In [7]:
```python
from sklearn.datasets import load_boston
boston = load_boston()
```

In [8]:
```python
print(boston.data.shape)
```

```
(506, 13)
```

In [9]:
```python
print(boston.feature_names)
```

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

In [10]:
```python
print(boston.target)
```

```
[24.  21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 15.  18.9 21.7 20.4
 18.2 19.9 23.1 17.5 20.2 18.2 13.6 19.6 15.2 14.5 15.6 13.9 16.6 14.8
 18.4 21.  12.7 14.5 13.2 13.1 13.5 18.9 20.  21.  24.7 30.8 34.9 26.6
 25.3 24.7 21.2 19.3 20.  16.6 14.4 19.4 19.7 20.5 25.  23.4 18.9 35.4
 24.7 31.6 23.3 19.6 18.7 16.  22.2 25.  33.  23.5 19.4 22.  17.4 20.9
 24.2 21.7 22.8 23.4 24.1 21.4 20.  20.8 21.2 20.3 28.  23.9 24.8 22.9
 23.9 26.6 22.5 22.2 23.6 28.7 22.6 22.  22.9 25.  20.6 28.4 21.4 38.7
 43.8 33.2 27.5 26.5 18.6 19.3 20.1 19.5 19.5 20.4 19.8 19.4 21.7 22.8
 18.8 18.7 18.5 18.3 21.2 19.2 20.4 19.3 22.  20.3 20.5 17.3 18.8 21.4
 15.7 16.2 18.  14.3 19.2 19.6 23.  18.4 15.6 18.1 17.4 17.1 13.3 17.8
 14.  14.4 13.4 15.6 11.8 13.8 15.6 14.6 17.8 15.4 21.5 19.6 15.3 19.4
 17.  15.6 13.1 41.3 24.3 23.3 27.  50.  50.  50.  22.7 25.  50.  23.8
 23.8 22.3 17.4 19.1 23.1 23.6 22.6 29.4 23.2 24.6 29.9 37.2 39.8 36.2
 37.9 32.5 26.4 29.6 50.  32.  29.8 34.9 37.  30.5 36.4 31.1 29.1 50.
 33.3 30.3 34.6 34.9 32.9 24.1 42.3 48.5 50.  22.6 24.4 22.5 24.4 20.
 21.7 19.3 22.4 28.1 23.7 25.  23.3 28.7 21.5 23.  26.7 21.7 27.5 30.1
 44.8 50.  37.6 31.6 46.7 31.5 24.3 31.7 41.7 48.3 29.  24.  25.1 31.5
 23.7 23.3 22.  20.1 22.2 23.7 17.6 18.5 24.3 20.5 24.5 26.2 24.4 24.8
 29.6 42.8 21.9 20.9 44.  50.  36.  30.1 33.8 43.1 48.8 31.  36.5 22.8
 30.7 50.  43.5 20.7 21.1 25.2 24.4 35.2 32.4 32.  33.2 33.1 29.1 35.1
```

```
 45.4 35.4 46.  50.  32.2 22.  20.1 23.2 22.3 24.8 28.5 37.3 27.9 23.9
 21.7 28.6 27.1 20.3 22.5 29.  24.8 22.  26.4 33.1 36.1 28.4 33.4 28.2
 22.8 20.3 16.1 22.1 19.4 21.6 23.8 16.2 17.8 19.8 23.1 21.  23.8 23.1
 20.4 18.5 25.  24.6 23.  22.2 19.3 22.6 19.8 17.1 19.4 22.2 20.7 21.1
 19.5 18.5 20.6 19.  18.7 32.7 16.5 23.9 31.2 17.5 17.2 23.1 24.5 26.6
 22.9 24.1 18.6 30.1 18.2 20.6 17.8 21.7 22.7 22.6 25.  19.9 20.8 16.8
 21.9 27.5 21.9 23.1 50.  50.  50.  50.  50.  13.8 13.8 15.  13.9 13.3
 13.1 10.2 10.4 10.9 11.3 12.3  8.8  7.2 10.5  7.4 10.2 11.5 15.1 23.2
  9.7 13.8 12.7 13.1 12.5  8.5  5.   6.3  5.6  7.2 12.1  8.3  8.5  5.
 11.9 27.9 17.2 27.5 15.  17.2 17.9 16.3  7.   7.2  7.5 10.4  8.8  8.4
 16.7 14.2 20.8 13.4 11.7  8.3 10.2 10.9 11.   9.5 14.5 14.1 16.1 14.3
 11.7 13.4  9.6  8.7  8.4 12.8 10.5 17.1 18.4 15.4 10.8 11.8 14.9 12.6
 14.1 13.  13.4 15.2 16.1 17.8 14.9 14.1 12.7 13.5 14.9 20.  16.4 17.7
 19.5 20.2 21.4 19.9 19.  19.1 19.1 20.1 19.9 19.6 23.2 29.8 13.8 13.3
 16.7 12.  14.6 21.4 23.  23.7 25.  21.8 20.6 21.2 19.1 20.6 15.2  7.
  8.1 13.6 20.1 21.8 24.5 23.1 19.7 18.3 21.2 17.5 16.8 22.4 20.6 23.9
 22.  11.9]
```

In [11]:
```
print(boston.DESCR)
```

```
.. _boston_dataset:

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

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Va
lue (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 2
5,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds ri
ver; 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 19
40
        - 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 blac
ks 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. 'Hedoni
c
prices and the demand for clean air', J. Environ. Economics & Managemen
t,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagn
ostics
...', 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 pape
rs that address regression
problems.

.. topic:: References
```

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

In [12]:

```python
import pandas as pd
bos = pd.DataFrame(boston.data)
print(bos.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  1
5.3
1  0.02731   0.0  7.07  0.0  0.469  6.421  78.9  4.9671  2.0  242.0  1
7.8
2  0.02729   0.0  7.07  0.0  0.469  7.185  61.1  4.9671  2.0  242.0  1
7.8
3  0.03237   0.0  2.18  0.0  0.458  6.998  45.8  6.0622  3.0  222.0  1
8.7
4  0.06905   0.0  2.18  0.0  0.458  7.147  54.2  6.0622  3.0  222.0  1
8.7

       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
```

In [15]:

```python
bos.columns  = ["CRIM", "ZN", "INDUS", "CHAS","NOX","RM","AGE","DIS","R
AD","TAX","PTRATIO","B","LSTAT"]
bos.head(3)
```

Out[15]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTA

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTA
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.9
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.1
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.0

In [16]:

```
bos['PRICE'] = boston.target

X = bos.drop('PRICE', axis = 1)
Y = bos['PRICE']
```

In [26]:

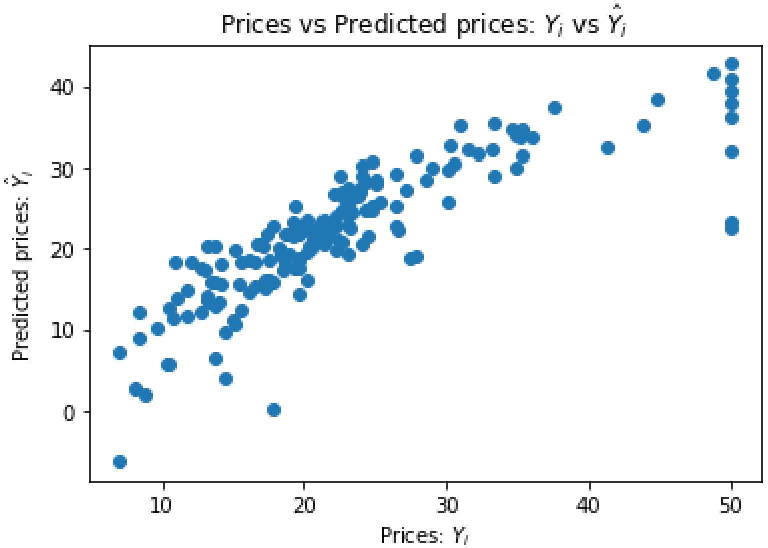
```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size =
0.33, random_state = 5)
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

```
(339, 13)
(167, 13)
(339,)
(167,)
```

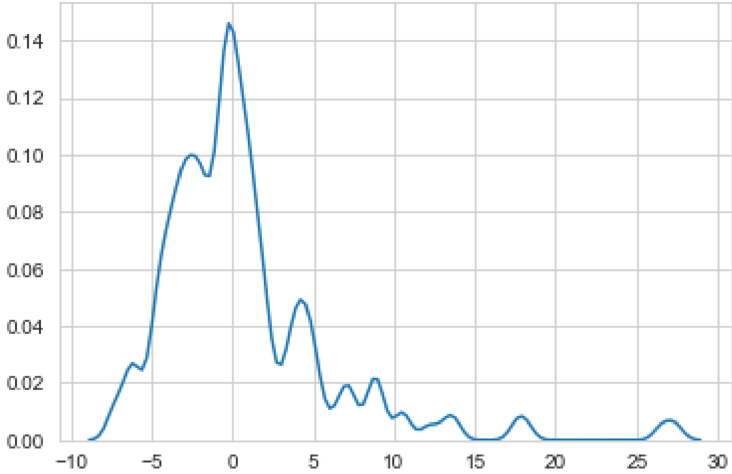
In [29]:

```
# 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
import matplotlib.pyplot as plt
lm = LinearRegression()
lm.fit(X_train, Y_train)

Y_pred = lm.predict(X_test)

plt.scatter(Y_test, Y_pred)
plt.xlabel("Prices: $Y_i$")
plt.ylabel("Predicted prices: $\hat{Y}_i$")
```

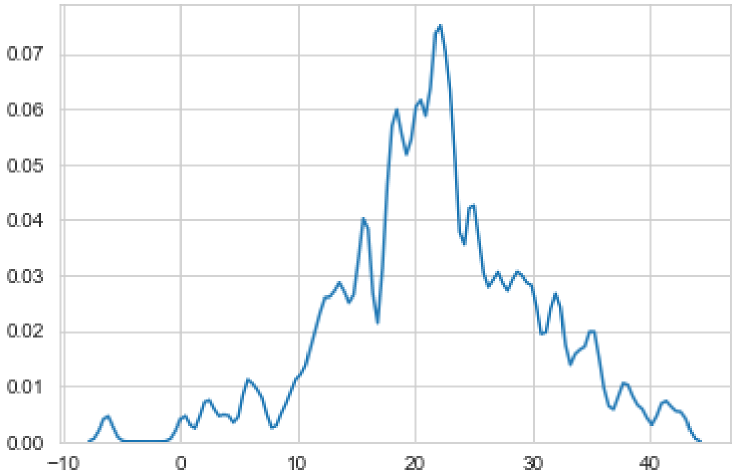
```
plt.title("Prices vs Predicted prices: $Y_i$ vs $\hat{Y}_i$")
plt.show()
```



In [30]:
```
delta_y = Y_test - Y_pred;

import seaborn as sns;
import numpy as np;
sns.set_style('whitegrid')
sns.kdeplot(np.array(delta_y), bw=0.5)
plt.show()
```



In [31]:
```
sns.set_style('whitegrid')
sns.kdeplot(np.array(Y_pred), bw=0.5)
plt.show()
```



In [32]:
```
from sklearn.preprocessing import StandardScaler
scalar = StandardScaler()
```

```
X_train = scalar.fit_transform(X_train)
X_test = scalar.transform(X_test)
```

In [33]:
```
"""class sklearn.linear_model.SGDRegressor(loss='squared_loss', penalty
='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None,
 tol=None, shuffle=True, verbose=0, epsilon=0.1, random_state=None, lea
rning_rate='invscaling', eta0=0.01, power_t=0.25, early_stopping=False,
 validation_fraction=0.1, n_iter_no_change=5, warm_start=False, average
=False, n_iter=None)[source]"""

from sklearn.linear_model import SGDRegressor
sgd = SGDRegressor(loss='squared_loss')
sgd.fit(X_train,Y_train)
y_pred = sgd.predict(X_test)
```

```
C:\Users\ishaan\Anaconda3\lib\site-packages\sklearn\linear_model\stocha
stic_gradient.py:166: FutureWarning: max_iter and tol parameters have b
een added in SGDRegressor in 0.19. If both are left unset, they default
to max_iter=5 and tol=None. If tol is not None, max_iter defaults to ma
x_iter=1000. From 0.21, default max_iter will be 1000, and default tol
will be 1e-3.
  FutureWarning)
```

In [34]:
```
y_pred.shape
```

Out[34]:
```
(167,)
```

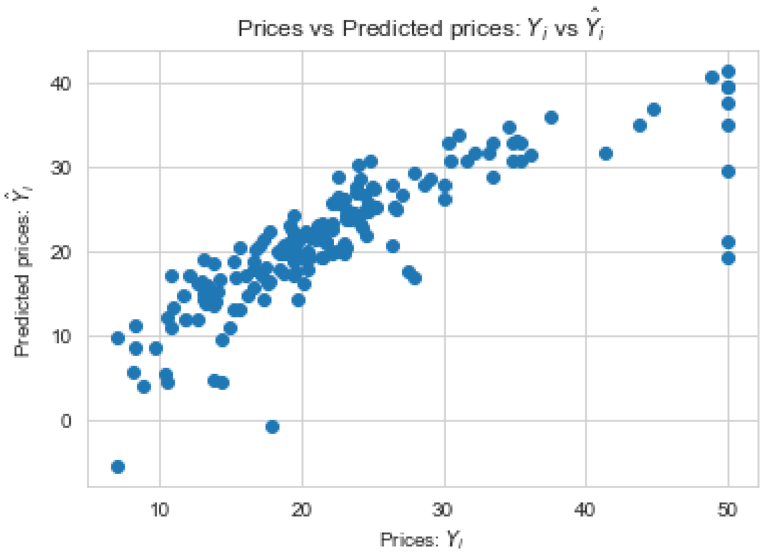
In [36]:
```
y_pred
```

Out[36]:
```
array([35.98778717, 29.35102374, 26.29992509,  4.6585246 , 32.98614839,
        5.34482803, 26.74744682, 28.60250521, 26.22287493, 20.50137456,
       31.71390341, 20.84809516, 22.45507499, 31.42319873, 26.69033853,
       17.02097144, -0.7991628 , 19.38976122, 14.27927189, 10.97184552,
        5.58549865, 19.83261694, 36.94231903, 23.85870775, 31.61413647,
       10.96511681, 24.21712976, 23.05981356, 22.6789443 , 23.31668085,
       14.16034581,  9.80243137, 16.18129984, 22.34524978, 27.92632529,
       19.22465653, 27.21954042,  8.62472008, 40.76020837, 32.72554599,
       18.61600049,  4.42512596, 27.78295167, 11.94537554, 26.51517181,
       30.57925235, -5.51448343, 17.89111287, 22.53868154, 14.03507283,
```

```
       19.84445757, 19.717407  , 23.31175951, 13.90360678, 18.80810837,
       25.70364301, 34.98658884, 15.30776099, 27.89280597, 21.1423233 ,
       20.25130106, 24.89720395, 14.76693418, 31.52057773, 20.18951969,
       11.04908753, 19.83403452, 24.66546839, 21.16350152, 19.26768653,
       18.78138827, 25.5603198 , 17.04568505, 17.49428081, 16.84472354,
       26.11419967, 21.35348116, 16.79534948, 33.83261775, 16.73942207,
       20.04634004, 39.47132922, 20.79337365, 14.68678913, 24.33443225,
       16.5090932 , 18.04232553,  8.4752252 , 19.0740912 , 19.67585381,
       35.02149577, 17.51961561, 19.25314147, 17.89032747, 25.78689003,
       27.50330816, 13.13986483, 24.84072095, 20.07591967, 14.95453649,
       20.92901335, 21.76298659, 14.13223153, 41.4937935 ,  4.10523285,
       21.53537504, 17.04473891, 19.18068146, 28.47651289, 17.20237235,
       27.62388902, 22.85904909, 18.92315368, 32.86125593, 18.84158734,
       14.74167359, 20.59387868, 17.79873079, 19.68431714, 16.18922742,
       20.77912923, 32.84117827, 22.18056888, 20.60836572, 23.84273521,
       25.1838154 , 19.19537012, 21.54774187, 22.97288554, 39.46854089,
       37.54189572, 26.22475714, 12.03590648, 16.35168997, 17.06840596,
       21.3397632 , 13.54325091,  4.40205007, 23.28297518, 30.56782092,
       21.97295346, 20.39253833, 16.11971941, 22.51722145, 34.81479717,
       21.25247618, 30.15062491, 17.30672473, 21.81394003, 28.87816059,
       13.78030746, 29.56530756, 11.85224679, 13.24868422, 24.40926827,
       30.78231851, 13.06790387, 24.68129013, 28.86553875, 30.61434227,
       15.90383086, 30.78508791,  9.5089426 , 32.90311461, 25.07836301,
       19.87341561, 15.68128981])
```

In [37]:

```
plt.scatter(Y_test, y_pred)
plt.xlabel("Prices: $Y_i$")
plt.ylabel("Predicted prices: $\hat{Y}_i$")
plt.title("Prices vs Predicted prices: $Y_i$ vs $\hat{Y}_i$")
plt.show()
```



In [39]:
```
from sklearn.metrics import mean_squared_error
error_one = mean_squared_error(Y_test,y_pred)
print(error_one)
```

```
31.021371841344322
```

In [ ]:

In [63]:
```
# Predicting

def predict(row,coefficients):
    y_dash = coefficients[0]
    for i in range(len(row)-1):
        y_dash += coefficients[i+1] * row[i]
    return y_dash

# estimating coefficients using SGD

def coeff(data_X,data_Y,step_size,n_epochs):
    coef = np.random.rand(len(data_X[0])+1)
```

```
    for epoch in range(n_epochs):
        err_sum = 0
        j=0
        for row in data_X:
            y_dash = predict(row,coef)
            err = data_Y.iloc[j] - y_dash
          #  err_sum += err**2
            coef[0] = coef[0] + step_size * err
            j=j+1
            for i in range(len(row)-1):
                coef[i+1] = coef[i+1] + step_size * err * row[i]

        print('# of epoch=%d, step_size=%.3f, Squared_error=%.3f' % (ep
och, step_size, err))


    return coef
```

In [64]:

```
step_size = 0.0001
n_epochs = 150
#X_data = np.random.rand(X_train)
coef = coeff(X_train,Y_train, step_size, n_epochs)
print(coef)
```

```
# of epoch=0, step_size=0.000, Squared_error=16.500
# of epoch=1, step_size=0.000, Squared_error=14.831
# of epoch=2, step_size=0.000, Squared_error=13.424
# of epoch=3, step_size=0.000, Squared_error=12.235
# of epoch=4, step_size=0.000, Squared_error=11.227
# of epoch=5, step_size=0.000, Squared_error=10.369
# of epoch=6, step_size=0.000, Squared_error=9.637
# of epoch=7, step_size=0.000, Squared_error=9.009
# of epoch=8, step_size=0.000, Squared_error=8.468
# of epoch=9, step_size=0.000, Squared_error=7.999
# of epoch=10, step_size=0.000, Squared_error=7.591
# of epoch=11, step_size=0.000, Squared_error=7.234
# of epoch=12, step_size=0.000, Squared_error=6.919
# of epoch=13, step_size=0.000, Squared_error=6.640
# of epoch=14, step_size=0.000, Squared_error=6.392
```

```
# of epoch=15, step_size=0.000, Squared_error=6.168
# of epoch=16, step_size=0.000, Squared_error=5.967
# of epoch=17, step_size=0.000, Squared_error=5.784
# of epoch=18, step_size=0.000, Squared_error=5.617
# of epoch=19, step_size=0.000, Squared_error=5.463
# of epoch=20, step_size=0.000, Squared_error=5.321
# of epoch=21, step_size=0.000, Squared_error=5.189
# of epoch=22, step_size=0.000, Squared_error=5.066
# of epoch=23, step_size=0.000, Squared_error=4.950
# of epoch=24, step_size=0.000, Squared_error=4.842
# of epoch=25, step_size=0.000, Squared_error=4.740
# of epoch=26, step_size=0.000, Squared_error=4.643
# of epoch=27, step_size=0.000, Squared_error=4.552
# of epoch=28, step_size=0.000, Squared_error=4.465
# of epoch=29, step_size=0.000, Squared_error=4.382
# of epoch=30, step_size=0.000, Squared_error=4.304
# of epoch=31, step_size=0.000, Squared_error=4.228
# of epoch=32, step_size=0.000, Squared_error=4.157
# of epoch=33, step_size=0.000, Squared_error=4.088
# of epoch=34, step_size=0.000, Squared_error=4.023
# of epoch=35, step_size=0.000, Squared_error=3.960
# of epoch=36, step_size=0.000, Squared_error=3.900
# of epoch=37, step_size=0.000, Squared_error=3.843
# of epoch=38, step_size=0.000, Squared_error=3.788
# of epoch=39, step_size=0.000, Squared_error=3.735
# of epoch=40, step_size=0.000, Squared_error=3.685
# of epoch=41, step_size=0.000, Squared_error=3.637
# of epoch=42, step_size=0.000, Squared_error=3.592
# of epoch=43, step_size=0.000, Squared_error=3.548
# of epoch=44, step_size=0.000, Squared_error=3.506
# of epoch=45, step_size=0.000, Squared_error=3.467
# of epoch=46, step_size=0.000, Squared_error=3.429
# of epoch=47, step_size=0.000, Squared_error=3.393
# of epoch=48, step_size=0.000, Squared_error=3.359
# of epoch=49, step_size=0.000, Squared_error=3.326
# of epoch=50, step_size=0.000, Squared_error=3.296
# of epoch=51, step_size=0.000, Squared_error=3.267
# of epoch=52, step_size=0.000, Squared_error=3.239
# of epoch=53, step_size=0.000, Squared_error=3.213
```

```
# of epoch=54, step_size=0.000, Squared_error=3.189
# of epoch=55, step_size=0.000, Squared_error=3.166
# of epoch=56, step_size=0.000, Squared_error=3.144
# of epoch=57, step_size=0.000, Squared_error=3.124
# of epoch=58, step_size=0.000, Squared_error=3.105
# of epoch=59, step_size=0.000, Squared_error=3.088
# of epoch=60, step_size=0.000, Squared_error=3.071
# of epoch=61, step_size=0.000, Squared_error=3.056
# of epoch=62, step_size=0.000, Squared_error=3.042
# of epoch=63, step_size=0.000, Squared_error=3.029
# of epoch=64, step_size=0.000, Squared_error=3.018
# of epoch=65, step_size=0.000, Squared_error=3.007
# of epoch=66, step_size=0.000, Squared_error=2.998
# of epoch=67, step_size=0.000, Squared_error=2.989
# of epoch=68, step_size=0.000, Squared_error=2.981
# of epoch=69, step_size=0.000, Squared_error=2.975
# of epoch=70, step_size=0.000, Squared_error=2.969
# of epoch=71, step_size=0.000, Squared_error=2.964
# of epoch=72, step_size=0.000, Squared_error=2.960
# of epoch=73, step_size=0.000, Squared_error=2.957
# of epoch=74, step_size=0.000, Squared_error=2.955
# of epoch=75, step_size=0.000, Squared_error=2.953
# of epoch=76, step_size=0.000, Squared_error=2.952
# of epoch=77, step_size=0.000, Squared_error=2.952
# of epoch=78, step_size=0.000, Squared_error=2.952
# of epoch=79, step_size=0.000, Squared_error=2.954
# of epoch=80, step_size=0.000, Squared_error=2.956
# of epoch=81, step_size=0.000, Squared_error=2.958
# of epoch=82, step_size=0.000, Squared_error=2.961
# of epoch=83, step_size=0.000, Squared_error=2.965
# of epoch=84, step_size=0.000, Squared_error=2.969
# of epoch=85, step_size=0.000, Squared_error=2.973
# of epoch=86, step_size=0.000, Squared_error=2.979
# of epoch=87, step_size=0.000, Squared_error=2.984
# of epoch=88, step_size=0.000, Squared_error=2.991
# of epoch=89, step_size=0.000, Squared_error=2.997
# of epoch=90, step_size=0.000, Squared_error=3.004
# of epoch=91, step_size=0.000, Squared_error=3.012
# of epoch=92, step_size=0.000, Squared_error=3.020
```

```
# of epoch=93, step_size=0.000, Squared_error=3.028
# of epoch=94, step_size=0.000, Squared_error=3.036
# of epoch=95, step_size=0.000, Squared_error=3.045
# of epoch=96, step_size=0.000, Squared_error=3.055
# of epoch=97, step_size=0.000, Squared_error=3.064
# of epoch=98, step_size=0.000, Squared_error=3.074
# of epoch=99, step_size=0.000, Squared_error=3.084
# of epoch=100, step_size=0.000, Squared_error=3.095
# of epoch=101, step_size=0.000, Squared_error=3.106
# of epoch=102, step_size=0.000, Squared_error=3.117
# of epoch=103, step_size=0.000, Squared_error=3.128
# of epoch=104, step_size=0.000, Squared_error=3.139
# of epoch=105, step_size=0.000, Squared_error=3.151
# of epoch=106, step_size=0.000, Squared_error=3.163
# of epoch=107, step_size=0.000, Squared_error=3.175
# of epoch=108, step_size=0.000, Squared_error=3.187
# of epoch=109, step_size=0.000, Squared_error=3.200
# of epoch=110, step_size=0.000, Squared_error=3.212
# of epoch=111, step_size=0.000, Squared_error=3.225
# of epoch=112, step_size=0.000, Squared_error=3.238
# of epoch=113, step_size=0.000, Squared_error=3.251
# of epoch=114, step_size=0.000, Squared_error=3.264
# of epoch=115, step_size=0.000, Squared_error=3.278
# of epoch=116, step_size=0.000, Squared_error=3.291
# of epoch=117, step_size=0.000, Squared_error=3.304
# of epoch=118, step_size=0.000, Squared_error=3.318
# of epoch=119, step_size=0.000, Squared_error=3.332
# of epoch=120, step_size=0.000, Squared_error=3.346
# of epoch=121, step_size=0.000, Squared_error=3.359
# of epoch=122, step_size=0.000, Squared_error=3.373
# of epoch=123, step_size=0.000, Squared_error=3.387
# of epoch=124, step_size=0.000, Squared_error=3.401
# of epoch=125, step_size=0.000, Squared_error=3.415
# of epoch=126, step_size=0.000, Squared_error=3.430
# of epoch=127, step_size=0.000, Squared_error=3.444
# of epoch=128, step_size=0.000, Squared_error=3.458
# of epoch=129, step_size=0.000, Squared_error=3.472
# of epoch=130, step_size=0.000, Squared_error=3.486
# of epoch=131, step_size=0.000, Squared_error=3.501
```

```
# of epoch=132, step_size=0.000, Squared_error=3.515
# of epoch=133, step_size=0.000, Squared_error=3.529
# of epoch=134, step_size=0.000, Squared_error=3.544
# of epoch=135, step_size=0.000, Squared_error=3.558
# of epoch=136, step_size=0.000, Squared_error=3.572
# of epoch=137, step_size=0.000, Squared_error=3.587
# of epoch=138, step_size=0.000, Squared_error=3.601
# of epoch=139, step_size=0.000, Squared_error=3.615
# of epoch=140, step_size=0.000, Squared_error=3.630
# of epoch=141, step_size=0.000, Squared_error=3.644
# of epoch=142, step_size=0.000, Squared_error=3.658
# of epoch=143, step_size=0.000, Squared_error=3.672
# of epoch=144, step_size=0.000, Squared_error=3.686
# of epoch=145, step_size=0.000, Squared_error=3.701
# of epoch=146, step_size=0.000, Squared_error=3.715
# of epoch=147, step_size=0.000, Squared_error=3.729
# of epoch=148, step_size=0.000, Squared_error=3.743
# of epoch=149, step_size=0.000, Squared_error=3.757
[22.40083475 -1.48392325  0.50075339 -0.75685621  0.33082352 -0.8299004
3
  4.45235849 -1.18306402 -1.85384349  0.48278019 -0.44824399 -2.0037521
9
  1.27228706  0.16805863]
```

In [65]:

```
predictions=[]
for i in range(len(X_test)):
    pred = predict(X_test[i],coef)
    predictions.append(pred)
```

In [66]:

```
test_err = mean_squared_error(Y_test,predictions)

print("Test error = %.3f"%(test_err))
```

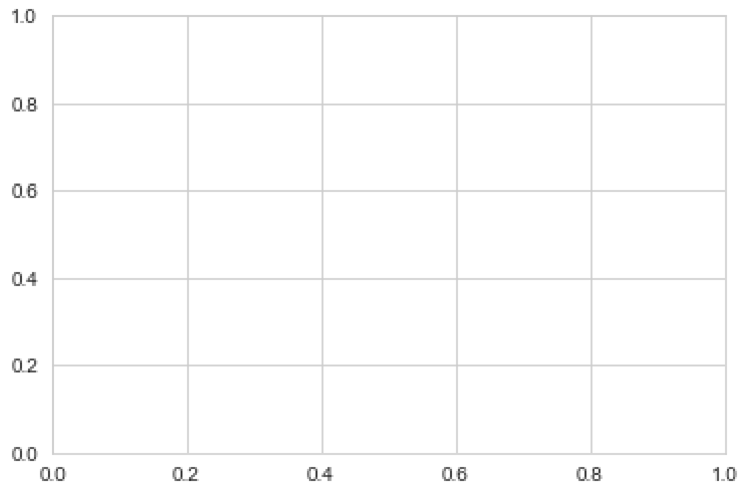
```
Test error = 36.887
```

In [87]:

```
sns.set_style('whitegrid')
sns.scatterplot(y_pred, predictions,color =["red","green"])
plt.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py in scatter(self, x, y, s, c, marker, cmap, norm, vmin, vmax, alpha, linewidths, verts, edgecolors, **kwargs)
   4237                     valid_shape = False
-> 4238                     raise ValueError
   4239             except ValueError:

ValueError: 

During handling of the above exception, another exception occurred:

ValueError                                Traceback (most recent call last)
<ipython-input-87-2249d9b41c16> in <module>
      1 sns.set_style('whitegrid')
----> 2 sns.scatterplot(y_pred, predictions,color =["red","green"])
      3 plt.show()

~\Anaconda3\lib\site-packages\seaborn\relational.py in scatterplot(x, y, hue, style, size, data, palette, hue_order, hue_norm, sizes, size_order, size_norm, markers, style_order, x_bins, y_bins, units, estimator, ci, n_boot, alpha, x_jitter, y_jitter, legend, ax, **kwargs)
   1339         ax = plt.gca()
   1340 
-> 1341     p.plot(ax, kwargs)
   1342 
   1343     return ax

~\Anaconda3\lib\site-packages\seaborn\relational.py in plot(self, ax, kws)
    877         # function will advance the axes property cycle.
    878 
--> 879         scout = ax.scatter([], [], **kws)
    880         s = kws.pop("s", scout.get_sizes())
    881         c = kws.pop("c", scout.get_facecolors())
```

```
~\Anaconda3\lib\site-packages\matplotlib\__init__.py in inner(ax, data,
 *args, **kwargs)
   1808                         "the Matplotlib list!)" % (label_namer,
 func.__name__),
   1809                         RuntimeWarning, stacklevel=2)
-> 1810             return func(ax, *args, **kwargs)
   1811
   1812         inner.__doc__ = _add_data_doc(inner.__doc__,

~\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py in scatter(self,
 x, y, s, c, marker, cmap, norm, vmin, vmax, alpha, linewidths, verts,
 edgecolors, **kwargs)
   4243                         "acceptable for use with 'x' with size
 {xs}, "
   4244                         "'y' with size {ys}."
-> 4245                         .format(nc=n_elem, xs=x.size, ys=y.size
)
   4246                     )
   4247                 # Both the mapping *and* the RGBA conversion fa
iled: pretty

ValueError: 'c' argument has 2 elements, which is not acceptable for us
e with 'x' with size 0, 'y' with size 0.
```



In [ ]:

```

```

In [ ]:

```

```