In [276]:

```
import warnings
warnings.filterwarnings("ignore")
from sklearn.datasets import load_boston
from random import seed
from random import randrange
from csv import reader
from math import sqrt
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
import random
from sklearn.preprocessing import StandardScaler
```

In [277]:

```
X = load_boston().data
Y = load_boston().target
```

In [278]:

```
X.shape
```

Out[278]:

```
(506, 13)
```

In [279]:

```
Y.shape
```

Out[279]:

```
(506,)
```

In [280]:

```
#Splitting whole data into train and test
from sklearn.model_selection  import train_test_split
X_train, X_test, y_train, y_test=train_test_split(X, Y, test_size=0.1, random_state=5)
```

In [281]:

```
scaler = StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)

X_test = scaler.transform(X_test)
```

In [282]:

```
# creating dataframe

boston_df_train = pd.DataFrame(X_train)

boston_df_train['Price'] = y_train
```

In [283]:

```
boston_df_train.shape
```

Out[283]:

```
(455, 14)
```

In [284]:

```
boston_df_train.head(5)
```

Out[284]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | |
|---|---|---|---|---|---|---|---|---|---|
| **0** | -0.391042 | -0.494611 | -0.535603 | -0.275046 | -0.518897 | -0.289406 | -0.852121 | 0.744305 | -0.5 |
| **1** | -0.408514 | -0.494611 | 0.273127 | -0.275046 | -1.000683 | 0.006805 | -0.806308 | 0.311299 | -0.5 |
| **2** | -0.226784 | -0.494611 | -0.422826 | -0.275046 | -0.131748 | -0.270437 | 1.022686 | -0.040871 | -0.6 |
| **3** | -0.162560 | -0.494611 | 1.274767 | -0.275046 | 2.733156 | -0.835134 | 1.079071 | -1.053859 | -0.5 |
| **4** | -0.417443 | 3.150410 | -1.520920 | -0.275046 | -1.232972 | 0.148344 | -1.134047 | 2.532514 | -0.8 |

In [285]:

```
boston_df_train.shape
```

Out[285]:

```
(455, 14)
```

# Implementing our own sgd based on the formulaes derived in theory

In [286]:

```
# initialising the variables firstly

W = np.random.randn(1,13) # here we are giving shape as (1,13) because we have 13 featu
res in our data starting from index 0 to 12

intercept_b = 0  # let us initialise intercept with 0 initially

iter = 360 # assumed to run the code for 1000 iterations first

learning_rate = 0.001  # learning rate is the term "r" from theory

#k = 30 # as we are performing SGD hence  we need to mention the batch size ---> k=50
```

