Chapter 5

Question 1

Chapter 5 Question 6

We continue to consider the use of a logistic regression model to predict the probability of default using income and balance on the Default data set. In particular, we will now compute estimates for the standard errors of the income and balance logistic regression coefficients in two different ways: (1) using the bootstrap, and (2) using the standard formula for computing the standard errors in the glm() function. Do not forget to set a random seed before beginning your analysis.

Part a

Using the summary() and glm() functions, determine the esti mated standard errors for the coefficients associated with income and balance in a multiple logistic regression model that uses both predictors.

In [1]:
```
import pandas as pd
import statsmodels.api as sm
import statsmodels.formula.api as smf
import numpy as np

import warnings
warnings.filterwarnings('ignore')
```

```
C:\Users\saiom\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:7: FutureWarning: pandas.Int64Index is deprecated and
will be removed from pandas in a future version. Use pandas.Index with the appropriate dtype instead.
  from pandas import (to_datetime, Int64Index, DatetimeIndex, Period,
C:\Users\saiom\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:7: FutureWarning: pandas.Float64Index is deprecated a
nd will be removed from pandas in a future version. Use pandas.Index with the appropriate dtype instead.
  from pandas import (to_datetime, Int64Index, DatetimeIndex, Period,
```

In [2]:
```
default_data = pd.read_csv('PS3_Data-Default.csv')

display(default_data.head())
print(default_data.shape, '\n')
default_data.value_counts('default')
```

	default	student	balance	income
0	No	No	729.526495	44361.625074
1	No	Yes	817.180407	12106.134700
2	No	No	1073.549164	31767.138947
3	No	No	529.250605	35704.493935
4	No	No	785.655883	38463.495879

```
(10000, 4)
```

Out[2]:
```
default
No     9667
Yes     333
dtype: int64
```

In [3]:
```
default_data.head()
```

Out[3]:

	default	student	balance	income
0	No	No	729.526495	44361.625074
1	No	Yes	817.180407	12106.134700
2	No	No	1073.549164	31767.138947
3	No	No	529.250605	35704.493935
4	No	No	785.655883	38463.495879

In [4]:
```
encoding_dict = {'Yes': 1,'No': 0}
for col in ['default', 'student']:
    default_data[col] = default_data[col].map(encoding_dict)

default_data.head()
```

Out[4]:

	default	student	balance	income
0	0	0	729.526495	44361.625074

	default	student	balance	income
1	0	1	817.180407	12106.134700
2	0	0	1073.549164	31767.138947
3	0	0	529.250605	35704.493935
4	0	0	785.655883	38463.495879

In [5]:

```
X = default_data[['balance', 'income']]
X = sm.add_constant(X)

y = default_data['default']

display(X.head(), y.head())

results = sm.Logit(y, X).fit()
print(results.summary())
```

	const	balance	income
0	1.0	729.526495	44361.625074
1	1.0	817.180407	12106.134700
2	1.0	1073.549164	31767.138947
3	1.0	529.250605	35704.493935
4	1.0	785.655883	38463.495879

```
0    0
1    0
2    0
3    0
4    0
Name: default, dtype: int64
Optimization terminated successfully.
         Current function value: 0.078948
         Iterations 10
                           Logit Regression Results
==============================================================================
Dep. Variable:                default   No. Observations:                10000
```

```
Model:                          Logit   Df Residuals:                     9997
Method:                           MLE   Df Model:                            2
Date:                Wed, 15 Feb 2023   Pseudo R-squ.:                  0.4594
Time:                        20:39:47   Log-Likelihood:                -789.48
converged:                       True   LL-Null:                       -1460.3
Covariance Type:            nonrobust   LLR p-value:                4.541e-292
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const        -11.5405      0.435    -26.544      0.000     -12.393     -10.688
balance        0.0056      0.000     24.835      0.000       0.005       0.006
income      2.081e-05   4.99e-06      4.174      0.000     1.1e-05    3.06e-05
==============================================================================

Possibly complete quasi-separation: A fraction 0.14 of observations can be
perfectly predicted. This might indicate that there is complete
quasi-separation. In this case some parameters will not be identified.
```

From the regression results, we can observe that the standard erroe is 0.000 for the variable balance and 4.99e-06 for the income variable.

Part b

(b) Write a function, boot.fn(), that takes as input the Default data set as well as an index of the observations, and that outputs the coefficient estimates for income and balance in the multiple logistic regression model

In [6]:

```python
def get_indices(data, n):

    assert type(data) == pd.DataFrame
    assert type(n) == int, 'n must be an integer'

    indices = np.random.choice(
        data.index,         # Indices as the input
        int(n),             # Number of indices per sample
        replace=True        # Draw samples with replacement
    )
    return indices

help(get_indices)
get_indices(default_data, 5)
```

```
Help on function get_indices in module __main__:
```

```
get_indices(data, n)
```

Out[6]:
```
array([6280,  537, 8176, 6066, 2878], dtype=int64)
```

In [7]:
```
np.asarray(np.unique(get_indices(default_data, 10000), return_counts=True)).T[-10:]
```

Out[7]:
```
array([[9985,    1],
       [9986,    2],
       [9989,    2],
       [9990,    2],
       [9991,    2],
       [9992,    2],
       [9993,    1],
       [9994,    1],
       [9997,    3],
       [9999,    5]], dtype=int64)
```

In [30]:
```
def boot_fn(data, index, constant=True, features=['balance','income'], target='default'):

    X = data[features].loc[index]
    if constant:
        X = sm.add_constant(X)
    y = data[target].loc[index]

    lr = sm.Logit(y,X).fit(disp=0)
    coefficients = [lr.params[0], lr.params[1], lr.params[2], lr.bse[0], lr.bse[1], lr.bse[2]]
    #sds = [lr.bse[0], lr.bse[1], lr.bse[2]]
    return coefficients

intercept, coef_balance, coef_income, sd_intercept, sd_coef_balance, sd_coef_income = boot_fn(default_data, get_indices(default_da
#sd_intercept, sd_coef_balance, sd_coef_income = boot_fn(default_data, get_indices(default_data, 10000))

print(f'Coefficients of a single subsample:\n\tIntercept:\t{intercept}\n\tBalance:\t{round(coef_balance,4)}\n\tIncome:\t\t{coef_in

print(f'Sds of a single subsample:\n\tIntercept:\t{sd_intercept}\n\tBalance:\t{round(sd_coef_balance,4)}\n\tIncome:\t\t{sd_coef_in
```

```
Coefficients of a single subsample:
        Intercept:      -11.635379501169187
        Balance:        0.0057
        Income:         2.4257075190162356e-05
Sds of a single subsample:
```

```
Intercept:      0.43240179793679373
Balance:        0.0002
Income:         4.93672790015799e-06
```

Coefficients of a single subsample

```
Intercept:  -11.667098131357776

Balance:    0.0055

Income:     2.079794928333737e-05
```

Sds of a single subsample

```
Intercept:  0.4231851158303451

Balance:    0.0002

Income:     4.915106410001708e-06
```

Part c

(c) Use the boot() function together with your boot.fn() function to estimate the standard errors of the logistic regression coefficients for income and balance.

In [9]:

```
def boot(data, func, B):
    coef_intercept = []
    coef_balance = []
    coef_income = []

    coefs = ['intercept', 'balance', 'income']
    output = {coef: [] for coef in coefs}
    for i in range(B):
        reg_out = func(data, get_indices(data, len(data)))
        for i, coef in enumerate(coefs):
            output[coef].append(reg_out[i])

    results = {}
    for coef in coefs:
```

```
        results[coef] = {
            'estimate': round(np.mean(output[coef]),4),
            'std_err': np.std(output[coef])
        }

    return results
```

In [10]:

```
results = boot(default_data, boot_fn, 1000)
for i, x in results.items():
    print(f"{i.capitalize()}:\n\tEstimate: {x['estimate']}\n\tStandard Error: {x['std_err']}")
```

```
Intercept:
        Estimate: -11.5871
        Standard Error: 0.43915594454133955
Balance:
        Estimate: 0.0057
        Standard Error: 0.00023078828720352152
Income:
        Estimate: 0.0
        Standard Error: 4.721997155091395e-06
```

Intercept: Estimate: -11.5471 Standard Error: 0.43915594454133955

Balance: Estimate: 0.0057 Standard Error: 0.00023078828720352152

Income: Estimate: 0.0 Standard Error: 4.721997155091395e-06

Part d

(d) Comment on the estimated standard errors obtained using the glm() function and using your bootstrap function

From part b

Sds of a single subsample

```
Intercept:  0.4231851158303451

Balance:    0.0002

Income:     4.915106410001708e-06
```

From part c

```
Intercept:
    Standard Error: 0.43915594454133955

Balance:
    Standard Error: 0.00023078828720352152

Income:
    Standard Error:  4.721997155091395e-06
```

We observe that in both the methods using Logit and using bootstrap function the standard errors are similar for Balance and Income varibles. For Balance variable in Logit method we get a standard error of 0.0002 and 0.00023 in bootstrap method. Similarly, for income variable, we observe a 4.92e-06 in Logit method and 4.71e-06 in bootstrap method.

Question 2

Chapter 5 Question 8

Part a

(a) Generate a simulated data set as follows

set.seed (1) x <- rnorm (100) y <- x - 2 * x^2 + rnorm (100)

In this data set, what is n and what is p? Write out the model used to generate the data in equation form.

