Homework 3, Part II Learning, Inference, Decisions

Nikhil Pereira nmp54

April 18th, 2022

Problem 4

Part A - Download the Data and Load into Pandas

In [17]:
```
import pandas as pd
import warnings
warnings.filterwarnings("ignore")
```

In [18]:
```
red = pd.read_csv('winequality-red.csv', sep=';')
red['Color'] = 1
white = pd.read_csv('winequality-white.csv', sep=';')
white['Color'] = 0
name_mapper = {'fixed acidity':'fixed_acidity','citric acid':'citric_acid', 'total sulfur dioxide
df = pd.concat([red,white], axis=0,ignore_index=True).rename(columns=name_mapper)
indices = list(df.index)
```

Part B - Random Sample for Train Test Split

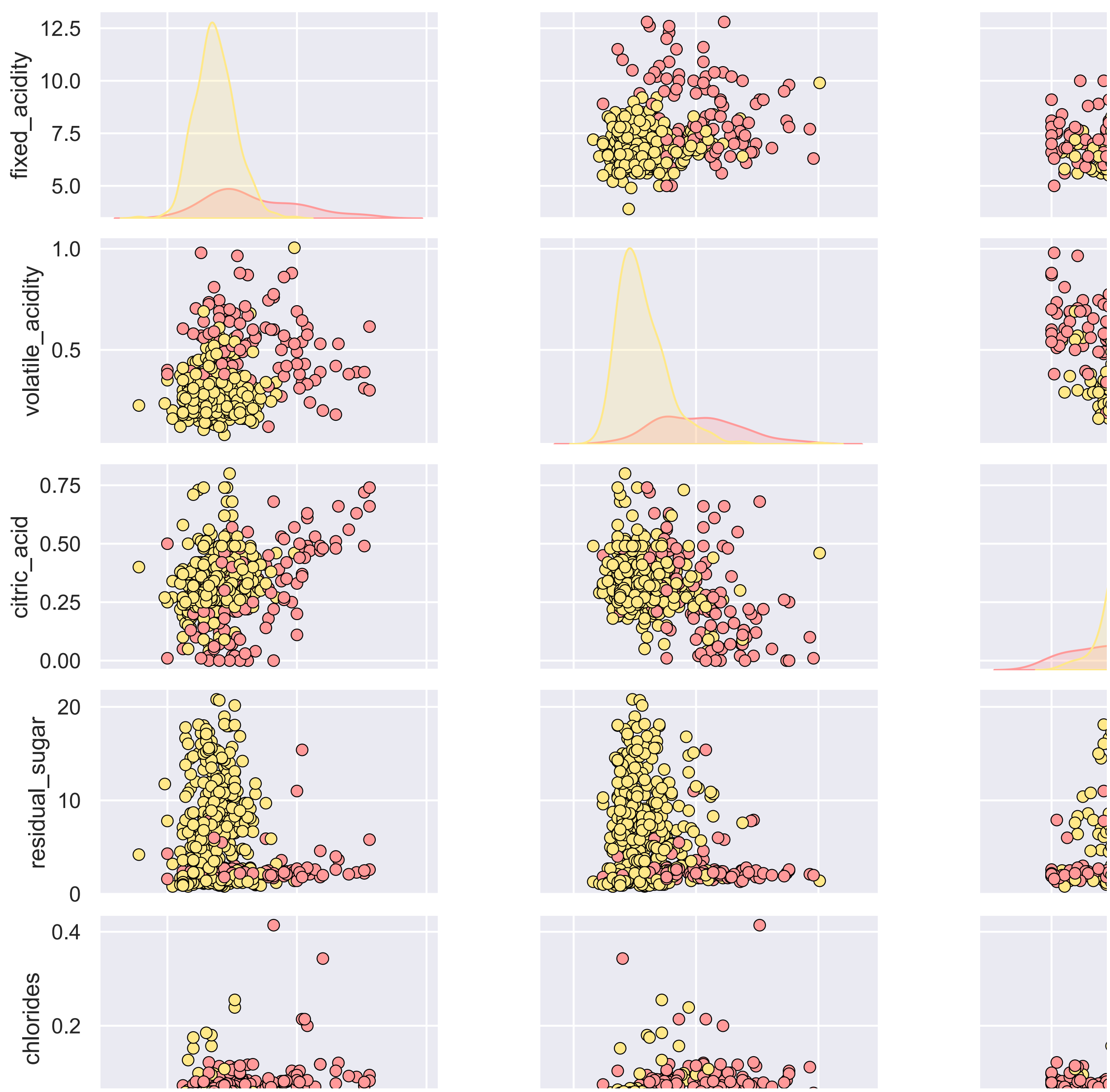
In [19]:
```
from random import sample
train_indices = list(sample(list(df.index), 5000))
test_indices = list(set(list(df.index)) - set(train_indices))
train_set = df.iloc[train_indices,:]
test_set = df.iloc[test_indices,:]
```

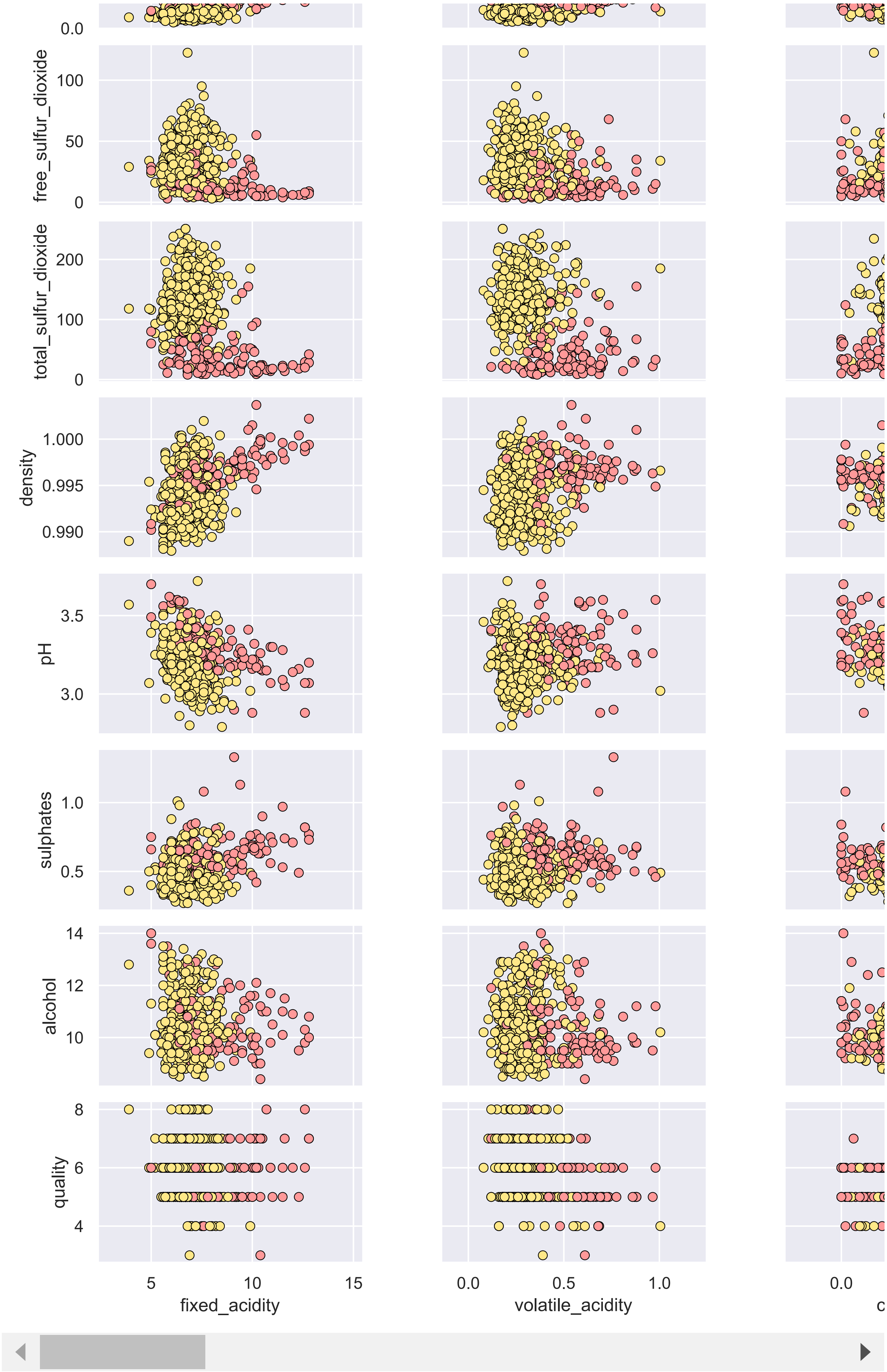
Part C - Visualizing the Wine Set All Covariates

