Project 6 ¶

In addition to answering the bolded questions on Coursera, also attach your notebook, both as `.ipynb` and `.html`.

In the following exercise, we will perform model selection to find the best model for two datasets.

In this assignment, we will be using PennGrader, a Python package built by a former TA for autograding Python notebooks. PennGrader was developed to provide students with instant feedback on their answer. You can submit your answer and know whether it's right or wrong instantly. We then record your most recent answer in our backend database. You will have 100 attempts per test case, which should be more than sufficient.

NOTE: Please remember to remove the

```
raise notImplementedError
```

after your implementation, otherwise the cell will not compile.

Getting Setup

Please run the below cells to get setup with the autograder. If you need to install packages, please do it below!

In [32]:
```
# %%capture
# !pip install penngrader --user
```

Let's try PennGrader out! Fill in the cell below with your PennID and then run the following cell to initialize the grader.

Warning: Please make sure you only have one copy of the student notebook in your directory in Codio upon submission. The autograder looks for the variable `STUDENT_ID` across all notebooks, so if there is a duplicate notebook, it will fail.

In [33]:
```
#PLEASE ENSURE YOUR STUDENT_ID IS ENTERED AS AN INT (NOT A STRING). IF NOT, TH
E AUTOGRADER WON'T KNOW WHO
#TO ASSIGN POINTS TO YOU IN OUR BACKEND

STUDENT_ID = 49731093                    # YOUR 8-DIGIT PENNID GOES HERE
STUDENT_NAME = "Newman Ilgenfritz"       # YOUR FULL NAME GOES HERE
```

In [34]:
```
import penngrader.grader

grader = penngrader.grader.PennGrader(homework_id = 'ESE542_Online_Spring_2021
_HW6', student_id = STUDENT_ID)
```

Part A

First, we will run multiple linear regression on the Auto dataset and use subset selection to find the best model. This dataset contains the following nine columns from 392 cars:

Column	Description
mpg	continuous
cylinders	multi-valued discrete
displacement	continuous
horsepower	continuous
weight	continuous
acceleration	continuous
model year	multi-valued discrete
origin	multi-valued discrete
car name	string

In [35]:
```
# Feel free to import your own libraries!
import pandas as pd
```

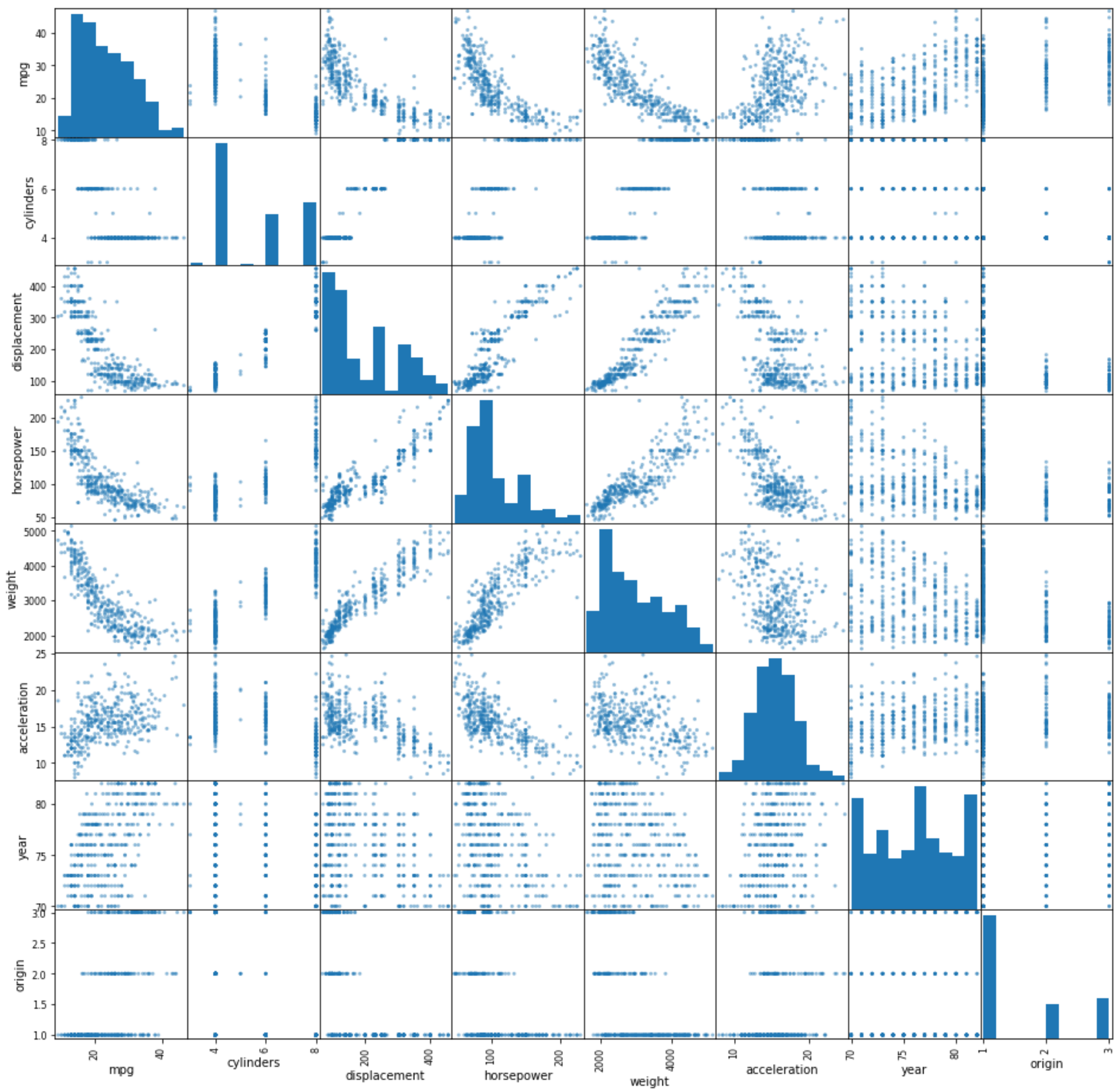
In [36]:
```
auto_data_raw = pd.read_csv("Auto.csv")
auto = auto_data_raw.copy()
auto.head()
```

Out[36]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino

1. Produce a scatter plot matrix which includes all the variables in the dataset. Comment on your observations. Are there any variables in particular which seem to be strongly correlated? Store your observation in `observed` as one string. If you do not get full credit for your observations, try to keep adding more comments.

In [37]:
```
# Add your codes here
pd.plotting.scatter_matrix(auto, alpha=0.50, figsize=(16,16), range_padding=0.
050)
observed = " strongly correlated vars:  displacement: horsepower (pos); displc
ement: weight(pos); horsepower: weight(pos); horsewpower: acceleration(neg); m
pg: horsepower(neg); mpg: weight(neg); mpg: displacement(neg); mpg: cylinders
(neg)"
```



In [38]:
```
grader.grade(test_case_id = 'test_correlation_obs', answer = observed)
```

```
Correct! You earned 1/1 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

1. Compute a matrix of correlations between the variables using the `pandas` and `corr()` functions.

In [39]:
```
# Add your comments here

# matrix of correlations:
auto.corr()
```

Out[39]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year
mpg	1.000000	-0.777618	-0.805127	-0.778427	-0.832244	0.423329	0.580541
cylinders	-0.777618	1.000000	0.950823	0.842983	0.897527	-0.504683	-0.345647
displacement	-0.805127	0.950823	1.000000	0.897257	0.932994	-0.543800	-0.369855
horsepower	-0.778427	0.842983	0.897257	1.000000	0.864538	-0.689196	-0.416361
weight	-0.832244	0.897527	0.932994	0.864538	1.000000	-0.416839	-0.309120
acceleration	0.423329	-0.504683	-0.543800	-0.689196	-0.416839	1.000000	0.290316
year	0.580541	-0.345647	-0.369855	-0.416361	-0.309120	0.290316	1.000000
origin	0.565209	-0.568932	-0.614535	-0.455171	-0.585005	0.212746	0.181528

1. Using `Stats Models` , perform linear regression with 'mpg' as the response variable and all other variables except 'name' as predictors. Print the results of your regression analysis. Please answer the following questions based on your model.

   - **Which predictors appear to have a statistically significant relationship with the response variable at a 95% confidence level?** Please store them in **significant_predictor** as a list of strings.
   - **What does the coefficient for the 'year' variable suggest?** Comment on your observations and store your findings in `year_coef` as a single string. If you do not get full credit for your observations, try to keep adding more comments.

In [47]:
```
# Add your codes here
import numpy as np

#Logistic Regression
import statsmodels.api as sm
import statsmodels.formula.api as smf
from sklearn.linear_model import LogisticRegression

#Plotting
import matplotlib.pyplot as plt

#Statistics
from scipy import stats

RANDOM_STATE=42
%matplotlib inline

X = auto[['cylinders','displacement', 'horsepower', 'weight', 'acceleration',
'year', 'origin']]
y = auto['mpg']

X = sm.add_constant(X)

model = sm.OLS(y, X).fit()
predictions = model.predict(X)

print_model = model.summary()
print(print_model)


year_coef = "The year coef equals 0.7508. This means there is a strong and sta
t. sig. pos. relationship b/w year model and miles per gallon."

significant_predictor = ['displacement', 'origin', 'weight', 'year']
```

```
                            OLS Regression Results
==============================================================================
=
Dep. Variable:                    mpg   R-squared:                       0.82
1
Model:                            OLS   Adj. R-squared:                  0.81
8
Method:                 Least Squares   F-statistic:                     252.
4
Date:                Fri, 12 Mar 2021   Prob (F-statistic):          2.04e-13
9
Time:                        01:32:00   Log-Likelihood:                -1023.
5
No. Observations:                 392   AIC:                             206
3.
Df Residuals:                     384   BIC:                             209
5.
Df Model:                           7
Covariance Type:            nonrobust
==============================================================================
===
                   coef    std err          t      P>|t|      [0.025      0.9
75]
------------------------------------------------------------------------------
---
const          -17.2184      4.644     -3.707      0.000     -26.350      -8.
087
cylinders       -0.4934      0.323     -1.526      0.128      -1.129       0.
142
displacement     0.0199      0.008      2.647      0.008       0.005       0.
035
horsepower      -0.0170      0.014     -1.230      0.220      -0.044       0.
010
weight          -0.0065      0.001     -9.929      0.000      -0.008      -0.
005
acceleration     0.0806      0.099      0.815      0.415      -0.114       0.
275
year             0.7508      0.051     14.729      0.000       0.651       0.
851
origin           1.4261      0.278      5.127      0.000       0.879       1.
973
==============================================================================
=
Omnibus:                       31.906   Durbin-Watson:                   1.30
9
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               53.10
0
Skew:                           0.529   Prob(JB):                    2.95e-1
2
Kurtosis:                       4.460   Cond. No.                    8.59e+0
4
==============================================================================
=

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

```
[2] The condition number is large, 8.59e+04. This might indicate that there a
re
strong multicollinearity or other numerical problems.


/usr/local/lib/python3.6/dist-packages/numpy/core/fromnumeric.py:2389: Future
Warning: Method .ptp is deprecated and will be removed in a future version. U
se numpy.ptp instead.
  return ptp(axis=axis, out=out, **kwargs)
```

In [48]:
```
grader.grade(test_case_id = 'test_sig_predictor', answer = significant_predict
or)
```

```
Correct! You earned 2/2 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

In [49]:
```
grader.grade(test_case_id = 'test_year_ob', answer = year_coef)
```

```
Correct! You earned 1/1 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

1. Select the optimal model by manually performing forward stepwise selection. The goal of this exercise is to show the sheer number of models needed for forward stepwise selection. To do this, first split the dataset into a training set and a test set, with a `test_size` of 20% and `random_state = 42`. It is important to only use the training set to train the model. You may use the `processSubset` function from the recitation, but you should at least run 3 iterations of forward stepwise selection manually, as we wish to see each step of forward stepwise selection.

First, run linear regression with one variable. Select the best model using training RSS as the performance metric. Using that first variable, continue adding variables, one at a time, until your linear model includes all of the variables. Afterwards, calculate the test RSS of all your models and select the one that **minimizes** test RSS. *Hint*: You can use the result of linear regression from `Stats Models` to calculate RSS by looking at the sum of the squared residuals.

Store the listed models **for each round of selection** within `chosen_models`. Store your list of selected variables within **predictors_forward**. **Please do not capitalize your column names**.

Hint:

1. Your `chosen_models` should be of size 1*7, the first model has two coefficients(intercept and a predictor), and the last model has 8 coefficients(intercept and the entire predictor space).
2. Find the RSS for each of your chosen models. Choose the model with the least RSS as your selected model.

In [43]:
```
# Add your codes here
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import scale
#Metrics
import itertools
import time
from sklearn.metrics import mean_squared_error

def processSubset(feature_set):
    # Fit model on feature_set and calculate RSS
    # add in a column of ones as intercept
    X_t = sm.add_constant(X_train[list(feature_set)])
    model = sm.OLS(y_train, X_t)
    regression = model.fit()
    # RSS is the Residual Sum of Squares
    RSS = (regression.resid ** 2).sum()
    return {
        'model': regression,
        'RSS': RSS
    }

def getBest(k):
    tic = time.time()
    results = []
    # Fit all p choose k models with k predictors
    for combo in itertools.combinations(X_train.columns, k):
        results.append(processSubset(combo))

    # Wrap everything up in a nice dataframe
    models = pd.DataFrame(results)

    # Choose the model with the smallest RSS
    best_model = models.loc[models['RSS'].argmin()]

    toc = time.time()
    print ('Processed ', models.shape[0],
          ' models on ', k, ' predictors in ',
          (toc - tic), ' seconds.') #count the time in between

    # Return the best model, along with other useful information
    return best_model

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando
m_state=RANDOM_STATE)
X_test = sm.add_constant(X_test)
print('X_test.head: ')
print(X_test, '\n')

# First, run linear regression with one variable.
#     Select the best model using training RSS as the performance metric.

models = pd.DataFrame(columns=['RSS', 'model'])

feature_set = pd.DataFrame(auto['displacement'])
```

```
for i in range(1, len(X_train.columns)+1): #reduce this range if running too l
ong
    models.loc[i] = getBest(i)

print('models: ')
print(models, '\n')

#print (models.loc[2, "model"].summary())
print (models.loc[8, "model"].summary())

chosen_models = []
```

```
/usr/local/lib/python3.6/dist-packages/numpy/core/fromnumeric.py:2389: Future
Warning: Method .ptp is deprecated and will be removed in a future version. U
se numpy.ptp instead.
  return ptp(axis=axis, out=out, **kwargs)
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:33: FutureWarnin
g:
The current behaviour of 'Series.argmin' is deprecated, use 'idxmin'
instead.
The behavior of 'argmin' will be corrected to return the positional
minimum in the future. For now, use 'series.values.argmin' or
'np.argmin(np.array(values))' to get the position of the minimum
row.
```

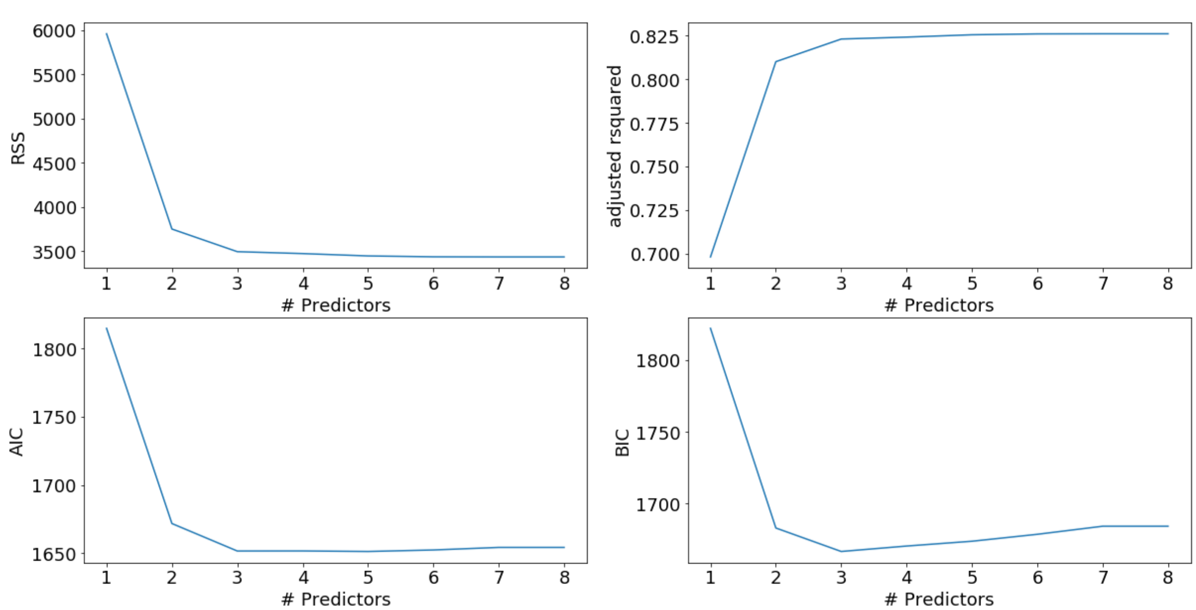
```
X_test.head:
     const  cylinders  displacement  horsepower  weight  acceleration  year
\
78     1.0          4          96.0          69    2189          18.0    72
274    1.0          4         121.0         115    2795          15.7    78
246    1.0          4          91.0          60    1800          16.4    78
55     1.0          4          91.0          70    1955          20.5    71
387    1.0          4         140.0          86    2790          15.6    82
..     ...        ...           ...         ...     ...           ...   ...
361    1.0          6         225.0          85    3465          16.6    81
82     1.0          4          98.0          80    2164          15.0    72
114    1.0          8         350.0         145    4082          13.0    73
3      1.0          8         304.0         150    3433          12.0    70
18     1.0          4          97.0          88    2130          14.5    70

     origin
78        2
274       2
246       3
55        1
387       1
..      ...
361       1
82        1
114       1
3         1
18        3

[79 rows x 8 columns]

Processed  8  models on  1  predictors in  0.025419235229492188  seconds.
Processed  28  models on  2  predictors in  0.09360504150390625  seconds.
Processed  56  models on  3  predictors in  0.20744943618774414  seconds.
Processed  70  models on  4  predictors in  0.2979397773742676  seconds.
Processed  56  models on  5  predictors in  0.25406455993652344  seconds.
Processed  28  models on  6  predictors in  0.1518688201904297  seconds.
Processed  8  models on  7  predictors in  0.04626941680908203  seconds.
Processed  1  models on  8  predictors in  0.006840705871582031  seconds.
models:
           RSS                                              model
1  5961.118240  <statsmodels.regression.linear_model.Regressio...
2  3752.032912  <statsmodels.regression.linear_model.Regressio...
3  3495.974375  <statsmodels.regression.linear_model.Regressio...
4  3474.203553  <statsmodels.regression.linear_model.Regressio...
5  3447.726465  <statsmodels.regression.linear_model.Regressio...
6  3437.837334  <statsmodels.regression.linear_model.Regressio...
7  3436.507079  <statsmodels.regression.linear_model.Regressio...
8  3436.507079  <statsmodels.regression.linear_model.Regressio...

                            OLS Regression Results
==============================================================================
=
Dep. Variable:                    mpg   R-squared:                       0.82
6
Model:                            OLS   Adj. R-squared:                  0.82
2
Method:                 Least Squares   F-statistic:                     206.
```

```
8
Date:                  Fri, 12 Mar 2021   Prob (F-statistic):          8.43e-11
2
Time:                          01:27:09   Log-Likelihood:               -819.1
0
No. Observations:                   313   AIC:                            165
4.
Df Residuals:                       305   BIC:                            168
4.
Df Model:                             7
Covariance Type:              nonrobust
==============================================================================
===
                   coef    std err          t      P>|t|      [0.025      0.9
75]
------------------------------------------------------------------------------
---
const          -18.4994      5.392     -3.431      0.001     -29.109      -7.
890
cylinders       -0.3458      0.373     -0.928      0.354      -1.079       0.
388
displacement     0.0151      0.008      1.780      0.076      -0.002       0.
032
horsepower      -0.0213      0.016     -1.362      0.174      -0.052       0.
009
weight          -0.0061      0.001     -8.529      0.000      -0.008      -0.
005
acceleration     0.0380      0.110      0.344      0.731      -0.179       0.
255
year             0.7677      0.059     12.958      0.000       0.651       0.
884
origin           1.6135      0.312      5.175      0.000       1.000       2.
227
==============================================================================
=
Omnibus:                         30.025   Durbin-Watson:                   2.03
2
Prob(Omnibus):                    0.000   Jarque-Bera (JB):               43.88
9
Skew:                             0.640   Prob(JB):                     2.95e-1
0
Kurtosis:                         4.315   Cond. No.                     8.85e+0
4
==============================================================================
=

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correc
tly specified.
[2] The condition number is large, 8.85e+04. This might indicate that there a
re
strong multicollinearity or other numerical problems.
```

In [44]:
```
import matplotlib.pyplot as plt

def forward(predictors):
    # Pull out predictors we still need to process
    remaining_predictors= [p for p in X_train.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 smallest RSS
    best_model = models.loc[models['RSS'].argmin()]

    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 th
e model
    return best_model

models2 = pd.DataFrame(columns=["RSS", "model"])

tic = time.time()
predictors = []

for i in range(1,len(X_train.columns)+1):
    models2.loc[i] = forward(predictors)
    predictors = models2.loc[i]["model"].model.exog_names[1:]

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

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)
plt.plot(models2["RSS"])
plt.xlabel('# Predictors')
plt.ylabel('RSS')

rsquared = models2.apply(lambda row: row[1].rsquared, axis=1)
plt.subplot(2, 2, 2)
plt.plot(rsquared)
plt.xlabel('# Predictors')
plt.ylabel('adjusted rsquared')


aic = models2.apply(lambda row: row[1].aic, axis=1)
```

```
plt.subplot(2, 2, 3)
plt.plot(aic)
plt.xlabel('# Predictors')
plt.ylabel('AIC')

bic = models2.apply(lambda row: row[1].bic, axis=1)
plt.subplot(2, 2, 4)
plt.plot(bic)
plt.xlabel('# Predictors')
plt.ylabel('BIC')

plt.show()
plt.close()

predictors_forward = ['weight','year']
```

```
/usr/local/lib/python3.6/dist-packages/numpy/core/fromnumeric.py:2389: Future
Warning: Method .ptp is deprecated and will be removed in a future version. U
se numpy.ptp instead.
  return ptp(axis=axis, out=out, **kwargs)
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:18: FutureWarnin
g:
The current behaviour of 'Series.argmin' is deprecated, use 'idxmin'
instead.
The behavior of 'argmin' will be corrected to return the positional
minimum in the future. For now, use 'series.values.argmin' or
'np.argmin(np.array(values))' to get the position of the minimum
row.

Processed  8  models on  1  predictors in  0.02644491195678711  seconds.
Processed  7  models on  2  predictors in  0.023945093154907227  seconds.
Processed  6  models on  3  predictors in  0.02156662940979004  seconds.
Processed  5  models on  4  predictors in  0.019430160522460938  seconds.
Processed  4  models on  5  predictors in  0.015909671783447266  seconds.
Processed  3  models on  6  predictors in  0.0140380859375  seconds.
Processed  2  models on  7  predictors in  0.010801076889038086  seconds.
Processed  1  models on  8  predictors in  0.006035566329956055  seconds.
Total elapsed time: 0.15853357315063477 seconds.
```



In [45]:
```
# Please do not change the code below
chosen_models_params = [x.params for x in chosen_models]
```

In [46]:
```
grader.grade(test_case_id = 'test_chosen_models', answer = list(chosen_models_
params))
```

```
Error: Test case failed. Test case function could not complete due to an erro
r in your answer.
Error Hint: list index out of range
```

In [49]:
```
grader.grade(test_case_id = 'test_forward_selection', answer = predictors_forw
ard)
```

```
Correct! You earned 2/2 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

1. Using the full dataset, fit a linear regression model with interaction effects between 'displacement', 'weight', 'year', and 'origin'. Do any interactions appear to be statistically significant? *Hint*: In addition to the full model with all seven predictors, your model should include six more interaction terms. List your list of significant interaction terms in **predictor_interaction** in alphabetical order.

Hint:

1. Only store the interaction terms, your output format should be ["a:b","a:c"...].
2. Manually create a list of interaction terms first and then use '+'.join(list) to build your model.

In [50]:

```
# Add your codes here
import seaborn as sns
import statsmodels.formula.api as smf

model = smf.ols(formula='mpg ~ cylinders + displacement + horsepower + weight
 + acceleration + year + origin', data=auto).fit()
#summary = model.summary()
#print(summary.tables[1])
#print('\n')

interTerms = ['displacement:weight', 'displacement:year', 'displacement:origi
n', 'weight:year + weight:origin', 'year:origin']

model_interaction = smf.ols(formula='mpg ~ cylinders + displacement + horsepow
er + weight + acceleration + year + origin \
    + displacement:weight + displacement:year + displacement:origin + weight:y
ear + weight:origin + year:origin', data=auto).fit()
#model_interaction = smf.ols(formula='mpg ~ cylinders + displacement + horsepo
wer + weight + acceleration + year + origin' + join(interTerms), data=auto).fi
t()
summary = model_interaction.summary()
print(summary.tables[1])
print('\n')


predictor_interaction = ["displacement:weight"]
print(predictor_interaction)
#raise NotImplementedError
```

```
==============================================================================
==========
                          coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
----------
Intercept             -41.8056     24.907     -1.678      0.094     -90.779
7.168
cylinders               0.4525      0.304      1.486      0.138      -0.146
1.051
displacement            0.0809      0.083      0.979      0.328      -0.082
0.244
horsepower             -0.0444      0.013     -3.519      0.000      -0.069
-0.020
weight                 -0.0066      0.011     -0.590      0.556      -0.028
0.015
acceleration            0.1131      0.087      1.301      0.194      -0.058
0.284
year                    1.2796      0.323      3.961      0.000       0.644
1.915
origin                 -1.6700      5.243     -0.319      0.750     -11.979
8.639
displacement:weight  2.202e-05   2.83e-06      7.772      0.000    1.65e-05
2.76e-05
displacement:year      -0.0022      0.001     -1.897      0.059      -0.004
7.96e-05
displacement:origin     0.0119      0.012      0.955      0.340      -0.013
0.036
weight:year         -5.534e-05      0.000     -0.368      0.713      -0.000
0.000
weight:origin           0.0002      0.001      0.213      0.831      -0.002
0.002
year:origin             0.0050      0.066      0.075      0.940      -0.126
0.136
==============================================================================
==========
```

```
['displacement:weight']
```

In [51]:

```
grader.grade(test_case_id = 'test_interactions', answer = predictor_interactio
n)
```

```
Correct! You earned 2/2 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

Part B

Next, we will use the College dataset to predict the number of applications ('Apps') received using the other variables in the College dataset. We will then use regularization to study their effects on our model.

In [52]:
```
data = pd.read_csv('College.csv').copy()
data.set_index('Names', inplace = True)
data['Private'] = [1 if x=="Yes" else 0 for x in data['Private']]
data = data.rename(columns = {'Grad.Rate':'Grad_Rate',
                              'S.F.Ratio': 'S_F_Ratio',
                              'perc.alumni':'perc_alumni',
                              'Room.Board':'Room_Board',
                              'F.Undergrad':'F_Undergrad',
                              'P.Undergrad':'P_Undergrad'})
data.head()
```

Out[52]:

	Private	Apps	Accept	Enroll	Top10perc	Top25perc	F_Undergrad	P_Undergrad	Ou
Names									
Abilene Christian University	1	1660	1232	721	23	52	2885	537	
Adelphi University	1	2186	1924	512	16	29	2683	1227	
Adrian College	1	1428	1097	336	22	50	1036	99	
Agnes Scott College	1	417	349	137	60	89	510	63	
Alaska Pacific University	1	193	146	55	16	44	249	869	

1. Split the dataset into a training set and a test set, with a `test_size` of 20% and `random_state=1` .

In [53]:
```
import pandas as pd
import numpy as np
import math
import statsmodels.api as sm
from statsmodels.tools.eval_measures import rmse
import sklearn
from sklearn.linear_model import Ridge
from sklearn.linear_model import RidgeCV
from sklearn.linear_model import Lasso
from sklearn.linear_model import LassoCV
from sklearn.model_selection import train_test_split


train, test = train_test_split(data, test_size=0.2, random_state=1)
print('train.head: ')
print(train.head(), '\n')

print('test.shape: ')
print(test.shape, '\n')



#raise NotImplementedError
```

```
train.head:
                           Private  Apps  Accept  Enroll  Top10perc  \
Names
Sweet Briar College              1   462     402     146         36
Eureka College                   1   560     454     113         36
Manhattanville College           1   962     750     212         21
Lenoir-Rhyne College             1   979     743     259         25
West Liberty State College       0  1164    1062     478         12

                           Top25perc  F_Undergrad  P_Undergrad  Outstate  \
Names
Sweet Briar College               68          527           41     14500
Eureka College                    56          484           16     10955
Manhattanville College            54          830          150     14700
Lenoir-Rhyne College              46         1188          166     10100
West Liberty State College        25         2138          227      4470

                           Room_Board  Books  Personal  PhD  Terminal  \
Names
Sweet Briar College              6000    500       600   91        99
Eureka College                   3450    330       670   62        87
Manhattanville College           6550    450       400   97        97
Lenoir-Rhyne College             4000    400      1000   88        92
West Liberty State College       2890    600      1210   33        33

                           S_F_Ratio  perc_alumni  Expend  Grad_Rate
Names
Sweet Briar College              6.5           48   18953         61
Eureka College                  10.6           31    9552         53
Manhattanville College          11.3           24   11291         70
Lenoir-Rhyne College            12.0           20    8539         66
West Liberty State College      16.3           10    4249         60

test.shape:
(156, 18)
```

In [54]:
```
grader.grade(test_case_id = 'test_train_test_split', answer = test)
```

```
Correct! You earned 0.5/0.5 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

1. Fit a linear model using `Stats Models` on the training set where the target variable is `Apps` , and report the test MSE obtained. Name this variable `test_MSE` .

In [55]:
```
# Add your codes here
import math
import statsmodels.api as sm
from statsmodels.tools.eval_measures import rmse
import pandas as pd
import numpy as np
import statsmodels.api as sm
from statsmodels.tools.eval_measures import rmse
import sklearn
from sklearn.linear_model import Ridge
from sklearn.linear_model import RidgeCV
from sklearn.linear_model import Lasso
from sklearn.linear_model import LassoCV
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

#X = data.drop(['Apps'], axis=1)
X = train.drop(['Apps'], axis=1)
X = sm.add_constant(X)
#print('X.head: ')
#print(X.head(), '\n')
print('X shape: ', X.shape, '\n')
#y = data['Apps']
y = train['Apps']
#print('y.head: ')
#print(y.head(), '\n')
print('y shape: ', y.shape, '\n')

X_test = test.drop(['Apps'], axis=1)
X_test = sm.add_constant(X_test)
#print('X.head: ')
#print(X_test.head(), '\n')
print('X_test shape: ', X_test.shape, '\n')
#y = data['Apps']
y_test = test['Apps']
print('y_test.head: ')
print(y_test.head(), '\n')
print('y_test shape: ', y_test.shape, '\n')

model = sm.OLS(y,X)
results = model.fit()
#print(results.summary(), '\n')

#ypred = model.predict(X)
ypred = results.predict(X_test)
print('ypred shape: ', ypred.shape, '\n')
print('ypred.head: ')
print(ypred.head(), '\n')

#mse = rmse**2
mse = mean_squared_error(y_test, ypred)
print('mse: ', mse, '\n')

# calc rmse
rmse = rmse(y_test, ypred, axis=0)
```

```
print('rmse: ', rmse, '\n')

test_MSE = mse

#raise NotImplementedError
```

```
X shape:  (621, 18)

y shape:  (621,)

X_test shape:  (156, 18)

y_test.head:
Names
Mississippi College                           594
Saint Francis College                        1046
University of Southern Colorado              1401
Wake Forest University                       5661
North Carolina State University at Raleigh  10634
Name: Apps, dtype: int64

y_test shape:  (156,)

ypred shape:  (156,)

ypred.head:
Names
Mississippi College                           877.607904
Saint Francis College                        1227.759279
University of Southern Colorado              1552.690755
Wake Forest University                       7396.943867
North Carolina State University at Raleigh  10855.232465
dtype: float64

mse:  640045.0279060608

rmse:  800.0281419463073


/usr/local/lib/python3.6/dist-packages/numpy/core/fromnumeric.py:2389: Future
Warning: Method .ptp is deprecated and will be removed in a future version. U
se numpy.ptp instead.
  return ptp(axis=axis, out=out, **kwargs)
```

In [14]:
```
grader.grade(test_case_id = 'test_MSE', answer = test_MSE)
```

```
Correct! You earned 2/2 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

1. What is the MSE if you fit a ridge regression with a λ parameter of 0? Store your answer in **ridge_lambda_0**. *Hint*: you can either take an informed guess and hard code your answer!

In [15]:
```
# Add your codes here
# from recitation:
import math
import statsmodels.api as sm
from statsmodels.tools.eval_measures import rmse
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error



X_train, X_test , y_train, y_test = train_test_split(X, y, test_size=0.2, rand
om_state=1)
print('X_train shape: ', X_train.shape, '\n')
print('X_test shape: ', X_test.shape, '\n')
print('y_test.head: ')
print(y_test.head(), '\n')
print('y_test shape: ', y_test.shape, '\n')

from sklearn.linear_model import Ridge, RidgeCV, Lasso, LassoCV

alphas = 10**np.linspace(10, -2, 100) * 0.5
alphas

ridge = Ridge(normalize=True)
coeffs = []

for a in alphas:
    ridge.set_params(alpha=a)
    ridge.fit(X, y)
    coeffs.append(ridge.coef_)

np.shape(coeffs)



print('X_train shape: ', X_train.shape, '\n')
print('X_test shape: ', X_test.shape, '\n')
#y = data['Apps']
#y_test = test['Apps']
print('y_test.head: ')
print(y_test.head(), '\n')
print('y_test shape: ', y_test.shape, '\n')

ridge2 = Ridge(alpha=0, normalize=True)
ridge2.fit(X_train, y_train)
pred2 = ridge2.predict(X_test)
print (pd.Series(ridge2.coef_, index=X_test.columns))
mse = mean_squared_error(y_test, pred2)
print ('mean_squared_error(y_test, pred2): ',mse , '\n')


ridge_lambda_0 =  640045.0279060608
```

```
X_train shape:  (496, 18)

X_test shape:  (125, 18)

y_test.head:
Names
Alaska Pacific University      193
Judson College                 313
Carnegie Mellon University    8728
Carleton College              2694
La Salle University           2929
Name: Apps, dtype: int64

y_test shape:  (125,)

X_train shape:  (496, 18)

X_test shape:  (125, 18)

y_test.head:
Names
Alaska Pacific University      193
Judson College                 313
Carnegie Mellon University    8728
Carleton College              2694
La Salle University           2929
Name: Apps, dtype: int64

y_test shape:  (125,)

const            0.000000
Private       -280.986318
Accept           1.618825
Enroll          -0.900719
Top10perc       45.190867
Top25perc      -11.899295
F_Undergrad      0.060178
P_Undergrad      0.019134
Outstate        -0.107078
Room_Board       0.223723
Books           -0.167106
Personal         0.052436
PhD            -11.057322
Terminal         0.672855
S_F_Ratio       15.858847
perc_alumni      0.692413
Expend           0.082110
Grad_Rate        7.749656
dtype: float64
mean_squared_error(y_test, pred2):  1353363.5780857229
```

In [16]:
```
grader.grade(test_case_id = 'test_ridge_lambda_0', answer = ridge_lambda_0)
```

```
Correct! You earned 0.5/0.5 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

1. Fit a ridge regression model on the training set, with λ chosen by cross-validation. Report the test error obtained. *Hint*: Look at the recitation guides for how to implement cross-validation with `RidgeCV`. `RidgeCV` essentially performs hyper-parameter optimization (more on this in the next recitation) by testing all possible parameters through cross validation. For its parameters, specify `KFold cross validation` with ten folds, `scoring` with mean squared error, `normalization` set to true, and 50 equally spaced λ values ranging from 10^2 to 10^3. Name the selected value of λ as `ridge_select` and calculate the corresponding test MSE as `test_MSE_ridge`.

Hint: Use np.linspace() to generate lambdas. Please refer to [sklearn.linear_model.ridgeCV (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.RidgeCV.html)](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.RidgeCV.html) for ridgeCV's documentation.

In [18]:
```
# Add your codes here


n_alphas = 50
alphas = np.linspace(10**2, 10**3, n_alphas)
#print('alphas: ')
#print(alphas, '\n')


X_train, X_test , y_train, y_test = train_test_split(X, y, test_size=0.5, rand
om_state=1)
print('X_train shape: ', X_train.shape, '\n')
print('X_test shape: ', X_test.shape, '\n')
print('y_test.head: ')
print(y_test.head(), '\n')
print('y_test shape: ', y_test.shape, '\n')


#ridgecv = RidgeCV(alphas=alphas, cv=None, scoring='neg_mean_squared_error', n
ormalize=True)
ridgecv = RidgeCV(alphas=alphas, cv=10, scoring='neg_mean_squared_error', norm
alize=True)
ridgecv.fit(X_train, y_train)
ridgecv.alpha_ #find the best alpha

ridge_select = ridgecv.alpha_
print('\nridge_select: ', ridge_select, '\n')

ridge5 = Ridge(alpha=ridgecv.alpha_, normalize=True)
ridge5.fit(X_train, y_train)
mse = mean_squared_error(y_test, ridge5.predict(X_test))
print('mse: ', mse, '\n')


test_MSE_ridge = 13146544.50
```

```
X_train shape:  (310, 18)

X_test shape:  (311, 18)

y_test.head:
Names
Alaska Pacific University      193
Judson College                 313
Carnegie Mellon University    8728
Carleton College              2694
La Salle University           2929
Name: Apps, dtype: int64

y_test shape:  (311,)


ridge_select:  100.0

mse:  10635049.461799858
```

In [19]:
```
grader.grade(test_case_id = 'test_ridgeCV', answer = (ridge_select, test_MSE_r
idge))
```

```
Correct! You earned 2.0/2 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

1. Compare the ridge regression coefficients when using $\lambda = 0$ and the value for λ given by `RidgeCV`. Comment on your observations.

In [37]:
```
print (pd.Series(ridge5.coef_, index=X.columns))


# Add your comment here
"""

"""

#raise NotImplementedError
```

```
const           0.000000
Private       -40.229651
Accept          0.014604
Enroll          0.035604
Top10perc       0.744416
Top25perc       0.706283
F_Undergrad     0.006398
P_Undergrad     0.012226
Outstate       -0.000054
Room_Board      0.004400
Books           0.029379
Personal        0.011452
PhD             0.997689
Terminal        1.025713
S_F_Ratio       1.285628
perc_alumni    -0.280390
Expend          0.001765
Grad_Rate       0.381617
dtype: float64
```

Out[37]:
```
'\n  \n'
```

1. Fit a lasso model on the training set, with λ chosen by cross-validation. Specify `KFold cross validation` with ten folds, `normalization` set to true, and 50 equally spaced λ values ranging from 10^2 to 10^3. Name the selected value of λ as `lasso_select` and calculate the corresponding test MSE as `test_MSE_lasso`. Also report the number of **non-zero** coefficient estimates by looking at the output of

```
pd.Series(lasso.coef_,index=x.columns)
```

and store your answer in `num_nonzero`

In [25]:
```
import math
import statsmodels.api as sm
from statsmodels.tools.eval_measures import rmse
import pandas as pd
import numpy as np
import statsmodels.api as sm
from statsmodels.tools.eval_measures import rmse
import sklearn
from sklearn.linear_model import Ridge
from sklearn.linear_model import RidgeCV
from sklearn.linear_model import Lasso
from sklearn.linear_model import LassoCV
from sklearn.model_selection import train_test_split

n_alphas = 50
alphas = np.linspace(10**2, 10**3, n_alphas)

lasso_cv = LassoCV(alphas=alphas, cv=10, max_iter=100000, normalize=True)
lasso_cv.fit(X_train, y_train)


print('lassocv.alpha_: ',lassocv.alpha_ , '\n')
predictions = lassocv.predict(X_test)
print("Test Error: " +str(mean_squared_error(y_test, predictions)))
print("Model coefficients: " + str(lassocv.coef_))

lasso_select = lassocv.alpha_

mse = mean_squared_error(y_test, predictions)
print('mse: ', mse, '\n')

test_MSE_lasso = mse

coeffs = pd.Series(lassocv.coef_,index=X.columns)
print('coeffs: ')
print(coeffs, '\n')
#num_nonzero =  # by visual inspection of output of

#raise NotImplementedError
```

```
lassocv.alpha_:  100.0

Test Error: 3029492.35757198
Model coefficients: [ 0.          -0.           0.87992925  0.           0.
0.
  0.           0.           0.           0.           0.           0.
  0.           0.          -0.           0.           0.           0.        ]
mse:  3029492.35757198

coeffs:
const           0.000000
Private        -0.000000
Accept          0.879929
Enroll          0.000000
Top10perc       0.000000
Top25perc       0.000000
F_Undergrad     0.000000
P_Undergrad     0.000000
Outstate        0.000000
Room_Board      0.000000
Books           0.000000
Personal        0.000000
PhD             0.000000
Terminal        0.000000
S_F_Ratio      -0.000000
perc_alumni     0.000000
Expend          0.000000
Grad_Rate       0.000000
dtype: float64
```

In [27]:
```
grader.grade(test_case_id = 'test_lassoCV', answer = (lasso_select, test_MSE_lasso))
```

```
You earned 1.5/2 points.

But, don't worry you can re-submit and we will keep only your latest score.
```

In [43]:
```
grader.grade(test_case_id = 'test_lasso_nonzero', answer = num_nonzero)
```

```
Correct! You earned 1/1 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

1. Comment on the results obtained. How accurately can we predict the number of college applications received? Is there much difference among the test errors resulting from these three approaches?

After commenting on your observations, please answer following questions:

- In ridge regression, what will be the effect on coefficients if we have an infinitely large λ compared to OLS:

  A. same coefficient values

  B. coefficients will shrink close to zero but not equal to zero

  C. some coefficients will shrink to zero

  D. all coefficients equal to zero

- Assume the model complexity remains unchanged, what is the effect of increasing λ using ridge and lasso regressions:

  A. Increasing bias and increasing variance

  B. Decreasing bias and decreasing variance

  C. Increasing bias and decreasing variance

  D. Decreasing bias and increasing variance

  E. None of the above

Please enter your answer in `answers` as a list of characters, (['F','F'])

In [44]:
```
# Add your comments and choises here
answers = ['b','c']
```

In [45]:
```
grader.grade(test_case_id = 'test_regu_ob', answer = answers)
```

```
Correct! You earned 1.0/1 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

In [ ]: