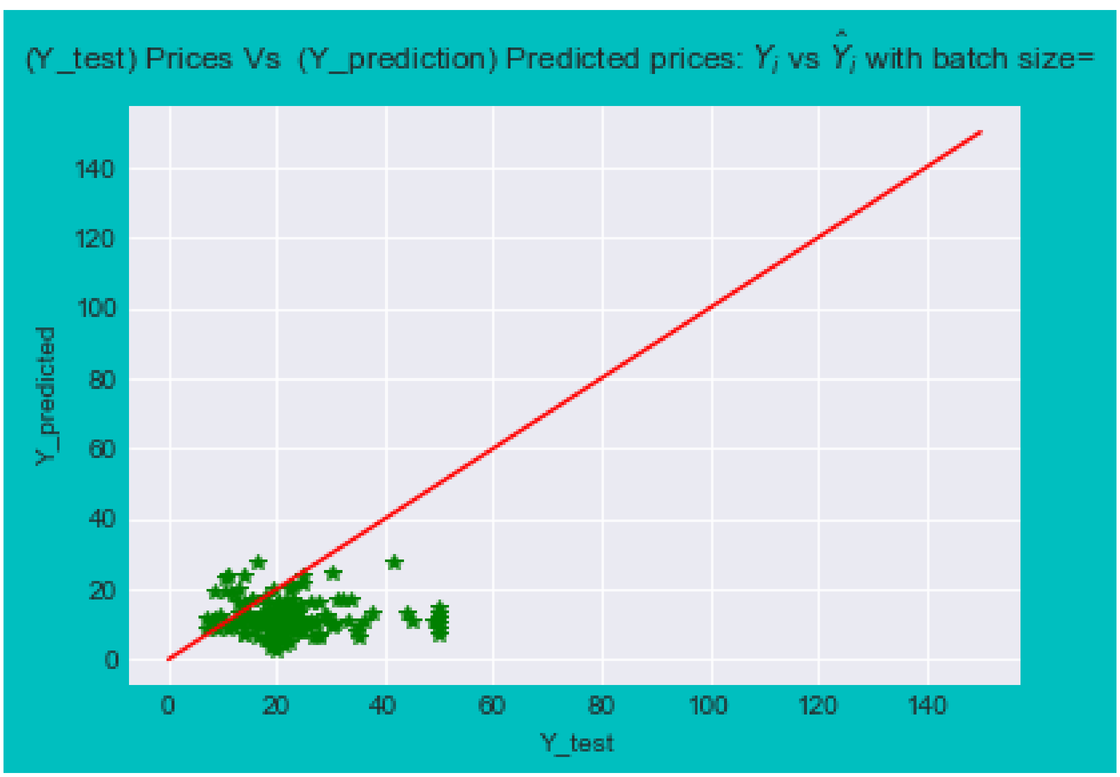
[35]:
```
models_performence1 = {
    'Model':[],
    'Batch_Size':[],
    'RMSE': [],
    'MSE':[],
    'Iteration':[],
    'Optimal learning Rate':[],


}
columns = ["Model","Batch_Size","RMSE","MSE", "Iteration", "Optimal learning␣
↪Rate"]
pd.DataFrame(models_performence1, columns=columns)
```

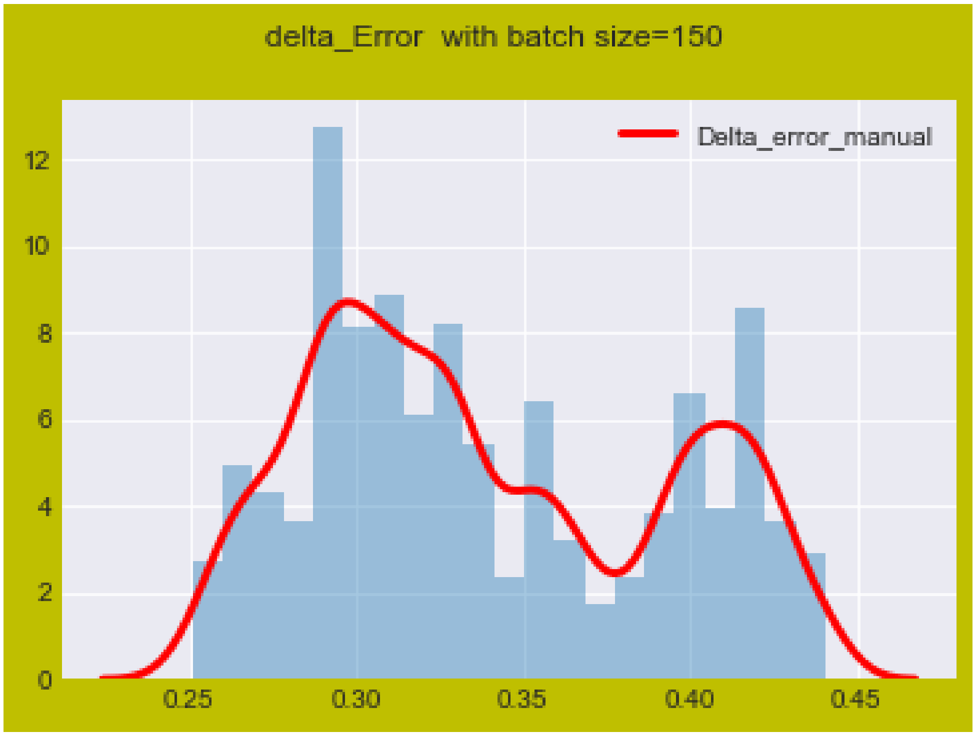
[35]:
```
Empty DataFrame
Columns: [Model, Batch_Size, RMSE, MSE, Iteration, Optimal learning Rate]
Index: []
```

