Module 5 Homework

Problem 1

1a) For this problem, we are going to do a similar analysis for the lasso as was done in the lecture for the ridge regression.

Read the Hitters.csv data into a dataframe, drop the rows with NaNs, and create dummy variables to replace the three category columns (use the drop_first=True parameter in the Pandas get_dummies() function.

In [1]:

```
import math;
import numpy;
import pandas;
import matplotlib.pyplot as plt;
from sklearn.model_selection import train_test_split;
from sklearn.linear_model import Ridge, RidgeCV, Lasso, LassoCV, LinearRegression;
from sklearn.metrics import mean_squared_error;
from itertools import combinations;
import statsmodels.api as sm;
import warnings;
warnings.filterwarnings("ignore");


df1 = pandas.read_csv("Hitters.csv");

df1_na = df1.dropna();
df1_na_dm = pandas.get_dummies(df1_na, columns = ["League", "Division", "NewLeague"], drop_first = True);
print(df1_na_dm);
```

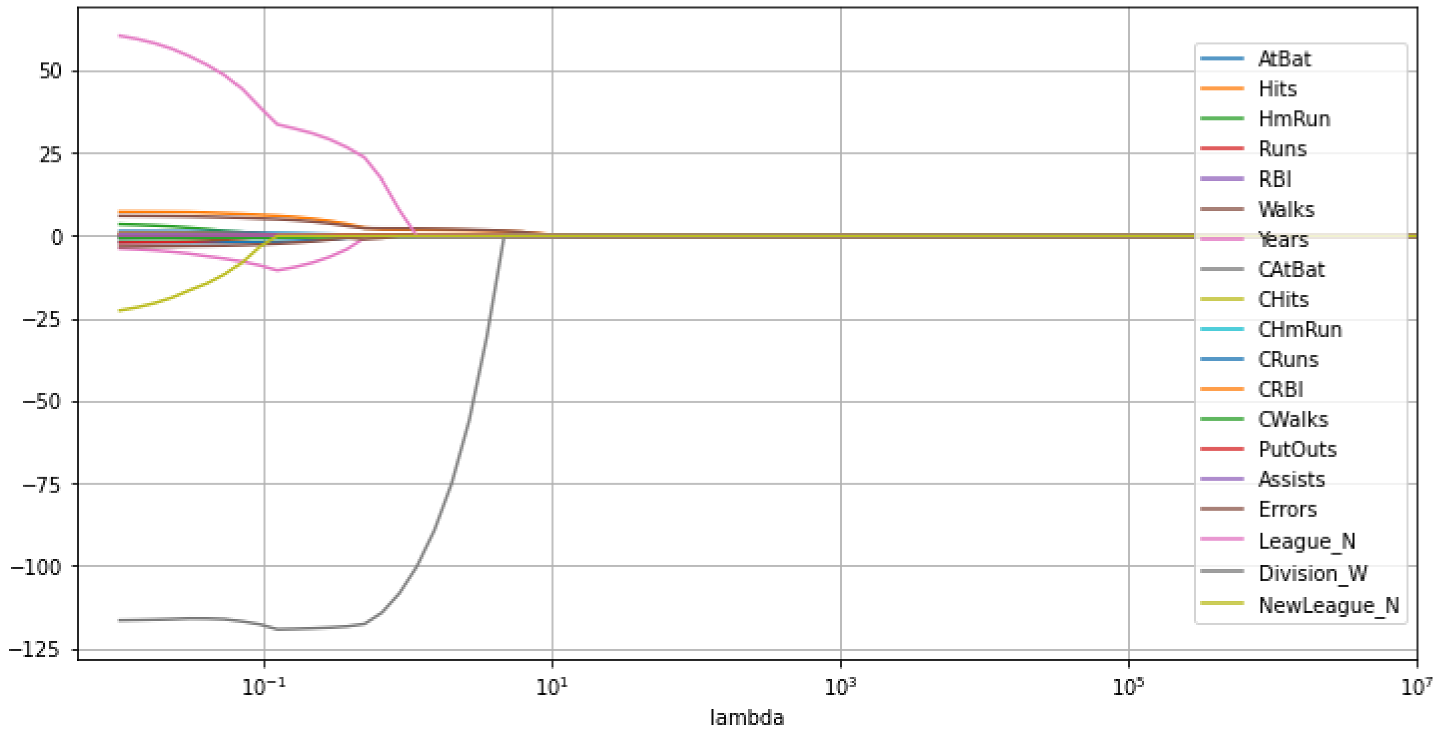
```
     AtBat  Hits  HmRun  Runs  RBI  Walks  Years  CAtBat  CHits  CHmRun  \
1      315    81      7    24   38     39     14    3449    835      69
2      479   130     18    66   72     76      3    1624    457      63
3      496   141     20    65   78     37     11    5628   1575     225
4      321    87     10    39   42     30      2     396    101      12
5      594   169      4    74   51     35     11    4408   1133      19
..     ...   ...    ...   ...  ...    ...    ...     ...    ...     ...
317    497   127      7    65   48     37      5    2703    806      32
318    492   136      5    76   50     94     12    5511   1511      39
319    475   126      3    61   43     52      6    1700    433       7
320    573   144      9    85   60     78      8    3198    857      97
321    631   170      9    77   44     31     11    4908   1457      30

     CRuns  CRBI  CWalks  PutOuts  Assists  Errors  Salary  League_N  \
1      321   414     375      632       43      10   475.0         1
2      224   266     263      880       82      14   480.0         0
3      828   838     354      200       11       3   500.0         1
4       48    46      33      805       40       4    91.5         1
5      501   336     194      282      421      25   750.0         0
..     ...   ...     ...      ...      ...     ...     ...       ...
317    379   311     138      325        9       3   700.0         1
318    897   451     875      313      381      20   875.0         0
319    217    93     146       37      113       7   385.0         0
320    470   420     332     1314      131      12   960.0         0
321    775   357     249      408        4       3  1000.0         0

     Division_W  NewLeague_N
1             1            1
2             1            0
3             0            1
4             0            1
5             1            0
..          ...          ...
317           0            1
318           0            0
319           1            0
320           0            0
321           1            0

[263 rows x 20 columns]
```