In [11]:

```
%pylab
%matplotlib inline
%config InlineBackend.figure_format = 'svg'
import seaborn as sns
sns.set()
wt = {0:'white',1:'red'}
smaller_train_set = train_set.sample(n=500)
smaller_train_set['wine_type'] = smaller_train_set.Color.map(wt)
cols = ['fixed_acidity', 'volatile_acidity', 'citric_acid', 'residual_sugar',
       'chlorides', 'free_sulfur_dioxide', 'total_sulfur_dioxide', 'density',
       'pH', 'sulphates', 'alcohol','quality', 'wine_type']
pp = sns.pairplot(smaller_train_set[cols], hue='wine_type', height=1.8, aspect=1.8,
                  palette={"red": "#FF9999", "white": "#FFE888"},
                  plot_kws=dict(edgecolor="black", linewidth=0.5))
fig = pp.fig
fig.subplots_adjust(top=0.93, wspace=0.3)
t = fig.suptitle('Wine Attributes Pairwise Plots', fontsize=14)
```

```
Using matplotlib backend: Qt5Agg
Populating the interactive namespace from numpy and matplotlib
```




free_sulfur_dioxide
total_sulfur_dioxide
density
pH
sulphates
alcohol
quality
fixed_acidity
volatile_acidity

Analyzing Important Covariates to be Transformed

In [5]:

```
for c in train_set.columns:
    print('Column:', c, ' Range:', round(train_set[c].max()-train_set[c].min(),2), ' Mean: ',roun
```

```
Column: fixed_acidity  Range: 11.7  Mean:  7.22  Std:  1.32
Column: volatile_acidity  Range: 1.5  Mean:  0.34  Std:  0.16
Column: citric_acid  Range: 1.0  Mean:  0.32  Std:  0.14
Column: residual_sugar  Range: 31.0  Mean:  5.4  Std:  4.69
Column: chlorides  Range: 0.46  Mean:  0.06  Std:  0.03
Column: free_sulfur_dioxide  Range: 288.0  Mean:  30.58  Std:  18.06
Column: total_sulfur_dioxide  Range: 434.0  Mean:  116.03  Std:  56.73
Column: density  Range: 0.02  Mean:  0.99  Std:  0.0
Column: pH  Range: 1.29  Mean:  3.22  Std:  0.16
Column: sulphates  Range: 1.72  Mean:  0.53  Std:  0.14
Column: alcohol  Range: 6.9  Mean:  10.49  Std:  1.19
Column: quality  Range: 6  Mean:  5.82  Std:  0.88
Column: Color  Range: 1  Mean:  0.24  Std:  0.43
```

Comments on Scatter Matrix

Some notable positive correlations are density vs fixed acidity, density vs sulphates whereas notable negative correlated variables are fixed acidity vs pH and citric acid vs pH which makes sense (lower pH is less acidic).Free_sulfur_dioxide and total_sulfur_dioxide are two covariates that should be linearly transformed since there range and standard deviations are very high indiciating a poor regression if no changes are made.

Part D - Fitting a Linear Model on All Covariates to establish a baseline model

In [20]:

```
import statsmodels.stats.api as sms
import statsmodels.api as sm
import statsmodels.formula.api as smf
linear_formula = 'quality ~' + '+'.join(list(set(df.columns)-{'quality'}))
lm = smf.ols(linear_formula, data=train_set).fit()
print('In Sample R2 Baseline: ', round(lm.rsquared,3))
```

```
In Sample R2 Baseline:  0.291
```

In [21]:

```python
### Performing CV and Measuring Out Sample R^2
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.model_selection import train_test_split
import numpy as np

#Function that uses 5 fold cross validation by creating random train test splits at a 20 % thresh
# and then fits an ols model and measures the  r_2 and MSE between the y_true and y_predicted in
# Finally it returns the means of r_2 and MSE
def CV_R2(features,linear_formula,data,split):
    #Create r_2 and MSE vectors
    r_2 = np.zeros(split)
    mses= np.zeros(split)
    for i in range(split):
        #Split the data into 5 folds
        X_train, X_test, y_train, y_test = train_test_split(
        data.loc[:,features], data.quality, test_size=0.2, random_state=i)
        full_train = pd.concat([X_train,y_train],axis=1)
        #Fit Model on train set
        model = smf.ols(formula=linear_formula, data=full_train).fit()
        #Gather model predictions from test set
        predictions = model.predict(X_test)
        #Calculate the R_2 and MSE
        r_2[i] = r2_score(y_test,predictions)
        mses[i] = mean_squared_error(y_test, predictions)
    #Return the mean r_2 and mean MSE found
    return r_2.mean(), mses.mean()

features = list(set(df.columns)-{'quality'})
print('Out of Sample R^2 via Cross Validation Baseline ', CV_R2(features,linear_formula,train_set
print('Out of Sample MSE via Cross Validation Baseline', CV_R2(features,linear_formula,train_set,
```

```
Out of Sample R^2 via Cross Validation Baseline  0.28760073577986595
Out of Sample MSE via Cross Validation Baseline 0.5356495740401874
```

Part E - Linear Transformation for features from Baseline Model

In [22]:

```python
transformed = train_set.copy()
#log transform the sulfur dioxide columns as they have very large ranges
transformed['free_sulfur_dioxide'] = np.log(transformed['free_sulfur_dioxide'])
transformed['total_sulfur_dioxide'] = np.log(transformed['total_sulfur_dioxide'])

linear_formula = 'quality ~' + '+'.join(list(set(df.columns)-{'quality'}))
lm2 = smf.ols(linear_formula, data=transformed).fit()
print('In Sample R2 after linear transform: ', round(lm2.rsquared,3))
```

```
In Sample R2 after linear transform:  0.302
```

In [23]:

```python
features = list(set(df.columns)-{'quality'})
print('Out of Sample R^2 via Cross Validation on Transformed Sulfur Dioxide Columns ', CV_R2(feat
print('Out of Sample MSE via Cross Validation on Transformed Sulfur Dioxide Columns', CV_R2(featu
```

```
Out of Sample R^2 via Cross Validation on Transformed Sulfur Dioxide Columns  0.2974783257839001
5
Out of Sample MSE via Cross Validation on Transformed Sulfur Dioxide Columns 0.5282591535172012
```

After transforming the sulfur dioxide columns and performing the same procedure as in part D, the r-squared went up a percentage point and the mean squared error became smaller by very little. This is not a significant improvement.

Part F - Fitting a linear model with 2-Way Interaction Terms (Report In Sample and Out Sample R^2)

In [27]:

```
import itertools
#add two way interactions to the linear formula
linear_formula = 'quality ~' + '+'.join(list(set(df.columns)-{'quality'})+['%s:%s'%v for v in ite
lm3 = smf.ols(linear_formula, data=train_set).fit()
print('In Sample R2 with 2-Way Interaction Terms: ', round(lm3.rsquared,3))
```

```
In Sample R2 with 2-Way Interaction Terms:  0.357
```

In [28]:

```
features = list(set(df.columns)-{'quality'})
print('Out of Sample R^2 via Cross Validation with 2_way Interaction ', CV_R2(features,linear_for
print('Out of Sample MSE via Cross Validation with 2_way Interaction ', CV_R2(features,linear_for
```

```
Out of Sample R^2 via Cross Validation with 2_way Interaction  -0.047743417571907788
Out of Sample MSE via Cross Validation with 2_way Interaction  0.7828630408021304
```

After adding interaction terms the in sample R^2 improved by 6 percentage points but the out of sample r-squared and MSE declined as the model has more complexity and started to overfit.

Part G - Stepwise regression with AIC criterion based on 2-Way Interaction Model Report Estimates of Prediction Error

In [29]:

```
singlefeatures = list(set(df.columns)-{'quality'})
interactions = ['%s:%s'%v for v in itertools.combinations(singlefeatures,2)]
```

Forward Stepwise Regression

In [30]:

```
def fitmodel(S,data):
    return smf.ols('quality ~ '+('+'.join(S) if len(S)>0 else '1'), data=data).fit()
## forward stepwise
Sfwd = set()
# features = set(singlefeatures)
features = set(singlefeatures).union(set(interactions))
while len(Sfwd)<len(features):
    f = max(features - Sfwd, key = lambda f: fitmodel(Sfwd.union({f}),train_set).aic)
    after = fitmodel(Sfwd.union({f}),train_set).aic
    before = fitmodel(Sfwd,train_set).aic
    if after > before:
        Sfwd = Sfwd.union({f})
    else:
        break
print('Forward Stepwise Terms:', Sfwd)
```

```
Forward Stepwise Terms: {'residual_sugar:alcohol', 'fixed_acidity:sulphates', 'density:pH'}
```

Backwards Stepwise Regression

In [32]:

```
## backward stepwise
# features = set(singlefeatures)
features = set(singlefeatures).union(set(interactions))
Sbwd = set(features)
while len(Sbwd)>0:
    f = max(Sbwd, key = lambda f: fitmodel(Sbwd-{f},train_set).aic)
    after = fitmodel(Sbwd-{f},train_set).aic
    before = fitmodel(Sbwd,train_set).aic
    if after - before > 5:
        Sbwd = Sbwd-{f}
    else:
        break
print('Backwards Stepwise Terms:', Sbwd)
```

```
Backwards Stepwise Terms: {'fixed_acidity:residual_sugar', 'total_sulfur_dioxide:pH', 'alcohol',
'residual_sugar:volatile_acidity', 'pH', 'volatile_acidity:chlorides', 'citric_acid:Color', 'res
idual_sugar:pH', 'fixed_acidity', 'residual_sugar:Color', 'Color', 'total_sulfur_dioxide:residua
l_sugar', 'chlorides', 'fixed_acidity:sulphates', 'density:Color', 'fixed_acidity:free_sulfur_di
oxide', 'free_sulfur_dioxide:alcohol', 'free_sulfur_dioxide:volatile_acidity', 'free_sulfur_diox
ide', 'density:alcohol', 'density:volatile_acidity', 'citric_acid:volatile_acidity', 'total_sulf
ur_dioxide:chlorides', 'sulphates', 'residual_sugar:alcohol', 'sulphates:alcohol', 'sulphates:ch
lorides', 'sulphates:volatile_acidity', 'fixed_acidity:alcohol', 'fixed_acidity:total_sulfur_dio
xide', 'alcohol:Color', 'citric_acid:total_sulfur_dioxide', 'sulphates:pH', 'citric_acid:residua
l_sugar', 'total_sulfur_dioxide:density', 'fixed_acidity:density', 'citric_acid', 'fixed_acidit
y:volatile_acidity', 'total_sulfur_dioxide', 'density:pH', 'citric_acid:free_sulfur_dioxide', 'c
itric_acid:density', 'citric_acid:sulphates', 'sulphates:residual_sugar', 'residual_sugar', 'fre
e_sulfur_dioxide:chlorides', 'volatile_acidity', 'citric_acid:chlorides', 'alcohol:pH', 'Color:c
hlorides', 'total_sulfur_dioxide:Color', 'density:chlorides', 'total_sulfur_dioxide:alcohol', 'r
esidual_sugar:chlorides', 'free_sulfur_dioxide:density', 'total_sulfur_dioxide:volatile_acidit
y', 'fixed_acidity:chlorides', 'free_sulfur_dioxide:Color', 'free_sulfur_dioxide:sulphates', 'fr
ee_sulfur_dioxide:pH', 'sulphates:Color', 'free_sulfur_dioxide:residual_sugar', 'density:sulphat
es', 'alcohol:chlorides', 'pH:volatile_acidity'}
```

Considering the optimal AIC from Forward and Backward Stepwise

In [33]:

```
## stepwise
Sboth = Sfwd if fitmodel(Sfwd,train_set).aic > fitmodel(Sbwd,train_set).aic else Sbwd
print(Sboth)
```

```
{'residual_sugar:alcohol', 'fixed_acidity:sulphates', 'density:pH'}
```

OLS Model with Forward Stepwise Terms

In [34]:

```
#fitting model based on optimal AIC stepwise regression
linear_formula = 'quality ~' + '+'.join(list(Sboth))
lm4 = smf.ols(linear_formula, data=train_set).fit()
print('In Sample R2 After AIC Stepwise: ', round(lm4.rsquared,3))
```

```
In Sample R2 After AIC Stepwise:  0.0
```

In [35]:

```
features = list(set(df.columns)-{'quality'})
print('Out of Sample R^2 via Cross Validation with AIC Stepwise Regression ', CV_R2(features,line
print('Out of Sample MSE via Cross Validation with AIC Stepwise Regression ', CV_R2(features,line
```

```
Out of Sample R^2 via Cross Validation with AIC Stepwise Regression  -0.0009112009929356901
Out of Sample MSE via Cross Validation with AIC Stepwise Regression  0.752974226749382
```

Since the forward regression only produced an optimal AIC with {'residual_sugar:alcohol', 'fixed_acidity:sulphates', 'density:pH'} terms, the MSE is decently high and the R-squared is near 0 because the coefficients are nearly 0. Indicating that this model is mostly a horizontal line.