for batch size 150

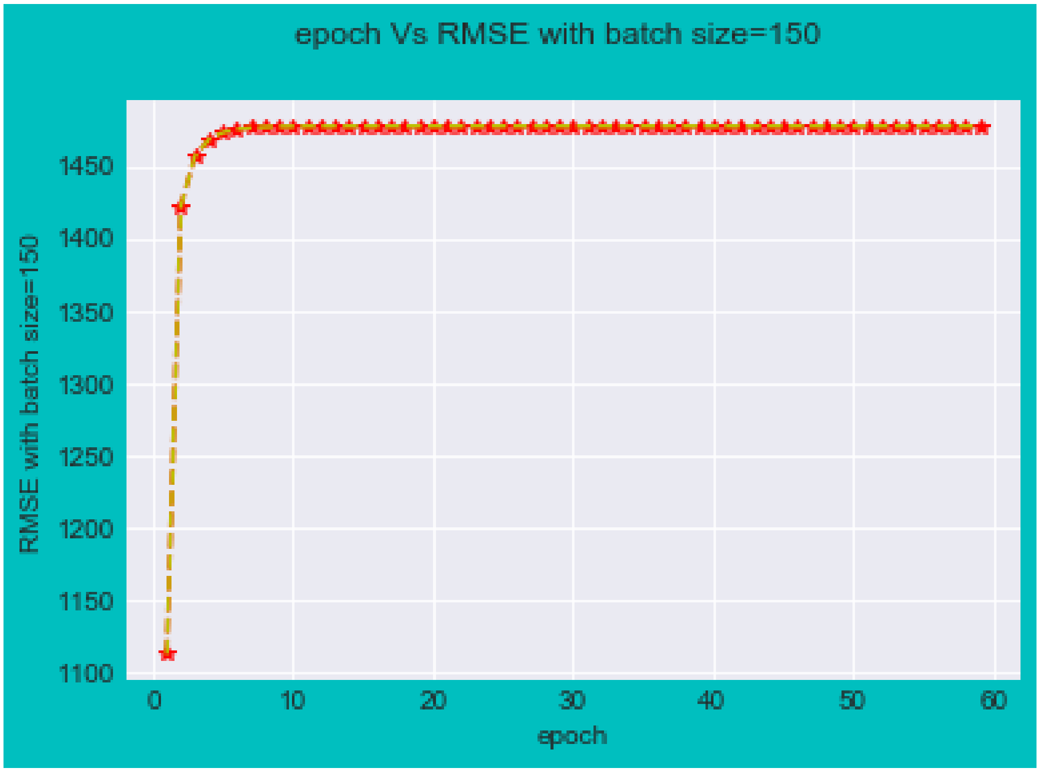
[36]:
```
SGD(150)
```

```
For batch size150
```