1b) Use the same array of 100 possible values of lambdas from 10^{-2} to 10^{10} that we used in the class example, create a plot of lasso coefficients as a function of lambda. Use the full dataset.

In [2]:

```
lambdaList = 10 ** numpy.linspace(10, -2, 100);

X_columnList = df1_na_dm.columns.drop("Salary");
X = df1_na_dm[X_columnList];
Y = df1_na_dm["Salary"];

coefList = [];
df1_na_dm_lasso_model = Lasso(normalize = True);

for l in lambdaList:
    df1_na_dm_lasso_model.set_params(alpha = l);
    df1_na_dm_lasso_model.fit(X, Y);
    coefList.append(df1_na_dm_lasso_model.coef_);
```

```
df_lasso_coef = pandas.DataFrame(coefList);
df_lasso_coef.columns = X_columnList;
df_lasso_coef.index = lambdaList;
df_lasso_coef.index.name = "lambda";

df_lasso_coef.plot(figsize = (12, 6), grid = True, logx = True, xlim = (0.01 / 2, 10 ** 7));
plt.legend(loc = "right");
```



1c) Using the LassoCV function and the full dataset, find the best possible lambda value for your dataset and the MSE associated with that model. Use a k-fold validation with 10 partitions.

In [3]:
```
df1_na_dm_lassocv_model = LassoCV(alphas = lambdaList, cv = 10, normalize = True);
df1_na_dm_lassocv_model.fit(X, Y);
print("best lambda: ", df1_na_dm_lassocv_model.alpha_);

Y_lassocv_pred_list = df1_na_dm_lassocv_model.predict(X);
lassocv_mse = mean_squared_error(Y, Y_lassocv_pred_list);
print("MSE: ", lassocv_mse);
```

```
best lambda:  0.16297508346206402
MSE:  94500.70207115081
```

1d) For the lasso model with the best lambda, which coefficients have been driven to 0? (5 points)

In [4]:
```
df1_na_dm_best_lasso_model = Lasso(alpha = df1_na_dm_lassocv_model.alpha_, normalize = True);
df1_na_dm_best_lasso_model.fit(X, Y);
df_best_lasso_coef = pandas.DataFrame(df1_na_dm_best_lasso_model.coef_, index = X_columnList, columns = ["Best_Lasso_Coef"]);
print(df_best_lasso_coef);
print("The following coefficients have been driven to 0: ");
print("HmRun, Runs, RBI, CAtBat, CHits, NewLeague_N");
```

```
             Best_Lasso_Coef
AtBat              -1.569100
Hits                5.715904
HmRun               0.000000
Runs               -0.000000
RBI                 0.000000
Walks               4.759498
Years              -9.468912
CAtBat             -0.000000
CHits               0.000000
CHmRun              0.539003
CRuns               0.668122
CRBI                0.381605
CWalks             -0.536725
PutOuts             0.273088
Assists             0.175832
Errors             -2.051931
League_N           32.289784
Division_W       -119.132165
NewLeague_N        -0.000000
The following coefficients have been driven to 0:
HmRun, Runs, RBI, CAtBat, CHits, NewLeague_N
```

1e) Which coefficients remain in the model (are non-zero)?

