PROJECT 4: UNSUPERVISED LEARNING

In this project, we analyse the Boston housing dataset. The goal is *dimensionality reduction*. We will try to reduce the number of feature columns to a representative set of smaller size.

Data description

The data set has 506 rows and 14 columns:

In [3]:
```
print(df.shape)
df.head()
```

```
(506, 14)
```

Out[3]:

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	b	lstat	medv
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2

For details of the variables and the initial exploration, see Project 2.

MEDV is the median value of houses (in thousands). It is usually the target for predictions. In this project, we will examine the possibility of predicting MEDV with a smaller number of features.

Baseline performance

Here we do a straightforward linear regression on all the features to establish a baseline. We used a train-test split with 20% test size.

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

X_train, X_test, y_train, y_test = train_test_split(Xall, y, test_size=0.2)
lr = LinearRegression()
lr.fit(X_train,y_train)
print('MSE: {}'.format(mean_squared_error(y_test,lr.predict(X_test))))
```

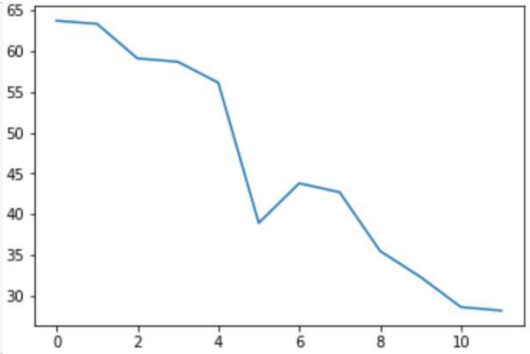
```
MSE: 26.25836813474162
```

Our goal after dimensionality reduction will be to try and have a feature sets that allows for similar (if not better) performance.

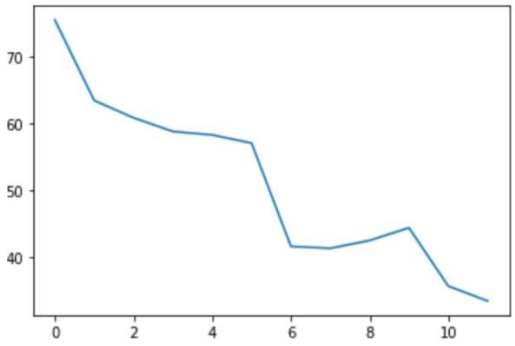
Unsupervised models

We used two dimensionality reduction techniques to reduce the feature set: PCA and NMF. For each technique, we tried reducing the feature set to n features, for n from 1 to 12. Each time, we trained a basic linear regression model on the resulting feature set and assessed its MSE. First, PCA:

```
from sklearn.decomposition import PCA

errorsP = []

for i in range(1,13):
    pca = PCA(n_components = i)
    pca.fit(X_train)
    X1 = pca.transform(X_train)

    lr = LinearRegression()
    lr.fit(X1,y_train)

    X2 = pca.transform(X_test)
    errorsP.append(mean_squared_error(y_test,lr.predict(X2)))

plt.plot(errorsP)
```



Here are the results for NMF:



Comparison

We compare the dimensionality reduction techniques for the various numbers of parameters

```
results = pd.DataFrame({'PCA': errorsP, 'NMF': errorsN}, index = list(range(1,13)))
results
```

	PCA	NMF
1	63.712254	75.430636
2	63.334535	63.423244
3	59.103642	60.842682
4	58.690266	58.797003
5	56.125337	58.288509
6	38.950594	57.065089
7	43.816914	41.667520
8	42.720706	41.379200
9	35.532769	42.541446
10	32.346934	44.440038
11	28.662952	35.749540
12	28.227046	33.527491

Some observations

1. Roughly, the MSE is decreasing as number of parameters increase. This is unsurprising since prediction may be expected to improve when more predictors are considered.
2. For high and low numbers of parameters, PCA does better than NMF. The reverse is true for 7 or 8 components, but there the difference is small. On the whole we'd say that PCA does better.
3. We saw some points at which the MSE increases with more parameters. This happened for PCA at 6 components, and NMF at 8 components.
4. At high number of components, PCA comes closer to the baseline than does NMF.