In [11]:
```
np.random.seed(1)

y = np.random.normal(size=100)
x = np.random.normal(size=100)

e = np.random.normal(size=100)

y = x - 2 * x**2 + e
```

n = number of observations = 100

p = number of predictors = 2 (X and X^2)

Model in equation form:

$Y = X - 2(X^2) + \epsilon$

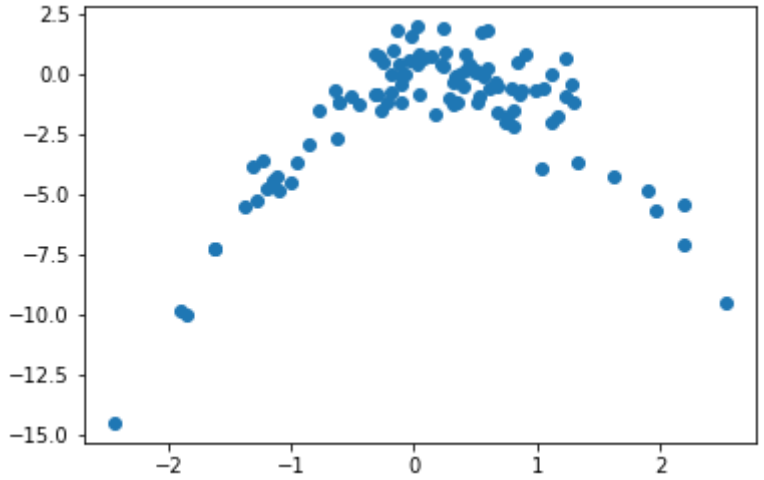
Part b

Create a scatterplot of X against Y . Comment on what you find

In [12]:
```
import matplotlib.pyplot as plt

plt.scatter(x, y)
```

Out[12]:
```
<matplotlib.collections.PathCollection at 0x153a8f8b340>
```



From the scatter, we can say that the plot looks like a reverse U shaped quadratic plot. It is convex in nature with a negative concavity to the plot. This is similar to what we have defined the relation as, a quadratic relation between y and x. The maximum value of y that the plot reaches is close to 0.125,x here is approaximately 0.25. This can also be observed by finding the first order derivative of the equation we have. From the figures, X seems to be having a range of approximately -2.5 to 2.5. We can use the ranges of x to find the minimum value of y that approaches to -15 when x approaches its minimum.While what we observe from scatter is approaximates, we can verify the values by using range function on x and y.

Part c

(c) Set a random seed, and then compute the LOOCV errors that result from fitting the following four models using least squares

i. $Y = \beta0 + \beta1X + \epsilon$

ii. $Y = \beta0 + \beta1X + \beta2X2 + \epsilon$

iii. $Y = \beta0 + \beta1X + \beta2X2 + \beta3X3 + \epsilon$

iv. $Y = \beta0 + \beta1X + \beta2X2 + \beta3X3 + \beta4X4 + \epsilon.$

Note you may find it helpful to use the data.frame() function to create a single data set containing both X and Y .

In [13]:
```
from sklearn.model_selection import LeaveOneOut
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline
```

```
from sklearn.metrics import mean_squared_error
import statsmodels.api as sm


np.random.seed(1)
loo = LeaveOneOut()
lookv_data = pd.DataFrame({'x':x, 'y':y})


scores = []

for i in range(1, 5):
    for train, test in loo.split(lookv_data):
        X_train = lookv_data['x'][train]
        y_train = lookv_data['y'][train]
        X_test = lookv_data['x'][test]
        y_test = lookv_data['y'][test]
        model = Pipeline([('poly', PolynomialFeatures(degree = i)),
                          ('linear', LinearRegression())])
        model.fit(X_train[:,np.newaxis], y_train)
        score = mean_squared_error(y_test, model.predict(X_test[:,np.newaxis]))
        scores.append(score)
    print('MSE')
    print('%i: %f' % (i,np.mean(scores)))
    scores = []
```

```
MSE
1: 8.292212
MSE
2: 1.017096
MSE
3: 1.046553
MSE
4: 1.057493
```

MSE 1: 8.292212

MSE 2: 1.017096

MSE 3: 1.046553

MSE 4: 1.057493

Part d

(d) Repeat (c) using another random seed, and report your results. Are your results the same as what you got in (c)? Why

In [14]:

```
np.random.seed(2)

loo = LeaveOneOut()
lookv_data = pd.DataFrame({'x':x, 'y':y})


scores = []

for i in range(1, 5):
    for train, test in loo.split(lookv_data):
        X_train = lookv_data['x'][train]
        y_train = lookv_data['y'][train]
        X_test = lookv_data['x'][test]
        y_test = lookv_data['y'][test]
        model = Pipeline([('poly', PolynomialFeatures(degree = i)),
                          ('linear', LinearRegression())])
        model.fit(X_train[:,np.newaxis], y_train)
        score = mean_squared_error(y_test, model.predict(X_test[:,np.newaxis]))
        scores.append(score)
    print('MSE')
    print('%i: %f' % (i,np.mean(scores)))
    scores = []
```

```
MSE
1: 8.292212
MSE
2: 1.017096
MSE
3: 1.046553
MSE
4: 1.057493
```

MSE 1: 8.292212

MSE 2: 1.017096

MSE 3: 1.046553

MSE 4: 1.057493

We have also observed that the results of LOOCV stayed the same even after changing the random seed to 2. The results are exactly the same. We also observe that there is no random effect resulting from the observations used in the test set.

Part e

Which of the models in (c) had the smallest LOOCV error? Is this what you expected? Explain your answer.

We observe that model 2 has the smallest LOOCV error with and MSE value of 1.017. This is because model 2 represents the quadratic polynomial of second degree, i.e same as the polynomial relation we setup while generating x and y using random normal functions. This is what is expected because the model has same polynomial form of relation between x and y. However, we cannot say whether the results will stay the same if we were to use a random seed of 0 or a random seed of higher value. This is because the random seed value can affect the epsilons generated and thus can affect the model.

Part f

(f) Comment on the statistical significance of the coefficient esti mates that results from fitting each of the models in (c) using least squares. Do these results agree with the conclusions drawn based on the cross-validation results?

In [15]:
```
for i in range(1, 5):
    pol = PolynomialFeatures(degree = i)
    X = pol.fit_transform(lookv_data['x'][:,np.newaxis])
    y = lookv_data['y']
    model = sm.OLS(y, X)
    results = model.fit()
    print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.088
Model:                            OLS   Adj. R-squared:                  0.079
Method:                 Least Squares   F-statistic:                     9.460
Date:                Wed, 15 Feb 2023   Prob (F-statistic):            0.00272
Time:                        20:40:24   Log-Likelihood:                -242.69
No. Observations:                 100   AIC:                             489.4
Df Residuals:                      98   BIC:                             494.6
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -1.7609      0.280     -6.278      0.000      -2.317      -1.204
```

```
x1             0.9134      0.297      3.076      0.003       0.324       1.503
==============================================================================
Omnibus:                       40.887   Durbin-Watson:                   1.957
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               83.786
Skew:                          -1.645   Prob(JB):                     6.40e-19
Kurtosis:                       6.048   Cond. No.                         1.19
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.882
Model:                            OLS   Adj. R-squared:                  0.880
Method:                 Least Squares   F-statistic:                     362.9
Date:                Wed, 15 Feb 2023   Prob (F-statistic):           9.26e-46
Time:                        20:40:24   Log-Likelihood:                -140.40
No. Observations:                 100   AIC:                             286.8
Df Residuals:                      97   BIC:                             294.6
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.0216      0.122     -0.177      0.860      -0.264       0.221
x1             1.2132      0.108     11.238      0.000       0.999       1.428
x2            -2.0014      0.078    -25.561      0.000      -2.157      -1.846
==============================================================================
Omnibus:                        0.094   Durbin-Watson:                   2.221
Prob(Omnibus):                  0.954   Jarque-Bera (JB):                0.009
Skew:                          -0.022   Prob(JB):                        0.995
Kurtosis:                       2.987   Cond. No.                         2.26
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.883
Model:                            OLS   Adj. R-squared:                  0.880
Method:                 Least Squares   F-statistic:                     242.1
Date:                Wed, 15 Feb 2023   Prob (F-statistic):           1.26e-44
Time:                        20:40:24   Log-Likelihood:                -139.91
No. Observations:                 100   AIC:                             287.8
```

```
Df Residuals:                       96   BIC:                             298.2
Df Model:                            3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0046      0.125      0.037      0.971      -0.244       0.253
x1             1.0639      0.189      5.636      0.000       0.689       1.439
x2            -2.0215      0.081    -24.938      0.000      -2.182      -1.861
x3             0.0550      0.057      0.965      0.337      -0.058       0.168
==============================================================================
Omnibus:                        0.034   Durbin-Watson:                   2.253
Prob(Omnibus):                  0.983   Jarque-Bera (JB):                0.050
Skew:                           0.032   Prob(JB):                        0.975
Kurtosis:                       2.911   Cond. No.                         6.55
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.885
Model:                            OLS   Adj. R-squared:                  0.880
Method:                 Least Squares   F-statistic:                     182.4
Date:                Wed, 15 Feb 2023   Prob (F-statistic):           1.13e-43
Time:                        20:40:24   Log-Likelihood:                -139.24
No. Observations:                 100   AIC:                             288.5
Df Residuals:                      95   BIC:                             301.5
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0866      0.144      0.600      0.550      -0.200       0.373
x1             1.0834      0.189      5.724      0.000       0.708       1.459
x2            -2.2455      0.214    -10.505      0.000      -2.670      -1.821
x3             0.0436      0.058      0.755      0.452      -0.071       0.158
x4             0.0482      0.043      1.132      0.260      -0.036       0.133
==============================================================================
Omnibus:                        0.102   Durbin-Watson:                   2.214
Prob(Omnibus):                  0.950   Jarque-Bera (JB):                0.117
Skew:                           0.069   Prob(JB):                        0.943
Kurtosis:                       2.906   Cond. No.                         17.5
==============================================================================
```

```
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

When we have a first order polynomial relation, we have x1 coeff of -0.9134. The t statistic is 0.297. The p value of 0.003 says the estimate is statistically significant.

When we have the second order polynomial relation, we have x1 coeff of 1.21 and x2 coeff of -2.0014. The t values respectively are 11.238 and -25.561. both the coefficient estimates for x1 and x2 have p-values of 0.000, which are less than 0.05, indicating that they are statistically significant.

When we have a thrid order polynomial relation, the coefficients observed are 1.0639, -2.0215 and 0.055 respectively for x1, x2 and x3. The t statistics are 5.636, -24.938 and 0.965 for x1 , x2 and x3 respectively. The coefficient estimates for x1 and x2 have p-values of 0.000, which are less than 0.05, indicating that they are statistically significant. The coefficient estimate for x3 has a p-value of 0.337, which is greater than 0.05, indicating that it is not statistically significant.

When we have a fourth order polynomial relation, the coefficients observed are 1.0834, -2.2455, 0.0436 and 0.0482 respectively for x1, x2, x3 and x4. The t statistics are 5.724, -10.505, 0.755 and 1,132 respectively. Similar trend of coefficients of x1 and x2 being statistically significant and coefficients of x3 and x4 not being statistically significant is observed.

Thus we can say that x2 and x1 are the variables with relevance compared to the other variables x3 and x4. These results agree to conclusions drawn based on the cross validation results, we can say that first and second order terms are the most significant among the variables.

Chapter 6

Question 3

Chapter 6 Question 11

Part a

We will now try to predict per capita crime rate in the Boston dataset

(a) Try out some of the regression methods explored in this chapter, such as best subset selection, the lasso, ridge regression, and PCR. Present and discuss results for the approaches that you consider

In [16]:
```
boston_data = pd.read_csv('ProblemSet3_Boston.csv')
```