OLS Model with Backwards Stepwise Terms

In [37]:
```
#fitting model based on optimal AIC stepwise regression via Stepwise Backwards
linear_formula = 'quality ~' + '+'.join(list(Sbwd))
lm5= smf.ols(linear_formula, data=train_set).fit()
print('In Sample R2 After AIC Stepwise Backward Regression: ', round(lm5.rsquared,3))
```

```
In Sample R2 After AIC Stepwise Backward Regression:  0.321
```

In [38]:
```
features = list(set(df.columns)-{'quality'})
print('Out of Sample R^2 via Cross Validation with AIC Stepwise Regression Backwards ', CV_R2(fea
print('Out of Sample MSE via Cross Validation with AIC Stepwise Regression Backwards ', CV_R2(fea
```

```
Out of Sample R^2 via Cross Validation with AIC Stepwise Regression Backwards  0.153212368236144
56
Out of Sample MSE via Cross Validation with AIC Stepwise Regression Backwards  0.634475431278334
3
```

Since the backward stepwise regression produced an optimal AIC with many terms, the MSE and r-squared out of sample did not improve. Indicating that this model is adding more noise then true correlations and overfitting the training set. The backwards stepwise regression contains these terms:

{'fixed_acidity:residual_sugar', 'total_sulfur_dioxide:pH', 'alcohol', 'residual_sugar:volatile_acidity', 'pH', 'volatile_acidity:chlorides', 'citric_acid:Color', 'residual_sugar:pH', 'fixed_acidity', 'residual_sugar:Color', 'Color', 'total_sulfur_dioxide:residual_sugar', 'chlorides', 'fixed_acidity:sulphates', 'density:Color', 'fixed_acidity:free_sulfur_dioxide', 'free_sulfur_dioxide:alcohol', 'free_sulfur_dioxide:volatile_acidity', 'free_sulfur_dioxide', 'density:alcohol', 'density:volatile_acidity', 'citric_acid:volatile_acidity', 'total_sulfur_dioxide:chlorides', 'sulphates', 'residual_sugar:alcohol', 'sulphates:alcohol', 'sulphates:chlorides', 'sulphates:volatile_acidity', 'fixed_acidity:alcohol', 'fixed_acidity:total_sulfur_dioxide', 'alcohol:Color', 'citric_acid:total_sulfur_dioxide', 'sulphates:pH', 'citric_acid:residual_sugar', 'total_sulfur_dioxide:density', 'fixed_acidity:density', 'citric_acid', 'fixed_acidity:volatile_acidity', 'total_sulfur_dioxide', 'density:pH', 'citric_acid:free_sulfur_dioxide', 'citric_acid:density', 'citric_acid:sulphates', 'sulphates:residual_sugar', 'residual_sugar', 'free_sulfur_dioxide:chlorides', 'volatile_acidity', 'citric_acid:chlorides', 'alcohol:pH', 'Color:chlorides', 'total_sulfur_dioxide:Color', 'density:chlorides', 'total_sulfur_dioxide:alcohol', 'residual_sugar:chlorides', 'free_sulfur_dioxide:density', 'total_sulfur_dioxide:volatile_acidity', 'fixed_acidity:chlorides', 'free_sulfur_dioxide:Color', 'free_sulfur_dioxide:sulphates', 'free_sulfur_dioxide:pH', 'sulphates:Color', 'free_sulfur_dioxide:residual_sugar', 'density:sulphates', 'alcohol:chlorides', 'pH:volatile_acidity'}

Part H - Fit a Lasso model of 2-Way Interaction terms Report Estimates of Prediction Error

In [29]:
```
from sklearn import linear_model, model_selection, tree, ensemble
from sklearn.linear_model import LassoCV

features = set(singlefeatures).union(set(interactions))
X = train_set.loc[:,list(set(df.columns)-{'quality'})]
#adding the interaction columns because Sklearn needs every column unlike stats models
for terms in interactions:
    first,second=terms.split(':')
    #creating a multiplication for the two columns as a two way interaction
    X[terms] = X[first]*X[second]

y = train_set.quality

lasso = LassoCV(cv=5, random_state=0).fit(X, y)
lassofeatures = list(zip(lasso.feature_names_in_,np.abs(lasso.coef_)>1e-10)) #getting the lasso f
lassofeatures = [x[0] for x in lassofeatures if x[1] == True] #filtering by the ones that are tru
lamb = lasso.alpha_
print('Lambda via CV Lasso ', lamb)
print('Lasso Features ', lassofeatures)
```

```
Lambda via CV Lasso  0.03914400323000001
Lasso Features  ['total_sulfur_dioxide', 'free_sulfur_dioxide:alcohol', 'free_sulfur_dioxide:p
H', 'free_sulfur_dioxide:Color', 'free_sulfur_dioxide:total_sulfur_dioxide', 'free_sulfur_dioxid
e:residual_sugar', 'free_sulfur_dioxide:fixed_acidity', 'alcohol:volatile_acidity', 'alcohol:sul
phates', 'alcohol:pH', 'alcohol:total_sulfur_dioxide', 'alcohol:residual_sugar', 'alcohol:fixed_
acidity', 'chlorides:total_sulfur_dioxide', 'volatile_acidity:total_sulfur_dioxide', 'sulphates:
total_sulfur_dioxide', 'pH:total_sulfur_dioxide', 'Color:total_sulfur_dioxide', 'total_sulfur_di
oxide:residual_sugar', 'total_sulfur_dioxide:fixed_acidity', 'residual_sugar:fixed_acidity']
```

In [30]:
```
linear_formula = 'quality ~' + '+'.join(list(lassofeatures))
lm6= smf.ols(linear_formula, data=train_set).fit()
print('In Sample R2 After Lasso Feature Selection ', round(lm6.rsquared,3))
```

```
In Sample R2 After Lasso Feature Selection  0.319
```

In [31]:
```
features = list(set(df.columns)-{'quality'})
print('Out of Sample R^2 via Cross Validation with Lasso Features', CV_R2(features,linear_formula
print('Out of Sample MSE via Cross Validation with Lasso Features', CV_R2(features,linear_formula
```

```
Out of Sample R^2 via Cross Validation with Lasso Features 0.3009848094030164
Out of Sample MSE via Cross Validation with Lasso Features 0.5267910390150287
```