In [287]:

```
# running the iterations

for i in range(0,iter,1):  # we will be using 200 iterations in order to make our sgd o
ptimisation converge
    # considering some temporary variables for calculation purposes
    w_temp = W
    intercept_temp = intercept_b
    a = np.zeros((1,13))  # you will see its purpose in the coming part of code
    p = 0

    # as we are persofming SGD hence we need a batch size which i have mentioned in k w
here k = 50 (initially taken)
    k = random.randrange(2,455,1)

    boston_sampled_data = boston_df_train.sample(k)

    # now in order to perform numpy multiplication we will need data in the form of num
py arrays instea of dataframe columns

    data_array_sampled_train = np.array(boston_sampled_data.drop('Price',axis=1))

    price_array__sampled_train = np.array(boston_sampled_data['Price'])

    # now as we have considered k data points for our batch size hence we need to do su
mmation k number of times hence
    # we will be running this loop k number of times

    for j in range(0,k,1):
        # vector 'a' will be storing the weight vector while we calculate it using summ
ation while vector 'b' will store the intercept

        # Don't be afraid of the formula below it is just a simple mathematical equatio
n for partial derivative whose image
        # I will be adding in the code below

        # expression for dL/dW ---> here L stands for the Loss function and W is the we
ight vector
        # note that we are performing the partial derivative here

        a = a + data_array_sampled_train[j] * (price_array__sampled_train[j] - (np.dot(
w_temp,data_array_sampled_train[j])- intercept_temp))


        # expression for dL/db ---> here L stands for the Loss function and b is the in
tercept.
        # note that we are performing the partial derivative here

        p = p + (price_array__sampled_train[j] - (np.dot(w_temp,data_array_sampled_trai
n[j]) - intercept_temp))

        # now after running out from this loop we will need to put the values received
 in a and p in our formula for sgd

    W = (w_temp - (learning_rate*(a))*(-2)/k)

    intercept_b = (intercept_temp - (learning_rate*(p))*-2/k)

    y_pred=np.dot(pd.DataFrame(X_train),w_temp.T) + intercept_temp
    print(mean_squared_error(y_train,y_pred),"iteration number = ",i+1)




print("Code Executed")
```

```
603.6001610592499 iteration number =  1
600.7091141322175 iteration number =  2
597.8949416915794 iteration number =  3
595.244360788315 iteration number =  4
592.3247114351765 iteration number =  5
589.5482757987952 iteration number =  6
586.837085222752 iteration number =  7
584.1401610664016 iteration number =  8
581.3348861640975 iteration number =  9
578.5080503643554 iteration number =  10
576.1409215429951 iteration number =  11
573.1611046246285 iteration number =  12
570.4559478049658 iteration number =  13
567.9013537769368 iteration number =  14
565.3649403524621 iteration number =  15
563.2059430846996 iteration number =  16
560.4729394944012 iteration number =  17
557.980619998044 iteration number =  18
555.347672502883 iteration number =  19
552.7774084808702 iteration number =  20
550.2854260728427 iteration number =  21
547.8231851306717 iteration number =  22
545.2575527811075 iteration number =  23
542.8257009776514 iteration number =  24
540.0575715008837 iteration number =  25
537.5174569622599 iteration number =  26
534.98863568459 iteration number =  27
532.6328025737545 iteration number =  28
530.2236479244906 iteration number =  29
527.8797488832054 iteration number =  30
525.5988255717111 iteration number =  31
523.0197182112087 iteration number =  32
520.5150259476808 iteration number =  33
518.9679457941696 iteration number =  34
516.5523574736436 iteration number =  35
514.0229053711772 iteration number =  36
511.7814566725989 iteration number =  37
509.0974752688316 iteration number =  38
506.76353065725925 iteration number =  39
504.1513950257342 iteration number =  40
502.1534971224124 iteration number =  41
499.93329383397736 iteration number =  42
497.6836372203744 iteration number =  43
495.1768828226593 iteration number =  44
492.9679386670418 iteration number =  45
490.5991257762221 iteration number =  46
488.17632579007096 iteration number =  47
486.497165974873 iteration number =  48
484.31061875164835 iteration number =  49
481.9453047856567 iteration number =  50
479.58816681735794 iteration number =  51
477.213055006689 iteration number =  52
474.9249786992541 iteration number =  53
472.63338966678225 iteration number =  54
470.288056217432 iteration number =  55
467.8245695867255 iteration number =  56
465.3427714373352 iteration number =  57
463.05185209409825 iteration number =  58
461.1825211061075 iteration number =  59
458.84501780586214 iteration number =  60
457.019864330761 iteration number =  61
```

```
454.605871681512 iteration number =  62
452.222832745884 iteration number =  63
449.91037437279226 iteration number =  64
447.7305684137721 iteration number =  65
445.3535983533317 iteration number =  66
443.04279461624947 iteration number =  67
441.01720879087765 iteration number =  68
438.78158471033817 iteration number =  69
436.5155821300476 iteration number =  70
434.2862074058741 iteration number =  71
432.1866054623632 iteration number =  72
429.9186964413318 iteration number =  73
427.5823923679939 iteration number =  74
425.32992512663066 iteration number =  75
423.08942338941796 iteration number =  76
420.7817242200979 iteration number =  77
418.54911619851504 iteration number =  78
416.36733016812684 iteration number =  79
414.01082857232035 iteration number =  80
411.78828314261585 iteration number =  81
409.57513289309406 iteration number =  82
407.46856118706194 iteration number =  83
405.16439246360153 iteration number =  84
403.01185539132285 iteration number =  85
400.7233762853219 iteration number =  86
398.51505223405 iteration number =  87
396.21078417499916 iteration number =  88
393.98256627221036 iteration number =  89
391.7426152002545 iteration number =  90
389.3825590804789 iteration number =  91
387.27383218311684 iteration number =  92
385.08021150453857 iteration number =  93
382.95842667831175 iteration number =  94
380.92847450488057 iteration number =  95
378.7794619645359 iteration number =  96
376.61888446201084 iteration number =  97
374.3697768210733 iteration number =  98
372.2362237432884 iteration number =  99
370.08193817402446 iteration number =  100
368.18006319999074 iteration number =  101
365.4141682296724 iteration number =  102
363.6578844047239 iteration number =  103
361.91005217447565 iteration number =  104
359.7818667428714 iteration number =  105
357.7533316534359 iteration number =  106
355.60010898883587 iteration number =  107
353.5567144003025 iteration number =  108
351.62547745206126 iteration number =  109
349.2850469091085 iteration number =  110
347.07832294297583 iteration number =  111
344.9713720441152 iteration number =  112
342.8734494806798 iteration number =  113
340.71940675215876 iteration number =  114
338.7804589483301 iteration number =  115
336.7123699132987 iteration number =  116
334.6362078508337 iteration number =  117
332.56523366519235 iteration number =  118
330.51309944874333 iteration number =  119
328.4904883482994 iteration number =  120
326.4068825237439 iteration number =  121
324.52836787009073 iteration number =  122
```

```
322.4712927481684 iteration number =  123
320.54643656617407 iteration number =  124
318.43612804747875 iteration number =  125
316.4411349674163 iteration number =  126
314.3368013435098 iteration number =  127
312.39373657423414 iteration number =  128
310.2965314146363 iteration number =  129
308.32045459586675 iteration number =  130
306.4203261747033 iteration number =  131
304.4277352457297 iteration number =  132
302.0550849350878 iteration number =  133
300.0889511314946 iteration number =  134
298.07450298993575 iteration number =  135
296.087627909856 iteration number =  136
294.18867963789427 iteration number =  137
292.09374010457424 iteration number =  138
289.94312634338144 iteration number =  139
287.9761707138396 iteration number =  140
286.1233811729458 iteration number =  141
283.97227238583247 iteration number =  142
282.1652579152873 iteration number =  143
280.4870758298104 iteration number =  144
278.5451312431076 iteration number =  145
276.5974860425635 iteration number =  146
274.5355879894933 iteration number =  147
272.599412904597 iteration number =  148
270.53410256026564 iteration number =  149
268.5557279768976 iteration number =  150
266.68950737388695 iteration number =  151
264.8727929047778 iteration number =  152
262.97320650680933 iteration number =  153
260.9376806103733 iteration number =  154
259.0056254683782 iteration number =  155
257.014416505557 iteration number =  156
255.06251960801637 iteration number =  157
253.1333686684946 iteration number =  158
251.24578412241786 iteration number =  159
249.48800133046737 iteration number =  160
247.6418817337134 iteration number =  161
245.6668632607822 iteration number =  162
243.7224560075704 iteration number =  163
241.75395660996904 iteration number =  164
239.89462712973764 iteration number =  165
237.96981719202418 iteration number =  166
236.1058222309487 iteration number =  167
234.2724601716487 iteration number =  168
232.37696146376015 iteration number =  169
230.6014577274139 iteration number =  170
228.80992763339367 iteration number =  171
227.01038663557912 iteration number =  172
225.18921783497012 iteration number =  173
223.42596476371028 iteration number =  174
221.58649012456993 iteration number =  175
219.88801383527053 iteration number =  176
218.0519646213355 iteration number =  177
216.23275023500955 iteration number =  178
214.49932882457833 iteration number =  179
212.8964674461095 iteration number =  180
211.1396309802273 iteration number =  181
209.3213000908755 iteration number =  182
207.5748768239596 iteration number =  183
```

```
205.98249014706184 iteration number =  184
204.30483875811177 iteration number =  185
202.5987865457595 iteration number =  186
200.85536440358905 iteration number =  187
199.09419619536928 iteration number =  188
197.36862173668644 iteration number =  189
195.6677491200074 iteration number =  190
194.22513128176027 iteration number =  191
192.4925150327582 iteration number =  192
190.7406599415242 iteration number =  193
189.05475638256155 iteration number =  194
187.37600588106753 iteration number =  195
185.75670336069706 iteration number =  196
184.17541023868736 iteration number =  197
182.5329335838756 iteration number =  198
180.876664587177 iteration number =  199
179.24132786446947 iteration number =  200
177.59177201987006 iteration number =  201
176.04690369353617 iteration number =  202
174.27879271025324 iteration number =  203
172.65496812622732 iteration number =  204
171.03109846946433 iteration number =  205
169.408854382274 iteration number =  206
167.8114715564885 iteration number =  207
166.15720001051142 iteration number =  208
164.86193184305895 iteration number =  209
163.30502801625858 iteration number =  210
161.6811265993266 iteration number =  211
160.08953063345416 iteration number =  212
158.442718676343 iteration number =  213
156.85989037713097 iteration number =  214
155.31782604478124 iteration number =  215
153.8327750206396 iteration number =  216
152.26160855228102 iteration number =  217
150.80296124370537 iteration number =  218
149.26596838109987 iteration number =  219
147.7670830728325 iteration number =  220
146.2350968007032 iteration number =  221
144.79052712413036 iteration number =  222
143.26886080689889 iteration number =  223