```
display(boston_data.head())
print(boston_data.shape, '\n')
#boston_data.value_counts('default')
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MDEV
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	4.98	24.0
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	9.14	21.6
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	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

```
(506, 14)
```

In [17]:
```
boston_data.columns
```

Out[17]:
```
Index(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX',
       'PTRATIO', 'B', 'LSTAT', 'MDEV'],
      dtype='object')
```

In [18]:
```
from itertools import combinations
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression,LinearRegression, Lasso
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import cross_val_score, cross_validate
```

Forward Stepwise Selection

In [19]:
```
# Add constant to dataframe
boston_data['constant'] = 1

# Specify target
Y = boston_data['CRIM']

# Variables to use in forward propagation
vars_left_add = ['ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX','PTRATIO', 'B', 'LSTAT', 'MDEV']
```

```
# Regression type
ols = LinearRegression()

# Starting variables (constant initially)
current_vars = ['constant']

X = boston_data[current_vars]
benchmark_error = np.mean(-1*cross_val_score(ols, X, Y, cv = 5, scoring = 'neg_mean_squared_error'))
print(' Initial run with only one var (constant term/only bias weight):', current_vars)
print('      Benchmark error:', benchmark_error)
print('')


for iter in range(len(vars_left_add)):
    print('\033[1m'+ 'Iteration:', iter, '\033[0m')
    error_list = []
    for var in vars_left_add:
        X = boston_data[current_vars + [var]]
        error = np.mean(-1*cross_val_score(ols, X, Y, cv = 5, scoring = 'neg_mean_squared_error'))
        error_list.append(error)
        print(' Running model with:', current_vars + [var])
        print('      Error:', error)

    # Chose the smallest error
    min_error = min(error_list)
    chosen_col_index = error_list.index(min_error)

    # If our current smallest error is smaller than our previous error, than we add a variable
    if min_error < benchmark_error:
        print('          *** Variable selected:', vars_left_add[chosen_col_index])
        print('          *** Min error selected:', min_error)
        print('          *** Chose the variable that generated the min error + was lower than previous error')
        print('')
        # Add the variable that produced the smallest error to current_vars
        current_vars.append(vars_left_add[chosen_col_index])
        # Delete chosen variable from vars_left_add
        del vars_left_add[chosen_col_index]
        # Update benchmark_error
        benchmark_error = min_error

    # Otherwise, we stop our model
    else:
        print('          \033[4m*** No variable was selected', '\033[0m')
```

```
        print('          *** Previous error rate (', benchmark_error,') is lower than smallest error rate of this iteration (', mi
        print('          *** Break')
        break

print('')
print('Variables chosen for our model', current_vars)
```

```
 Initial run with only one var (constant term/only bias weight): ['constant']
      Benchmark error: 82.92758188752097

Iteration: 0
 Running model with: ['constant', 'ZN']
      Error: 80.27397805562121
 Running model with: ['constant', 'INDUS']
      Error: 74.7889065692895
 Running model with: ['constant', 'CHAS']
      Error: 83.41765212625346
 Running model with: ['constant', 'NOX']
      Error: 72.93199505968977
 Running model with: ['constant', 'RM']
      Error: 79.28682676023121
 Running model with: ['constant', 'AGE']
      Error: 76.67580886003633
 Running model with: ['constant', 'DIS']
      Error: 74.28733132761663
 Running model with: ['constant', 'RAD']
      Error: 45.82003611707332
 Running model with: ['constant', 'TAX']
      Error: 49.99382062232205
 Running model with: ['constant', 'PTRATIO']
      Error: 77.67773955150695
 Running model with: ['constant', 'B']
      Error: 76.79597319366995
 Running model with: ['constant', 'LSTAT']
      Error: 67.79590067769361
 Running model with: ['constant', 'MDEV']
      Error: 73.36961819142553
          *** Variable selected: RAD
          *** Min error selected: 45.82003611707332
          *** Chose the variable that generated the min error + was lower than previous error

Iteration: 1
 Running model with: ['constant', 'RAD', 'ZN']
      Error: 45.82198169501373
```

```
 Running model with: ['constant', 'RAD', 'INDUS']
      Error: 45.66935547677909
 Running model with: ['constant', 'RAD', 'CHAS']
      Error: 46.324779549788914
 Running model with: ['constant', 'RAD', 'NOX']
      Error: 45.58118857348601
 Running model with: ['constant', 'RAD', 'RM']
      Error: 45.76346608013206
 Running model with: ['constant', 'RAD', 'AGE']
      Error: 45.637573238661254
 Running model with: ['constant', 'RAD', 'DIS']
      Error: 45.51141190783985
 Running model with: ['constant', 'RAD', 'TAX']
      Error: 45.814201876450944
 Running model with: ['constant', 'RAD', 'PTRATIO']
      Error: 45.95542344588003
 Running model with: ['constant', 'RAD', 'B']
      Error: 47.09155124840813
 Running model with: ['constant', 'RAD', 'LSTAT']
      Error: 44.61848968213957
 Running model with: ['constant', 'RAD', 'MDEV']
      Error: 45.50653025939223
          *** Variable selected: LSTAT
          *** Min error selected: 44.61848968213957
          *** Chose the variable that generated the min error + was lower than previous error

Iteration: 2
 Running model with: ['constant', 'RAD', 'LSTAT', 'ZN']
      Error: 44.46585900848578
 Running model with: ['constant', 'RAD', 'LSTAT', 'INDUS']
      Error: 44.74615589188474
 Running model with: ['constant', 'RAD', 'LSTAT', 'CHAS']
      Error: 44.83961299896133
 Running model with: ['constant', 'RAD', 'LSTAT', 'NOX']
      Error: 44.8044669896804
 Running model with: ['constant', 'RAD', 'LSTAT', 'RM']
      Error: 45.57611123204303
 Running model with: ['constant', 'RAD', 'LSTAT', 'AGE']
      Error: 44.853361470969375
 Running model with: ['constant', 'RAD', 'LSTAT', 'DIS']
      Error: 44.89758208685341
 Running model with: ['constant', 'RAD', 'LSTAT', 'TAX']
      Error: 44.7824272582582
 Running model with: ['constant', 'RAD', 'LSTAT', 'PTRATIO']
```

```
      Error: 44.587894214563384
 Running model with: ['constant', 'RAD', 'LSTAT', 'B']
      Error: 46.37189489580936
 Running model with: ['constant', 'RAD', 'LSTAT', 'MDEV']
      Error: 44.94792059501778
          *** Variable selected: ZN
          *** Min error selected: 44.46585900848578
          *** Chose the variable that generated the min error + was lower than previous error

Iteration: 3
 Running model with: ['constant', 'RAD', 'LSTAT', 'ZN', 'INDUS']
      Error: 44.60690146833299
 Running model with: ['constant', 'RAD', 'LSTAT', 'ZN', 'CHAS']
      Error: 44.674547759255645
 Running model with: ['constant', 'RAD', 'LSTAT', 'ZN', 'NOX']
      Error: 44.664805819793514
 Running model with: ['constant', 'RAD', 'LSTAT', 'ZN', 'RM']
      Error: 45.44166482320341
 Running model with: ['constant', 'RAD', 'LSTAT', 'ZN', 'AGE']
      Error: 44.88114697040826
 Running model with: ['constant', 'RAD', 'LSTAT', 'ZN', 'DIS']
      Error: 44.64818414734465
 Running model with: ['constant', 'RAD', 'LSTAT', 'ZN', 'TAX']
      Error: 44.62155410125715
 Running model with: ['constant', 'RAD', 'LSTAT', 'ZN', 'PTRATIO']
      Error: 44.50780000555501
 Running model with: ['constant', 'RAD', 'LSTAT', 'ZN', 'B']
      Error: 46.111739982093845
 Running model with: ['constant', 'RAD', 'LSTAT', 'ZN', 'MDEV']
      Error: 44.74039481076157
          *** No variable was selected
          *** Previous error rate ( 44.46585900848578 ) is lower than smallest error rate of this iteration ( 44.50780000555501 )
          *** Break

 Variables chosen for our model ['constant', 'RAD', 'LSTAT', 'ZN']
```

We see that forward stepwise alongwith 5 Fold Cross Validation selection chooses the model with a 'constant', 'RAD', 'LSTAT', 'ZN' as the features while excluding others.

FSS AIC

In [20]:

```
# Add constant to dataframe
boston_data['constant'] = 1
```

```
# Specify target
Y = boston_data['CRIM']

# Variables to use in forward propagation
vars_left_add = ['ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX','PTRATIO', 'B', 'LSTAT', 'MDEV']

# Regression type
ols = LinearRegression()

# Starting variables (constant initially)
current_vars = ['constant']

X = boston_data[current_vars]
#benchmark_error = np.mean(-1*cross_val_score(ols, X, Y, cv = 5, scoring = 'neg_mean_squared_error'))
model = sm.OLS(Y, X).fit()
benchmark_error = model.aic
print(' Initial run with only one var (constant term/only bias weight):', current_vars)
print('      Benchmark error:', benchmark_error)
print('')


for iter in range(len(vars_left_add)):
    print('\033[1m'+ 'Iteration:', iter, '\033[0m')
    error_list = []
    for var in vars_left_add:
        X = boston_data[current_vars + [var]]
        #error = np.mean(-1*cross_val_score(ols, X, Y, cv = 5, scoring = 'neg_mean_squared_error'))
        model = sm.OLS(Y, X).fit()
        error = model.aic
        error_list.append(error)
        print(' Running model with:', current_vars + [var])
        print('      Error:', error)

    # Chose the smallest error
    min_error = min(error_list)
    chosen_col_index = error_list.index(min_error)

    # If our current smallest error is smaller than our previous error, than we add a variable
    if min_error < benchmark_error:
        print('          *** Variable selected:', vars_left_add[chosen_col_index])
        print('          *** Min error selected:', min_error)
        print('          *** Chose the variable that generated the min error + was lower than previous error')
        print('')
```

```
        # Add the variable that produced the smallest error to current_vars
        current_vars.append(vars_left_add[chosen_col_index])
        # Delete chosen variable from vars_left_add
        del vars_left_add[chosen_col_index]
        # Update benchmark_error
        benchmark_error = min_error

    # Otherwise, we stop our model
    else:
        print('          \033[4m*** No variable was selected', '\033[0m')
        print('          *** Previous error rate (', benchmark_error,') is lower than smallest error rate of this iteration (', mi
        print('          *** Break')
        break

print('')
print('Variables chosen for our model', current_vars)
```

```
Initial run with only one var (constant term/only bias weight): ['constant']
     Benchmark error: 3614.1694785562504

Iteration: 0
Running model with: ['constant', 'ZN']
     Error: 3595.627661797868
Running model with: ['constant', 'INDUS']
     Error: 3525.7755232365867
Running model with: ['constant', 'CHAS']
     Error: 3614.619980334153
Running model with: ['constant', 'NOX']
     Error: 3519.243655311626
Running model with: ['constant', 'RM']
     Error: 3591.08064952379
Running model with: ['constant', 'AGE']
     Error: 3549.7294697160696
Running model with: ['constant', 'DIS']
     Error: 3538.1963952494225
Running model with: ['constant', 'RAD']
     Error: 3368.6040266329437
Running model with: ['constant', 'TAX']
     Error: 3409.056271006969
Running model with: ['constant', 'PTRATIO']
     Error: 3572.2770418517935
Running model with: ['constant', 'B']
     Error: 3538.4366380342935
Running model with: ['constant', 'LSTAT']
```

```
      Error: 3500.4025345835294
 Running model with: ['constant', 'MDEV']
      Error: 3534.609458612529
          *** Variable selected: RAD
          *** Min error selected: 3368.6040266329437
          *** Chose the variable that generated the min error + was lower than previous error
```

Iteration: 1

```
 Running model with: ['constant', 'RAD', 'ZN']
      Error: 3370.577192490913
 Running model with: ['constant', 'RAD', 'INDUS']
      Error: 3369.0997551175096
 Running model with: ['constant', 'RAD', 'CHAS']
      Error: 3368.476911961381
 Running model with: ['constant', 'RAD', 'NOX']
      Error: 3368.7778437770467
 Running model with: ['constant', 'RAD', 'RM']
      Error: 3363.654841833111
 Running model with: ['constant', 'RAD', 'AGE']
      Error: 3365.8869887889996
 Running model with: ['constant', 'RAD', 'DIS']
      Error: 3365.1821077706627
 Running model with: ['constant', 'RAD', 'TAX']
      Error: 3369.7421749160776
 Running model with: ['constant', 'RAD', 'PTRATIO']
      Error: 3370.603297418971
 Running model with: ['constant', 'RAD', 'B']
      Error: 3360.0178393224005
 Running model with: ['constant', 'RAD', 'LSTAT']
      Error: 3346.194660602593
 Running model with: ['constant', 'RAD', 'MDEV']
      Error: 3348.8546253325494
          *** Variable selected: LSTAT
          *** Min error selected: 3346.194660602593
          *** Chose the variable that generated the min error + was lower than previous error
```

Iteration: 2

```
 Running model with: ['constant', 'RAD', 'LSTAT', 'ZN']
      Error: 3346.0024088660293
 Running model with: ['constant', 'RAD', 'LSTAT', 'INDUS']
      Error: 3347.0018288339215
 Running model with: ['constant', 'RAD', 'LSTAT', 'CHAS']
      Error: 3346.7380613433907
 Running model with: ['constant', 'RAD', 'LSTAT', 'NOX']
```

```
      Error: 3347.5404976340633
 Running model with: ['constant', 'RAD', 'LSTAT', 'RM']
      Error: 3348.048586281651
 Running model with: ['constant', 'RAD', 'LSTAT', 'AGE']
      Error: 3348.1285205194026
 Running model with: ['constant', 'RAD', 'LSTAT', 'DIS']
      Error: 3347.6491334753628
 Running model with: ['constant', 'RAD', 'LSTAT', 'TAX']
      Error: 3348.0017977641664
 Running model with: ['constant', 'RAD', 'LSTAT', 'PTRATIO']
      Error: 3347.199616915427
 Running model with: ['constant', 'RAD', 'LSTAT', 'B']
      Error: 3342.3675148809493
 Running model with: ['constant', 'RAD', 'LSTAT', 'MDEV']
      Error: 3344.970144517545
          *** Variable selected: B
          *** Min error selected: 3342.3675148809493
          *** Chose the variable that generated the min error + was lower than previous error

Iteration: 3
 Running model with: ['constant', 'RAD', 'LSTAT', 'B', 'ZN']
      Error: 3342.2757471020996
 Running model with: ['constant', 'RAD', 'LSTAT', 'B', 'INDUS']
      Error: 3342.879344708228
 Running model with: ['constant', 'RAD', 'LSTAT', 'B', 'CHAS']
      Error: 3343.1215136593323
 Running model with: ['constant', 'RAD', 'LSTAT', 'B', 'NOX']
      Error: 3343.340191792265
 Running model with: ['constant', 'RAD', 'LSTAT', 'B', 'RM']
      Error: 3344.343265366195
 Running model with: ['constant', 'RAD', 'LSTAT', 'B', 'AGE']
      Error: 3344.306643079205
 Running model with: ['constant', 'RAD', 'LSTAT', 'B', 'DIS']
      Error: 3343.9187447775876
 Running model with: ['constant', 'RAD', 'LSTAT', 'B', 'TAX']
      Error: 3344.0469959165653
 Running model with: ['constant', 'RAD', 'LSTAT', 'B', 'PTRATIO']
      Error: 3343.6917247812735
 Running model with: ['constant', 'RAD', 'LSTAT', 'B', 'MDEV']
      Error: 3341.8597369091076
          *** Variable selected: MDEV
          *** Min error selected: 3341.8597369091076
          *** Chose the variable that generated the min error + was lower than previous error
```

```
Iteration: 4
 Running model with: ['constant', 'RAD', 'LSTAT', 'B', 'MDEV', 'ZN']
      Error: 3341.3080618441213
 Running model with: ['constant', 'RAD', 'LSTAT', 'B', 'MDEV', 'INDUS']
      Error: 3342.126718411293
 Running model with: ['constant', 'RAD', 'LSTAT', 'B', 'MDEV', 'CHAS']
      Error: 3343.188536063383
 Running model with: ['constant', 'RAD', 'LSTAT', 'B', 'MDEV', 'NOX']
      Error: 3342.963278561945
 Running model with: ['constant', 'RAD', 'LSTAT', 'B', 'MDEV', 'RM']
      Error: 3342.7793351316604
 Running model with: ['constant', 'RAD', 'LSTAT', 'B', 'MDEV', 'AGE']
      Error: 3343.8588436969617
 Running model with: ['constant', 'RAD', 'LSTAT', 'B', 'MDEV', 'DIS']
      Error: 3342.733214095413
 Running model with: ['constant', 'RAD', 'LSTAT', 'B', 'MDEV', 'TAX']
      Error: 3343.107791161233
 Running model with: ['constant', 'RAD', 'LSTAT', 'B', 'MDEV', 'PTRATIO']
      Error: 3341.3853906584372
          *** Variable selected: ZN
          *** Min error selected: 3341.3080618441213
          *** Chose the variable that generated the min error + was lower than previous error

Iteration: 5
 Running model with: ['constant', 'RAD', 'LSTAT', 'B', 'MDEV', 'ZN', 'INDUS']
      Error: 3342.6955843801916
 Running model with: ['constant', 'RAD', 'LSTAT', 'B', 'MDEV', 'ZN', 'CHAS']
      Error: 3342.8418874967692
 Running model with: ['constant', 'RAD', 'LSTAT', 'B', 'MDEV', 'ZN', 'NOX']
      Error: 3343.1452662991264
 Running model with: ['constant', 'RAD', 'LSTAT', 'B', 'MDEV', 'ZN', 'RM']
      Error: 3342.427443911504
 Running model with: ['constant', 'RAD', 'LSTAT', 'B', 'MDEV', 'ZN', 'AGE']
      Error: 3342.735681402325
 Running model with: ['constant', 'RAD', 'LSTAT', 'B', 'MDEV', 'ZN', 'DIS']
      Error: 3336.587929677827
 Running model with: ['constant', 'RAD', 'LSTAT', 'B', 'MDEV', 'ZN', 'TAX']
      Error: 3342.481042829595
 Running model with: ['constant', 'RAD', 'LSTAT', 'B', 'MDEV', 'ZN', 'PTRATIO']
      Error: 3341.7829067515004
          *** Variable selected: DIS
          *** Min error selected: 3336.587929677827
          *** Chose the variable that generated the min error + was lower than previous error
```

```
Iteration: 6
 Running model with: ['constant', 'RAD', 'LSTAT', 'B', 'MDEV', 'ZN', 'DIS', 'INDUS']
      Error: 3333.8715257631347
 Running model with: ['constant', 'RAD', 'LSTAT', 'B', 'MDEV', 'ZN', 'DIS', 'CHAS']
      Error: 3337.83803089632
 Running model with: ['constant', 'RAD', 'LSTAT', 'B', 'MDEV', 'ZN', 'DIS', 'NOX']
      Error: 3333.4044316181694
 Running model with: ['constant', 'RAD', 'LSTAT', 'B', 'MDEV', 'ZN', 'DIS', 'RM']
      Error: 3337.9166118518488
 Running model with: ['constant', 'RAD', 'LSTAT', 'B', 'MDEV', 'ZN', 'DIS', 'AGE']
      Error: 3338.272922140982
 Running model with: ['constant', 'RAD', 'LSTAT', 'B', 'MDEV', 'ZN', 'DIS', 'TAX']
      Error: 3335.667059782153
 Running model with: ['constant', 'RAD', 'LSTAT', 'B', 'MDEV', 'ZN', 'DIS', 'PTRATIO']
      Error: 3337.8424601859633
          *** Variable selected: NOX
          *** Min error selected: 3333.4044316181694
          *** Chose the variable that generated the min error + was lower than previous error

Iteration: 7
 Running model with: ['constant', 'RAD', 'LSTAT', 'B', 'MDEV', 'ZN', 'DIS', 'NOX', 'INDUS']
      Error: 3332.973115088829
 Running model with: ['constant', 'RAD', 'LSTAT', 'B', 'MDEV', 'ZN', 'DIS', 'NOX', 'CHAS']
      Error: 3335.0440070836794
 Running model with: ['constant', 'RAD', 'LSTAT', 'B', 'MDEV', 'ZN', 'DIS', 'NOX', 'RM']
      Error: 3334.671907275078
 Running model with: ['constant', 'RAD', 'LSTAT', 'B', 'MDEV', 'ZN', 'DIS', 'NOX', 'AGE']
      Error: 3335.399611515146
 Running model with: ['constant', 'RAD', 'LSTAT', 'B', 'MDEV', 'ZN', 'DIS', 'NOX', 'TAX']
      Error: 3333.610269065178
 Running model with: ['constant', 'RAD', 'LSTAT', 'B', 'MDEV', 'ZN', 'DIS', 'NOX', 'PTRATIO']
      Error: 3332.396301742922
          *** Variable selected: PTRATIO
          *** Min error selected: 3332.396301742922
          *** Chose the variable that generated the min error + was lower than previous error

Iteration: 8
 Running model with: ['constant', 'RAD', 'LSTAT', 'B', 'MDEV', 'ZN', 'DIS', 'NOX', 'PTRATIO', 'INDUS']
      Error: 3332.701557391131
 Running model with: ['constant', 'RAD', 'LSTAT', 'B', 'MDEV', 'ZN', 'DIS', 'NOX', 'PTRATIO', 'CHAS']
      Error: 3333.9638374558717
 Running model with: ['constant', 'RAD', 'LSTAT', 'B', 'MDEV', 'ZN', 'DIS', 'NOX', 'PTRATIO', 'RM']
      Error: 3333.737183528465
 Running model with: ['constant', 'RAD', 'LSTAT', 'B', 'MDEV', 'ZN', 'DIS', 'NOX', 'PTRATIO', 'AGE']
```

```
     Error: 3334.358658826476
Running model with: ['constant', 'RAD', 'LSTAT', 'B', 'MDEV', 'ZN', 'DIS', 'NOX', 'PTRATIO', 'TAX']
     Error: 3333.0184122535993
         *** No variable was selected
         *** Previous error rate ( 3332.396301742922 ) is lower than smallest error rate of this iteration ( 3332.701557391131 )
         *** Break

Variables chosen for our model ['constant', 'RAD', 'LSTAT', 'B', 'MDEV', 'ZN', 'DIS', 'NOX', 'PTRATIO']
```

When AIC is used as the error criteria, then the variables choosen by the FSS for the model are 'constant', 'RAD', 'LSTAT', 'B', 'MDEV', 'ZN', 'DIS', 'NOX', 'PTRATIO'