In [5]:
```
print("The following coefficients remain in the model: ");
print("AtBat, Hits, Walks, Years, CHmRun, CRuns, CRBI, CWalks, PutOuts, Assists, Errors, League_N, Division_W");
```

```
The following coefficients remain in the model:
AtBat, Hits, Walks, Years, CHmRun, CRuns, CRBI, CWalks, PutOuts, Assists, Errors, League_N, Division_W
```

1f) Fit a lasso model to this data using a lambda value of 5. For this model, which coefficients have been driven to 0? (5 points)

In [6]:
```
df1_na_dm_5lambda_lasso_model = Lasso(alpha = 5, normalize = True);
df1_na_dm_5lambda_lasso_model.fit(X, Y);
```

```
df_5lambda_lasso_coef = pandas.DataFrame(df1_na_dm_5lambda_lasso_model.coef_, index = X_columnList, columns = ["5Lambda_Lasso_Co
print(df_5lambda_lasso_coef);
print("The following coefficients have been driven to 0: ");
print("AtBat, HmRun, Runs, RBI, Years, CAtBat, CHits, CHmRun, CWalks, Assists, Errors, League_N, Division_W, NewLeague_N");
```

```
             5Lambda_Lasso_Coef
AtBat                  0.000000
Hits                   1.317257
HmRun                  0.000000
Runs                   0.000000
RBI                    0.000000
Walks                  1.433696
Years                  0.000000
CAtBat                 0.000000
CHits                  0.000000
CHmRun                 0.000000
CRuns                  0.142543
CRBI                   0.326828
CWalks                 0.000000
PutOuts                0.053166
Assists                0.000000
Errors                -0.000000
League_N               0.000000
Division_W            -0.000000
NewLeague_N            0.000000
The following coefficients have been driven to 0:
AtBat, HmRun, Runs, RBI, Years, CAtBat, CHits, CHmRun, CWalks, Assists, Errors, League_N, Division_W, NewLeague_N
```

1g) Which of these two models (best lambda and lambda = 5) had more coefficients driven to 0? Explain why that happened.

The Lasso model with lambda 5 has more coefficients driven to 0. Because larger lambda means larger penalty, if we want to minimize (the sum of residual sqaured + penalty), we need to turn more coefficients to 0.

Problem 2

2) For this probem, we are going to perform a best subset selection using the same dataset as in problem 1.

2a) How many different combinations of 5 predictors are there in this dataset? Hint: the comb() function in the Python math library can make this easy

In [7]:
```
df2 = pandas.read_csv("Hitters.csv");
df2_na = df2.dropna();
df2_na_dm = pandas.get_dummies(df2_na, columns = ["League", "Division", "NewLeague"], drop_first = True);

X_columnList = df2_na_dm.columns.drop("Salary");
X = df2_na_dm[X_columnList];
Y = df2_na_dm["Salary"];

nCk = math.comb(len(X_columnList), 5);
print("combinations: ", nCk);
```

```
combinations:  11628
```

2b) Using sklearn, fit a regression model to the first 5 predictors ('AtBat', 'Hits', 'HmRun', 'Runs', 'RBI'). What is the MSE of your resulting model (using the full dataset)? (10 points)

In [8]:
```
X_5pdt = df2_na_dm[["AtBat", "Hits", "HmRun", "Runs", "RBI"]];

df2_5pdt_lr_model = LinearRegression();
df2_5pdt_lr_model.fit(X_5pdt, Y);

Y_5pdt_lr_pred_list = df2_5pdt_lr_model.predict(X_5pdt);
fpdt_lr_mse = mean_squared_error(Y, Y_5pdt_lr_pred_list);
print("MSE: ", fpdt_lr_mse);
```

```
MSE:  153253.62757614415
```

2c) The itertools package provides a variety of "iterators" for use in loops. The combinations iterator provides an iterator that can be used in for loops, For example, here is the list of all of the combinations of 2 predictors in our dataset:

In [9]:
```
for predictor_comb in combinations(X_columnList, 2):
    print(predictor_comb);
```

```
('AtBat', 'Hits')
('AtBat', 'HmRun')
('AtBat', 'Runs')
('AtBat', 'RBI')
('AtBat', 'Walks')
('AtBat', 'Years')
('AtBat', 'CAtBat')
('AtBat', 'CHits')
('AtBat', 'CHmRun')
('AtBat', 'CRuns')
('AtBat', 'CRBI')
('AtBat', 'CWalks')
('AtBat', 'PutOuts')
('AtBat', 'Assists')
('AtBat', 'Errors')
('AtBat', 'League_N')
('AtBat', 'Division_W')
('AtBat', 'NewLeague_N')
('Hits', 'HmRun')
('Hits', 'Runs')
('Hits', 'RBI')
('Hits', 'Walks')
('Hits', 'Years')
('Hits', 'CAtBat')
('Hits', 'CHits')
('Hits', 'CHmRun')
('Hits', 'CRuns')
('Hits', 'CRBI')
('Hits', 'CWalks')
('Hits', 'PutOuts')
('Hits', 'Assists')
('Hits', 'Errors')
('Hits', 'League_N')
('Hits', 'Division_W')
('Hits', 'NewLeague_N')
('HmRun', 'Runs')
('HmRun', 'RBI')
('HmRun', 'Walks')
('HmRun', 'Years')
('HmRun', 'CAtBat')
('HmRun', 'CHits')
('HmRun', 'CHmRun')
('HmRun', 'CRuns')
('HmRun', 'CRBI')
('HmRun', 'CWalks')
('HmRun', 'PutOuts')
('HmRun', 'Assists')
('HmRun', 'Errors')
('HmRun', 'League_N')
('HmRun', 'Division_W')
('HmRun', 'NewLeague_N')
('Runs', 'RBI')
('Runs', 'Walks')
('Runs', 'Years')
('Runs', 'CAtBat')
('Runs', 'CHits')
('Runs', 'CHmRun')
('Runs', 'CRuns')
('Runs', 'CRBI')
('Runs', 'CWalks')
('Runs', 'PutOuts')
('Runs', 'Assists')
('Runs', 'Errors')
('Runs', 'League_N')
('Runs', 'Division_W')
('Runs', 'NewLeague_N')
('RBI', 'Walks')
('RBI', 'Years')
('RBI', 'CAtBat')
('RBI', 'CHits')
('RBI', 'CHmRun')
('RBI', 'CRuns')
('RBI', 'CRBI')
('RBI', 'CWalks')
('RBI', 'PutOuts')
('RBI', 'Assists')
('RBI', 'Errors')
('RBI', 'League_N')
('RBI', 'Division_W')
('RBI', 'NewLeague_N')
('Walks', 'Years')
('Walks', 'CAtBat')
('Walks', 'CHits')
('Walks', 'CHmRun')
('Walks', 'CRuns')
('Walks', 'CRBI')
('Walks', 'CWalks')
('Walks', 'PutOuts')
('Walks', 'Assists')
('Walks', 'Errors')
('Walks', 'League_N')
('Walks', 'Division_W')
```

```
('Walks', 'NewLeague_N')
('Years', 'CAtBat')
('Years', 'CHits')
('Years', 'CHmRun')
('Years', 'CRuns')
('Years', 'CRBI')
('Years', 'CWalks')
('Years', 'PutOuts')
('Years', 'Assists')
('Years', 'Errors')
('Years', 'League_N')
('Years', 'Division_W')
('Years', 'NewLeague_N')
('CAtBat', 'CHits')
('CAtBat', 'CHmRun')
('CAtBat', 'CRuns')
('CAtBat', 'CRBI')
('CAtBat', 'CWalks')
('CAtBat', 'PutOuts')
('CAtBat', 'Assists')
('CAtBat', 'Errors')
('CAtBat', 'League_N')
('CAtBat', 'Division_W')
('CAtBat', 'NewLeague_N')
('CHits', 'CHmRun')
('CHits', 'CRuns')
('CHits', 'CRBI')
('CHits', 'CWalks')
('CHits', 'PutOuts')
('CHits', 'Assists')
('CHits', 'Errors')
('CHits', 'League_N')
('CHits', 'Division_W')
('CHits', 'NewLeague_N')
('CHmRun', 'CRuns')
('CHmRun', 'CRBI')
('CHmRun', 'CWalks')
('CHmRun', 'PutOuts')
('CHmRun', 'Assists')
('CHmRun', 'Errors')
('CHmRun', 'League_N')
('CHmRun', 'Division_W')
('CHmRun', 'NewLeague_N')
('CRuns', 'CRBI')
('CRuns', 'CWalks')
('CRuns', 'PutOuts')
('CRuns', 'Assists')
('CRuns', 'Errors')
('CRuns', 'League_N')
('CRuns', 'Division_W')
('CRuns', 'NewLeague_N')
('CRBI', 'CWalks')
('CRBI', 'PutOuts')
('CRBI', 'Assists')
('CRBI', 'Errors')
('CRBI', 'League_N')
('CRBI', 'Division_W')
('CRBI', 'NewLeague_N')
('CWalks', 'PutOuts')
('CWalks', 'Assists')
('CWalks', 'Errors')
('CWalks', 'League_N')
('CWalks', 'Division_W')
('CWalks', 'NewLeague_N')
('PutOuts', 'Assists')
('PutOuts', 'Errors')
('PutOuts', 'League_N')
('PutOuts', 'Division_W')
('PutOuts', 'NewLeague_N')
('Assists', 'Errors')
('Assists', 'League_N')
('Assists', 'Division_W')
('Assists', 'NewLeague_N')
('Errors', 'League_N')
('Errors', 'Division_W')
('Errors', 'NewLeague_N')
('League_N', 'Division_W')
('League_N', 'NewLeague_N')
('Division_W', 'NewLeague_N')
```

2d) Using this combinations function and sklearn, construct a loop that fits a linear regression model to every possible combination of three predictors. For each combination of predictors, save the model, its predictors, and its MSE in a dataframe called "models" and display the first 10 rows of the models dataframe.