After performing Lasso for two-way interactions and single dimension features with 5-fold cv, the lambda is 0.039. The insample r-squared was 0.319 and the out of sample r_squared is 0.3 with an MSE of 0.5216. This is the best performance we have seen so far. The Lasso terms includes ['total_sulfur_dioxide', 'free_sulfur_dioxide:alcohol', 'free_sulfur_dioxide:pH', 'free_sulfur_dioxide:Color', 'free_sulfur_dioxide:total_sulfur_dioxide', 'free_sulfur_dioxide:residual_sugar', 'free_sulfur_dioxide:fixed_acidity', 'alcohol:volatile_acidity', 'alcohol:sulphates', 'alcohol:pH', 'alcohol:total_sulfur_dioxide', 'alcohol:residual_sugar', 'alcohol:fixed_acidity', 'chlorides:total_sulfur_dioxide', 'volatile_acidity:total_sulfur_dioxide', 'sulphates:total_sulfur_dioxide', 'pH:total_sulfur_dioxide', 'Color:total_sulfur_dioxide', 'total_sulfur_dioxide:residual_sugar', 'total_sulfur_dioxide:fixed_acidity', 'residual_sugar:fixed_acidity']

Part I - Pick favorite model and compare to the previous models

In [32]:
```
linear_formula = 'quality ~' + '+'.join(list(lassofeatures) + singlefeatures)
lm7= smf.ols(linear_formula, data=train_set).fit()
print('In Sample R2 After Lasso + Single Feature Selection ', round(lm7.rsquared,3))
features = list(set(df.columns)-{'quality'})
print('Out of Sample R^2 via Cross Validation with Lasso Features', CV_R2(features,linear_formula
print('Out of Sample MSE via Cross Validation with Lasso Features', CV_R2(features,linear_formula
```

```
In Sample R2 After Lasso + Single Feature Selection  0.333
Out of Sample R^2 via Cross Validation with Lasso Features 0.3070809199544662
Out of Sample MSE via Cross Validation with Lasso Features 0.5221288590003825
```

This model that combines the Lasso 2-way interaction terms with all single features yields the highest out of sample r-squared at 0.307 compare to previous models. The MSE for out of sample test errors in 0.522 which is marginally better than the LassoCV model.

Part J - Create a 95% Wald predictive confidence for wine quality

In [48]:

```
#Function that generates the Wald 95% CI and tests to see if true wine quality is in interval for
#Uses the model trained in part F for all single and two-way interaction terms
def generateWaldConfidenceIntervals(train_set, test_set):
    #Model from part F single and two way interactions
    linear_formula = 'quality ~' + '+'.join(list(set(df.columns)-{'quality'})+['%s:%s'%v for v in
    lm3 = smf.ols(linear_formula, data=train_set).fit()
    #Gets the list of confidence intervals for the test_set records using Stat's Modles conf_int
    confidence_intervals = lm3.get_prediction(test_set).conf_int(obs=True, alpha=0.05)
    #Obtain the true wine quality
    true_quality = list(test_set.quality)
    correct = 0
    #Loop through the two lists and count whether the true wine quality is in the interval
    for i,interval in enumerate(confidence_intervals):
        if interval[0] <= true_quality[i] <= interval[1]:
            correct+=1
    #Returns the percentage of true wine quality
    return correct/len(true_quality)
correct = generateWaldConfidenceIntervals(train_set, test_set)
print('One run of WI and percentage of times contains true wine quality ', correct)
```

```
One run of WI and percentage of times contains true wine quality  0.9465597862391449
```

Repeating Wald CI Experiment 100x over randomly generated train-test splits

In [47]:

```
#running experiment 100 times
from random import sample
waldIntervalsCorrect = np.zeros(100)
for i in range(100):
    #Obtains random samples
    train_indices = list(sample(list(df.index), 5000))
    test_indices = list(set(list(df.index)) - set(train_indices))
    train_set = df.iloc[train_indices,:]
    test_set = df.iloc[test_indices,:]
    #Store percentage correct for each experiment run
    waldIntervalsCorrect[i] = generateWaldConfidenceIntervals(train_set, test_set)
print('Percentage WI contrains true quality', waldIntervalsCorrect.mean(), 'Standard Deviation',w
```

```
Percentage WI contrains true quality 0.9458984635938544 Standard Deviation 0.005453533066307316
```

Part K - Repeat Part J but with Stepwise Regression

I am using forward stepwise regression because it is computationally faster than backwards and has the lowest AIC. After getting the Sfwd features I run an OLS model to get predictions on test set and determine if the actual wine quality lies in the interval.

In [55]:

```python
#Function that does the same steps as above but calls Forward Stepwise regression because in Part
#forward stepwise regression was chosen over backwards as it is computationally faster
def generateWaldConfidenceIntervalsStepWise(train_set, test_set, Sfwd):
    linear_formula = 'quality ~' + '+'.join(list(Sfwd))
    lm4 = smf.ols(linear_formula, data=train_set).fit()
    confidence_intervals = lm4.get_prediction(test_set).conf_int(obs=True, alpha=0.05)
    true_quality = list(test_set.quality)
    correct = 0
    for i,interval in enumerate(confidence_intervals):
        if interval[0] <= true_quality[i] <= interval[1]:
            correct+=1
    return correct/len(true_quality)

def fitmodel(S,data):
    return smf.ols('quality ~ '+('+'.join(S) if len(S)>0 else '1'), data=data).fit()


def forwardStepWise(singlefeatures, interactions, train_set):
    ## forward stepwise
    Sfwd = set()
    # features = set(singlefeatures)
    features = set(singlefeatures).union(set(interactions))
    while len(Sfwd)<len(features):
        f = max(features - Sfwd, key = lambda f: fitmodel(Sfwd.union({f}),train_set).aic)
        after = fitmodel(Sfwd.union({f}),train_set).aic
        before = fitmodel(Sfwd,train_set).aic
        if after > before:
            Sfwd = Sfwd.union({f})
        else:
            break
    return Sfwd

singlefeatures = list(set(df.columns)-{'quality'})
interactions = ['%s:%s'%v for v in itertools.combinations(singlefeatures,2)]
Sfwd = forwardStepWise(singlefeatures, interactions, train_set)
correct = generateWaldConfidenceIntervalsStepWise(train_set, test_set, Sfwd)
print('One run of Stepwise trained WI and percentage of times contains true wine quality ', corre
```

```
One run of Stepwise trained WI and percentage of times contains true wine quality  0.93319973279
89312
```

Repeating Wald CI Experiment 10 over randomly generated train-test splits.

My computer was taking very long to run 100 experiments. I also use forward stepwise regression because the algorithm finds features much faster than backwards. After getting the Sfwd features I run an OLS model to get predictions on test set and determine if the actual wine quality lies in the interval.

In [56]:

```python
#running experiment 10 times
from random import sample
n = 10
waldIntervalsCorrect = np.zeros(n)
for i in range(n):
    train_indices = list(sample(list(df.index), 5000))
    test_indices = list(set(list(df.index)) - set(train_indices))
    train_set = df.iloc[train_indices,:]
    test_set = df.iloc[test_indices,:]
    Sfwd = forwardStepWise(singlefeatures, interactions, train_set)
    waldIntervalsCorrect[i] = generateWaldConfidenceIntervalsStepWise(train_set, test_set, Sfwd)
print('Percentage WI contrains true quality', waldIntervalsCorrect.mean(), 'Standard Deviation',w
```

```
Percentage WI contrains true quality 0.9345357381429527 Standard Deviation 0.007744222710054484
```

Part L - Use Bootstrap to generate confidence intervals

Approach: Since the question was not entirely clear I assumed that the goal is to use bootstrap to get a pivotal CI for the mean of wine quality. I am using the stepwise regression from part K to fit a model and then calculate the mean of the predicted wine quality and testing if its in the bootstrap CI. I run this experiment 25 times of speed.

