**Homework 6**

**Yue Peng**

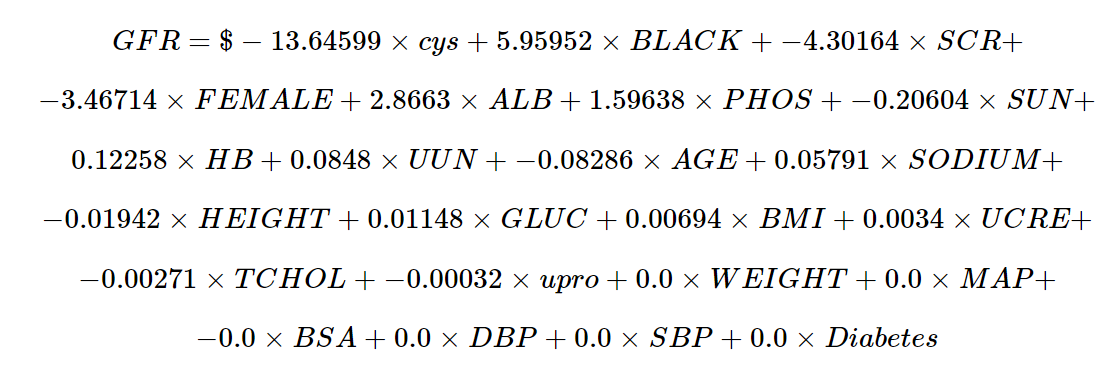
Use the data from the iod.csv file you analyzed in the previous homework to create a model for GFR that considers nonlinear transformations (polynomials, step functions, splines, generalized additive models) for continuous predictors. Try each type of nonlinear function and give statistical support (e.g, by using a hypothesis test) to your final choice. If the function is nonparametric or hard to interpret, try to represent it using a function that can be interpreted straightforwardly.

We will now perform lasso regression to predict GFR with feature selection on the data. We now perform 10-fold cross-validation to choose the best alpha, refit the model, and compute the associated test error

We use the LassoCV object that sets its λ parameter automatically from the data by internal cross-validation (i.e. it performs cross-validation on the training data it receives).

The test data MSE is 149.89

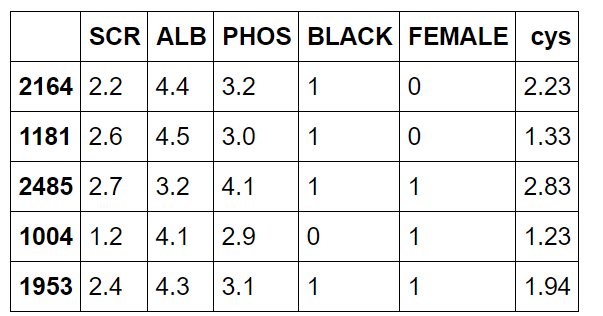
The test data R-squared is 0.72



Also, the lasso has a substantial advantage over ridge regression in that the resulting coefficient estimates are sparse. Here we see that the coefficient estimates of **WEIGHT**, **MAP**, **BSA**, **DBP**, **SBP** and **Diabetes** are exactly zero.

The other factors are predictive. This model predicts the data quite well with MSE = 149.89and R-Squared = 0.72

We selected some significant variables whose absolute value of their coefficient are greater than one.



We could know that **cys**, **ALB**, **PHOS** and **SCR** are continuous predictors.

Quadratic polynomial

We have done quadratic transformation on all continuous variables.

The test data MSE is 118.37

The test data R-squared is 0.78

Cubic polynomial

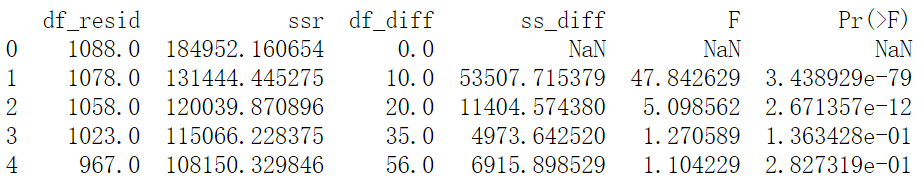
We have done cubic transformation on all continuous variables