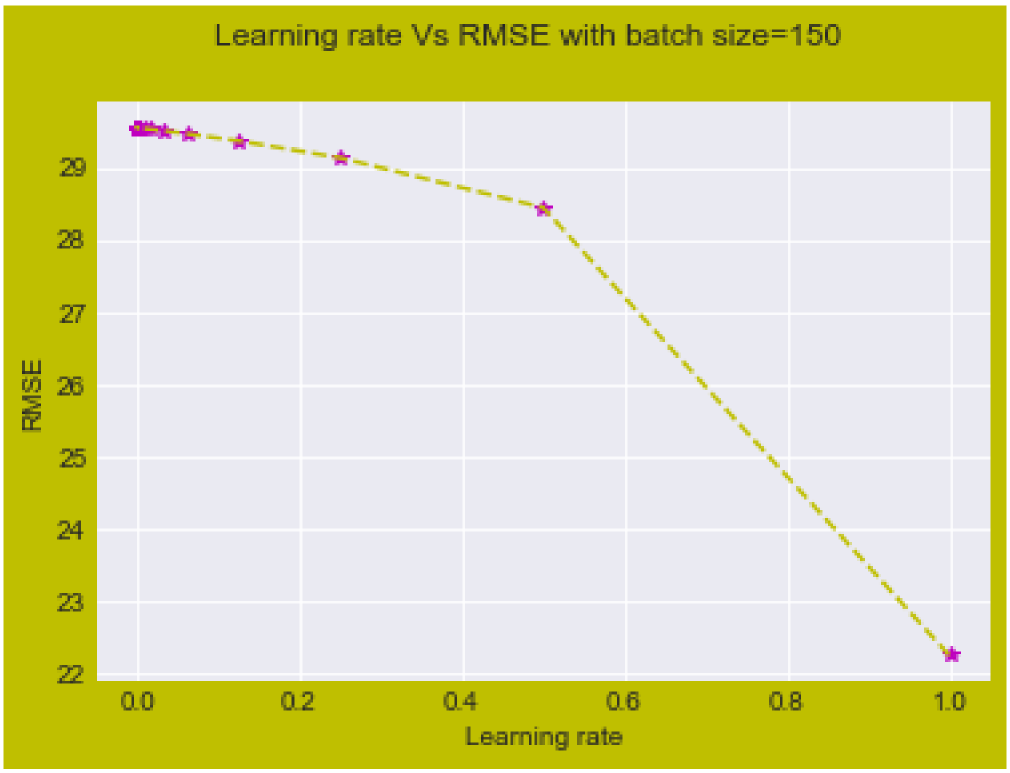


delta_Error with batch size=150
Delta_error_manual
0
2
4
6
8
10
12
0.25
0.30
0.35
0.40
0.45



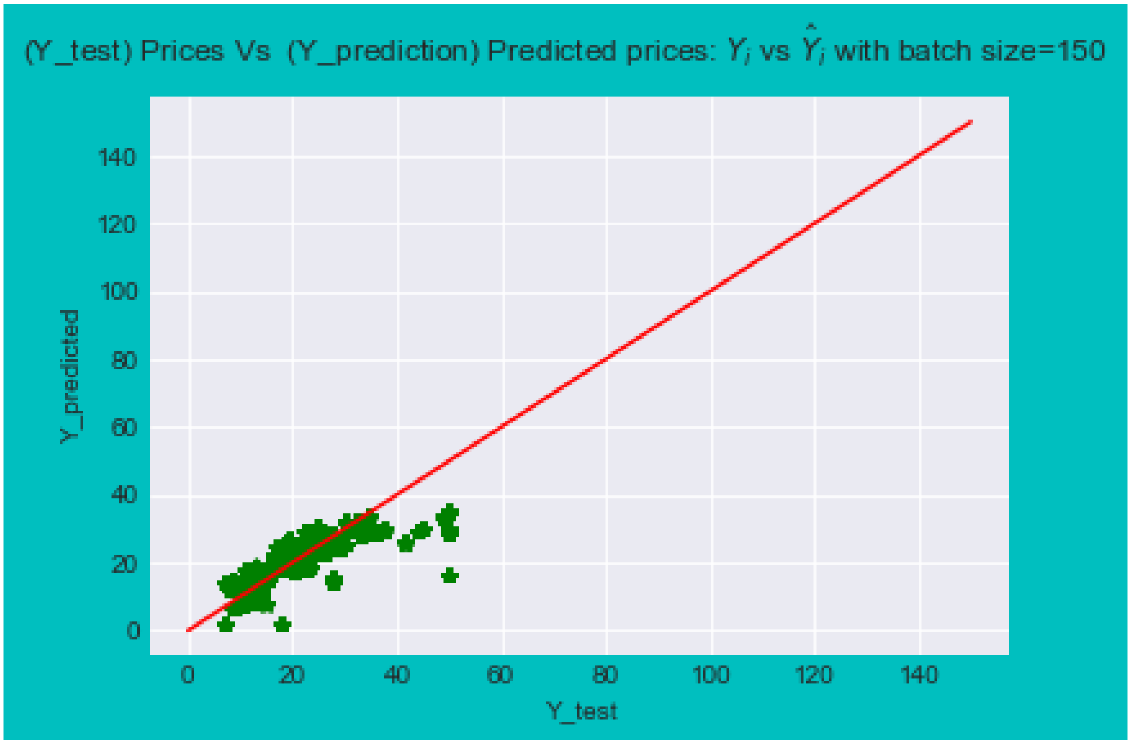
epoch Vs RMSE with batch size=150
RMSE with batch size=150
1100
1150
1200
1250
1300
1350
1400
1450
0
10
20
30
40
50
60
epoch