In [10]:
```python
tpdt_lr_model_list = [];
tpdt_list = [];
tpdt_lr_mse_list = [];

for pdt_cmb in combinations(X_columnList, 3):
    pdt_cmb_list = list(pdt_cmb);
    tpdt_list.append(pdt_cmb_list);
```

```
    X_3pdt = df2_na_dm[pdt_cmb_list];
    df2_3pdt_lr_model = LinearRegression().fit(X_3pdt, Y);
    tpdt_lr_model_list.append(df2_3pdt_lr_model);

    Y_3pdt_pred_list = df2_3pdt_lr_model.predict(X_3pdt);
    tpdt_lr_mse = mean_squared_error(Y, Y_3pdt_pred_list);
    tpdt_lr_mse_list.append(tpdt_lr_mse);

models = pandas.DataFrame(
    {
        "model": tpdt_lr_model_list,
        "predictor": tpdt_list,
        "mse": tpdt_lr_mse_list
    }
);

print(models.head(10));
```

```
                model              predictor            mse
0  LinearRegression()   [AtBat, Hits, HmRun]  156688.795581
1  LinearRegression()    [AtBat, Hits, Runs]  160409.737289
2  LinearRegression()     [AtBat, Hits, RBI]  153639.314956
3  LinearRegression()   [AtBat, Hits, Walks]  146818.048331
4  LinearRegression()   [AtBat, Hits, Years]  130532.248019
5  LinearRegression()  [AtBat, Hits, CAtBat]  120676.905337
6  LinearRegression()   [AtBat, Hits, CHits]  118979.738096
7  LinearRegression()  [AtBat, Hits, CHmRun]  117596.537771
8  LinearRegression()   [AtBat, Hits, CRuns]  115800.957920
9  LinearRegression()    [AtBat, Hits, CRBI]  113402.254897
```

2e) Find the best model (lowest MSE) and answer the following questions:

- What are the predictors?
- What is the MSE of the model?

In [11]:
```
min_mse_idx = models["mse"].idxmin();

min_mse_row = models.iloc[min_mse_idx];
print("The predictors are: ", min_mse_row["predictor"]);
print("The MSE is: ", min_mse_row["mse"]);
```

```
The predictors are:  ['Hits', 'CRBI', 'PutOuts']
The MSE is:  111214.05648618752
```

2f) Now, use your code to create a function get_best_model(k) which takes the number of predictors as an input (k) and returns a list consisting of the model, its predictors, and its corresponding MSE for the best model with k predictors. Call the function with k = 3 to test that it returns the same results (predictors and model MSE) as your answer to part e.

In [12]:
```
def get_best_model(k):
    kpdt_lr_model_list = [];
    kpdt_list = [];
    kpdt_lr_mse_list = [];

    kpdt_min_mse = 2 * 10 ** 9;
    kpdt_min_mse_idx = -1;
    cmb_list = list(combinations(X_columnList, k));
    for idx in range(len(cmb_list)):
        pdt_cmb = cmb_list[idx];
        pdt_cmb_list = list(pdt_cmb);
        kpdt_list.append(pdt_cmb_list);

        X_kpdt = df2_na_dm[pdt_cmb_list];
        df2_kpdt_lr_model = LinearRegression().fit(X_kpdt, Y);
        kpdt_lr_model_list.append(df2_kpdt_lr_model);

        Y_kpdt_pred_list = df2_kpdt_lr_model.predict(X_kpdt);
        kpdt_lr_mse = mean_squared_error(Y, Y_kpdt_pred_list);
        kpdt_lr_mse_list.append(kpdt_lr_mse);

        if kpdt_lr_mse < kpdt_min_mse:
            kpdt_min_mse = kpdt_lr_mse;
            kpdt_min_mse_idx = idx;

    return pandas.DataFrame(
        {
            "model": [kpdt_lr_model_list[kpdt_min_mse_idx]],
            "predictor": [kpdt_list[kpdt_min_mse_idx]],
            "mse": [kpdt_lr_mse_list[kpdt_min_mse_idx]]
        }
    );

models2 = get_best_model(3);
print(models2);
```

```
                model              predictor            mse
0  LinearRegression()  [Hits, CRBI, PutOuts]  111214.056486
```

2g) We are going to wish to evaluate the candidate models using AIC, BIC, and Adjusted R^2 which are not available in sklearn. Re-write you function from part f above using statsmodels instead of sklearn. Call the function with k = 3 to test that it returns the same results (predictors and model MSE) as your answers to parts e and f.

