In [29]:

```
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 sklearn

import seaborn as sns
sns.set_style('whitegrid')
sns.set(rc={'figure.figsize':(9,7)})
```

In [2]:

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

In [3]:

```
scaler = preprocessing.StandardScaler().fit(X)
X = scaler.transform(X)
```

In [4]:

```
sgd = SGDRegressor()
sgd.fit(X, Y)
Y_sklearn_sgd = sgd.predict(X)
print("MSE from Sklearn SGD:", mean_squared_error(Y, sgd.predict(X)))
```

```
MSE from Sklearn SGD: 22.613601327676935
```

Implementation of SGD

In [23]:

```
class SGD:

    def __init__(self):
        self.w = np.zeros(shape=(1, X.shape[1]))
        self.b = 0
        pass

    def cost(self, X, y, w, b):
        inner = np.power(((X @ w.T)+b - y), 2)
        return np.sum(inner) / (2 * len(X))

    def fit(self, X, Y, lr=1e-2, iters=1e3):

        # To store costs for each iteration
        costs = []

        for _ in range(int(iters)):
            w_old = self.w
            b_old = self.b
```

```
            # Gradient of weights
            w_grad = np.zeros(shape=(1, X.shape[1]))

            # Bias of weights
            b_grad = 0

            # For sampling
            sample = np.random.randint(X.shape[0], size=(20))

            # The stochastic part
            for i in sample:

                # to calculate the grad
                y_ = w_old @ X[i].T + b_old

                w_grad += X[i] * (Y[i] - y_)
                b_grad += Y[i] - y_

            # Gradient Descent
            self.w = w_old - (-2/len(sample))*lr*w_grad
            self.b = b_old - (-2/len(sample))*lr*b_grad

        return self.w, self.b

    def predict(self, X):
        y_ = []
        for i in X:
            y_.append(i@self.w.T+self.b)
        return y_
```

In [24]:

```
clf = SGD()
```

In [25]:

```
w, b = clf.fit(X, Y, iters=2000)
```

In [26]:

```
y_manual_sgd = clf.predict(X)
```

In [27]:

```
print("MSE from Manual SGD",mean_squared_error(Y, y_manual_sgd))
```

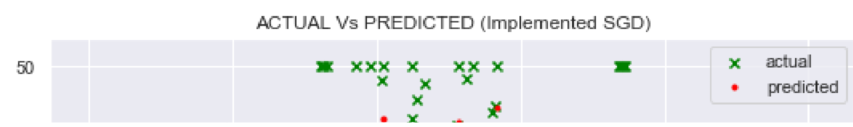
```
MSE from Manual SGD 22.12428053707617
```

Visualizations

Predictions of Implemented SGD

In [30]:

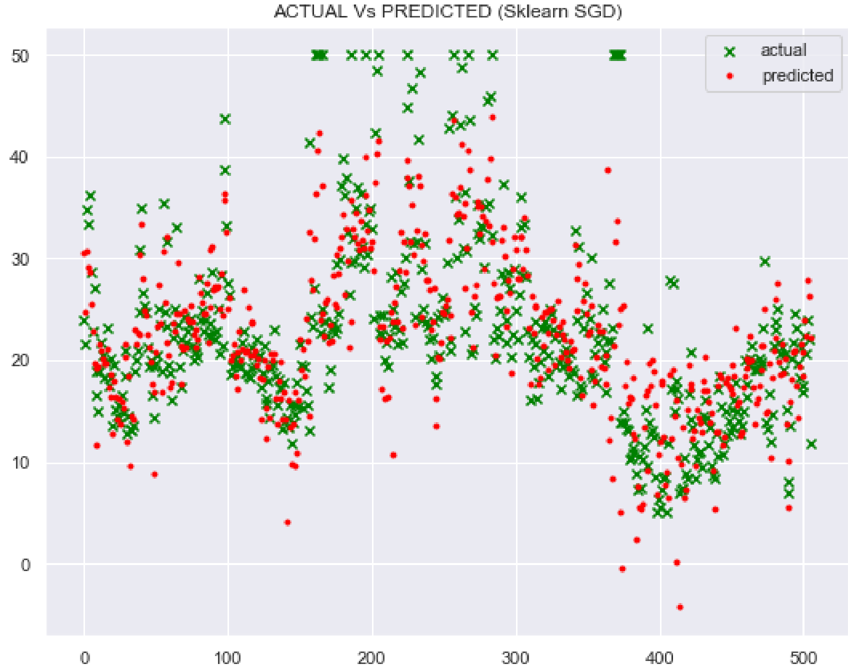
```
plt.scatter([i for i in range(506)], Y, marker="x", c='green')
plt.scatter([i for i in range(506)], y_manual_sgd, marker='.', c='red')
plt.legend(['actual', 'predicted'])
plt.title("ACTUAL Vs PREDICTED (Implemented SGD)")
plt.show()
```



Predictions of Skleran SGD

In [31]:

```
plt.scatter([i for i in range(506)], Y, marker="x", c='green')
plt.scatter([i for i in range(506)], Y_sklearn_sgd , marker='.', c='red')
plt.legend(['actual', 'predicted'])
plt.title("ACTUAL Vs PREDICTED (Sklearn SGD)")
plt.show()
```