In [39]:
```
from arch.bootstrap import IIDBootstrap
```

In [69]:
```
def generateBootStrapCI(train_set, test_set, Sfwd):
    #Fit a model based on Sfwd features
    linear_formula = 'quality ~' + '+'.join(list(Sfwd))
    lm4 = smf.ols(linear_formula, data=train_set).fit()
    #Theta function set to the mean
    thetafn = lambda z: np.mean(z)
    #Get the predctions from the model
    predictions = list(lm4.predict(test_set))
    thetahat = thetafn(predictions)
    #Using arch bootstrap function to generate pivotal CI
    CI = IIDBootstrap(test_set.quality).conf_int(
        thetafn,
        1000,
        method='basic')
    #return 1 if the the is in the interval or not
    if CI[0] <= thetahat <= CI[1]:
        return 1
    else:
        return 0

from random import sample
n = 25
bootIntervalCorrect = np.zeros(n)
for i in range(n):
    #resample the train and test sets
    train_indices = list(sample(list(df.index), 5000))
    test_indices = list(set(list(df.index)) - set(train_indices))
    train_set = df.iloc[train_indices,:]
    test_set = df.iloc[test_indices,:]
    #get the forward step wise features
    Sfwd = forwardStepWise(singlefeatures, interactions, train_set)
    #Calculate if the mean of the predictions is in the pivotal bootstrap CI
    bootIntervalCorrect[i] = generateBootStrapCI(train_set, test_set, Sfwd)

print('Percentage Bootstrap CI for mean contrains true quality', bootIntervalCorrect.mean(), 'Sta
```

```
Percentage Bootstrap CI for mean contrains true quality 0.88 Standard Deviation 0.32496153618543
844
```

Part M - Repeat Pivotal Bootstrap for Mean using Lasso Model

Repeated the general steps in part K except train a 5-Fold Lasso CV model. Counting the number of times the predicted wine quality mean lies in the Bootstrap CI generated.

In [70]:

```python
from sklearn import linear_model, model_selection, tree, ensemble
from sklearn.linear_model import LassoCV

def generateBootStrapCILasso(train_set, test_set, lassofeatures):
    #Fit a model based on Sfwd features
    linear_formula = 'quality ~' + '+'.join(list(lassofeatures))
    lm6= smf.ols(linear_formula, data=train_set).fit()
    #Theta function set to the mean
    thetafn = lambda z: np.mean(z)
    #Get the predctions from the model
    predictions = list(lm6.predict(test_set))
    thetahat = thetafn(predictions)
    #Using arch bootstrap function to generate pivotal CI
    CI = IIDBootstrap(test_set.quality).conf_int(
        thetafn,
        1000,
        method='basic')
    #return 1 if the mean of the predictions is in the bootstrap pivotal interval
    if CI[0] <= thetahat <= CI[1]:
        return 1
    else:
        return 0

def lassoFeatures(train_set,df,singlefeatures,interactions):
    features = set(singlefeatures).union(set(interactions))
    X = train_set.loc[:,list(set(df.columns)-{'quality'})]
    #adding the interaction columns because Sklearn needs every column unlike stats models
    for terms in interactions:
        first,second=terms.split(':')
        #creating a multiplication for the two columns as a two way interaction
        X[terms] = X[first]*X[second]

    y = train_set.quality
    lasso = LassoCV(cv=5, random_state=0).fit(X, y)
    lassofeatures = list(zip(lasso.feature_names_in_,np.abs(lasso.coef_)>1e-10)) #getting the las
    lassofeatures = [x[0] for x in lassofeatures if x[1] == True] #filtering by the ones that are
    return lassofeatures

#running Lasso Experiment
from random import sample
n = 25
bootIntervalCorrect = np.zeros(n)
for i in range(n):
    #resample the train and test sets
    train_indices = list(sample(list(df.index), 5000))
    test_indices = list(set(list(df.index)) - set(train_indices))
    train_set = df.iloc[train_indices,:]
    test_set = df.iloc[test_indices,:]
    #get the forward step wise features
    lasso = lassoFeatures(train_set,df,singlefeatures,interactions)
    #Calculate if the mean of the predictions is in the pivotal bootstrap CI
    bootIntervalCorrect[i] = generateBootStrapCILasso(train_set, test_set, lasso)

print('Percentage Lasso Bootstrap CI for mean contrains true quality', bootIntervalCorrect.mean()
```

```
Percentage Lasso Bootstrap CI for mean contrains true quality 0.96 Standard Deviation 0.19595917
942265428
```

Problem 5

In [10]:

```python
### Import data
import pandas as pd
import statsmodels.api as sm
df2 = pd.DataFrame(sm.datasets.star98.load().data)
```

Part A - Simple OLS Student Math Scores vs Per-Pupil Spending

Run a simple regression to investigate whether student outcomes, as the fraction of students above the national mean, are affected by per-pupil spending (hint: if A and B are two column names, you can use I(A/(A+B))~... as a specification of the response variable in statsmodels formula). Interpret the results – do you think there is a causal effect? Why or why not?

In [18]:

```
import statsmodels.formula.api as smf
response = 'I((NABOVE)/(NABOVE+NBELOW)) ~ '
features = 'PERSPENK'
linear_formula = response + features
lm = smf.ols(linear_formula, data=df2).fit()
perspenkCoeff = lm.params['PERSPENK']
print('PERSPENK COEFF: ', perspenkCoeff)
lm.summary()
```

```
PERSPENK COEFF:  0.04917284368069601
```

Out[18]:

OLS Regression Results

Dep. Variable:	I((NABOVE) / (NABOVE + NBELOW))	**R-squared:**	0.025
Model:	OLS	**Adj. R-squared:**	0.022
Method:	Least Squares	**F-statistic:**	7.665
Date:	Mon, 18 Apr 2022	**Prob (F-statistic):**	0.00598
Time:	10:54:41	**Log-Likelihood:**	86.908
No. Observations:	303	**AIC:**	-169.8
Df Residuals:	301	**BIC:**	-162.4
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	0.2245	0.077	2.899	0.004	0.072	0.377
PERSPENK	0.0492	0.018	2.769	0.006	0.014	0.084

Omnibus:	8.083	**Durbin-Watson:**	1.646
Prob(Omnibus):	0.018	**Jarque-Bera (JB):**	6.923
Skew:	0.293	**Prob(JB):**	0.0314
Kurtosis:	2.546	**Cond. No.**	33.9

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

Interpretation of Simple OLS PERSPENK

Looking at the OLS Regression Results we can see that the R^2 is very small 0.025 which means that the variation of Y is not well explained by the covariate set 'PERSPENK'. While this coefficient is significant and positively correlated with math score we cannot conclude causal effects as we have not enforced randomization of assignment to treatment or used ignorability principles to compare like-to-like counties.