Backward Stepwise Selection

In [21]:

```
# Add constant to dataframe
boston_data['constant'] = 1

# Specify the target
Y = boston_data['CRIM']

# Variables to remove iteratively
vars_left_to_drop = ['ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX','PTRATIO', 'B', 'LSTAT', 'MDEV']

# Regression type
ols = LinearRegression()

# Starting variables (all)
current_vars = ['constant'] + vars_left_to_drop

X = boston_data[current_vars]
benchmark_error = np.mean(-1*cross_val_score(ols, X, Y, cv = 5, scoring = 'neg_mean_squared_error'))
print(' Initial run with all vars:', current_vars)
print('      Benchmark error:', benchmark_error)
print('')

# Keep removing the worst variables (until no improvement can be made)
for iter in range(len(vars_left_to_drop)):
    print('\033[1m'+ 'Iteration:', iter, '\033[0m')
    error_list = []
    # Iterate over all the variables left to remove
    for var in vars_left_to_drop:
        # Modify X according to current iteration
        vars_to_be_used = ['constant'] + [i for i in vars_left_to_drop if i != var]
```

```
        X = boston_data[['constant'] + [i for i in vars_left_to_drop if i != var]]
        # Perform 5-fold CV to get errors
        error = np.mean(-1*cross_val_score(ols, X, Y, cv = 5, scoring = 'neg_mean_squared_error'))
        error_list.append(error)
        print(' Running model with:', vars_to_be_used)
        print('      Error:', error)

    # Chose the largest error
    min_error = min(error_list)
    chosen_col_index = error_list.index(min_error)

    # If our current smallest error is smaller than our previous error, then we drop the variable associated with it
    if min_error < benchmark_error:
        print('          *** Will drop:', vars_left_to_drop[chosen_col_index])
        print('          *** Min error selected:', min_error)
        print('          *** Chose the variable that generated the min error + was lower than previous error')
        print('')
        # Delete chosen variable from current_vars and vars_left_to_drop
        del current_vars[chosen_col_index + 1]
        del vars_left_to_drop[chosen_col_index]
        # Update benchmark_error
        benchmark_error = min_error

    # If not, we keep our model
    else:
        print('          \033[4m*** No variable was selected', '\033[0m')
        print('          *** Previous error rate (', benchmark_error,') is lower than smallest error rate of this iteration (', mi
        print('          *** Break')
        break

print('')
print('Variables chosen for our model', current_vars)
```

```
 Initial run with all vars: ['constant', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT',
'MDEV']
      Benchmark error: 50.37783426726539

Iteration: 0
 Running model with: ['constant', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
      Error: 51.05837513173859
 Running model with: ['constant', 'ZN', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
      Error: 49.450337205176154
 Running model with: ['constant', 'ZN', 'INDUS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
      Error: 50.13861791255141
```

```
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
      Error: 49.56409044975212
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'NOX', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
      Error: 47.105664575653485
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
      Error: 49.39717909514341
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
      Error: 50.965489302090646
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
      Error: 54.689231752912974
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
      Error: 50.32593407984406
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'B', 'LSTAT', 'MDEV']
      Error: 50.6534783722195
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'LSTAT', 'MDEV']
      Error: 47.912510542862364
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'MDEV']
      Error: 50.04232666992431
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT']
      Error: 49.126879209918705
          *** Will drop: RM
          *** Min error selected: 47.105664575653485
          *** Chose the variable that generated the min error + was lower than previous error

Iteration: 1
 Running model with: ['constant', 'INDUS', 'CHAS', 'NOX', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
      Error: 48.02855491239959
 Running model with: ['constant', 'ZN', 'CHAS', 'NOX', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
      Error: 46.30179269187482
 Running model with: ['constant', 'ZN', 'INDUS', 'NOX', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
      Error: 46.949850279811514
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
      Error: 46.38851333762737
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'NOX', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
      Error: 46.8292901196068
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'NOX', 'AGE', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
      Error: 47.37482420952087
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'NOX', 'AGE', 'DIS', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
      Error: 53.43253062743734
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'NOX', 'AGE', 'DIS', 'RAD', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
      Error: 46.9872367964349
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'NOX', 'AGE', 'DIS', 'RAD', 'TAX', 'B', 'LSTAT', 'MDEV']
      Error: 47.282949018527276
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'NOX', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'LSTAT', 'MDEV']
```

```
      Error: 46.03706169147082
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'NOX', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'MDEV']
      Error: 47.29201315352875
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'NOX', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT']
      Error: 47.04779150975477
          *** Will drop: B
          *** Min error selected: 46.03706169147082
          *** Chose the variable that generated the min error + was lower than previous error

Iteration: 2
 Running model with: ['constant', 'INDUS', 'CHAS', 'NOX', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'LSTAT', 'MDEV']
      Error: 46.848225581267926
 Running model with: ['constant', 'ZN', 'CHAS', 'NOX', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'LSTAT', 'MDEV']
      Error: 45.065721813836184
 Running model with: ['constant', 'ZN', 'INDUS', 'NOX', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'LSTAT', 'MDEV']
      Error: 45.907858980264315
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'LSTAT', 'MDEV']
      Error: 44.94272114063311
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'NOX', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'LSTAT', 'MDEV']
      Error: 45.81358830999615
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'NOX', 'AGE', 'RAD', 'TAX', 'PTRATIO', 'LSTAT', 'MDEV']
      Error: 45.838163376823516
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'NOX', 'AGE', 'DIS', 'TAX', 'PTRATIO', 'LSTAT', 'MDEV']
      Error: 49.71420384354165
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'NOX', 'AGE', 'DIS', 'RAD', 'PTRATIO', 'LSTAT', 'MDEV']
      Error: 45.86296495802093
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'NOX', 'AGE', 'DIS', 'RAD', 'TAX', 'LSTAT', 'MDEV']
      Error: 46.2848808767164
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'NOX', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'MDEV']
      Error: 46.20665631048024
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'NOX', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'LSTAT']
      Error: 46.01178313707686
          *** Will drop: NOX
          *** Min error selected: 44.94272114063311
          *** Chose the variable that generated the min error + was lower than previous error

Iteration: 3
 Running model with: ['constant', 'INDUS', 'CHAS', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'LSTAT', 'MDEV']
      Error: 45.41948351730665
 Running model with: ['constant', 'ZN', 'CHAS', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'LSTAT', 'MDEV']
      Error: 44.922916467447905
 Running model with: ['constant', 'ZN', 'INDUS', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'LSTAT', 'MDEV']
      Error: 44.8930450007566
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'LSTAT', 'MDEV']
```

```
     Error: 44.69757025481981
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'AGE', 'RAD', 'TAX', 'PTRATIO', 'LSTAT', 'MDEV']
     Error: 45.4021488893526
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'AGE', 'DIS', 'TAX', 'PTRATIO', 'LSTAT', 'MDEV']
     Error: 49.55703370732756
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'AGE', 'DIS', 'RAD', 'PTRATIO', 'LSTAT', 'MDEV']
     Error: 44.9377289273967
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'AGE', 'DIS', 'RAD', 'TAX', 'LSTAT', 'MDEV']
     Error: 45.02637918465123
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'MDEV']
     Error: 45.10400403023092
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'LSTAT']
     Error: 45.34230394026995
         *** Will drop: AGE
         *** Min error selected: 44.69757025481981
         *** Chose the variable that generated the min error + was lower than previous error

Iteration: 4
 Running model with: ['constant', 'INDUS', 'CHAS', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'LSTAT', 'MDEV']
     Error: 45.21684629258205
 Running model with: ['constant', 'ZN', 'CHAS', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'LSTAT', 'MDEV']
     Error: 44.795595717823495
 Running model with: ['constant', 'ZN', 'INDUS', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'LSTAT', 'MDEV']
     Error: 44.62447779796319
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'RAD', 'TAX', 'PTRATIO', 'LSTAT', 'MDEV']
     Error: 44.90938059460607
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'DIS', 'TAX', 'PTRATIO', 'LSTAT', 'MDEV']
     Error: 49.40760770954502
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'DIS', 'RAD', 'PTRATIO', 'LSTAT', 'MDEV']
     Error: 44.71710934537782
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'DIS', 'RAD', 'TAX', 'LSTAT', 'MDEV']
     Error: 44.75665817699234
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'MDEV']
     Error: 44.947356167891414
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'LSTAT']
     Error: 45.08026159583899
         *** Will drop: CHAS
         *** Min error selected: 44.62447779796319
         *** Chose the variable that generated the min error + was lower than previous error

Iteration: 5
 Running model with: ['constant', 'INDUS', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'LSTAT', 'MDEV']
     Error: 45.160848357553945
 Running model with: ['constant', 'ZN', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'LSTAT', 'MDEV']
```

```
      Error: 44.73434504692936
 Running model with: ['constant', 'ZN', 'INDUS', 'RAD', 'TAX', 'PTRATIO', 'LSTAT', 'MDEV']
      Error: 44.79228935179958
 Running model with: ['constant', 'ZN', 'INDUS', 'DIS', 'TAX', 'PTRATIO', 'LSTAT', 'MDEV']
      Error: 49.29197501407013
 Running model with: ['constant', 'ZN', 'INDUS', 'DIS', 'RAD', 'PTRATIO', 'LSTAT', 'MDEV']
      Error: 44.57164670513221
 Running model with: ['constant', 'ZN', 'INDUS', 'DIS', 'RAD', 'TAX', 'LSTAT', 'MDEV']
      Error: 44.66944643257213
 Running model with: ['constant', 'ZN', 'INDUS', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'MDEV']
      Error: 44.86386005423638
 Running model with: ['constant', 'ZN', 'INDUS', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'LSTAT']
      Error: 45.006209304552485
          *** Will drop: TAX
          *** Min error selected: 44.57164670513221
          *** Chose the variable that generated the min error + was lower than previous error

Iteration: 6
 Running model with: ['constant', 'INDUS', 'DIS', 'RAD', 'PTRATIO', 'LSTAT', 'MDEV']
      Error: 45.00557592403743
 Running model with: ['constant', 'ZN', 'DIS', 'RAD', 'PTRATIO', 'LSTAT', 'MDEV']
      Error: 44.835543881109984
 Running model with: ['constant', 'ZN', 'INDUS', 'RAD', 'PTRATIO', 'LSTAT', 'MDEV']
      Error: 44.75419504015118
 Running model with: ['constant', 'ZN', 'INDUS', 'DIS', 'PTRATIO', 'LSTAT', 'MDEV']
      Error: 63.69552151762239
 Running model with: ['constant', 'ZN', 'INDUS', 'DIS', 'RAD', 'LSTAT', 'MDEV']
      Error: 44.64223506545175
 Running model with: ['constant', 'ZN', 'INDUS', 'DIS', 'RAD', 'PTRATIO', 'MDEV']
      Error: 44.86633776936436
 Running model with: ['constant', 'ZN', 'INDUS', 'DIS', 'RAD', 'PTRATIO', 'LSTAT']
      Error: 44.81536568056383
          *** No variable was selected
          *** Previous error rate ( 44.57164670513221 ) is lower than smallest error rate of this iteration ( 44.64223506545175 )
          *** Break

 Variables chosen for our model ['constant', 'ZN', 'INDUS', 'DIS', 'RAD', 'PTRATIO', 'LSTAT', 'MDEV']
```

Backward stepwise with the 5 fold cross validation criteria selection chooses the model with 'constant', 'ZN', 'INDUS', 'DIS', 'RAD', 'PTRATIO', 'LSTAT', 'MDEV' as the features while excluding others.