The test data MSE is 109.13

The test data R-squared is 0.79

Deciding on a degree

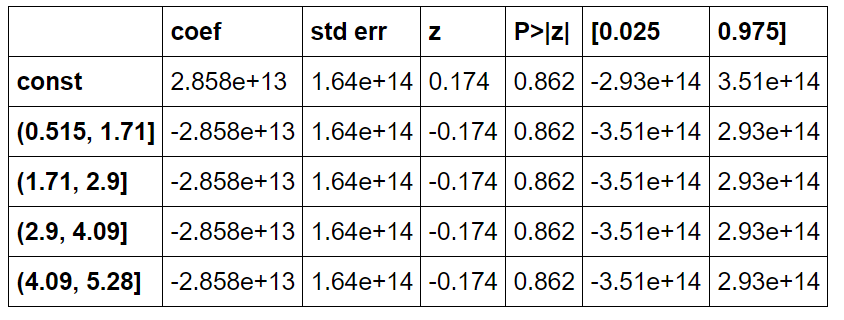
We can do this using the **anova\_lm()** function, which performs an analysis of variance (ANOVA, using an F-test) in order to test the null hypothesis that a model Model 1 is sufficient to explain the data against the alternative hypothesis that a more complex model Model 2 is required.



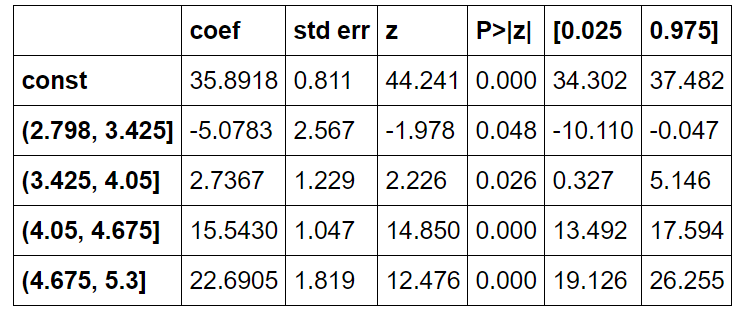
The p-value comparing the linear Model 1 to the quadratic Model 2 is essentially zero (10e−78), indicating that a linear fit is not sufficient. Similarly the p-value comparing the quadratic Model 2 to the cubic Model 3 is very low (<10e−11), so the quadratic fit is also insufficient. The p-value comparing the cubic and degree-4 polynomials, Model 3 and Model 4, is approximately 0.01 while the degree-5 polynomial Model 5 seems unnecessary because its p-value is 0.28. Hence, either a cubic or a quartic polynomial appear to provide a reasonable fit to the data, but lower- or higher-order models are not justified.

Step function

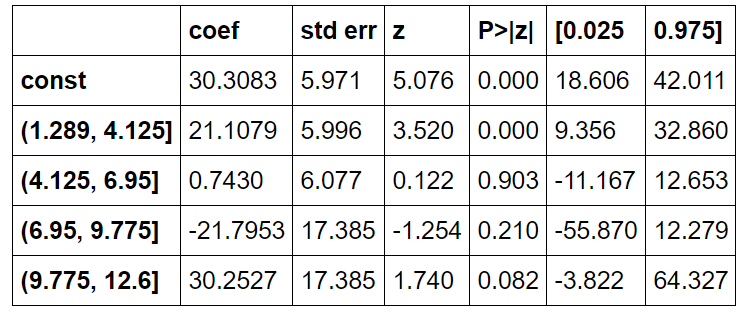
First, we deal with the **cys** predictor



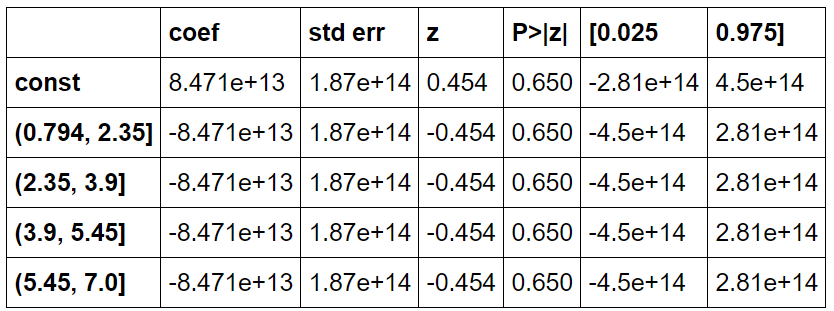
Then, we deal with the **ALB** predictor



Then, we deal with the **PHOS** predictor



Then, we deal with the **SCR** predictor



According to the tables above, we choose **ALB** variables to analyze since its p-values are all less than 0.05.