In [13]:
```
def get_best_model2(k):
    kpdt_list = [];
    kpdt_lr_model_list = [];
    kpdt_lr_mse_list = [];

    kpdt_min_mse = 2 * 10 ** 9;
    kpdt_min_mse_idx = -1;
    cmb_list = list(combinations(X_columnList, k));
    for idx in range(len(cmb_list)):
        pdt_cmb = cmb_list[idx];
        pdt_cmb_list = list(pdt_cmb);
        kpdt_list.append(pdt_cmb_list);

        X_kpdt = df2_na_dm[pdt_cmb_list];
        X_kpdt = sm.add_constant(X_kpdt);
        df2_kpdt_lr_model = sm.OLS(Y, X_kpdt).fit();
        kpdt_lr_model_list.append(df2_kpdt_lr_model);

        Y_kpdt_pred_list = df2_kpdt_lr_model.predict(X_kpdt);
        kpdt_lr_mse = mean_squared_error(Y, Y_kpdt_pred_list);
        kpdt_lr_mse_list.append(kpdt_lr_mse);

        if kpdt_lr_mse < kpdt_min_mse:
            kpdt_min_mse = kpdt_lr_mse;
            kpdt_min_mse_idx = idx;

    return pandas.DataFrame(
        {
            "model": [kpdt_lr_model_list[kpdt_min_mse_idx]],
            "predictor": [kpdt_list[kpdt_min_mse_idx]],
            "mse": [kpdt_lr_mse_list[kpdt_min_mse_idx]]
        }
    );

models3 = get_best_model2(3);
print(models3);
```

```
                                               model               predictor  \
0  <statsmodels.regression.linear_model.Regressio...  [Hits, CRBI, PutOuts]

             mse
0  111214.056486
```

2h) Set up a loop to call this function for every number of susbsets between 1 and 18 and store the results in a dataframe called models_best. This will take a long time to run (about 40 minutes on my computer), so I recommend that you test it using only a relatively small number of subsets (maybe 1 to 5) before running it on the full 19. Display the resulting models_best dataframe.

In [14]:
```
models_best_list = [];

for k in range(1, 18 + 1):
    models_best_list.append(get_best_model2(k));

models_best = pandas.concat(models_best_list);
print(models_best);
```

```
                                               model  \
0  <statsmodels.regression.linear_model.Regressio...
0  <statsmodels.regression.linear_model.Regressio...
0  <statsmodels.regression.linear_model.Regressio...
0  <statsmodels.regression.linear_model.Regressio...
0  <statsmodels.regression.linear_model.Regressio...
0  <statsmodels.regression.linear_model.Regressio...
0  <statsmodels.regression.linear_model.Regressio...
0  <statsmodels.regression.linear_model.Regressio...
0  <statsmodels.regression.linear_model.Regressio...
0  <statsmodels.regression.linear_model.Regressio...
0  <statsmodels.regression.linear_model.Regressio...
0  <statsmodels.regression.linear_model.Regressio...
0  <statsmodels.regression.linear_model.Regressio...
0  <statsmodels.regression.linear_model.Regressio...
0  <statsmodels.regression.linear_model.Regressio...
0  <statsmodels.regression.linear_model.Regressio...
0  <statsmodels.regression.linear_model.Regressio...
0  <statsmodels.regression.linear_model.Regressio...

                                           predictor            mse
0                                             [CRBI]  137565.320361
0                                       [Hits, CRBI]  116526.843690
0                              [Hits, CRBI, PutOuts]  111214.056486
0                  [Hits, CRBI, PutOuts, Division_W]  106353.048729
0           [AtBat, Hits, CRBI, PutOuts, Division_W]  103231.556776
0    [AtBat, Hits, Walks, CRBI, PutOuts, Division_W]   99600.395162
0  [Hits, Walks, CAtBat, CHits, CHmRun, PutOuts, ...   98503.982892
0  [AtBat, Hits, Walks, CHmRun, CRuns, CWalks, Pu...   95577.680376
0  [AtBat, Hits, Walks, CAtBat, CRuns, CRBI, CWal...   94350.005272
0  [AtBat, Hits, Walks, CAtBat, CRuns, CRBI, CWal...   93157.420296
0  [AtBat, Hits, Walks, CAtBat, CRuns, CRBI, CWal...   92727.547724
0  [AtBat, Hits, Runs, Walks, CAtBat, CRuns, CRBI...   92521.796119
0  [AtBat, Hits, Runs, Walks, CAtBat, CRuns, CRBI...   92354.174290
0  [AtBat, Hits, HmRun, Runs, Walks, CAtBat, CRun...   92200.229630
0  [AtBat, Hits, HmRun, Runs, Walks, CAtBat, CHit...   92148.963328
0  [AtBat, Hits, HmRun, Runs, RBI, Walks, CAtBat,...   92088.887730
0  [AtBat, Hits, HmRun, Runs, RBI, Walks, CAtBat,...   92051.128352
0  [AtBat, Hits, HmRun, Runs, RBI, Walks, Years, ...   92022.195280
```

2i) Create and populate a dataframe best_model_stats that has the following columns: "Model Size", "R2", "Adjusted R2", "AIC", "BIC". Display the dataframe.

In [15]:

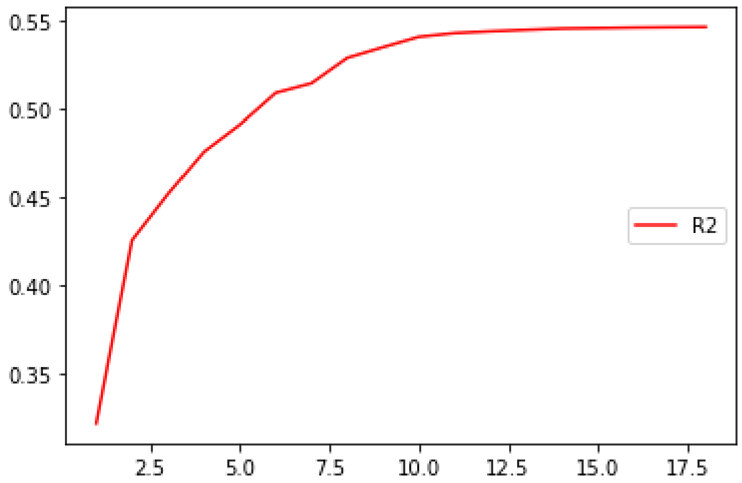
```python
model_size_list = [];
r_squared_list = [];
adjusted_r_sqaured_list = [];
aic_list = [];
bic_list = [];

for idx, row in models_best.iterrows():
    model_size_list.append(len(row["predictor"]));
    r_squared_list.append(row["model"].rsquared);
    adjusted_r_sqaured_list.append(row["model"].rsquared_adj);
    aic_list.append(row["model"].aic);
    bic_list.append(row["model"].bic);

best_model_stats = pandas.DataFrame(
    {
        "Model Size": model_size_list,
        "R2": r_squared_list,
        "Adjusted R2": adjusted_r_sqaured_list,
        "AIC": aic_list,
        "BIC": bic_list
    }
);
print(best_model_stats);
```

```
    Model Size        R2  Adjusted R2          AIC          BIC
0            1  0.321450     0.318850  3862.139307  3869.283615
1            2  0.425224     0.420802  3820.487305  3831.203767
2            3  0.451429     0.445075  3810.214440  3824.503056
3            4  0.475407     0.467273  3800.460294  3818.321064
4            5  0.490804     0.480897  3794.625624  3816.058548
5            6  0.508715     0.497200  3787.208000  3812.213078
6            7  0.514123     0.500785  3786.296813  3814.874046
7            8  0.528557     0.513708  3780.365349  3812.514735
8            9  0.534612     0.518057  3778.965286  3814.686826
9           10  0.540495     0.522261  3777.619775  3816.913469
10          11  0.542615     0.522571  3778.403359  3821.269208
11          12  0.543630     0.521724  3779.819145  3826.257147
12          13  0.544457     0.520674  3781.342235  3831.352392
13          14  0.545216     0.519543  3782.903476  3836.485787
14          15  0.545469     0.517866  3784.757199  3841.911664
15          16  0.545766     0.516222  3786.585683  3847.312301
16          17  0.545952     0.514446  3788.477822  3852.776595
17          18  0.546095     0.512610  3790.395144  3858.266071
```

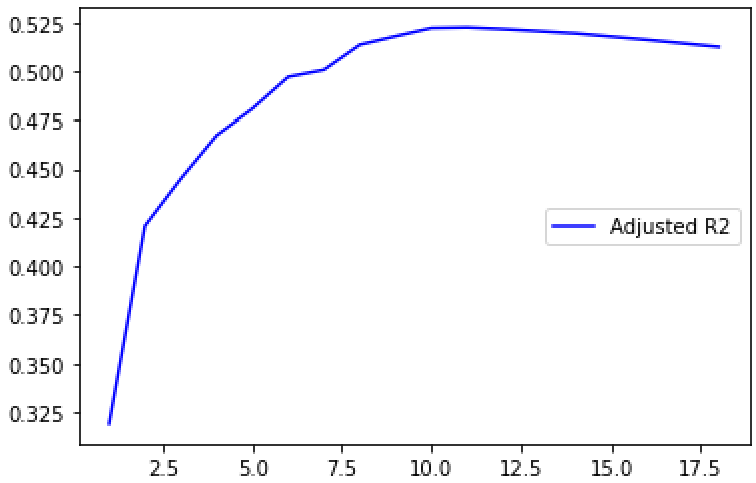
2j) Create four line plots showing the four assessment statistics as a function of model size (5 points)

In [16]:

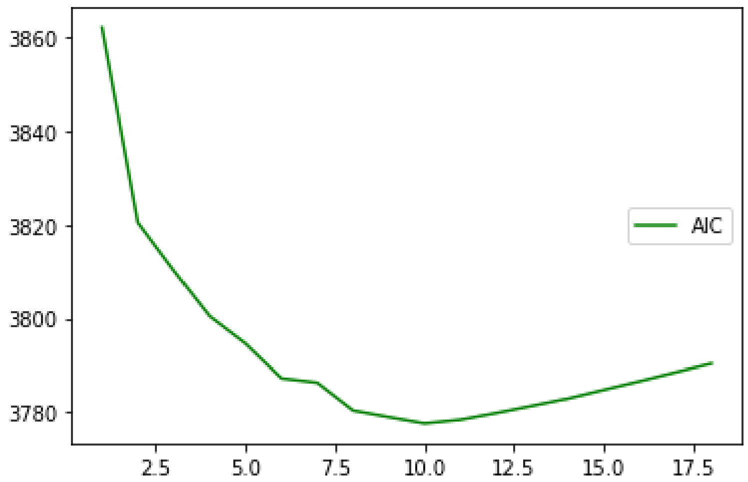
```python
plt.plot(best_model_stats["Model Size"], best_model_stats["R2"], color = "red", label = "R2");
plt.legend(loc = "right");
```



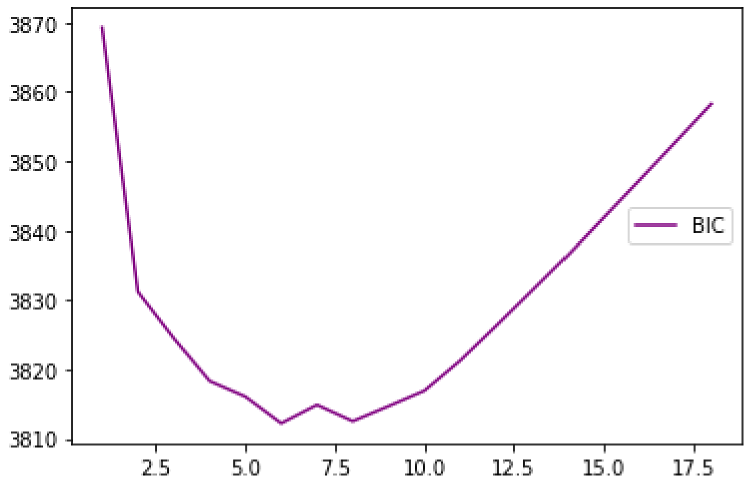
In [17]:
```
plt.plot(best_model_stats["Model Size"], best_model_stats["Adjusted R2"], color = "blue", label = "Adjusted R2");
plt.legend(loc = "right");
```



In [18]:
```
plt.plot(best_model_stats["Model Size"], best_model_stats["AIC"], color = "green", label = "AIC");
plt.legend(loc = "right");
```



In [19]:
```
plt.plot(best_model_stats["Model Size"], best_model_stats["BIC"], color = "purple", label = "BIC");
plt.legend(loc = "right");
```



2k. Based on this data, which model size would you select for each of the three assessment statistics (not including R2)? (5 points)

In [20]:
```
max_r2_idx = best_model_stats["R2"].idxmax();
max_r2_adj_idx = best_model_stats["Adjusted R2"].idxmax();
min_aic_idx = best_model_stats["AIC"].idxmin();
min_bic_idx = best_model_stats["BIC"].idxmin();

print("R2: ");
print(best_model_stats.iloc[max_r2_idx]);
print("Adjusted R2: ");
print(best_model_stats.iloc[max_r2_adj_idx]);
print("AIC: ");
print(best_model_stats.iloc[min_aic_idx]);
print("BIC: ");
print(best_model_stats.iloc[min_bic_idx]);
```

```
R2:
Model Size       18.000000
R2                0.546095
Adjusted R2       0.512610
AIC            3790.395144
BIC            3858.266071
Name: 17, dtype: float64
Adjusted R2:
Model Size       11.000000
R2                0.542615
Adjusted R2       0.522571
AIC            3778.403359
BIC            3821.269208
Name: 10, dtype: float64
AIC:
Model Size       10.000000
R2                0.540495
Adjusted R2       0.522261
AIC            3777.619775
BIC            3816.913469
Name: 9, dtype: float64
BIC:
Model Size        6.000000
R2                0.508715
Adjusted R2       0.497200
AIC            3787.208000
BIC            3812.213078
Name: 5, dtype: float64
```

- Adjusted R^2: 11
- AIC: 10
- BIC: 6

2l. How does this compare to the number of predictors using the Lasso "best model" that you found in question 1? (5 points)

The predictor number of each three assessment statistics is less than that of the Lasso "best model". This indicates the Lasso is not accurated as the best subset selection, because the best subset selection tries all possibile predictor combinations, while the Lasso just introduces bias. However, the subset selection is very time-comsuming and memory-comsuming, while the Lasso just uses a little resourse.