Part B - Simple OLS Student Math Scores vs Pupil-Teacher Ratio

In [12]:

```
response2 = 'I((NABOVE)/(NABOVE+NBELOW)) ~ '
features2 = 'PTRATIO'
linear_formula2 = response2 + features2
lm2 = smf.ols(linear_formula2, data=df2).fit()
ptratioCoeff = lm2.params['PTRATIO']
print('PTRATIO COEFF: ', ptratioCoeff)
lm2.summary()
```

```
PTRATIO COEFF:  -0.014078019457326896
```

Out[12]:

OLS Regression Results

Dep. Variable:	I((NABOVE) / (NABOVE + NBELOW))	**R-squared:**	0.029
Model:	OLS	**Adj. R-squared:**	0.025
Method:	Least Squares	**F-statistic:**	8.853
Date:	Fri, 15 Apr 2022	**Prob (F-statistic):**	0.00316
Time:	19:03:14	**Log-Likelihood:**	87.490
No. Observations:	303	**AIC:**	-171.0
Df Residuals:	301	**BIC:**	-163.6
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	0.7532	0.107	7.052	0.000	0.543	0.963
PTRATIO	-0.0141	0.005	-2.975	0.003	-0.023	-0.005

Omnibus:	5.689	**Durbin-Watson:**	1.643
Prob(Omnibus):	0.058	**Jarque-Bera (JB):**	5.583
Skew:	0.294	**Prob(JB):**	0.0613
Kurtosis:	2.689	**Cond. No.**	231.

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

Interpreation of OLS PTRatio

Looking at the OLS Regression Results we can see that the R^2 is very small 0.020 which means that the variation of Y is not well explained by the covariate set 'PTRATIO' alone. While this coefficient is significant and negatively correlated with math score we cannot conclude causal effects as we have not enforced randomization of assignment to treatment or used ignorability principles to compare like-to-like counties.

Part C - Adding Features to the OLS Model

For each of part (a) and part (b), try including additional variables in the following order:

Average teacher experience

Average teacher experience and student race

Finally, run a regression against all variables (but exclude the interactions terms). For each case, report the change in the effect coefficient for each model.

In [13]:
```
# Case 1 Part A with AVYRSEXP
response = 'I((NABOVE)/(NABOVE+NBELOW)) ~ '
features = 'PERSPENK' + '+' + 'AVYRSEXP'
linear_formula = response + features
lm = smf.ols(linear_formula, data=df2).fit()
perspenkCoeff1 = lm.params['PERSPENK']
print('PERSPENK COEFF CASE 1: ', perspenkCoeff1)
print('PERSPENK Coefficient Difference after adding AVYRSEXP ', round(perspenkCoeff1 - perspenkCo
```

```
PERSPENK COEFF CASE 1:  0.028124399361265873
PERSPENK Coefficient Difference after adding AVYRSEXP  -0.021048
```

In [14]:
```
# Case 2 Part A with AVYRSEXP + Student Race
response = 'I((NABOVE)/(NABOVE+NBELOW)) ~ '
features = ['PERSPENK','AVYRSEXP', 'PERASIAN', 'PERBLACK', 'PERHISP']
linear_formula = response + '+'.join(features)
lm = smf.ols(linear_formula, data=df2).fit()
perspenkCoeff2 = lm.params['PERSPENK']
print('PERSPENK COEFF CASE 2: ', perspenkCoeff2)
print('PERSPENK Coefficient Difference after adding AVYRSEXP + STUDENT RACE ', round(perspenkCoef
```

```
PERSPENK COEFF CASE 2:  -0.002637582935566419
PERSPENK Coefficient Difference after adding AVYRSEXP + STUDENT RACE  -0.05181
```

In [15]:
```
# Case 1 Part B with AVYRSEXP
response2 = 'I((NABOVE)/(NABOVE+NBELOW)) ~ '
features2 = 'PTRATIO' '+' + 'AVYRSEXP'
linear_formula2 = response2 + features2
lm2 = smf.ols(linear_formula2, data=df2).fit()
ptratioCoeff1 = lm2.params['PTRATIO']
print('PTRATIO COEFF CASE 1: ', ptratioCoeff1)
print('PTRATIO Coefficient Difference after adding AVYRSEXP ', round(ptratioCoeff1 - ptratioCoeff
```

```
PTRATIO COEFF CASE 1:  -0.01480897162953032
PTRATIO Coefficient Difference after adding AVYRSEXP  -0.000731
```

In [16]:
```
# Case 2 Part B with AVYRSEXP + Student Race
response2 = 'I((NABOVE)/(NABOVE+NBELOW)) ~ '
features2 = ['PTRATIO','AVYRSEXP', 'PERASIAN', 'PERBLACK', 'PERHISP']
linear_formula2 = response2 + '+'.join(features2)
lm2 = smf.ols(linear_formula2, data=df2).fit()
ptratioCoeff2 = lm2.params['PTRATIO']
print('PTRATIO COEFF CASE 2: ', ptratioCoeff2)
print('PTRATIO Coefficient Difference after adding AVYRSEXP + Student Race ', round(ptratioCoeff2
```

```
PTRATIO COEFF CASE 2:  0.002653121266625309
PTRATIO Coefficient Difference after adding AVYRSEXP + Student Race  0.016731
```

In [17]:
```
# Case 3 - Finally, run a regression against all variables (but exclude the interactions terms).
features = list(df2.columns)
single_features = [x for x in features if '_' not in x][2:]
response = 'I((NABOVE)/(NABOVE+NBELOW)) ~ '
linear_formula = response + '+'.join(single_features)
lm = smf.ols(linear_formula, data=df2).fit()
perspenkCoeff3 = lm.params['PERSPENK']
print('PERSPENK COEFF CASE 3: ', perspenkCoeff3)
ptratioCoeff3 = lm.params['PTRATIO']
print('PTRATIO COEFF CASE 3: ', ptratioCoeff3)
print('PERSPENK Coefficient Difference after All Single Features ', round(perspenkCoeff3 - perspe
print('PTRATIO Coefficient Difference after All Single Features', round(ptratioCoeff3 - ptratioCo
```

```
PERSPENK COEFF CASE 3:  0.01024358094521226
PTRATIO COEFF CASE 3:  -0.000980990866091552
PERSPENK Coefficient Difference after All Single Features  -0.038929
PTRATIO Coefficient Difference after All Single Features 0.013097
```

Conclusion of Coefficient Analysis

As we added more terms for Model A the PERSPENK coefficient had a smaller effect than the original model. For Model B the PTRATIO started from a negative effect and grew closer to 0 as we added terms. For both experiments the addition of Student Race covariates made the coefficients flip (PERSPENK positive to negative and PTRATIO negative to positive).

Part D - Experimenting with Model and Interaction Terms and Observing changes in coefficients

Experiment with the data to come up with your favorite model for student outcomes. Try adding the interaction terms. What do you observe? How do the coefficients change?

Running a model against full feature set

In [51]:

```python
response = 'I((NABOVE)/(NABOVE+NBELOW)) ~ '
features = df2.columns
#Removing features that are not significant
features = set(features) #- {'NABOVE','NBELOW','PERMINTE','AVYRSEXP','AVSALK','PCTAF','PCTCHRT','
interactions = '+'.join(list(interaction))
linear_formula = response + '+'.join(list(features))
lm9 = smf.ols(linear_formula, data=df2).fit()
lm9.summary()
```

Out[51]:

OLS Regression Results

Dep. Variable:	I((NABOVE) / (NABOVE + NBELOW))	**R-squared:**	0.830
Model:	OLS	**Adj. R-squared:**	0.817
Method:	Least Squares	**F-statistic:**	62.15
Date:	Mon, 18 Apr 2022	**Prob (F-statistic):**	3.28e-94
Time:	11:15:36	**Log-Likelihood:**	351.58
No. Observations:	303	**AIC:**	-657.2
Df Residuals:	280	**BIC:**	-571.7
Df Model:	22		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	0.5039	0.916	0.550	0.583	-1.299	2.306
PERASIAN	0.0021	0.001	3.390	0.001	0.001	0.003
PERSPENK	0.0036	0.155	0.023	0.982	-0.302	0.309
NABOVE	9.986e-05	1.95e-05	5.124	0.000	6.15e-05	0.000
PERMINTE_AVYRSEXP	-0.0012	0.002	-0.480	0.632	-0.006	0.004
PERBLACK	-0.0042	0.001	-5.760	0.000	-0.006	-0.003
AVSALK	-0.0019	0.011	-0.162	0.872	-0.024	0.021
PCTCHRT	-0.0008	0.001	-0.986	0.325	-0.002	0.001
PERSPEN_PCTAF	0.0004	0.004	0.108	0.914	-0.007	0.007
LOWINC	-0.0042	0.000	-10.934	0.000	-0.005	-0.003
PCTAF	0.0045	0.017	0.262	0.793	-0.029	0.038
AVYRSEXP_AVSAL	0.0004	0.001	0.453	0.651	-0.001	0.002
PTRATIO	0.0170	0.034	0.506	0.613	-0.049	0.083
PERMINTE_AVSAL	-0.0004	0.001	-0.743	0.458	-0.002	0.001
PERMINTE	0.0267	0.035	0.765	0.445	-0.042	0.095
PERSPEN_PTRATIO_PCTAF	5.647e-05	0.000	0.321	0.749	-0.000	0.000
PCTYRRND	-0.0004	0.000	-1.819	0.070	-0.001	3.42e-05
NBELOW	-3.456e-05	8.65e-06	-3.995	0.000	-5.16e-05	-1.75e-05
PERHISP	-0.0019	0.000	-5.280	0.000	-0.003	-0.001
PERMINTE_AVYRSEXP_AVSAL	1.98e-05	4.1e-05	0.483	0.629	-6.09e-05	0.000
PTRATIO_PCTAF	-0.0005	0.001	-0.583	0.560	-0.002	0.001
PERSPEN_PTRATIO	-0.0023	0.008	-0.307	0.759	-0.017	0.013
AVYRSEXP	-0.0168	0.046	-0.367	0.714	-0.107	0.073

Omnibus:	5.088	**Durbin-Watson:**	1.927
Prob(Omnibus):	0.079	**Jarque-Bera (JB):**	7.017

Skew:	-0.019	**Prob(JB):**	0.0299
Kurtosis:	3.745	**Cond. No.**	3.56e+06

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.56e+06. This might indicate that there are
strong multicollinearity or other numerical problems.

Running a model after removing insignificant features based on p-val

In [52]:

```python
response = 'I((NABOVE)/(NABOVE+NBELOW)) ~ '
features = df2.columns
#Removing features that are not significant
features = set(features) - {'NABOVE','NBELOW','PERMINTE','AVYRSEXP','AVSALK','PCTAF','PCTCHRT','P
linear_formula = response + '+'.join(list(features))
lm10 = smf.ols(linear_formula, data=df2).fit()
lm10.summary()
```

Out[52]:

OLS Regression Results

Dep. Variable:	I((NABOVE) / (NABOVE + NBELOW))	**R-squared:**	0.808
Model:	OLS	**Adj. R-squared:**	0.800
Method:	Least Squares	**F-statistic:**	101.5
Date:	Mon, 18 Apr 2022	**Prob (F-statistic):**	3.07e-96
Time:	11:28:57	**Log-Likelihood:**	332.88
No. Observations:	303	**AIC:**	-639.8
Df Residuals:	290	**BIC:**	-591.5
Df Model:	12		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	0.5240	0.105	4.968	0.000	0.316	0.732
PERMINTE_AVYRSEXP	0.0005	0.000	1.756	0.080	-5.88e-05	0.001
PERBLACK	-0.0045	0.001	-6.428	0.000	-0.006	-0.003
PCTYRRND	-0.0004	0.000	-1.914	0.057	-0.001	1.24e-05
PERASIAN	0.0032	0.001	5.316	0.000	0.002	0.004
PERSPEN_PCTAF	0.0013	0.000	4.191	0.000	0.001	0.002
PTRATIO	0.0093	0.004	2.308	0.022	0.001	0.017
LOWINC	-0.0046	0.000	-11.840	0.000	-0.005	-0.004
PERHISP	-0.0021	0.000	-5.729	0.000	-0.003	-0.001
PERSPENK	-0.0349	0.017	-2.048	0.042	-0.068	-0.001
PERMINTE_AVYRSEXP_AVSAL	-7.327e-06	4.35e-06	-1.683	0.093	-1.59e-05	1.24e-06
PTRATIO_PCTAF	-0.0002	6.13e-05	-3.313	0.001	-0.000	-8.25e-05
AVYRSEXP_AVSAL	9.579e-05	4.65e-05	2.060	0.040	4.25e-06	0.000

Omnibus:	8.042	**Durbin-Watson:**	1.879
Prob(Omnibus):	0.018	**Jarque-Bera (JB):**	11.211
Skew:	-0.187	**Prob(JB):**	0.00368
Kurtosis:	3.865	**Cond. No.**	3.86e+05

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.86e+05. This might indicate that there are
strong multicollinearity or other numerical problems.

Based on the data, I chose a model where p-value indicated the term was significant wheras 'NABOVE','NBELOW','PERMINTE','AVYRSEXP','AVSALK','PCTAF','PCTCHRT','PERSPEN_PTRATIO','PERSPEN_PTRATIO_PCTA terms were not.

Model Coeffcients after experimenting with data:

- Intercept 0.524040
- PERMINTE_AVYRSEXP 0.000486
- PERBLACK -0.004522
- PCTYRRND -0.000441
- PERASIAN 0.003245
- PERSPEN_PCTAF 0.001301
- PTRATIO 0.009275
- LOWINC -0.004595
- PERHISP -0.002104
- PERSPENK -0.034890
- PERMINTE_AVYRSEXP_AVSAL -0.000007
- PTRATIO_PCTAF -0.000203
- AVYRSEXP_AVSAL 0.000096

The resulting model an in sample r-squared of 0.808 and the PERSPENK = -0.034 and PTRATIO = 0.009. With all features PERSPENK = 0.0036 and PTRATIO = 0.0170. Removing coefficients made PTRATIO smaller and PERSPENK negative and in absolute terms bigger.

Part E - Interpreting Causal Effects on student outcomes

Since PTRATIO has a positive confidence interval and PERSPENK has a negative confidence interval we have 95% confidence that the true coefficient for pupil-teacher ratio is slightly positive. Similarly we are 95% confident the per-puil spending has a negative effect on student outcomes. We can't say this is causal because there is not enforced randomization of treatment and measured outcomes like in an A/B test. Further investigation can be done because in this analysis these coefficients are near to 0 and switch signs depending on the number of features used in the model.