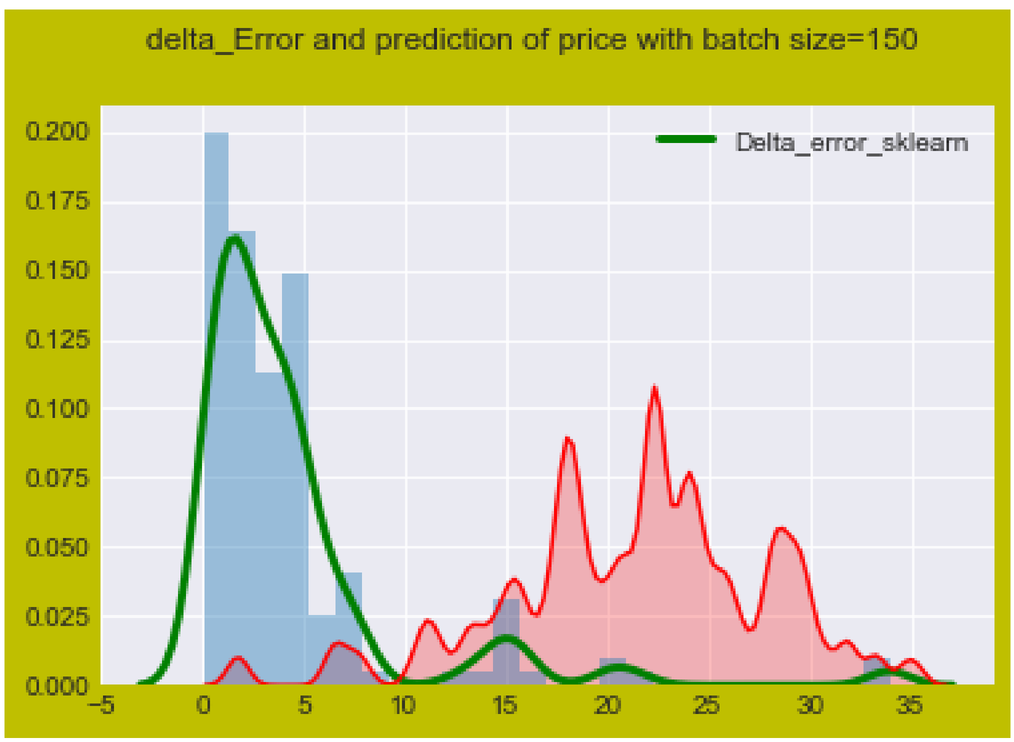
```
The best value of best_Learning_rate is 1.
MSE_value= 29.3977723217
RMSE =  5.42197125792
For batch size150
iteration = 60
Total number of learning_rate= 59
```

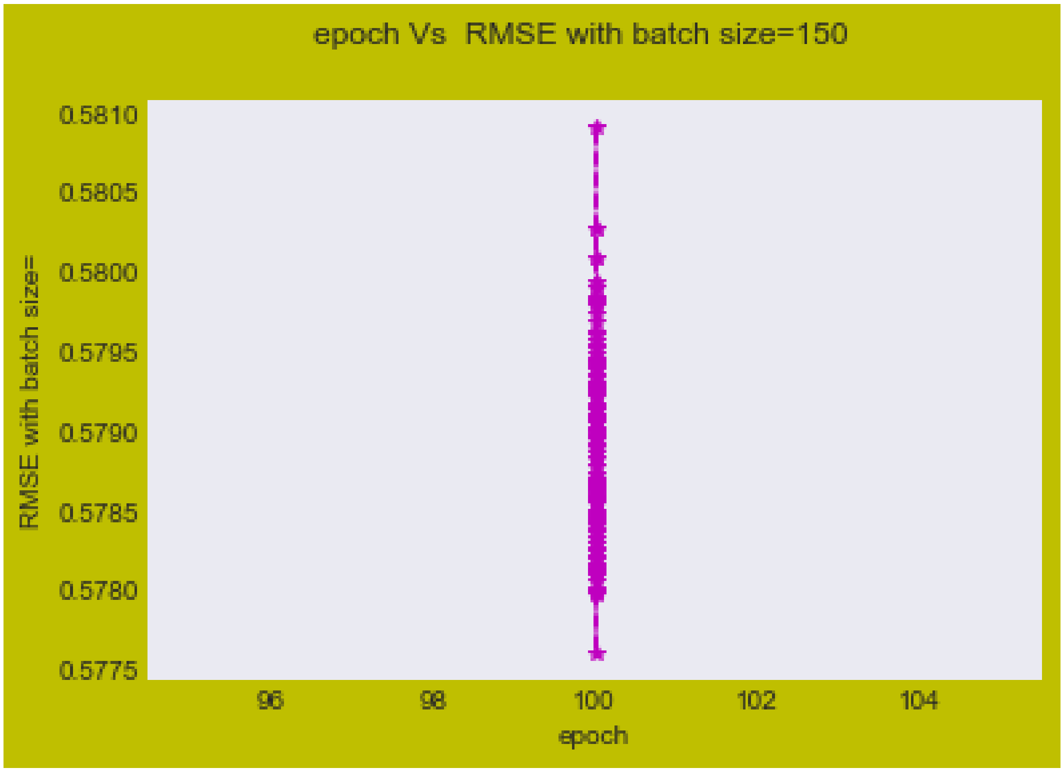


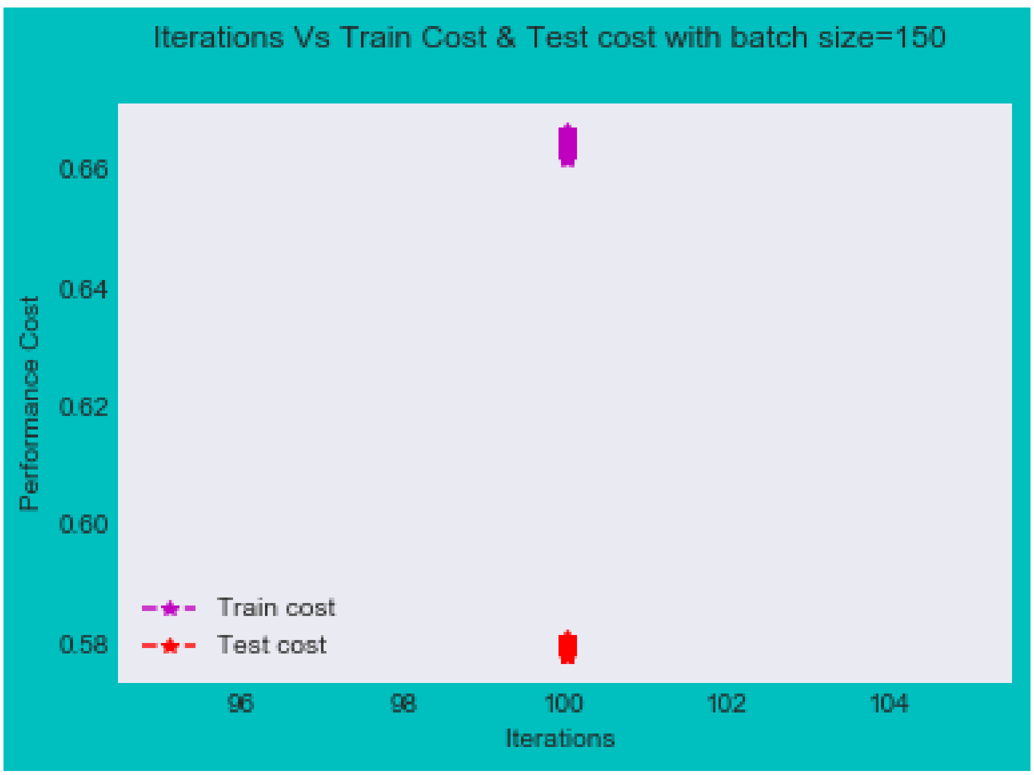
[37]:
```
sgdreg_function(1,150)
```

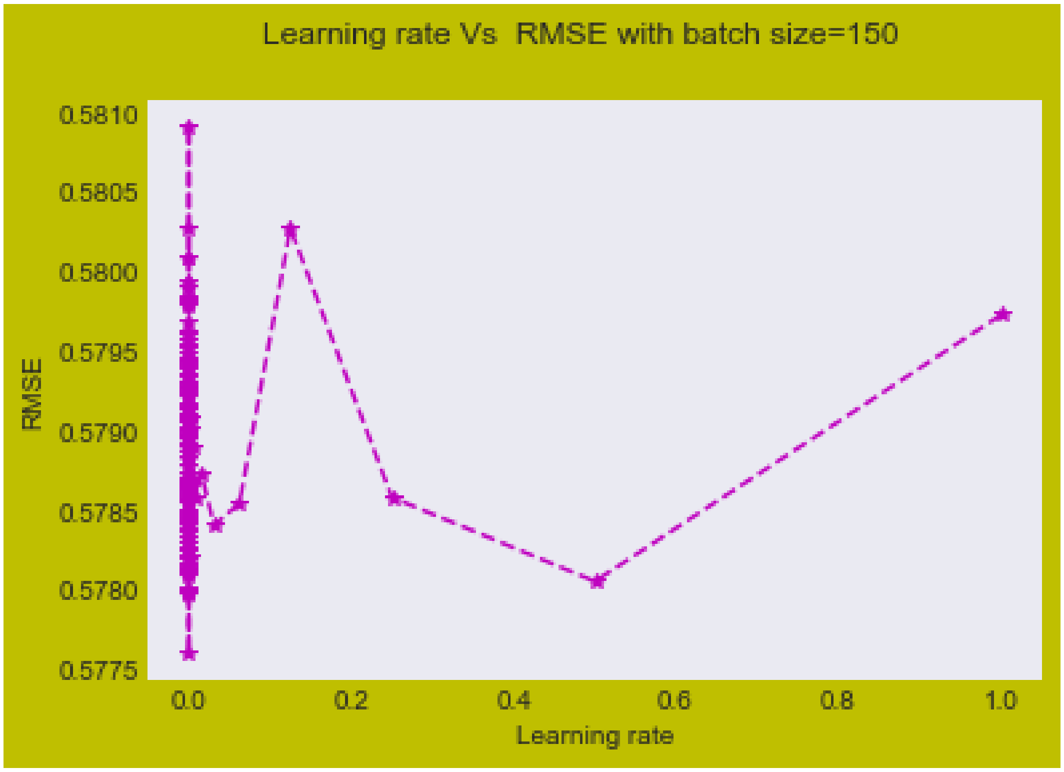
```
Training Error= 0.335742324564
Testing_error 0.4209298324
```


(Y_test) Prices Vs (Y_prediction) Predicted prices: Y_i vs $\hat{Y}_i$ with batch size=150
Y_predicted
Y_test
0
20
40
60
80
100
120
140


delta_Error and prediction of price with batch size=150
Delta_error_sklearn
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
−5
0
5
10
15
20
25
30
35


epoch Vs RMSE with batch size=150
RMSE with batch size=
0.5810
0.5805
0.5800
0.5795
0.5790
0.5785
0.5780
0.5775
96
98
100
102
104
epoch


Iterations Vs Train Cost & Test cost with batch size=150
Performance Cost
0.66
0.64
0.62
0.60
0.58
Train cost
Test cost
96
98
100
102
104
Iterations




```
The best value of best_Learning_rate is 0.
Batch Size 150
RMSE with batch size=150 5.38007871877
MSE with batch size=150 28.9452470202
```

3.1 Y_predicted using manual SGD Vs Y_predicted using Sklearn SGD

Y_predicted using manual SGD == y_hat_manual_SGD

Error(y-y_hat) for manual SGD == delta_Error

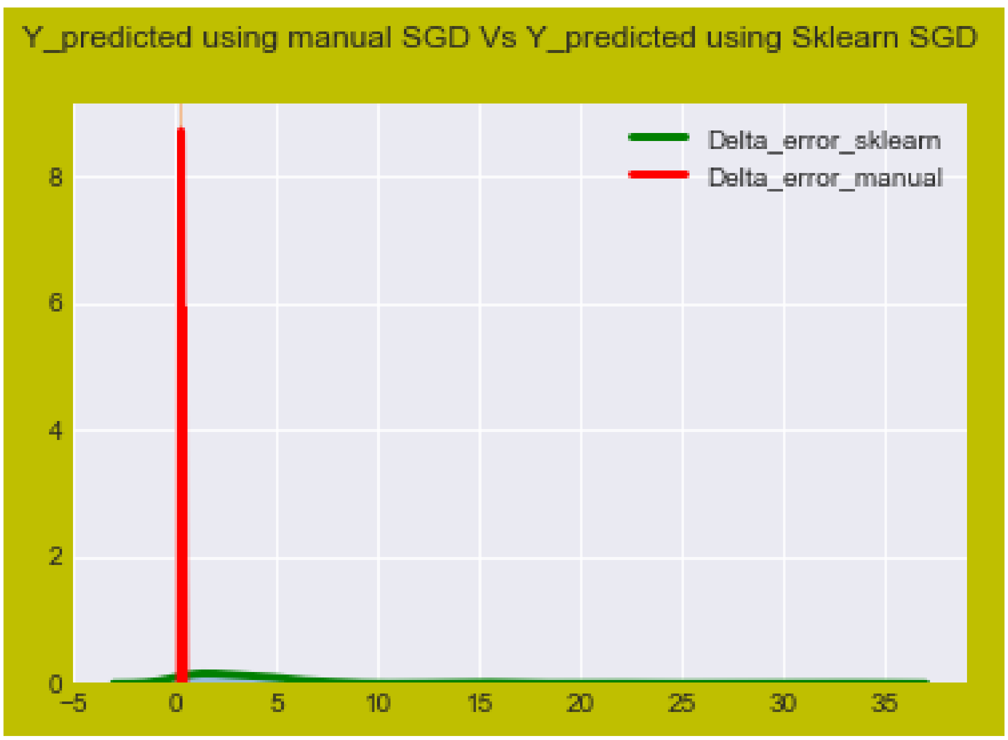
Y_predicted using Sklearn SGD == Y_hat_Predicted

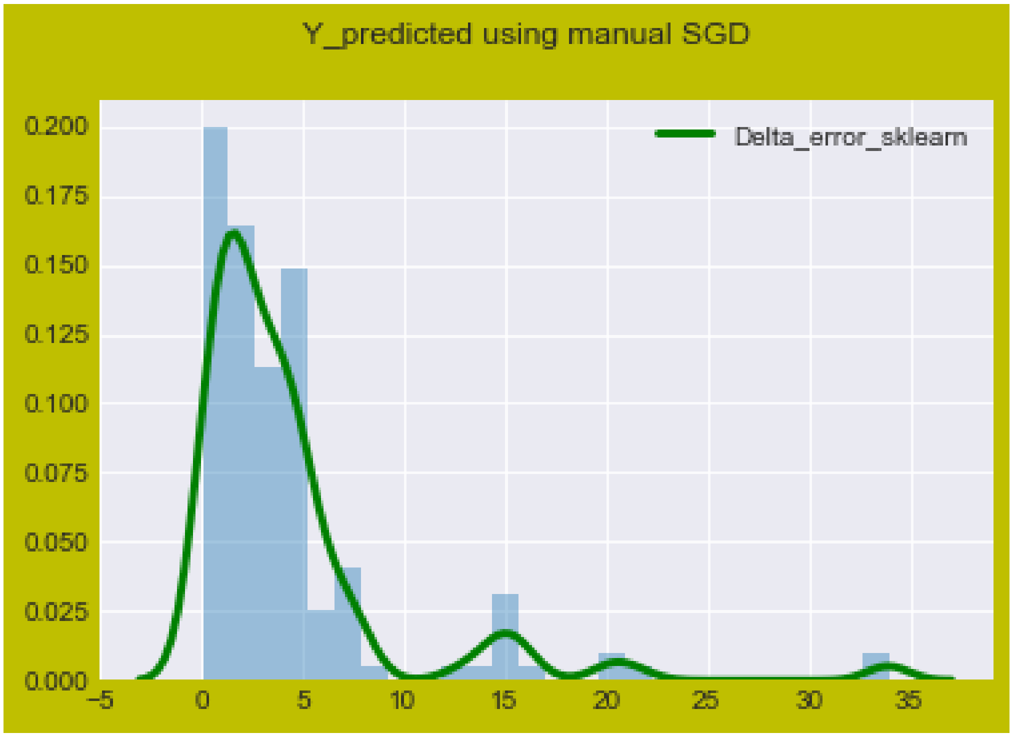
Error(y-y_hat) for SKlearn SGD == delta_error

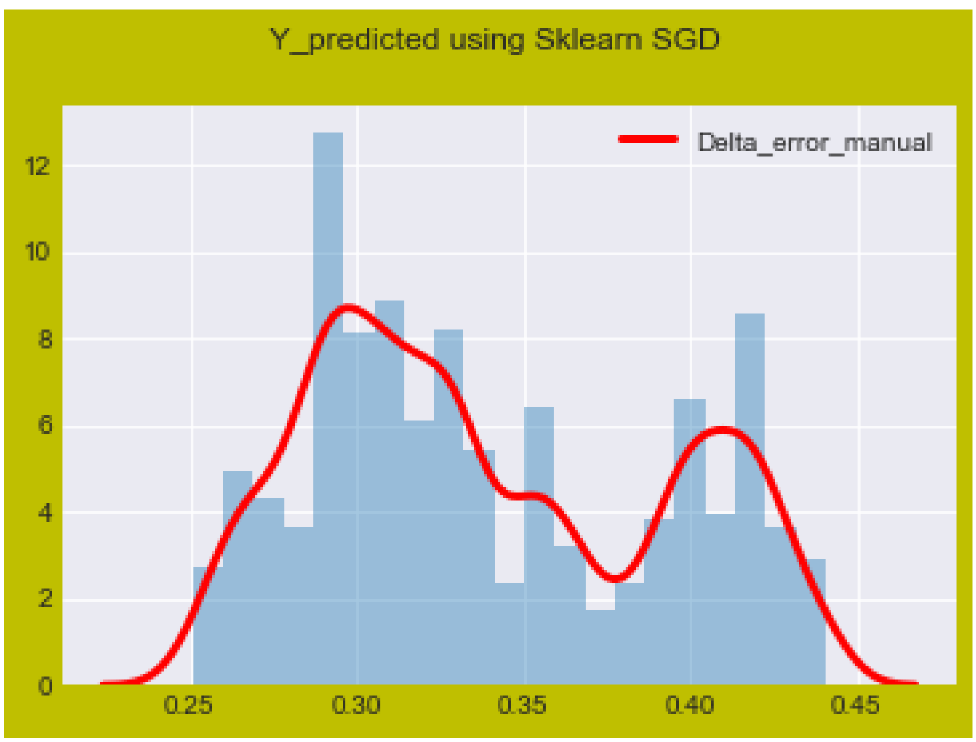
[41]:
```
def y_hat_cal(delta_error_sklearn,delta_Error_manual):
    fig41 = plt.figure( facecolor='y', edgecolor='k')
    fig41.suptitle('Y_predicted using manual SGD Vs Y_predicted using Sklearn␣
 ↪SGD ', fontsize=12)
```

```
    sns.set_style('darkgrid')
    Y_sklearn=np.array(sum(delta_error_sklearn)/len(delta_error_sklearn))

    Y_manual=np.array(delta_Error_manual)
    #print(Y_manual[0])
    sns.distplot(Y_sklearn,kde_kws={"color": "g", "lw": 3, "label":␣
↪"Delta_error_sklearn"} )
    sns.distplot(Y_manual,kde_kws={"color": "r", "lw": 3, "label":␣
↪"Delta_error_manual"} )
    fig51 = plt.figure( facecolor='y', edgecolor='k')
    fig51.suptitle('Y_predicted using manual SGD ', fontsize=12)
    sns.distplot(Y_sklearn,kde_kws={"color": "g", "lw": 3, "label":␣
↪"Delta_error_sklearn"} )

    fig41 = plt.figure( facecolor='y', edgecolor='k')
    fig41.suptitle(' Y_predicted using Sklearn SGD ', fontsize=12)
    sns.distplot(Y_manual,kde_kws={"color": "r", "lw": 3, "label":␣
↪"Delta_error_manual"} )
```