141.94439620670835 iteration number =  224
140.46807986593788 iteration number =  225
138.9920980441294 iteration number =  226
137.59417111824794 iteration number =  227
136.09817846474348 iteration number =  228
134.64476296535992 iteration number =  229
133.19145767360766 iteration number =  230
131.84252482998542 iteration number =  231
130.44631483849537 iteration number =  232
128.98588943105597 iteration number =  233
127.57597035031286 iteration number =  234
126.17622375455375 iteration number =  235
124.73948883243953 iteration number =  236
123.32558228395469 iteration number =  237
121.92640002016384 iteration number =  238
120.58353759357114 iteration number =  239
119.20344077736578 iteration number =  240
117.93028156672652 iteration number =  241
116.64693496281951 iteration number =  242
115.31143939652198 iteration number =  243
113.962125963364 iteration number =  244
```

```
112.63183017277456 iteration number =  245
111.60471015284337 iteration number =  246
110.32271732904694 iteration number =  247
108.98861916376721 iteration number =  248
107.71722662285707 iteration number =  249
106.49889252171634 iteration number =  250
105.24326157565072 iteration number =  251
103.97370618120769 iteration number =  252
102.72041327623688 iteration number =  253
101.4961549335525 iteration number =  254
100.31090522243277 iteration number =  255
99.20006531747751 iteration number =  256
98.00992804613229 iteration number =  257
96.80410932925018 iteration number =  258
95.61282955570013 iteration number =  259
94.52176911148615 iteration number =  260
93.29519081419193 iteration number =  261
92.12847251725057 iteration number =  262
91.05014129628587 iteration number =  263
89.9115310447714 iteration number =  264
88.62860110916749 iteration number =  265
87.51590107741563 iteration number =  266
86.35342976448044 iteration number =  267
85.25082681344868 iteration number =  268
84.15399971073356 iteration number =  269
82.99758336929573 iteration number =  270
82.09978344405899 iteration number =  271
81.04302595466056 iteration number =  272
80.02475009034629 iteration number =  273
78.96062731491962 iteration number =  274
77.91199310935488 iteration number =  275
76.88029326706655 iteration number =  276
75.84195709245681 iteration number =  277
74.89787228344163 iteration number =  278
73.86755283198686 iteration number =  279
72.92220702720303 iteration number =  280
71.9794588317343 iteration number =  281
71.01222213290245 iteration number =  282
70.09547873604075 iteration number =  283
69.14519498612857 iteration number =  284
68.24320562714713 iteration number =  285
67.39076994887995 iteration number =  286
66.4247267113553 iteration number =  287
65.4842291358392 iteration number =  288
64.5982368195941 iteration number =  289
63.51718665305984 iteration number =  290
62.71090662957579 iteration number =  291
61.86384149472624 iteration number =  292
60.98513156980659 iteration number =  293
60.22657201553491 iteration number =  294
59.412903908873616 iteration number =  295
58.63420698761819 iteration number =  296
57.811985350264095 iteration number =  297
57.019851567897014 iteration number =  298
56.52049881699428 iteration number =  299
55.78888156056508 iteration number =  300
55.09670478376379 iteration number =  301
54.33300917710411 iteration number =  302
53.61473009610232 iteration number =  303
52.93988116730454 iteration number =  304
52.18452529161068 iteration number =  305
```

```
51.48429301701234 iteration number =  306
50.84587731051139 iteration number =  307
50.25741010400873 iteration number =  308
49.5782825513834 iteration number =  309
49.00572740235409 iteration number =  310
48.339450975396566 iteration number =  311
47.87193651344102 iteration number =  312
47.192047029097004 iteration number =  313
46.63346890211067 iteration number =  314
46.10278163793352 iteration number =  315
45.515763270057136 iteration number =  316
45.030061137451575 iteration number =  317
44.52370951139097 iteration number =  318
44.21470761695214 iteration number =  319
43.71697683558183 iteration number =  320
43.23441387677681 iteration number =  321
42.781824152982416 iteration number =  322
42.3237409854164 iteration number =  323
41.91091399462295 iteration number =  324
41.50752870133089 iteration number =  325
41.13780203517053 iteration number =  326
40.77501584129707 iteration number =  327
40.357861396767426 iteration number =  328
39.976417150104034 iteration number =  329
39.630400843656666 iteration number =  330
39.372356558100996 iteration number =  331
39.018640719122985 iteration number =  332
38.76819842702773 iteration number =  333
38.489369141111844 iteration number =  334
38.212682608813964 iteration number =  335
37.96605553519017 iteration number =  336
37.734271952856645 iteration number =  337
37.570352745381285 iteration number =  338
37.371202001037695 iteration number =  339
37.22863303826986 iteration number =  340
37.020964107660284 iteration number =  341
36.873926046906355 iteration number =  342
36.706856616215845 iteration number =  343
36.59939434620735 iteration number =  344
36.53685631889381 iteration number =  345
36.398200155051185 iteration number =  346
36.36285590237975 iteration number =  347
36.29343636272431 iteration number =  348
36.265836877884624 iteration number =  349
36.23521868167403 iteration number =  350
36.15459571480167 iteration number =  351
36.168750169587845 iteration number =  352
36.20407896622409 iteration number =  353
36.237660170628025 iteration number =  354
36.3013307116 iteration number =  355
36.316790914830904 iteration number =  356
36.40598306528796 iteration number =  357
36.48427059949022 iteration number =  358
36.60708651662538 iteration number =  359
36.752632644974625 iteration number =  360
Code Executed
```