The test data MSE is 466.83

Since the MSE is quite large, it is not flexible and efficient to use step function here.

Spline

First, we used 3 knots and 3 degrees (cubic spline). The chosen variable is still **ALB according to the analysis above.**

The test data MSE is 468.96

Then we used 3 knots and 6 degrees. The chosen variable is still **ALB according to the analysis above.**

The test data MSE is 467.62

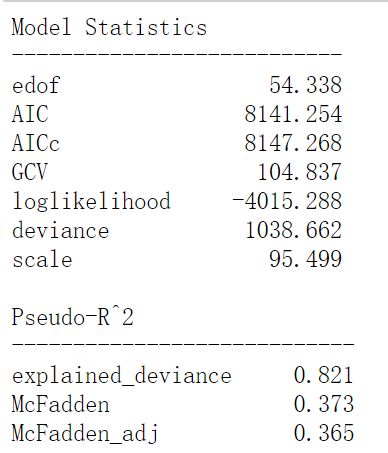
Then we used 5 knots and 3 degrees. The chosen variable is still **ALB** according to the analysis above.

The test data MSE is 466.59

The **MSE** is quite large compared to polynomial and decreases not obviously with higher degree or more knots.

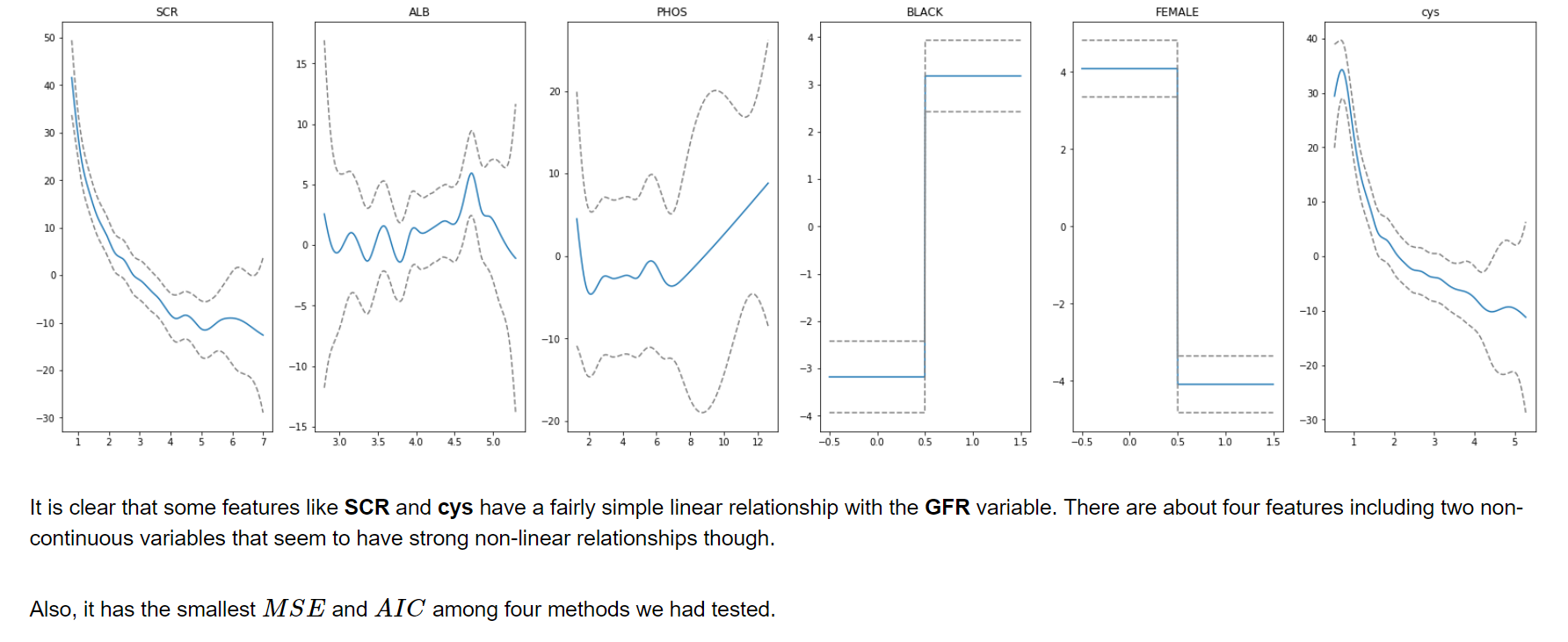
We decided to move to next methods.