BSS-AIC

In [22]:

```
# Add constant to dataframe
boston_data['constant'] = 1

# Specify the target
Y = boston_data['CRIM']

# Variables to remove iteratively
vars_left_to_drop = ['ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX','PTRATIO', 'B', 'LSTAT', 'MDEV']

# Regression type
ols = LinearRegression()

# Starting variables (all)
current_vars = ['constant'] + vars_left_to_drop

X = boston_data[current_vars]
#benchmark_error = np.mean(-1*cross_val_score(ols, X, Y, cv = 5, scoring = 'neg_mean_squared_error'))
model = sm.OLS(Y, X).fit()
benchmark_error = model.aic


print(' Initial run with all vars:', current_vars)
print('      Benchmark error:', benchmark_error)
print('')

# Keep removing the worst variables (until no improvement can be made)
for iter in range(len(vars_left_to_drop)):
    print('\033[1m'+ 'Iteration:', iter, '\033[0m')
    error_list = []
    # Iterate over all the variables left to remove
    for var in vars_left_to_drop:
        # Modify X according to current iteration
        vars_to_be_used = ['constant'] + [i for i in vars_left_to_drop if i != var]
        X = boston_data[['constant'] + [i for i in vars_left_to_drop if i != var]]
        # Perform AIC to get errors
        model = sm.OLS(Y, X).fit()
        #error = np.mean(-*cross_val_score(ols, X, Y, cv = 5, scoring = 'neg_mean_squared_error'))
        error = model.aic
        error_list.append(error)
        print(' Running model with:', vars_to_be_used)
        print('      Error:', error)
```

```
    # Chose the largest error
    min_error = min(error_list)
    chosen_col_index = error_list.index(min_error)

    # If our current smallest error is smaller than our previous error, then we drop the variable associated with it
    if min_error < benchmark_error:
        print('          *** Will drop:', vars_left_to_drop[chosen_col_index])
        print('          *** Min error selected:', min_error)
        print('          *** Chose the variable that generated the min error + was lower than previous error')
        print('')
        # Delete chosen variable from current_vars and vars_left_to_drop
        del current_vars[chosen_col_index + 1]
        del vars_left_to_drop[chosen_col_index]
        # Update benchmark_error
        benchmark_error = min_error

    # If not, we keep our model
    else:
        print('          \033[4m*** No variable was selected', '\033[0m')
        print('          *** Previous error rate (', benchmark_error,') is lower than smallest error rate of this iteration (', mi
        print('          *** Break')
        break

print('')
print('Variables chosen for our model', current_vars)
```

```
 Initial run with all vars: ['constant', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT',
'MDEV']
      Benchmark error: 3339.385162839013

Iteration: 0
 Running model with: ['constant', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
      Error: 3343.20771130992
 Running model with: ['constant', 'ZN', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
      Error: 3337.940028166776
 Running model with: ['constant', 'ZN', 'INDUS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
      Error: 3337.7870578241814
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
      Error: 3341.5159785696305
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'NOX', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
      Error: 3337.7788683231893
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
      Error: 3337.397987975169
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
```

```
     Error: 3349.9258617960922
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
     Error: 3381.0062976126505
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
     Error: 3337.9225600830205
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'B', 'LSTAT', 'MDEV']
     Error: 3339.656488421626
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'LSTAT', 'MDEV']
     Error: 3340.920320093771
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'MDEV']
     Error: 3339.9908199025413
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT']
     Error: 3348.305400313703
         *** Will drop: AGE
         *** Min error selected: 3337.397987975169
         *** Chose the variable that generated the min error + was lower than previous error

Iteration: 1
 Running model with: ['constant', 'INDUS', 'CHAS', 'NOX', 'RM', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
     Error: 3341.239133139702
 Running model with: ['constant', 'ZN', 'CHAS', 'NOX', 'RM', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
     Error: 3335.9526811508977
 Running model with: ['constant', 'ZN', 'INDUS', 'NOX', 'RM', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
     Error: 3335.7937324672725
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'RM', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
     Error: 3339.7015492431633
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'NOX', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
     Error: 3335.835080095834
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
     Error: 3349.22690309678
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'DIS', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
     Error: 3379.158287397374
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'DIS', 'RAD', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
     Error: 3335.930902970419
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'DIS', 'RAD', 'TAX', 'B', 'LSTAT', 'MDEV']
     Error: 3337.6564884492054
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'LSTAT', 'MDEV']
     Error: 3338.9203404988752
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'MDEV']
     Error: 3338.4091521221235
 Running model with: ['constant', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT']
     Error: 3346.3166280750397
         *** Will drop: CHAS
         *** Min error selected: 3335.7937324672725
```

```
        *** Chose the variable that generated the min error + was lower than previous error

Iteration: 2
 Running model with: ['constant', 'INDUS', 'NOX', 'RM', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
      Error: 3339.6794096371004
 Running model with: ['constant', 'ZN', 'NOX', 'RM', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
      Error: 3334.4534231786656
 Running model with: ['constant', 'ZN', 'INDUS', 'RM', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
      Error: 3338.3317417859644
 Running model with: ['constant', 'ZN', 'INDUS', 'NOX', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
      Error: 3334.2440856526337
 Running model with: ['constant', 'ZN', 'INDUS', 'NOX', 'RM', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
      Error: 3347.7146777143707
 Running model with: ['constant', 'ZN', 'INDUS', 'NOX', 'RM', 'DIS', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
      Error: 3377.1864533665025
 Running model with: ['constant', 'ZN', 'INDUS', 'NOX', 'RM', 'DIS', 'RAD', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
      Error: 3334.245842612034
 Running model with: ['constant', 'ZN', 'INDUS', 'NOX', 'RM', 'DIS', 'RAD', 'TAX', 'B', 'LSTAT', 'MDEV']
      Error: 3335.980335714623
 Running model with: ['constant', 'ZN', 'INDUS', 'NOX', 'RM', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'LSTAT', 'MDEV']
      Error: 3337.3996337999465
 Running model with: ['constant', 'ZN', 'INDUS', 'NOX', 'RM', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'MDEV']
      Error: 3336.7606023424505
 Running model with: ['constant', 'ZN', 'INDUS', 'NOX', 'RM', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT']
      Error: 3345.551277688976
          *** Will drop: RM
          *** Min error selected: 3334.2440856526337
          *** Chose the variable that generated the min error + was lower than previous error

Iteration: 3
 Running model with: ['constant', 'INDUS', 'NOX', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
      Error: 3338.37517684147
 Running model with: ['constant', 'ZN', 'NOX', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
      Error: 3333.018412253599
 Running model with: ['constant', 'ZN', 'INDUS', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
      Error: 3336.6690371307222
 Running model with: ['constant', 'ZN', 'INDUS', 'NOX', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
      Error: 3346.4915746415354
 Running model with: ['constant', 'ZN', 'INDUS', 'NOX', 'DIS', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
      Error: 3376.304753055805
 Running model with: ['constant', 'ZN', 'INDUS', 'NOX', 'DIS', 'RAD', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
      Error: 3332.701557391131
 Running model with: ['constant', 'ZN', 'INDUS', 'NOX', 'DIS', 'RAD', 'TAX', 'B', 'LSTAT', 'MDEV']
      Error: 3334.446734777459
```

```
 Running model with: ['constant', 'ZN', 'INDUS', 'NOX', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'LSTAT', 'MDEV']
      Error: 3336.3276398063344
 Running model with: ['constant', 'ZN', 'INDUS', 'NOX', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'MDEV']
      Error: 3334.827138337393
 Running model with: ['constant', 'ZN', 'INDUS', 'NOX', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT']
      Error: 3344.0688727538445
          *** Will drop: TAX
          *** Min error selected: 3332.701557391131
          *** Chose the variable that generated the min error + was lower than previous error

Iteration: 4
 Running model with: ['constant', 'INDUS', 'NOX', 'DIS', 'RAD', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
      Error: 3336.3822574539126
 Running model with: ['constant', 'ZN', 'NOX', 'DIS', 'RAD', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
      Error: 3332.3963017429214
 Running model with: ['constant', 'ZN', 'INDUS', 'DIS', 'RAD', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
      Error: 3335.267965969556
 Running model with: ['constant', 'ZN', 'INDUS', 'NOX', 'RAD', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
      Error: 3344.7801575145654
 Running model with: ['constant', 'ZN', 'INDUS', 'NOX', 'DIS', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
      Error: 3436.8787010006263
 Running model with: ['constant', 'ZN', 'INDUS', 'NOX', 'DIS', 'RAD', 'B', 'LSTAT', 'MDEV']
      Error: 3332.973115088829
 Running model with: ['constant', 'ZN', 'INDUS', 'NOX', 'DIS', 'RAD', 'PTRATIO', 'LSTAT', 'MDEV']
      Error: 3334.778862995495
 Running model with: ['constant', 'ZN', 'INDUS', 'NOX', 'DIS', 'RAD', 'PTRATIO', 'B', 'MDEV']
      Error: 3333.4644161482993
 Running model with: ['constant', 'ZN', 'INDUS', 'NOX', 'DIS', 'RAD', 'PTRATIO', 'B', 'LSTAT']
      Error: 3342.07359489322
          *** Will drop: INDUS
          *** Min error selected: 3332.3963017429214
          *** Chose the variable that generated the min error + was lower than previous error

Iteration: 5
 Running model with: ['constant', 'NOX', 'DIS', 'RAD', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
      Error: 3336.1902631263456
 Running model with: ['constant', 'ZN', 'DIS', 'RAD', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
      Error: 3337.8424601859633
 Running model with: ['constant', 'ZN', 'NOX', 'RAD', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
      Error: 3342.9612773917115
 Running model with: ['constant', 'ZN', 'NOX', 'DIS', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
      Error: 3434.9064476045396
 Running model with: ['constant', 'ZN', 'NOX', 'DIS', 'RAD', 'B', 'LSTAT', 'MDEV']
      Error: 3333.4044316181694
```

```
Running model with: ['constant', 'ZN', 'NOX', 'DIS', 'RAD', 'PTRATIO', 'LSTAT', 'MDEV']
     Error: 3334.3739982778625
Running model with: ['constant', 'ZN', 'NOX', 'DIS', 'RAD', 'PTRATIO', 'B', 'MDEV']
     Error: 3332.824449287047
Running model with: ['constant', 'ZN', 'NOX', 'DIS', 'RAD', 'PTRATIO', 'B', 'LSTAT']
     Error: 3341.066513914931
         *** No variable was selected
         *** Previous error rate ( 3332.3963017429214 ) is lower than smallest error rate of this iteration ( 3332.824449287047 )
         *** Break

 Variables chosen for our model ['constant', 'ZN', 'NOX', 'DIS', 'RAD', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
```

Backward stepwise with the AIC score criteria selection chooses the model with 'constant', 'ZN', 'NOX', 'DIS', 'RAD', 'PTRATIO', 'B', 'LSTAT', 'MDEV' as the features while excluding others. These variables are same as the variables selected in FSS-AIC model.

We have observed here the BSS and FSS methods selected the same features using AIC as the error criterion, it is likely that these features were the most important predictors in reducing the prediction error according to both methods. The fact that both the models selected same variables under this criterion indicates that they were the most useful features in the context of the model and data.

It's worth noting that different error criteria may be more appropriate in different contexts, and it can be a good practice to try multiple criteria and compare their performance on validation sets or through cross-validation.

Part b

(b) Propose a model (or set of models) that seem to perform well on this data set, and justify your answer. Make sure that you are evaluating model performance using validation set error, crossvalidation, or some other reasonable alternative, as opposed to using training error.

Variables selected in FSS-5FCV are ['constant', 'RAD', 'LSTAT', 'ZN']

Variables selected in FSS-AIC are ['constant', 'RAD', 'LSTAT', 'B', 'MDEV', 'ZN', 'DIS', 'NOX', 'PTRATIO']

Variables selected in BSS-5FCV are ['constant', 'ZN', 'INDUS', 'DIS', 'RAD', 'PTRATIO', 'LSTAT', 'MDEV']

Variables selected in BSS-AIC are ['constant', 'ZN', 'NOX', 'DIS', 'RAD', 'PTRATIO', 'B', 'LSTAT', 'MDEV']

Variables seleced in FSS-AIC and BSS-AIC are same.

In [23]:

```
x_vars = ['constant','ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX','PTRATIO', 'B', 'LSTAT', 'MDEV']
y_vars = ['CRIM']
```

```
X_mtrain, X_mtest, y_mtrain, y_mtest = train_test_split(boston_data[x_vars], boston_data[y_vars], test_size=0.2)
```

In [24]:

```
#Linear regression of all the 4 models from above

#Model 1, variables selected in FSS-5FCV

x_vars = ['constant', 'RAD', 'LSTAT', 'ZN']
y_vars = ['CRIM']

X_train = X_mtrain[x_vars]
X_test = X_mtest[x_vars]
y_train = y_mtrain[y_vars]
y_test = y_mtest[y_vars]

#X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.8)

model_FSS5FCV = LinearRegression()
model_FSS5FCV.fit(X_train, y_train)

y_pred = model_FSS5FCV.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print("Validation set error: FSS-5FCV ", mse)
cross_v_error = np.mean(-1*cross_val_score(ols, boston_data[x_vars], boston_data[y_vars], cv = 5, scoring = 'neg_mean_squared_erro
print("Cross Validation error: FSS-5FCV ", cross_v_error)
```

```
Validation set error: FSS-5FCV  30.22614162146663
Cross Validation error: FSS-5FCV  44.46585900848578
```

In [25]:

```
#Model 2, variables selected in FSS-AIC

x_vars = ['constant', 'RAD', 'LSTAT', 'B', 'MDEV', 'ZN', 'DIS', 'NOX', 'PTRATIO']
y_vars = ['CRIM']

X_train = X_mtrain[x_vars]
X_test = X_mtest[x_vars]
y_train = y_mtrain[y_vars]
y_test = y_mtest[y_vars]

model_FSSAIC = LinearRegression()
model_FSSAIC.fit(X_train, y_train)
```

```
y_pred = model_FSSAIC.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print("Validation set error: FSS-AIC ", mse)
cross_v_error = np.mean(-1*cross_val_score(ols, boston_data[x_vars], boston_data[y_vars], cv = 5, scoring = 'neg_mean_squared_erro
print("Cross Validation error: FSS-AIC ", cross_v_error)
```

```
Validation set error: FSS-AIC  26.801819151175337
Cross Validation error: FSS-AIC  45.62012009007763
```

In [26]:

```
#Model 3, variables selected in BSS-5FCV

x_vars = ['constant', 'ZN', 'INDUS', 'DIS', 'RAD', 'PTRATIO', 'LSTAT', 'MDEV']
y_vars = ['CRIM']

X_train = X_mtrain[x_vars]
X_test = X_mtest[x_vars]
y_train = y_mtrain[y_vars]
y_test = y_mtest[y_vars]

model_BSS5FCV = LinearRegression()
model_BSS5FCV.fit(X_train, y_train)

y_pred = model_BSS5FCV.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print("Validation set error: BSS-5FCV ", mse)
cross_v_error = np.mean(-1*cross_val_score(ols, boston_data[x_vars], boston_data[y_vars], cv = 5, scoring = 'neg_mean_squared_erro
print("Cross Validation error: BSS-5FCV ", cross_v_error)
```

```
Validation set error: BSS-5FCV  28.684570963909
Cross Validation error: BSS-5FCV  44.57164670513221
```

In [27]:

```
#Model 4, variables selected in FSS-AIC

x_vars = ['constant', 'ZN', 'NOX', 'DIS', 'RAD', 'PTRATIO', 'B', 'LSTAT', 'MDEV']
y_vars = ['CRIM']

X_train = X_mtrain[x_vars]
X_test = X_mtest[x_vars]
y_train = y_mtrain[y_vars]
y_test = y_mtest[y_vars]

model_BSSAIC = LinearRegression()
model_BSSAIC.fit(X_train, y_train)
```

```
y_pred = model_BSSAIC.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print("Validation set error: BSS-AIC ", mse)
cross_v_error = np.mean(-1*cross_val_score(ols, boston_data[x_vars], boston_data[y_vars], cv = 5, scoring = 'neg_mean_squared_erro
print("Cross Validation error: BSS-AIC ", cross_v_error)
```

```
Validation set error: BSS-AIC  26.80181915117488
Cross Validation error: BSS-AIC  45.62012009007754
```

Looking at the cross-validation errors, we can see that the FSS-5FCV model has the lowest error, followed by the BSS-5FCV model and the FSS/BSS-AIC model.

However, when we look at the validation set errors, we can see that the FSS/BSS-AIC model has the lowest error, followed by the BSS-5FCV model and the FSS-5FCV model.

Given these results, it's difficult to definitively determine which model is the best. If validation set errors are a beter indicator of performance, then the FSS/BSS AIC model may be the best choice since it has the lowest validation set error. However, if we believe that the cross-validation errors are a better indicator of future performance, then the FSS-5FCV model may be the best choice since it has the lowest cross-validation error.

Its usually very important to consider multiple metrics when comparing models and to take into account the specific goals and constraints of the problem at hand.

Part c

(c) Does your chosen model involve all of the features in the data set? Why or why not?

The advantage of cross-validation is that it provides a more robust estimate of the model's performance compared to a single validation set. By using multiple folds we can get a better idea of how the model performs on different subsets of the data. But this approach can be computationally expensive.

In general, if the dataset is small, the validation set error may be more reliable. If the dataset is large, cross-validation can provide a more robust estimate of the model's performance.

Considering in our case, cross validation score as a better indicator of performance, FSS-5FCV model may be the best choice since it has the lowest cross-validation error. Variables selected in FSS-5FCV are ['constant', 'RAD', 'LSTAT', 'ZN']. The chosen model doesnt involve all the features of the data set. This is likely because the feature selection method used in the model (forward stepwise selection) iteratively adds the feature that has the greatest impact on reducing the prediction error at each step. This is decided keeping in notice the collinearity between features. This is done until a stop criteria is met such as addition of new feature may no longer lead to a reduction in prediction error.

In this case, the forward stepwise selection method identified the features 'RAD', 'LSTAT', 'ZN' as the most important features in reducing the prediction error. This suggests that the other features may not be as important in predicting "CRIM". It can also be the case that the others may be redundant when the selected features features are present.

It is very important to note that different feature selection methods(Example: BSS or FSS) may result in selection of different subsets of features, and that the specific subset may not necessarily be the optimal set of features for all scenarios. Generally, it is a good practice to try multiple methods, compare the performance and then choose the correct model and correct set of features.