In [288]:

```
print(W)
```

```
[[-0.53041167 -0.83272991 -1.17259681  0.81550224 -1.28023008  2.07936695
  -0.73125111  0.38831873  0.66980055 -0.35939117 -2.11523758  0.63884163
  -0.91468516]]
```

In [289]:

```
print(intercept_b)
```

```
[23.71705586]
```

## Now we have received the weight vector and the intercept using sgd hence what we need now isto test how it works on our boston house data

In [290]:

```
# let us store all the predicted values using our equation which we received from above
in a list so that we can compare them
# with their actual values

predicted_y_list = []

# note that our prediction will be on our test data which is unseen till now

q = len(X_test)

print(q)
```

```
51
```

In [291]:

```
# hence we will need to run the loop for 152 times

for m in range(q):

    product = np.dot(W,X_test[m]) + intercept_b  # althought intercept_b is a scalar th
en also it will get added with the vector because numpy broadcasting will take place he
re

    # (np.dot(W,X_test[3]) + intercept_b) --- > this statement is giving an output of t
ype "numpy.ndarray" although it gives a single number only as output

    # hence in order to add it into the list we need to convert it into the scaler so t
hat we can subtract it from the actual y later on

    # refer this ---> https://docs.scipy.org/doc/numpy-1.14.1/reference/generated/nump
y.asscalar.html

    predicted_y_list.append(np.asscalar(product))
```

In [292]:

```
print(predicted_y_list[0:5])
```

```
[32.76088849096495, 26.80878437370018, 27.520855154957196, 11.571778332303
523, 33.800880549738174]
```

## Now as we have received the predicted y values and we already have the actual y values from our test dataset hence now we can easily plot a graph between them showing how much they vary.