GAM



We fit a GAM with 25 splines to use in each of the smooth function that is going to be fitted. The penalization term is 0.6 that is multiplied to the second derivative in the overall objective function. And no constraints.

The model is quite good with lowest AIC and MSE (92.71) and highest R-squared (0.821)



In all, the GAM method would be used in further analysis.

Appendix

%**matplotlib** inline

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

**import** **matplotlib.pyplot** **as** **plt**

**import** **statsmodels.api** **as** **sm**

*# from IPython.display import Markdown, display*

**from** **sklearn.model\_selection** **import** train\_test\_split

**from** **sklearn.linear\_model** **import** Lasso, LassoCV

**from** **sklearn.metrics** **import** mean\_squared\_error

**from** **sklearn.model\_selection** **import** KFold, cross\_val\_score

**from** **sklearn.preprocessing** **import** scale

**def** pretty\_print\_linear(coefs, names = **None**, sort = **False**):

**if** names == **None**:

names = ["X**%s**" % x **for** x **in** range(len(coefs))]

lst = zip(coefs, names)

**if** sort:

lst = sorted(lst, key = **lambda** x:-np.abs(x[0]))

**return** " + ".join("**%s** **\t**imes **%s**" % (round(coef, 5), name)

**for** coef, name **in** lst)

*# def printmd(string):*

*# display(Markdown(string))*

data = pd.read\_csv('iodatadev.csv')

vars = ["WEIGHT", "BMI", "GFR", "UCRE", "UUN", "UPHO", "SUN", "SCR", "TCHOL", "ALB", "HBA1C", "PHOS",

"TRIG", "LDL", "HDL", "HB", "MAP", "upro", "BSA", "SODIUM", "GLUC", "BLACK", "HEIGHT", "AGE",

"FEMALE", "cys", "DBP", "SBP", "CRP", "Diabetes", "hbpstatus"]

data = data[vars]

data.shape

**for** i **in** vars:

print('**%s** has **%d** missing values' % (i, sum(data[i].isna())))

dat = data[["WEIGHT", "BMI", "GFR", "UCRE", "UUN", "SUN", "SCR", "TCHOL", "ALB", "PHOS",

"HB", "MAP", "upro", "BSA", "SODIUM", "GLUC", "BLACK", "HEIGHT", "AGE",

"FEMALE", "cys", "DBP", "SBP", "Diabetes"]]

dat = dat.dropna()

X = dat[["WEIGHT", "BMI", "UCRE", "UUN", "SUN", "SCR", "TCHOL", "ALB", "PHOS",

"HB", "MAP", "upro", "BSA", "SODIUM", "GLUC", "BLACK", "HEIGHT", "AGE",

"FEMALE", "cys", "DBP", "SBP", "Diabetes"]]

y = dat["GFR"]

print("Finally we keep **%.2f** of data points with keeping **%d**/30 predicors" %(X.shape[0]/data.shape[0], X.shape[1]))

X\_train, X\_test , y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=123)

lassocv = LassoCV(alphas=**None**, cv=10, max\_iter=100000, normalize=**True**)

lassocv.fit(X\_train, y\_train)

lasso = Lasso(max\_iter=10000, normalize=**True**)

lasso.set\_params(alpha=lassocv.alpha\_)

lasso.fit(X\_train, y\_train)

print("The test data MSE is **%.2f**" % (mean\_squared\_error(y\_test, lasso.predict(X\_test))))

print("The test data R-squared is **%.2f**" % (lasso.score(X\_test, y\_test)))

lasso.fit(X, y)

pretty\_print\_linear(lasso.coef\_, list(X.columns), sort = **True**)

vars\_selected = ["SCR", "ALB", "PHOS", "BLACK", "FEMALE", "cys"]

X\_train = X\_train[vars\_selected]

X\_test = X\_test[vars\_selected]

X\_train.head()

**from** **sklearn** **import** linear\_model

**from** **sklearn.preprocessing** **import** PolynomialFeatures

poly\_reg = PolynomialFeatures(degree = 2)

X\_train\_poly = poly\_reg.fit\_transform(X\_train[["cys", "ALB", "PHOS", "SCR"]])

X\_test\_poly = poly\_reg.fit\_transform(X\_test[["cys", "ALB", "PHOS", "SCR"]])

lin\_reg\_2 = linear\_model.LinearRegression()

lin\_reg\_2.fit(X\_train\_poly, y\_train)

print("The test data MSE is **%.2f**" % (mean\_squared\_error(y\_test, lin\_reg\_2.predict(X\_test\_poly))))

print("The test data R-squared is **%.2f**" % (lin\_reg\_2.score(X\_test\_poly, y\_test)))

poly\_reg3 = PolynomialFeatures(degree = 3)

X\_train\_poly3 = poly\_reg3.fit\_transform(X\_train[["cys", "ALB", "PHOS", "SCR"]])

X\_test\_poly3 = poly\_reg3.fit\_transform(X\_test[["cys", "ALB", "PHOS", "SCR"]])

lin\_reg\_3 = linear\_model.LinearRegression()

lin\_reg\_3.fit(X\_train\_poly3, y\_train)

print("The test data MSE is **%.2f**" % (mean\_squared\_error(y\_test, lin\_reg\_3.predict(X\_test\_poly3))))

print("The test data R-squared is **%.2f**" % (lin\_reg\_3.score(X\_test\_poly3, y\_test)))

**import** **statsmodels.api** **as** **sm**

X1 = PolynomialFeatures(1).fit\_transform(X\_train[["cys", "ALB", "PHOS", "SCR"]])

X2 = PolynomialFeatures(2).fit\_transform(X\_train[["cys", "ALB", "PHOS", "SCR"]])

X3 = PolynomialFeatures(3).fit\_transform(X\_train[["cys", "ALB", "PHOS", "SCR"]])

X4 = PolynomialFeatures(4).fit\_transform(X\_train[["cys", "ALB", "PHOS", "SCR"]])

X5 = PolynomialFeatures(5).fit\_transform(X\_train[["cys", "ALB", "PHOS", "SCR"]])

fit\_1 = fit = sm.GLS(y\_train, X1).fit()

fit\_2 = fit = sm.GLS(y\_train, X2).fit()

fit\_3 = fit = sm.GLS(y\_train, X3).fit()

fit\_4 = fit = sm.GLS(y\_train, X4).fit()

fit\_5 = fit = sm.GLS(y\_train, X5).fit()

print(sm.stats.anova\_lm(fit\_1, fit\_2, fit\_3, fit\_4, fit\_5, typ=1))

df\_cut, bins = pd.cut(X\_train.cys, 4, retbins = **True**, right = **True**)

df\_cut.value\_counts(sort = **False**)

df\_steps = pd.concat([X\_train.cys, df\_cut, y\_train], keys = ['cys','cys\_cuts','GFR'], axis = 1)

*# Create dummy variables for the cys groups*

df\_steps\_dummies = pd.get\_dummies(df\_steps['cys\_cuts'])

*# Statsmodels requires explicit adding of a constant (intercept)*

df\_steps\_dummies = sm.add\_constant(df\_steps\_dummies)

fit1 = sm.GLM(df\_steps.GFR, df\_steps\_dummies).fit()

fit1.summary().tables[1]

df\_cut, bins = pd.cut(X\_train.ALB, 4, retbins = **True**, right = **True**)

df\_cut.value\_counts(sort = **False**)

df\_steps = pd.concat([X\_train.ALB, df\_cut, y\_train], keys = ['ALB','ALB\_cuts','GFR'], axis = 1)

*# Create dummy variables for the cys groups*

df\_steps\_dummies = pd.get\_dummies(df\_steps['ALB\_cuts'])

*# Statsmodels requires explicit adding of a constant (intercept)*

df\_steps\_dummies = sm.add\_constant(df\_steps\_dummies)

fit2 = sm.GLM(df\_steps.GFR, df\_steps\_dummies).fit()

fit2.summary().tables[1]

df\_cut, bins = pd.cut(X\_train.PHOS, 4, retbins = **True**, right = **True**)

df\_cut.value\_counts(sort = **False**)

df\_steps = pd.concat([X\_train.PHOS, df\_cut, y\_train], keys = ['PHOS','PHOS\_cuts','GFR'], axis = 1)

*# Create dummy variables for the cys groups*

df\_steps\_dummies = pd.get\_dummies(df\_steps['PHOS\_cuts'])

*# Statsmodels requires explicit adding of a constant (intercept)*

df\_steps\_dummies = sm.add\_constant(df\_steps\_dummies)

fit3 = sm.GLM(df\_steps.GFR, df\_steps\_dummies).fit()

fit3.summary().tables[1]

df\_cut, bins = pd.cut(X\_train.SCR, 4, retbins = **True**, right = **True**)

df\_cut.value\_counts(sort = **False**)

df\_steps = pd.concat([X\_train.SCR, df\_cut, y\_train], keys = ['SCR','SCR\_cuts','GFR'], axis = 1)

*# Create dummy variables for the cys groups*

df\_steps\_dummies = pd.get\_dummies(df\_steps['SCR\_cuts'])

*# Statsmodels requires explicit adding of a constant (intercept)*

df\_steps\_dummies = sm.add\_constant(df\_steps\_dummies)

fit4 = sm.GLM(df\_steps.GFR, df\_steps\_dummies).fit()

fit4.summary().tables[1]

df\_cut, bins = pd.cut(X\_test.ALB, 4, retbins = **True**, right = **True**)

df\_cut.value\_counts(sort = **False**)

df\_steps = pd.concat([X\_test.ALB, df\_cut, y\_test], keys = ['ALB','ALB\_cuts','GFR'], axis = 1)

*# Create dummy variables for the cys groups*

df\_steps\_dummies = pd.get\_dummies(df\_steps['ALB\_cuts'])

*# Statsmodels requires explicit adding of a constant (intercept)*

df\_steps\_dummies = sm.add\_constant(df\_steps\_dummies)

mean\_squared\_error(y\_test, fit2.predict(df\_steps\_dummies))

**from** **patsy** **import** dmatrix

*# Specifying 3 knots*

transformed\_x\_train = dmatrix("bs(X\_train.ALB, knots=(3, 4, 5), degree=3, include\_intercept=False)",

{"X\_train.ALB": X\_train.ALB}, return\_type='dataframe')

transformed\_x\_test = dmatrix("bs(X\_test.ALB, knots=(3, 4, 5), degree=3, include\_intercept=False)",

{"X\_test.ALB": X\_test.ALB}, return\_type='dataframe')

*# Build a regular linear model from the splines*

fits1 = sm.GLM(y\_train, transformed\_x\_train).fit()

fits1.params

mean\_squared\_error(y\_test, fits1.predict(transformed\_x\_test))

*# Specifying 6 degrees of freedom*

transformed\_x\_train1 = dmatrix("bs(X\_train.ALB, df=6, include\_intercept=False)",

{"X\_train.ALB": X\_train.ALB}, return\_type='dataframe')

transformed\_x\_test2 = dmatrix("bs(X\_test.ALB, df=6, include\_intercept=False)",

{"X\_test.ALB": X\_test.ALB}, return\_type='dataframe')

fits2 = sm.GLM(y\_train, transformed\_x\_train1).fit()

fits2.params

mean\_squared\_error(y\_test, fits2.predict(transformed\_x\_test2))

**from** **pygam** **import** LinearGAM

**from** **pygam.utils** **import** generate\_X\_grid

gam = LinearGAM().fit(X\_train, y\_train)

gam.summary()

mean\_squared\_error(y\_test, gam.predict(X\_test))

XX = generate\_X\_grid(gam)

plt.rcParams['figure.figsize'] = (28, 8)

fig, axs = plt.subplots(1, 6)

titles = X\_train.columns

**for** i, ax **in** enumerate(axs):

pdep, confi = gam.partial\_dependence(XX, feature=i+1, width=.95)

ax.plot(XX[:, i], pdep)

ax.plot(XX[:, i], confi[0][:, 0], c='grey', ls='--')

ax.plot(XX[:, i], confi[0][:, 1], c='grey', ls='--')

ax.set\_title(titles[i])

plt.show()