[44]:
```
y_hat_cal(delta_error,delta_Error)
```

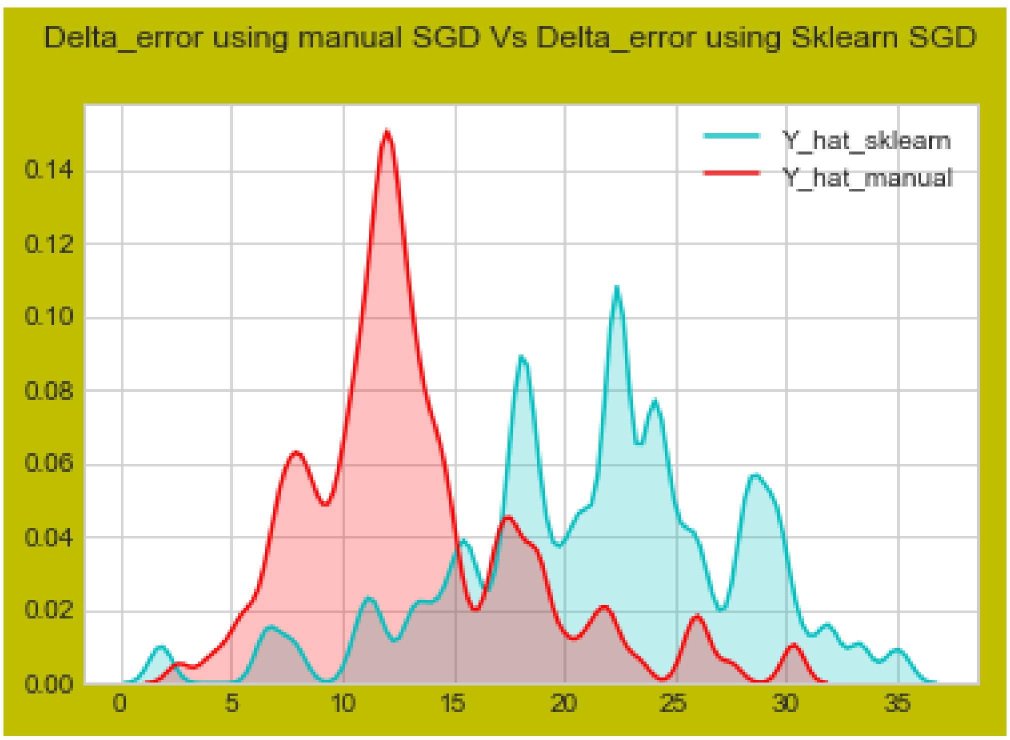



Y_predicted using manual SGD
Delta_error_sklearn
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
−5
0
5
10
15
20
25
30
35


Y_predicted using Sklearn SGD
Delta_error_manual
12
10
8
6
4
2
0
0.25
0.30
0.35
0.40
0.45

[43]:
```
def y_skl_maual(y_hat_sklearn,y_hat_maunal):
    fig41 = plt.figure( facecolor='y', edgecolor='k')
    fig41.suptitle('Delta_error using manual SGD Vs Delta_error using Sklearn␣
 ↪SGD ', fontsize=12)

    sns.set_style('whitegrid')
    Y_sklearn=np.array(sum(y_hat_sklearn)/len(y_hat_sklearn))

    Y_manual=np.array(scale*sum(y_hat_maunal)/len(y_hat_maunal))
    #print(Y_manual[0])

    sns.kdeplot(Y_sklearn,shade=True, color="c", bw=0.5,label='Y_hat_sklearn')
    sns.kdeplot(Y_manual[0],shade=True, color="r", bw=0.5,label='Y_hat_manual')
```

[45]:
```
y_skl_maual(Y_hat_Predicted,y_hat_manual_SGD)
```



[40]:
```
columns = ["Model","Batch_Size","RMSE","MSE", "Iteration", "Optimal learning␣
 ↪Rate"]
pd.DataFrame(models_performence1, columns=columns)
```

[40]:
```
                                Model  Batch_Size      RMSE        MSE  \
0                 SGD Manual Function         150  5.421971  29.397772
1  sklearn.linear_model.SGDRegressor         150  5.380079  28.945247
```

```
   Iteration  Optimal learning Rate
0       60.0           1.000000e+00
1      100.0           2.067952e-25
```

3.2 Observation

- In stochastic gradient descent Manual model(a user designed model),RMSE(root mean squared error) is varied as compared to sklearn designed stochastic gradient descent model for varied number of batch_size.
- Graphs between learning rate vs RMSE & Epoch Vs RMSE are plotted.
- From the graph , stochastic gradient descent model performance can be observed .

Comparision of SGD_sklearn and SGD_manual with batch_size=150 :-

```
* Distributions Plots for  errors(y - y_hat) and It is overlapping as shown in graph "y_hat_cal

* "Delta_error using manual SGD Vs Delta_error using Sklearn SGD" graph is plotted .Varience(sp

* RMSE Vs epoch for manual SGD graph looks like almost "L" shape.So, Model doesn't leads to ove

* RMSE value and MSE value for batch_size 150 is almost similar as seen in above table

*  Optimal learning rate is low for SGD sklearn and 1 which high in this case is for SGD manual
```