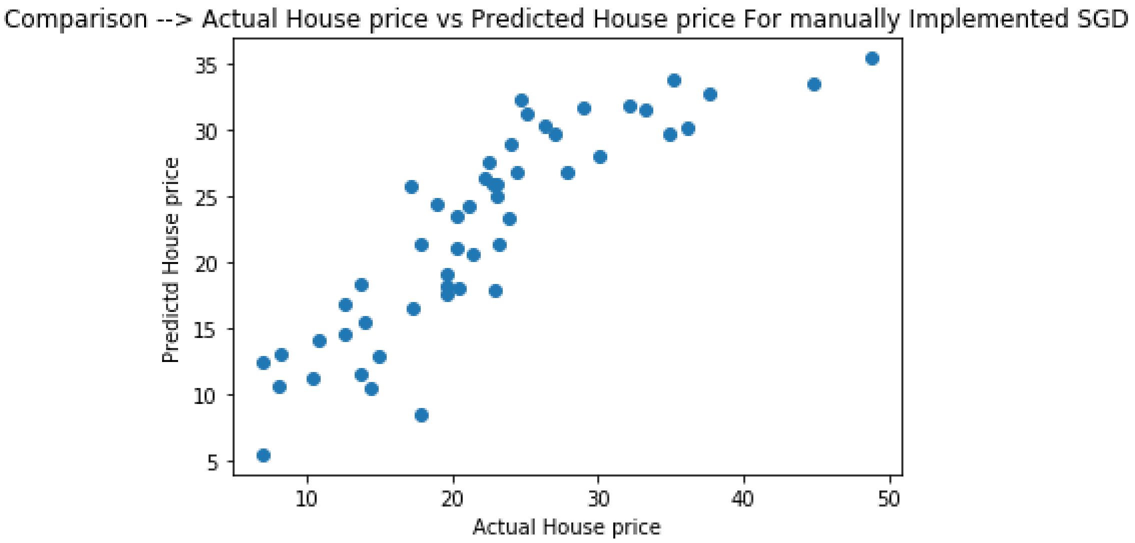
In [293]:

```
# plotting

plt.scatter(y_test,predicted_y_list)
plt.xlabel('Actual House price')
plt.ylabel('Predictd House price')
plt.title('Comparison --> Actual House price vs Predicted House price For manually Impl
emented SGD')
```

Out[293]:

```
Text(0.5, 1.0, 'Comparison --> Actual House price vs Predicted House price
For manually Implemented SGD')
```



## Now our immediate task is to see how far away are our predicted values from the actual values which directly means the erros there are various type of error measures but we will be using Mean Squared Error for this purpose

In [294]:

```
len(y_test)
```

Out[294]:

```
51
```

In [295]:

```
len(predicted_y_list)
```

Out[295]:

```
51
```

In [296]:

```
MSE = mean_squared_error(y_test,predicted_y_list)

print("The mean squared error for above data is ", MSE)
```

```
The mean squared error for above data is  20.104946398385035
```

# Using the inbuilt SGD of sklearn

In [297]:

```
clf = SGDRegressor()
clf.fit(X_train,y_train)
print(mean_squared_error(y_test, clf.predict(X_test)))
```

```
25.114630658659188
```

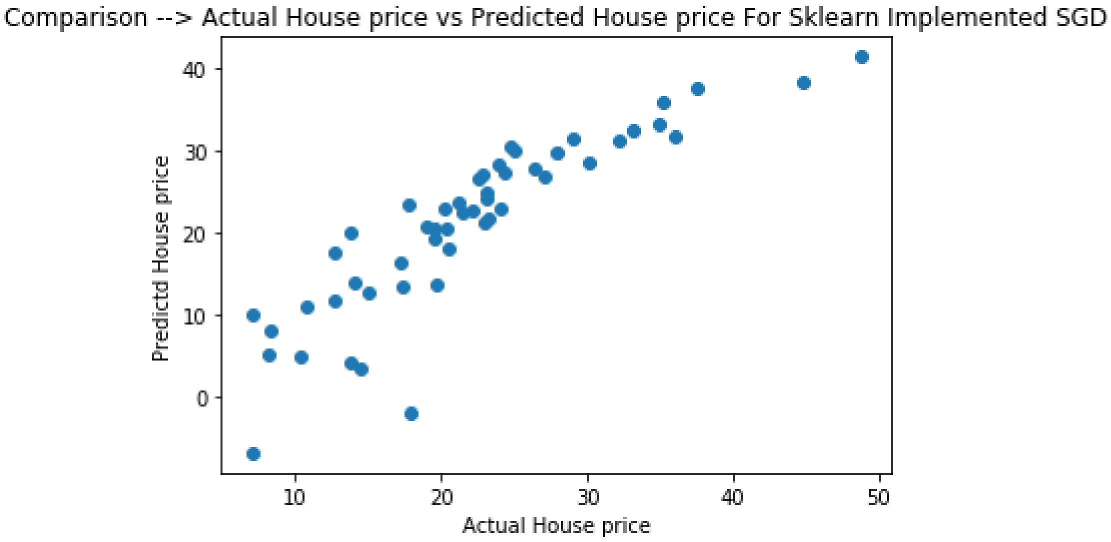
In [298]:

```
# plotting

plt.scatter(y_test,clf.predict(X_test))
plt.xlabel('Actual House price')
plt.ylabel('Predictd House price')
plt.title('Comparison --> Actual House price vs Predicted House price For Sklearn Imple
mented SGD')
```