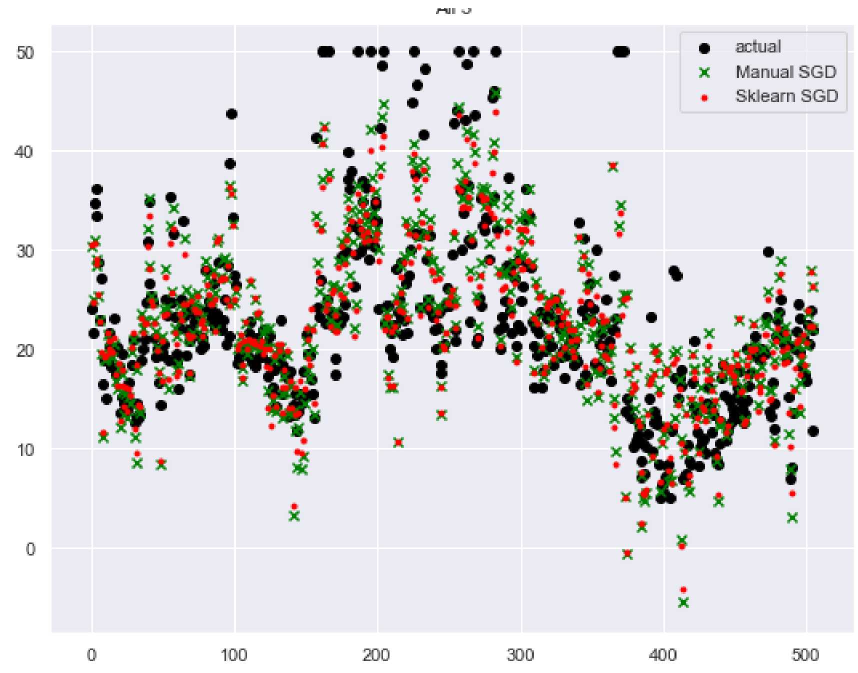


All the 3 Ys

In [32]:

```
plt.scatter([i for i in range(506)], Y, marker="o", c='black')
plt.scatter([i for i in range(506)], y_manual_sgd , marker="x", c='green')
plt.scatter([i for i in range(506)], Y_sklearn_sgd , marker='.', c='red')
plt.legend(['actual', 'Manual SGD', 'Sklearn SGD'])
plt.title("All 3")
plt.show()
```

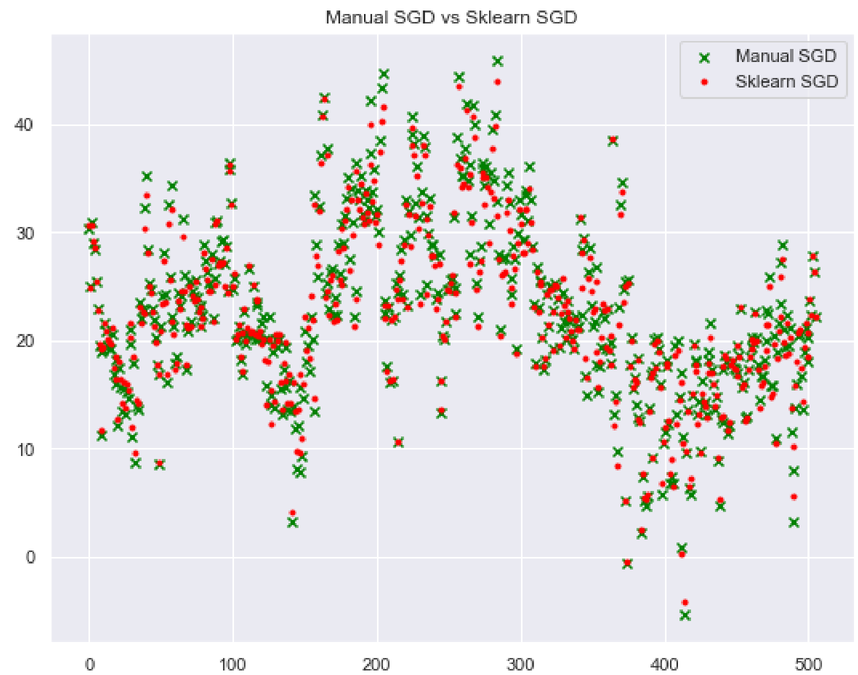
All 3



Predictions of Implemented SGD and Sklearn SGD

In [33]:

```
plt.scatter([i for i in range(506)], y_manual_sgd , marker="x", c='green')
plt.scatter([i for i in range(506)], Y_sklearn_sgd , marker='.', c='red')
plt.legend(['Manual SGD', 'Sklearn SGD'])
plt.title("Manual SGD vs Sklearn SGD")
plt.show()
```



Difference in the weights of Manual SGD and Sklearn SGD

In [34]:

```
results = pd.DataFrame(data=w.reshape(-1,1), columns=['Manual Weights'])
results['Sklean Weights'] = sgd.coef_.reshape(-1,1)
```

In [35]:

```
results
```

Out[35]:

	Manual Weights	Sklean Weights
0	-0.979874	-0.700175
1	1.264071	0.728309
2	0.134774	-0.443963
3	0.623677	0.766338
4	-1.916676	-1.084585
5	2.980117	3.065277
6	-0.051326	-0.128434
7	-2.953148	-2.331470
8	2.560922	0.925395
9	-1.970132	-0.540547
10	-2.014415	-1.809878
11	0.793275	0.851660
12	-3.791334	-3.527531

In [ ]:

```

```