Assignment 5 Complete

March 7, 2022

Instructions:

a. You may talk to a friend, discuss the questions and potential directions for solving them. However, you need to write your own solutions and code separately, and not as a group activity.

b. Do not write your name on the assignment. (1 point)

c. Export your Jupyter notebook as a PDF file. If you get an error, make sure you have downloaded the MikTex software (for windows) or MacTex (for mac). Note that after installing MikTex/MacTex, you will need to check for updates, install the updates if needed, and re-start your system. Submit the PDF file. (1 point)

d. Please include each question (or question number) followed by code and your answer (if applicable). Write your code in the 'Code' cells and your answer in the 'Markdown' cells of the Jupyter notebook. Ensure that the solution is well-organized, clear, and concise (3 points)

- 1. It's easy enough to identify different sections of the homework assignment (e.g., if there are different sections of an assignment, they're clearly distinguishable by section headers or the like)

- 2. It's clear which code/markdown blocks correspond to which questions.

- 3. There aren't excessively long outputs of extraneous information (e.g., no printouts of entire data frames without good reason)

e. This assignment is **due at 11:59pm on Monday, March 7th**. Good luck!

f. There are two parts in this assignment - Part 1 and Part 2. You have the **option to choose any one part** out of the two and submit the assignment. Each part is worth 35 points. Note that both parts are relevant for the final exam. However, for the purpose of this assignment you need to do any one part of your choice. If you do both parts, only the first part will be graded.

Submissions will be graded with a maximum of **40 points** – 35 points for code & answers, 5 points for anonymity and proper formatting.

Part 1

The datasets *ENB2012_Train.csv* and *ENB2012_Test.csv* provide data on energy analysis using 12 different building shapes simulated in Ecotect. The buildings differ with respect to the glazing area, the glazing area distribution, and the orientation, amongst other parameters.

1. X1 Relative Compactness
2. X2 Surface Area
3. X3 Wall Area
4. X4 Roof Area
5. X5 Overall Height
6. X6 Orientation
7. X7 Glazing Area
8. X8 Glazing Area Distribution
9. y1 Heating Load

For this part, if the 3 criteria suggest different numbers of predictors, consider the best model as per the BIC criterion.

Q1a

Suppose that we want to implement the best subset selection algorithm to find the first order predictors (X1-X8) that can predict heating load (y1). How many models for E(y1) are possible, if the model includes (i) one variable, (ii) three variables, and (iii) eight variables? Show your steps without running any code.

Note: The notation E(y1) means the expected value of y1 or the mean of y1

(3 points for answer)

(i) $\binom{8}{1} = 8$, (ii) $\binom{8}{3} = 56$, (iii) $\binom{8}{8} = 1$

Q1b

Implement the best subset selection algorithm to find the "best" first-order predictors of heating load (y1). Print out the model summary.
Use *ENB2012_Train.csv* and consider only the first-order terms.

(4 points for code)

[69]:
```python
import pandas as pd
import numpy as np
import statsmodels.formula.api as sm
import seaborn as sns
import matplotlib.pyplot as plt
import itertools
import time
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Ridge, RidgeCV, Lasso, LassoCV
```

```
[3]: ENB_train = pd.read_csv('ENB2012_Train.csv')
     ENB_test = pd.read_csv('ENB2012_Test.csv')
     ENB_train.head()
```

```
[3]:      X1     X2     X3      X4   X5  X6   X7  X8     Y1
     0  0.98  514.5  294.0  110.25  7.0   2  0.0   0  15.55
     1  0.98  514.5  294.0  110.25  7.0   3  0.0   0  15.55
     2  0.98  514.5  294.0  110.25  7.0   5  0.0   0  15.55
     3  0.90  563.5  318.5  122.50  7.0   2  0.0   0  20.84
     4  0.90  563.5  318.5  122.50  7.0   3  0.0   0  21.46
```

```
[4]: l= list(ENB_train.columns)
     l.remove('Y1')
     X=ENB_train[l]
```

```
[5]: #Function to develop a model based on all predictors in predictor_subset
     def processSubset(predictor_subset):
     # Fit model on feature_set and calculate R-squared
         model = sm.ols('Y1~' + '+'.join(predictor_subset),data = ENB_train).fit()
         Rsquared = model.rsquared
         return {"model":model, "Rsquared":Rsquared}
```

```
[6]: #Function to select the best model amongst all models with 'k' predictors
     def getBest_model(k):
         tic = time.time()
         results = []
         for combo in itertools.combinations(X.columns, k):
             results.append(processSubset(list(combo)))
         # Wrap everything up in a dataframe
         models = pd.DataFrame(results)
         # Choose the model with the highest RSS
         best_model = models.loc[models['Rsquared'].argmax()]
         toc = time.time()
         print("Processed", models.shape[0], "models on", k, "predictors␣
       ↪in",(toc-tic), "seconds.")
         return best_model
```

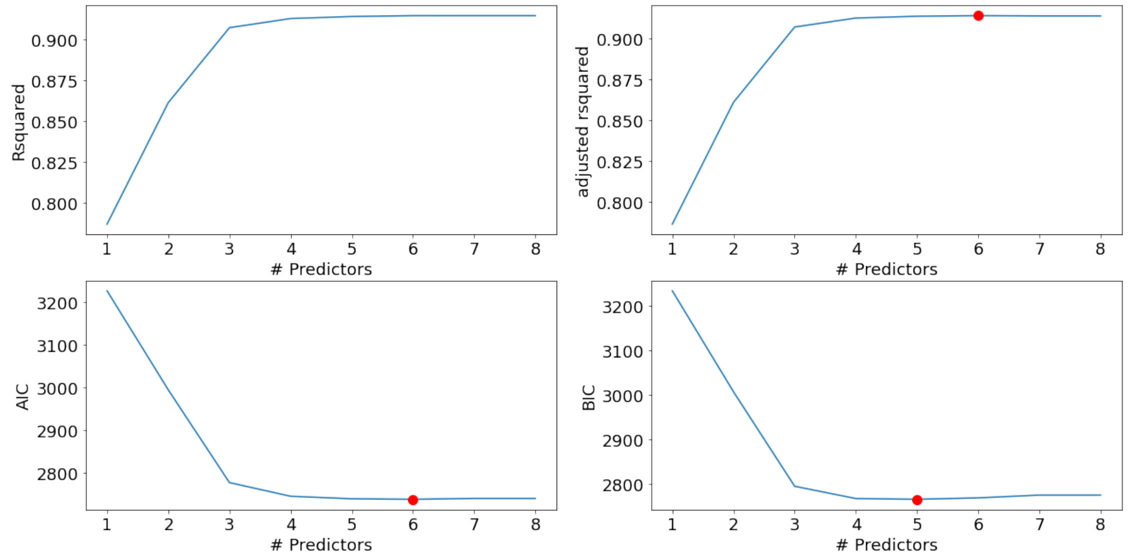
```
[7]: #Function to select the best model amongst the best models for 'k'␣
       ↪predictors,␣where k = 1,2,3,..
     models_best = pd.DataFrame(columns=["Rsquared", "model"])

     tic = time.time()
     for i in range(1,1+X.shape[1]):
         models_best.loc[i] = getBest_model(i)
     toc = time.time()
```

```
print("Total elapsed time:", (toc-tic), "seconds.")
```

```
Processed 8 models on 1 predictors in 0.044007301330566406 seconds.
Processed 28 models on 2 predictors in 0.11554336547851562 seconds.
Processed 56 models on 3 predictors in 0.3001894950866699 seconds.
Processed 70 models on 4 predictors in 0.47556424140930176 seconds.
Processed 56 models on 5 predictors in 0.43196749687194824 seconds.
Processed 28 models on 6 predictors in 0.20800161361694336 seconds.
Processed 8 models on 7 predictors in 0.06399798393249512 seconds.
Processed 1 models on 8 predictors in 0.00800013542175293 seconds.
Total elapsed time: 1.686894416809082 seconds.
```

[8]:
```
def best_sub_plots():
    plt.figure(figsize=(20,10))
    plt.rcParams.update({'font.size': 18, 'lines.markersize': 10})
    # Set up a 2x2 grid so we can look at 4 plots at once
    plt.subplot(2, 2, 1)
    # We will now plot a red dot to indicate the model with the largest␣
 ↪adjusted R^2 statistic.
    # The argmax() function can be used to identify the location of the maximum␣
 ↪point of a vector
    plt.plot(models_best["Rsquared"])
    plt.xlabel('# Predictors')
    plt.ylabel('Rsquared')
    # We will now plot a red dot to indicate the model with the largest␣
 ↪adjusted R^2 statistic.
    # The argmax() function can be used to identify the location of the maximum␣
 ↪point of a vector
    rsquared_adj = models_best.apply(lambda row: row[1].rsquared_adj, axis=1)
    plt.subplot(2, 2, 2)
    plt.plot(rsquared_adj)
    plt.plot(1+rsquared_adj.argmax(), rsquared_adj.max(), "or")
    plt.xlabel('# Predictors')
    plt.ylabel('adjusted rsquared')
    # We'll do the same for AIC and BIC, this time looking for the models with␣
 ↪the SMALLEST statistic
    aic = models_best.apply(lambda row: row[1].aic, axis=1)
    plt.subplot(2, 2, 3)
    plt.plot(aic)
    plt.plot(1+aic.argmin(), aic.min(), "or")
    plt.xlabel('# Predictors')
    plt.ylabel('AIC')
    bic = models_best.apply(lambda row: row[1].bic, axis=1)
    plt.subplot(2, 2, 4)
    plt.plot(bic)
    plt.plot(1+bic.argmin(), bic.min(), "or")
    plt.xlabel('# Predictors')
    plt.ylabel('BIC')
best_sub_plots()
```



```
[9]: best_subset_model = models_best['model'][5]
     best_subset_model.summary()
```

```
[9]: <class 'statsmodels.iolib.summary.Summary'>
     """
                                 OLS Regression Results
     ==============================================================================
     Dep. Variable:                     Y1   R-squared:                       0.914
     Model:                            OLS   Adj. R-squared:                  0.914
     Method:                 Least Squares   F-statistic:                     1144.
     Date:                Thu, 24 Feb 2022   Prob (F-statistic):          2.87e-283
     Time:                        03:39:33   Log-Likelihood:                -1363.5
     No. Observations:                 542   AIC:                             2739.
     Df Residuals:                     536   BIC:                             2765.
     Df Model:                           5
     Covariance Type:            nonrobust
     ==============================================================================
                      coef    std err          t      P>|t|      [0.025      0.975]
     ------------------------------------------------------------------------------
     Intercept     46.1893      5.212      8.862      0.000      35.950      56.428
     X1           -45.8814      2.567    -17.876      0.000     -50.923     -40.839
     X4            -0.1078      0.014     -7.539      0.000      -0.136      -0.080
     X5             4.7438      0.318     14.902      0.000       4.118       5.369
     X7            20.1466      0.990     20.357      0.000      18.203      22.091
     X8             0.2421      0.086      2.809      0.005       0.073       0.411
```

```
==============================================================================
Omnibus:                        9.613   Durbin-Watson:                   0.869
Prob(Omnibus):                  0.008   Jarque-Bera (JB):               15.883
Skew:                          -0.031   Prob(JB):                     0.000356
Kurtosis:                       3.836   Cond. No.                     7.80e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 7.8e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
"""
```

The best predictors according to best subset selection algorithm are x1, x2, x3, x5, x7, x8.

Q1c

Should R-squared be used to select from among a set of models with different numbers of predictors? Justify your answer.

(1 point for answer, 2 points for justification)

No, R-squared cannot be used to select from among a set of models with different numbers of predictors. Since R-squared increases monotonically as the number of predictors increase, so it would always lead to selecting the model with all the predictors.

Q1d

Calculate the RMSE of the model found in (b). Compare it with the RMSE of the model using all first-order predictors. You will find that the two RMSEs are similar. Seems like the best subset model didn't help improve prediction. However, did it help improve inference? Justify your answer.

Hint: VIF!

(2 points for code, 2 points for answer)

[10]:
```
pred_y1 = best_subset_model.predict(ENB_test)
np.sqrt(((pred_y1 - ENB_test.Y1)**2).mean())
```

[10]:
```
2.778167195091445
```

[11]:
```
model = sm.ols('Y1~' + '+'.join(X.columns),data = ENB_train).fit()
pred_y1 = model.predict(ENB_test)
np.sqrt(((pred_y1 - ENB_test.Y1)**2).mean())
```

[11]:
```
2.774159872401328
```

[12]:
```
model.summary()
```

```
[12]: <class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                     Y1   R-squared:                       0.915
Model:                            OLS   Adj. R-squared:                  0.914
Method:                 Least Squares   F-statistic:                     819.5
Date:                Thu, 24 Feb 2022   Prob (F-statistic):          6.84e-281
Time:                        03:40:46   Log-Likelihood:                -1361.9
No. Observations:                 542   AIC:                             2740.
Df Residuals:                     534   BIC:                             2774.
Df Model:                           7
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     86.8999     23.677      3.670      0.000      40.389     133.410
X1           -68.0136     12.782     -5.321      0.000     -93.122     -42.905
X2            -0.0642      0.017     -3.842      0.000      -0.097      -0.031
X3             0.0363      0.005      7.617      0.000       0.027       0.046
X4            -0.0503      0.009     -5.335      0.000      -0.069      -0.032
X5             4.2704      0.417     10.243      0.000       3.451       5.089
X6             0.0223      0.118      0.189      0.850      -0.209       0.254
X7            20.1554      0.989     20.386      0.000      18.213      22.098
X8             0.2430      0.086      2.822      0.005       0.074       0.412
==============================================================================
Omnibus:                       10.570   Durbin-Watson:                   0.892
Prob(Omnibus):                  0.005   Jarque-Bera (JB):               18.131
Skew:                          -0.039   Prob(JB):                     0.000116
Kurtosis:                       3.893   Cond. No.                     2.31e+16
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The smallest eigenvalue is 5.97e-25. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
"""
```

The RMSE of the best subset model is close to the RMSE of the model with all the predictors. Note that X2, X3 and X6 are excluded in the best subset model. X6 is statistically insignificant, so it is not a useful predictor and is removed in the best subset. If we compute the VIF with either X2 or X3 included in the model, we find that X2 and X3 are multicollinear with other predictors in the model. Thus, they do not help explain any additional variation in the response, given other predictors. So, they are not necessary and that is why not there in the best subset model. Thus, the best subset model eliminates the issue of multicollinearity, and provides a more accurate inference.

[13]:
```
def vif(X_vif):
    X_vif = add_constant(X_vif)
    vif_data = pd.DataFrame()
    vif_data["feature"] = X_vif.columns

    for i in range(len(X_vif.columns)):
        vif_data.loc[i,'VIF'] = variance_inflation_factor(X_vif.values, i)

    print(vif_data)
```

[14]:
```
#VIF with X2 in the model
X_vif=X.iloc[:,[0,1,3,4,6,7]]
vif(X_vif)
```

```
  feature           VIF
0   const  33578.814492
1      X1    110.732796
2      X2    116.487267
3      X4     33.819552
4      X5     31.863278
5      X7      1.050550
6      X8      1.048236
```

[15]:
```
#VIF with X3 in the model
X_vif=X.iloc[:,[0,2,3,4,6,7]]
vif(X_vif)
```

```
  feature           VIF
0   const  33578.814492
1      X1    110.732796
2      X3     26.876039
3      X4    220.820817
4      X5     31.863278
5      X7      1.050550
6      X8      1.048236
```

[16]:
```
#VIF with X2 and X3 removed from the model
X_vif=X.iloc[:,[0,3,4,6,7]]
vif(X_vif)
```

```
  feature          VIF
0   const  1623.968159
1      X1     4.457484
2      X4    24.792621
3      X5    18.546746
4      X7     1.050511
5      X8     1.048155
```

Q1e

Let us consider adding all the 2-factor interactions of the predictors in the model. Please answer the following questions without running code.
i) How many predictors do we have in total?
ii) Assume best subset selection is used. How many models are fitted in total?
iii) Assume forward selection is used. How many models are fitted in total?
iv) Assume backward selection is used. How many models are fitted in total?
v) How many models will be developed in the iteration that contains exactly 10 predictors in each model – for best subsets, fwd/bwd regression?
vi) What approach would you choose for variable selection (amonst best subset, forward selection, backward selection)?

(8 points for answer)

i) $8 + \binom{8}{2} = 8 + 28 = 36$

ii) $2^p = 2^{36} = 68,719,476,736$

iii) $1 + p(p+1)/2 = 1 + 36 * 37/2 = 667$

iv) $1 + p(p+1)/2 = 1 + 36 * 37/2 = 667$

v) Best subset: 10 predictors, $\binom{36}{10} = 254,186,856$
Fwd selection: 10 predictors, p-k models for k+1 predictors, then 10 predictors corresponds to $k = 9$, $p - k = 36 - 9 = 27$
Bwd selection: 10 predictors, we consider k models that contains k+1 predictors, we have $10 + 1 = 11$ models that contains 10 predictors

vi) We should use forward or backward selection because best subset selection is very computationally heavy in this case.

Q1f

Use forward selection to find the "best" first-order predictors and 2-factor interactions of the predictors of y1 (Heating Load). Print out the model summary.

(5 points for code)

[20]:
```
X_backup=ENB_train[l]

for combo in itertools.combinations(X_backup.columns, 2):
    ENB_train['_'.join(combo)] = ENB_train[combo[0]]*ENB_train[combo[1]]
    ENB_test['_'.join(combo)] = ENB_test[combo[0]]*ENB_test[combo[1]]
    X.loc[:,'_'.join(combo)] = ENB_train.loc[:,'_'.join(combo)]
```

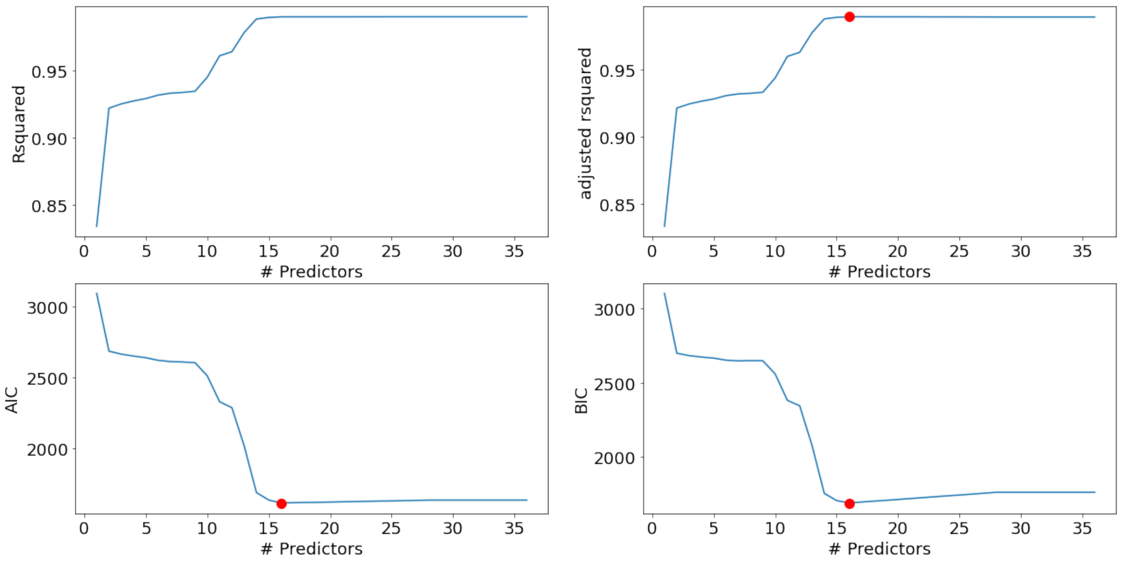
[19]:
```
#Function to find the best predictor out of p-k predictors and add it to the␣
↪model containing the k predictors
def forward(predictors):
    # Pull out predictors we still need to process
    remaining_predictors = [p for p in X.columns if p not in predictors]
    tic = time.time()
    results = []
    for p in remaining_predictors:
        results.append(processSubset(predictors+[p]))
    # Wrap everything up in a nice dataframe
    models = pd.DataFrame(results)
    # Choose the model with the highest RSS
    best_model = models.loc[models['Rsquared'].argmax()]
    toc = time.time()
    print("Processed ", models.shape[0], "models on",␣
 ↪len(predictors)+1,"predictors in", (toc-tic), "seconds.")
    # Return the best model, along with some other useful information about the␣
 ↪model
    return best_model
```

[21]:
```
def forward_selection():
    models_best = pd.DataFrame(columns=["Rsquared", "model"])
    tic = time.time()
    predictors = []
    for i in range(1,len(X.columns)+1):
        models_best.loc[i] = forward(predictors)
        predictors = list(models_best.loc[i]["model"].params.index[1:])
    toc = time.time()
    print("Total elapsed time:", (toc-tic), "seconds.")
    return models_best
```

[22]:
```
models_best = forward_selection()
```

```
Processed  36 models on 1 predictors in 0.197035551071167 seconds.
Processed  35 models on 2 predictors in 0.15621089935302734 seconds.
Processed  34 models on 3 predictors in 0.1815474033355713 seconds.
Processed  33 models on 4 predictors in 0.2255539894104004 seconds.
Processed  32 models on 5 predictors in 0.2816908359527588 seconds.
Processed  31 models on 6 predictors in 0.2708756923675537 seconds.
Processed  30 models on 7 predictors in 0.2542533874511719 seconds.
Processed  29 models on 8 predictors in 0.328472375869751 seconds.
Processed  28 models on 9 predictors in 0.3839991092681885 seconds.
Processed  27 models on 10 predictors in 0.31999826431274414 seconds.
Processed  26 models on 11 predictors in 0.3524012565612793 seconds.
Processed  25 models on 12 predictors in 0.3279991149902344 seconds.
Processed  24 models on 13 predictors in 0.3922576904296875 seconds.
Processed  23 models on 14 predictors in 0.316530704498291 seconds.
Processed  22 models on 15 predictors in 0.34543752670288086 seconds.
Processed  21 models on 16 predictors in 0.3589646816253662 seconds.
Processed  20 models on 17 predictors in 0.37851786613464355 seconds.
Processed  19 models on 18 predictors in 0.4036996364593506 seconds.
Processed  18 models on 19 predictors in 0.40822577476501465 seconds.
Processed  17 models on 20 predictors in 0.38858795166015625 seconds.
Processed  16 models on 21 predictors in 0.37599873542785645 seconds.
Processed  15 models on 22 predictors in 0.39428043365478516 seconds.
Processed  14 models on 23 predictors in 0.34958434104919434 seconds.
Processed  13 models on 24 predictors in 0.32555556297302246 seconds.
Processed  12 models on 25 predictors in 0.3119509220123291 seconds.
Processed  11 models on 26 predictors in 0.2833878993988037 seconds.
Processed  10 models on 27 predictors in 0.3082747459411621 seconds.
Processed  9 models on 28 predictors in 0.24689698219299316 seconds.
Processed  8 models on 29 predictors in 0.2558615207672119 seconds.
Processed  7 models on 30 predictors in 0.2569773197174072 seconds.
Processed  6 models on 31 predictors in 0.20799875259399414 seconds.
Processed  5 models on 32 predictors in 0.1919999122619629 seconds.
Processed  4 models on 33 predictors in 0.11999821662902832 seconds.
Processed  3 models on 34 predictors in 0.11210274696350098 seconds.
Processed  2 models on 35 predictors in 0.07199883460998535 seconds.
Processed  1 models on 36 predictors in 0.03196454048156738 seconds.
Total elapsed time: 10.258779764175415 seconds.
```

[23]:
```
best_sub_plots()
```



[24]:
```
#all criteria agree on having 16 predictors
best_fwd_reg_model = models_best['model'][16]
best_fwd_reg_model.summary()
```

[24]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
```

```
==============================================================================
Dep. Variable:                     Y1   R-squared:                       0.990
Model:                            OLS   Adj. R-squared:                  0.989
Method:                 Least Squares   F-statistic:                     3118.
Date:                Thu, 24 Feb 2022   Prob (F-statistic):               0.00
Time:                        03:42:56   Log-Likelihood:                -792.42
No. Observations:                 542   AIC:                             1619.
Df Residuals:                     525   BIC:                             1692.
Df Model:                          16
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept  -7705.1653    607.596    -12.681      0.000   -8898.783   -6511.548
X3_X5          1.5109      0.109     13.891      0.000       1.297       1.725
X5_X7         -2.7764      0.830     -3.346      0.001      -4.407      -1.146
X1_X2         -4.9300      0.652     -7.560      0.000      -6.211      -3.649
X2_X5         -0.9502      0.067    -14.222      0.000      -1.081      -0.819
X3_X8          0.0003      0.000      0.671      0.502      -0.001       0.001
X7_X8         -2.8188      0.204    -13.814      0.000      -3.220      -2.418
X7            84.3219      9.950      8.475      0.000      64.775     103.868
X4_X7         -0.2434      0.032     -7.592      0.000      -0.306      -0.180
X3_X4          0.0423      0.002     17.156      0.000       0.037       0.047
X3             7.2024      1.546      4.660      0.000       4.166      10.239
X1_X5        196.7862     36.713      5.360      0.000     124.663     268.910
X4            55.3400      3.229     17.139      0.000      48.997      61.683
X1          2968.1523    606.145      4.897      0.000    1777.385    4158.919
X2_X4         -0.0460      0.004    -12.152      0.000      -0.053      -0.039
X2_X3         -0.0047      0.001     -7.500      0.000      -0.006      -0.003
X1_X8          0.9009      0.191      4.723      0.000       0.526       1.276
==============================================================================
Omnibus:                        1.817   Durbin-Watson:                   0.832
Prob(Omnibus):                  0.403   Jarque-Bera (JB):                1.620
Skew:                           0.095   Prob(JB):                        0.445
Kurtosis:                       3.188   Cond. No.                     4.84e+09
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 4.84e+09. This might indicate that there are
strong multicollinearity or other numerical problems.
"""
```

Q1g

Calculate the RMSE of the model found in (f). Compare it with i) the RMSE of model you found in (b) and ii) the RMSE of the model using all first-order and 2-fatcor interaction terms and discuss about your finding.

(2 points for code, 2 points for answer)

```
[51]: full_model = models_best['model'][36]
```

```
[26]: pred_y1 = best_fwd_reg_model.predict(ENB_test)
      np.sqrt(((pred_y1 - ENB_test.Y1)**2).mean())
```

```
[26]: 1.0894404698394693
```

The RMSE is much smaller than the RMSE of the model we found in (b), which is 2.7781671950919242. We can see a huge improvement of both the adjusted R-squared (0.914 to 0.989) and RMSE (1.09 to 2.78) after adding the 2-factor interaction terms.

```
[27]: #calculate RMSE for all predictors
      pred_y1 = full_model.predict(ENB_test)
      np.sqrt(((pred_y1 - ENB_test.Y1)**2).mean())
```

```
[27]: 1.1088320527269544
```

We can see that the best model RMSE is very close to the RMSE of the full model. However, there are several insignificant predictors in the full model, which may be due to multicollinearity, or just due to the fact that those predictors do not explain the response. Thus, the model in (f) is better for the purpose of inference.

Q1h

Assume that we found another dataset of 32 variabels on the same set of 768 buildings (542 for training) that we would want to add into our model. We want find the "best" model of all 40 predictors and their 2-factor interaction terms. Would you choose forward or backward selection? Justify your answer.

(1 point for answer, 3 points for justification)

p $= 40 + \binom{40}{2} = 40 + 780 = 820$ which is larger than n $= 542$. We should use forward selection in this case since backward selection requires that the number of samples n is larger than the number of variables p (so that the full model can be fit). In contrast, forward stepwise can be used even when n<p, and so is the only viable subset method when p is very large.

Part 2

See https://exoplanetarchive.ipac.caltech.edu (for context/source). We are using the Composite Planetary Systems dataset

Q2a

Say we're interested in modeling the radius of exoplanets in kilometers, which is named as `pl_rade` in the data. Note that the variable `pl_rade` captures the radius of each plant as a proportion of Earth's radius, which is approximately 6,378.1370 km.

Develop a linear regression model to predict `pl_rade` using all the variables in *train_CompositePlanetarySystems.csv* except `pl_name`, `disc_facility` and `disc_locale`. Find the RMSE (Root mean squared error) of the model on *test1_CompositePlanetarySystems.csv* and *test2_CompositePlanetarySystems.csv*.

(4 points for code)

[82]:
```
train
```

[82]:
```
     sy_snum  sy_pnum  cb_flag  disc_year  rv_flag  pul_flag  ptv_flag  \
0          1        2        0       2014        0         0         0
1          1        3        0       2019        0         0         0
2          1        1        0       2021        0         0         0
3          1        1        0       2020        0         0         0
4          1        5        0       2014        0         0         0
..       ...      ...      ...        ...      ...       ...       ...
965        1        1        0       2021        0         0         0
966        1        2        0       2013        0         0         0
967        1        3        0       2012        0         0         0
968        1        2        0       2016        0         0         0
969        1        1        0       2021        0         0         0

     tran_flag  ast_flag  obm_flag  …  ima_flag  pl_controv_flag  \
0            1         0         0  …         0                0
1            1         0         0  …         0                0
2            1         0         0  …         0                0
3            1         0         0  …         0                0
4            1         0         0  …         0                0
..         …         …         …  …       …              …
965          1         0         0  …         0                0
966          1         0         0  …         0                0
967          1         0         0  …         0                0
968          1         0         0  …         0                0
969          1         0         0  …         0                0

      pl_orbper  pl_dens  ttv_flag  st_teff  st_rad  st_mass  st_lum  st_logg
0     10.613839    3.040         0   5818.0    0.89     0.91  -0.331     4.54
1      3.359605    4.980         0   5212.9    0.76     0.80  -0.401     4.60
2      6.892130    5.600         0   6164.0    1.45     1.29   0.437     4.23
3      3.262000    2.480         0   3752.0    0.51     0.52  -1.333     4.74
4     87.090195    2.300         0   4997.0    0.76     0.84  -0.496     4.58
..          …      …       …      …     …      …      …       …
965    4.626050    2.970         0   5834.0    0.80     0.85  -0.075     4.57
```

```
966  191.231800    0.152          1   5600.0    1.82     1.10   0.326     3.96
967   85.312000    0.030          1   6018.0    0.94     1.04  -0.142     4.51
968    8.752900    4.310          0   4326.0    0.55     0.61  -0.874     4.74
969    5.186720    2.420          0   5230.0    0.78     0.87  -0.394     4.59

[970 rows x 22 columns]
```

[94]:
```
train = pd.read_csv("train_CompositePlanetarySystems.csv")
test1 = pd.read_csv("test1_CompositePlanetarySystems.csv")
test2 = pd.read_csv("test2_CompositePlanetarySystems.csv")
```

[95]:
```
def data_prep(data):
    X = data.drop(columns =␣
 ↪['pl_name','pl_rade','disc_facility','disc_locale'],axis=1)
    return X
X = data_prep(train)
Xtest1 = data_prep(test1)
Xtest2 = data_prep(test2)
```

[96]:
```
model = sm.ols('pl_rade~' + '+'.join(X.columns), data = train).fit()
```

[97]:
```
def rmse(data,ytrue,model):
    pred = model.predict(data)
    rmse = np.sqrt(((pred-ytrue)**2).mean())
    return rmse
```

[98]:
```
rmse(test1,test1.pl_rade,model)
```

[98]:
```
127.36595496362311
```

RMSE on *test1.csv* is 127

[99]:
```
rmse(test2,test2.pl_rade,model)
```

[99]:
```
202.66277572552076
```

RMSE on *test2.csv* is 203

Q2b

Develop a ridge regression model to predict `pl_rade` using all the variables in *train_CompositePlanetarySystems.csv* except `pl_name`, `disc_facility` and `disc_locale`. What is the optimal value of the tuning parameter λ?

Hint: You may use the following grid of lambda values to find the optimal λ: `alphas = 10**np.linspace(2,0.5,200)*0.5`

Remember to standardize data before fitting the ridge regression model

(5 points for code)

```
[36]: #Standardizing predictors so that each of them have zero mean and unit variance

      #Defining a scaler object
      scaler = StandardScaler()

      #The scaler object will contain the mean and variance of each column␣
       ↪(predictor) of X. These values will be useful to scale
      #test data based on the same mean and variance as obtained on train data
      scaler.fit(X)

      #Using the scaler object (or the values of mean and variance stored in it) to␣
       ↪standardize X (or train data) and other datasets
      Xstd = pd.DataFrame(scaler.transform(X))
      Xtest1_std = pd.DataFrame(scaler.transform(Xtest1))
      Xtest2_std = pd.DataFrame(scaler.transform(Xtest2))
      Xstd.columns = X.columns
      Xtest1_std.columns = Xtest1.columns
      Xtest2_std.columns = Xtest2.columns

      y = train.pl_rade
```

```
[60]: #Let us use cross validation to find the optimal value of the tuning parameter␣
       ↪- lambda
      #For the optimal lambda, the cross validation error will be the least

      alphas = 10**np.linspace(2,0.5,200)*0.5
      ridgecv = RidgeCV(alphas = alphas,store_cv_values=True)
      ridgecv.fit(Xstd, y)

      #Optimal value of the tuning parameter - lamda
      ridgecv.alpha_
```

```
[60]: 3.221181754360687
```

The optimal value of λ is 3.22

Q2c

Use the optimal value of λ found in the previous question to develop a ridge regression model. What is the RMSE of the model on *test1_CompositePlanetarySystems.csv* and *test2_CompositePlanetarySystems.csv*?

(5 points for code)

```
[61]: #Using the developed ridge regression model to predict on test data
      ridge = Ridge(alpha = ridgecv.alpha_)
      ridge.fit(Xstd, y)
```

```
[61]: Ridge(alpha=3.221181754360687)
```

```
[62]: rmse(Xtest1_std,test1.pl_rade,ridge)
```

```
[62]: 3.218708879148552
```

```
[63]: rmse(Xtest2_std,test2.pl_rade,ridge)
```

```
[63]: 2.9898651678534853
```

The RMSE on *test1.csv* is 3.22 and on *test2.csv* is 2.99.

Q2d

Note that ridge regression has a much lower RMSE on test datasets as compared to Ordinary least squares (OLS) regression. Shrinking the coefficients has improved the model fit. Appreciate it. Which are the top two predictors for which the coefficients have shrunk the most?

To answer this question, find the ridge regression estimates for $\lambda = 10^{-10}$. Treat these estimates as OLS estimates and find the predictors for which these estimates have shrunk the most in the model developed in 2c.

(4 points for code, 1 point for answer)

```
[41]: ridge_ols = Ridge(alpha = 10**-10)
      ridge_ols.fit(Xstd, y)
```

```
[41]: Ridge(alpha=1e-10)
```

```
[42]: coef_diff = pd.Series(np.abs(ridge_ols.coef_) - np.abs(ridge.coef_))
```

```
[43]: pd.concat([pd.Series(Xstd.columns),coef_diff],axis=1).sort_values(by =␣
       ↪1,ascending=False)[0:2]
```

```
[43]:            0          1
      14  pl_orbper  75.809870
      12   ima_flag  75.800798
```

The coefficients of `pl_orbper` and `ima_flag` have shrunk the most.

Q2e

Why do you think the coefficients of the two variables indentified in the previous question shrunk the most?

Hint: VIF!

(4 points for justification - including any code used)

[44]:
```python
def vif(X):
    X = add_constant(X)
    vif_data = pd.DataFrame()
    vif_data["feature"] = X.columns

    for i in range(len(X.columns)):
        vif_data.loc[i,'VIF'] = variance_inflation_factor(X.values, i)

    print(vif_data)
```

[45]:
```python
vif(X)
```

```
              feature            VIF
0               const  293592.094382
1             sy_snum       1.162755
2             sy_pnum       1.146094
3             cb_flag       2.058390
4           disc_year       1.362591
5             rv_flag       2.014735
6            pul_flag            NaN
7            ptv_flag       2.610996
8           tran_flag       2.466560
9            ast_flag       1.091332
10           obm_flag       1.183595
11         micro_flag            NaN
12           etv_flag       7.440160
13           ima_flag   37675.155715
14   pl_controv_flag       1.698968
15          pl_orbper   37683.517115
16            pl_dens       1.017593
17           ttv_flag       1.110245
18            st_teff      13.944718
19             st_rad       3.257504
20            st_mass       4.139998
21             st_lum      12.507339
22            st_logg      10.421923
```

```
C:\Users\akl0407\Anaconda3\lib\site-
packages\statsmodels\regression\linear_model.py:1715: RuntimeWarning: invalid
value encountered in double_scalars
  return 1 - self.ssr/self.centered_tss
```

Note that `ima_flag` and `pl_orbper` are highly multicollinear, which results in a high variance of their coefficients. Thus, their cofficients have been shrunk the most in the ridge regression model, as shrinkage reduces the variance of the coefficients to improve the model fit.

Q2f

Develop a lasso model to predict `pl_rade` using all the variables in *train_CompositePlanetarySystems.csv* except `pl_name`, `disc_facility` and `disc_locale`. What is the optimal value of the tuning parameter λ?

Hint: You may use the following grid of lambda values to find the optimal λ: `alphas = 10**np.linspace(0,-2.5,200)*0.5`

(4 points for code)

```
[64]: #Let us use cross validation to find the optimal value of the tuning parameter␣
       ↪- lambda
      #For the optimal lambda, the cross validation error will be the least

      alphas = 10**np.linspace(0,-2.5,200)*0.5
      lassocv = LassoCV(alphas = alphas, cv = 10, max_iter = 100000)
      lassocv.fit(Xstd, y)

      #Optimal value of the tuning parameter - lamda
      lassocv.alpha_
```

```
[64]: 0.002051329052913595
```

Optimal λ for lasso is 0.002

Q2g

Use the optimal value of λ found in the previous question to develop a lasso model. What is the RMSE of the model on *test1_CompositePlanetarySystems.csv* and *test2_CompositePlanetarySystems.csv*?

(5 points for code)

```
[65]: #Using the developed lasso model to predict on test data
      lasso = Lasso(alpha = lassocv.alpha_)
      lasso.fit(Xstd, y)
```

```
[65]: Lasso(alpha=0.002051329052913595)
```

```
[66]: rmse(Xtest1_std,test1.pl_rade,lasso)
```

```
[66]: 3.220582381965326
```

```
[67]: rmse(Xtest2_std,test2.pl_rade,lasso)
```

```
[67]: 2.996713163709016
```

Q2h

Note that lasso has a much lower RMSE on test datasets as compared to Ordinary least squares (OLS) regression. Shrinking the coefficients has improved the model fit. Appreciate it. Which variables have been eliminated by lasso?

To answer this question, find the predictors whose coefficients are 0 in the lasso model.

(2 points for code, 1 point for answer)

[68]:
```
Xstd.columns[lasso.coef_==0]
```

[68]:
```
Index(['pul_flag', 'micro_flag', 'ima_flag'], dtype='object')
```

Coefficients of `pul_flag`, `micro_flag`, and `ima_flag` are reduced to zero. Thus, these predictors are eliminated by lasso.