Out[298]:

```
Text(0.5, 1.0, 'Comparison --> Actual House price vs Predicted House price
For Sklearn Implemented SGD')
```



# We can clearly see that there is a little difference in the mean squared error for the manual implementation of SGD and the sklearn implementation of SGD.

## Comparison of weights of W vector for both of the implementations

In [299]:

```
print("Absolute values of the manually implemented SGD weight vector",np.absolute(W))
```

```
Absolute values of the manually implemented SGD weight vector [[0.53041167
0.83272991 1.17259681 0.81550224 1.28023008 2.07936695
  0.73125111 0.38831873 0.66980055 0.35939117 2.11523758 0.63884163
  0.91468516]]
```

In [300]:

```
print("Absolute values of the Sklearn implemented SGD weight vector",np.absolute(clf.co
ef_))
```

```
Absolute values of the Sklearn implemented SGD weight vector [0.83763698
0.62333182 0.27496414 0.84832365 0.87892639 3.11320544
 0.10881286 2.16342448 0.8927688  0.38953697 1.74881751 1.06201685
 3.70227512]
```

In [301]:

```
len(np.absolute(clf.coef_))
```

Out[301]:

```
13
```

In [302]:

```
W.shape
```

Out[302]:

```
(1, 13)
```

In [303]:

```
np.absolute(W).shape
```

Out[303]:

```
(1, 13)
```

In [304]:

```
np.absolute(clf.coef_).shape
```

Out[304]:

```
(13,)
```

## Tabular representation of the same comparison

In [305]:

```
table = PrettyTable()
table.field_names = ['Manual implementation of SGD', 'Sklearn implementation of SGD']

for i in range(len(np.absolute(clf.coef_))):
    table.add_row([(np.absolute(W)[0][i]),np.absolute(clf.coef_)[i]])
print(table)
```

```
+------------------------------+-------------------------------+
| Manual implementation of SGD | Sklearn implementation of SGD |
+------------------------------+-------------------------------+
|      0.5304116716338811      |       0.8376369788936538      |
|      0.8327299142435407      |       0.6233318182825607      |
|      1.1725968106490288      |      0.27496413651612656      |
|      0.8155022444388706      |       0.8483236512786987      |
|      1.2802300811707545      |       0.878926391095405       |
|       2.07936695433442       |       3.113205439215406       |
|      0.7312511086774909      |      0.10881286487391076      |
|      0.3883187276120403      |       2.1634244789904082      |
|      0.6698005497954693      |       0.8927688047330182      |
|     0.35939117192544817      |       0.3895369705436371      |
|      2.1152375842994187      |       1.7488175139349766      |
|      0.6388416261594938      |       1.0620168495092048      |
|      0.9146851590794788      |       3.702275116102864       |
+------------------------------+-------------------------------+
```

# Summary and steps

## Here in order to find which number of iterations suits the best to our model for getting the minimun loss in case of manual implementation of sgd we printed our MSE value with every iterations and accordingly selected that iteration value to be the best where we found our MSE was converging to a minimum value and after that it started increasing gradually.

#### There is a slight difference between the sklearn implementation MSE and the manual implementation MSE. Sk learn implemenatation gives it out to be 25 and manual gives it out to be 20. I have tried every possible change and this is the best I am getting.

In [ ]: