Multiple Regression II

Today's Outline

- Multicollinearity
- Model Misspecification
- AIC and BIC
- Spread Level Plots
- Tests for heteroskedasticity
- Robust Standard Errors
- FGLS

Note: Assignment 2 is posted and will be due November 9th at 11:59pm

You must submit as a PDF.

Multicollinearity

- You've learned about *perfect multicollinearity* which occurs when variables have an exact linear relationship between them
  - Often occurs when you include the same variable measured using different units (feet and meters for example)
  - Additionally occurs with the dummy variable trap
- Multicollinearity can still cause problems when the relationship is very close but not perfect
  - Inflates variances and standard errors, making our hypothesis tests less sensitive
  - Can result in large changes in our regression coefficients

```
ceo_perf = ceo.copy()

# The original sales number is in millions, so here I make this the actual number
ceo_perf["Sales_Actual"] = ceo_perf["sales"]*1000000
```

Perfect Multicollinearity

- A note at the bottom says that the design matrix may be singular
- This tells us that there is almost certainly perfect multicollinearity in our model
- At this point we need to find the problematic variables and remove one of them

Dep. Variable:	np.log(salary)	**R-squared:**	0.079
Model:	OLS	**Adj. R-squared:**	0.075
Method:	Least Squares	**F-statistic:**	17.79
Date:	Thu, 27 Oct 2022	**Prob (F-statistic):**	3.70e-05
Time:	20:48:17	**Log-Likelihood:**	-168.63
No. Observations:	209	**AIC:**	341.3
Df Residuals:	207	**BIC:**	347.9
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	6.8467	0.045	152.138	0.000	6.758	6.935
sales	1.498e-17	3.55e-18	4.217	0.000	7.98e-18	2.2e-17
Sales_Actual	1.498e-11	3.55e-12	4.217	0.000	7.98e-12	2.2e-11

Omnibus:	57.347	**Durbin-Watson:**	1.893
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	205.912
Skew:	1.063	**Prob(JB):**	1.94e-45
Kurtosis:	7.373	**Cond. No.**	7.08e+21

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 6.69e-22. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

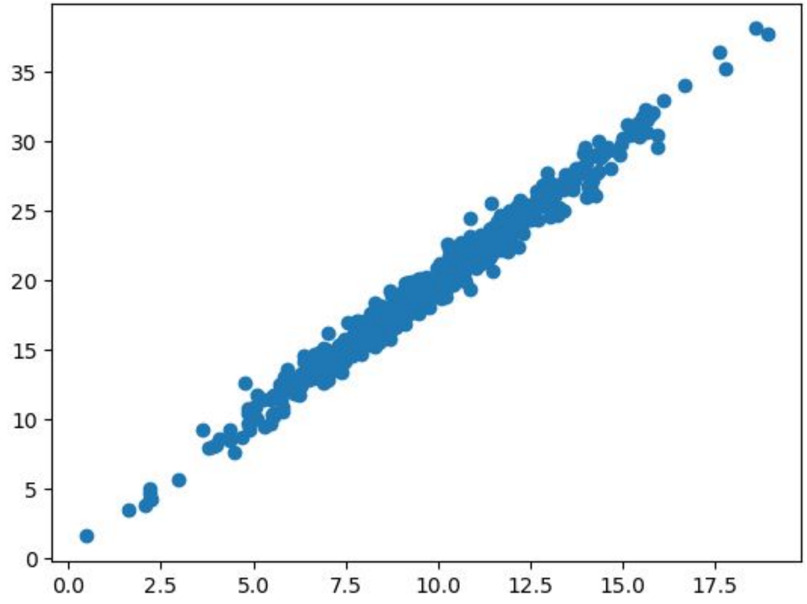
Imperfect Multicollinearity

- Let S1 and S2 be two uncorrelated random variables
- For simulation, I can use python to generate correlated random variables that will demonstrate the effect of multicollinearity

```
: synth_data= pd.DataFrame(m.T, columns = ['X', 'Z'])
```

```
: synth_data.corr()
```

	X	Z
X	1.000000	0.989449
Z	0.989449	1.000000

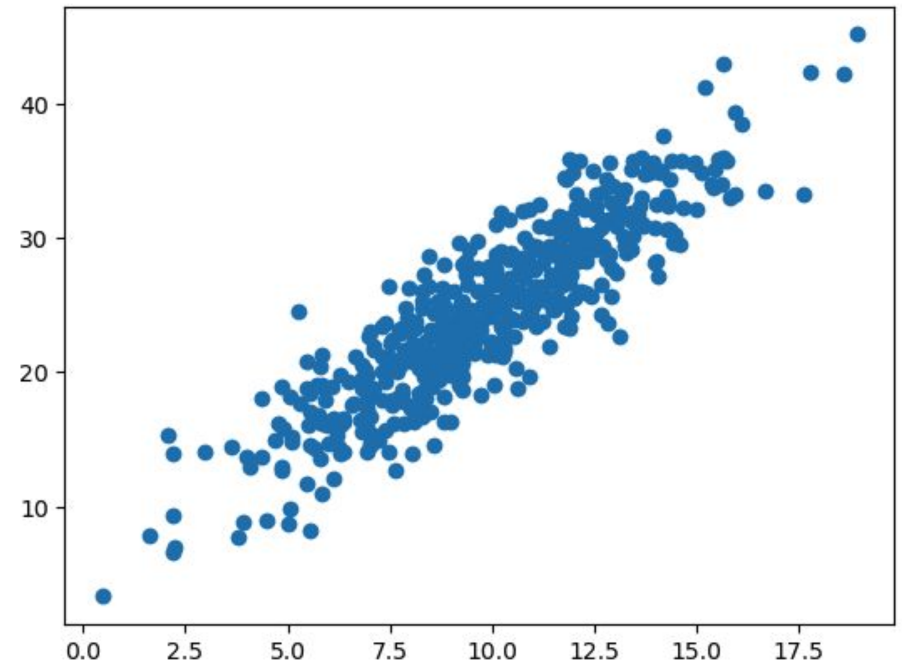


Imperfect Multicollinearity (Simulation)

- Below we create a random variable that is a function of X plus some random noise
- By construction B0 = 5, B1 = 2

```
# Case 1
synth_data['Y1'] = 5 + 2*synth_data['X'] + np.random.normal(0, 3, 500)

plt.scatter(synth_data.X, synth_data.Y1)
```

```
<matplotlib.collections.PathCollection at 0x7f94700cbeb0>
```



Imperfect Multicollinearity (Simulation Result)

- On the right we run two separate regressions
  - One just regressing Y on X
  - One regressing Y on X and the irrelevant variable Z
- Note that our estimates for the beta on X has gotten worse and the standard errors have also grown

```
: smf.ols('Y1 ~ X', data = synth_data).fit().summary()
```

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	5.3574	0.460	11.647	0.000	4.454	6.261
X	1.9324	0.044	43.839	0.000	1.846	2.019

```
]: smf.ols('Y1 ~ X + Z ', data = synth_data).fit().summary()
```

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	5.4664	0.461	11.862	0.000	4.561	6.372
X	2.6350	0.320	8.231	0.000	2.006	3.264
Z	-0.3570	0.161	-2.216	0.027	-0.674	-0.040

Variance Inflation Factor (VIF)

- We can also calculate the VIF, which is a metric for identifying imperfect multicollinearity

$$\widehat{var}(b_j) = \frac{\hat{\sigma}^2}{(n-1)s_j^2} \times \boxed{\frac{1}{1-R_j^2}}.$$

- Where $1/(1-R^2)$ *is* the VIF
- Simple to calculate by hand, just:
  - Regress a predictor on the other predictors
  - Plug the rsquared from that regression into the equation above

```
# Run the auxiliary Regressions
r_z = smf.ols('Z ~ X', data = synth_data).fit().rsquared
r_x = smf.ols('X ~ Z', data = synth_data).fit().rsquared
```

```
# calculate the VIF for each
1/(1-r_z)
```

```
53.157380925866796
```

```
1/(1-r_x)
```

```
53.157380925866796
```

VIF in Statsmodels

- The outliers_influence submodule in statsmodels.stats will also calculate VIF for us

```
statsmodels.stats.outliers_influence.variance_inflation_factor(
    exog,
    exog_idx
)                                                         [source]
```

Variance inflation factor, VIF, for one exogenous variable

The variance inflation factor is a measure for the increase of the variance of the parameter estimates if an additional variable, given by exog_idx is added to the linear regression. It is a measure for multicollinearity of the design matrix, exog.

One recommendation is that if VIF is greater than 5, then the explanatory variable given by exog_idx is highly collinear with the other explanatory variables, and the parameter estimates will have large standard errors because of this.

Parameters

exog : {`ndarray`, `DataFrame`}

design matrix with all explanatory variables, as for example used in regression

exog_idx : `int`

index of the exogenous variable in the columns of exog

VIF Code

```
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Get the design matrix (set of predictors + intercept)
synth_data['intercept'] = 1
X = synth_data[['intercept', 'X', 'Z']]

# Create place to store VIF values
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns

# calculating VIF for each feature
vif_data["VIF"] = [variance_inflation_factor(X.values, i)
                          for i in range(len(X.columns))]

print(vif_data)
```

```
     feature        VIF
0  intercept  12.246353
1          X  53.157381
2          Z  53.157381
```

Higher Order Predictors

- Note that regressions that include higher order terms can also create multicollinearity
- This is because higher order terms are naturally correlated with the original variable
- Below we have another simulated example where an output (Z) is a function of a 3rd order polynomial of X

```python
synthdata['Z2'] = X + X**2 +X**3 + np.random.normal(0,5, 40)
```

```python
np.corrcoef(synthdata["X"]**3, synthdata["X"]**2)
```

```
2]: array([[1.        , 0.4666816],
           [0.4666816, 1.        ]])
```

Higher Order Predictors

- Ideally our regression should detect that each predictor:
    - Has a coefficient = 1
    - Is statistically significant
- Unfortunately this doesn't happen
- We may erroneously conclude X is not a good predictor of Y
- Note the *very high* R-Squared term

```
results1 = smf.ols('Z2 ~ X + I(X**2) + I(X**3)', synthdata).fit()
results1.summary()
```

OLS Regression Results

Dep. Variable:	Z2	**R-squared:**	0.496
Model:	OLS	**Adj. R-squared:**	0.454
Method:	Least Squares	**F-statistic:**	11.80
Date:	Wed, 26 Oct 2022	**Prob (F-statistic):**	1.57e-05
Time:	21:15:55	**Log-Likelihood:**	-124.88
No. Observations:	40	**AIC:**	257.8
Df Residuals:	36	**BIC:**	264.5
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-0.6111	1.207	-0.506	0.616	-3.059	1.837
X	1.6622	1.682	0.988	0.330	-1.749	5.074
I(X ** 2)	1.2609	0.760	1.659	0.106	-0.281	2.803
I(X ** 3)	0.8160	0.531	1.537	0.133	-0.261	1.893

Omnibus:	7.833	**Durbin-Watson:**	2.105
Prob(Omnibus):	0.020	**Jarque-Bera (JB):**	7.484
Skew:	-0.683	**Prob(JB):**	0.0237
Kurtosis:	4.621	**Cond. No.**	7.60

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

Joint Hypothesis Testing (F-Test)

- Individual t-tests in this case will not be able to tell us whether X is a good predictor
- We can test whether, altogether, X is a good predictor of Y
- This is done in a joint hypothesis (or F) test
- If any of the predictors are significant we reject the null that for:

$$Z = \beta_0 + \beta_1 X_1 + \beta_2 X_1^2 + \beta_2 X_1^3 + e$$

$$H_0 : \beta_1 = \beta_2 = \beta_3 = 0$$

- If any are significant then we reject the null
- Note that this will not tell us *which* of the variables included are important

```
# write each of your hypotheses in a list
hypotheses = ['X = 0', "I(X ** 2) = 0", "I(X ** 3) = 0"]

# use the f-test method included in sm.ols().fit() objects
results1.f_test(hypotheses)
```

```
]: <class 'statsmodels.stats.contrast.ContrastResults'>
<F test: F=11.797252124012763, p=1.5688628957854124e-05, df_denom=36, df_num=3>
```

Multicollinearity Interactions

- Multicollinearity may also be a problem arising from including interaction variables
- Below we generate an indicator variable and interact it with the X variable generated previously
- We build another variable that is a linear function of each of these variables

```python
# generate an indicator variable
I =  np.random.choice([0,1], 40, p =  [.3, .7])
```

```python
I
```

```
3]: array([1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0,
           1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0])
```

```python
synthdata["IND"] = I

# create a dependent variable
synthdata['Z3'] = .6*I + .1*X + .5*I*X + np.random.normal(0,1, 40)
```

Multicollinearity Interactions

- One again, we are unable to detect any statistical significance
- From this we may conclude (mistakenly) that X is not related to Z3
- Below we can run an F-test on all terms including X
- This will tell us whether X is a relevant predictor

```
hypotheses = ['X = 0', 'IND:X = 0']
results3.f_test(hypotheses)
```

```
<class 'statsmodels.stats.contrast.ContrastResults'>
<F test: F=3.592673978535451, p=0.03779108809418408, df_denom=36, df_num=2>
```

OLS Regression Results

Dep. Variable:	Z3	R-squared:	0.211
Model:	OLS	Adj. R-squared:	0.145
Method:	Least Squares	F-statistic:	3.213
Date:	Wed, 26 Oct 2022	Prob (F-statistic):	0.0342
Time:	21:28:44	Log-Likelihood:	-54.743
No. Observations:	40	AIC:	117.5
Df Residuals:	36	BIC:	124.2
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-0.1621	0.347	-0.467	0.643	-0.866	0.542
IND	0.7425	0.404	1.838	0.074	-0.077	1.562
X	0.1163	0.281	0.414	0.682	-0.454	0.686
IND:X	0.4777	0.360	1.329	0.192	-0.252	1.207

Omnibus:	2.733	Durbin-Watson:	1.918
Prob(Omnibus):	0.255	Jarque-Bera (JB):	2.172
Skew:	0.571	Prob(JB):	0.338
Kurtosis:	2.969	Cond. No.	4.65

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

Exercise

- Using the wooldridge module, import the “mlb1” dataset (link to variable descriptions)
- Run the following regression:

$$\log\left(salary\right) = \beta_0 + \beta_1 years + \beta_2 gamesyr + \beta_3 bavg + \beta_4 hrunsyr + \beta_5 rbisyr + e$$

- Based on the individuals t-tests, are baseball statistics related to a player’s salary?
- Does VIF indicate any problems with multicollinearity in the data?
- Test whether performance statistics, as a whole, matter for determining a baseball player’s salary (use a joint hypothesis test for the the last three variables in the regression)

Component Plus Residuals Plots

- We will often be interested in only modelling the relationship between one predictor and y
- A component-plus-residuals plot attempts to remove the effects of other predictors
- The component plus residuals is simply given by:

$$e_{partial,ij} = e_i + b_j x_{ij}$$

- Below we replicate an example using the prestige dataset from econ 430 in python

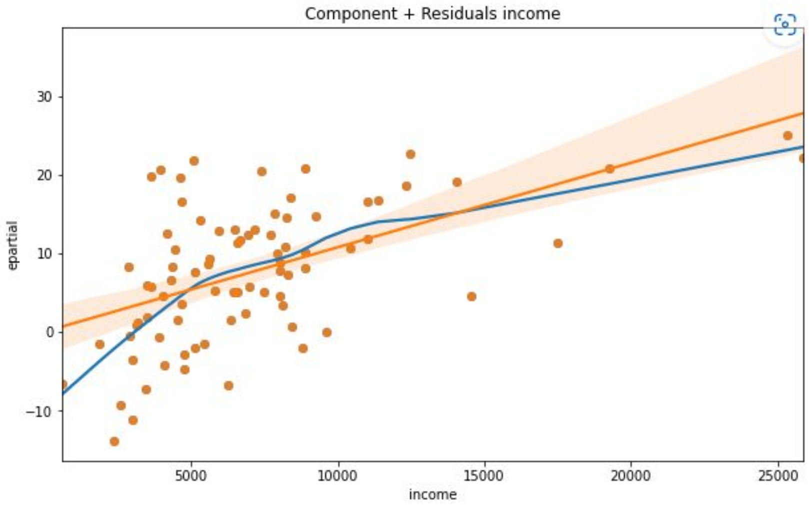
```
prestige = pd.read_csv("Prestige_miss.csv")
fit2 = smf.ols('prestige ~ income +education + women', prestige).fit()
```

Component Plus Residuals Plots

- Producing a CCPR plot simply entails
  - Pulling the residuals from a regression
  - Multiplying the variable of interest by its coefficient
  - Adding the two vectors together
- To reproduce plots similar to the ones shown in R we can use the regplot() function to:
  - Show a line of best fit
  - Add a lowess smoother to model any nonlinearities
- These plots can help tell us about any remaining nonlinear relationships

```
def ccpr_plot(model, data, variable):
    df_copy = data.copy()

    df_copy["epartial"] = model.resid + model.params[variable]*data[variable]

    plt.figure(figsize = (10, 6))

    sns.regplot(x = variable, y = "epartial", data =df_copy, lowess = True)
    sns.regplot(x = variable, y = "epartial", data =df_copy)

    plt.title("Component + Residuals "+variable)
```

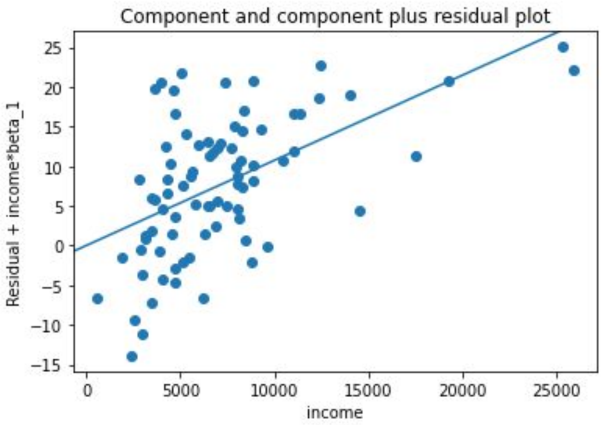
```
ccpr_plot(fit2, prestige, "income")
```



Component Plus Residuals Plot (Statsmodels)

- The out-of-the-box method given by statsmodels will only give us a line of best fit
- Takes a fitted model and a variable name as arguments

```
import statsmodels.api as sm
sm.graphics.plot_ccpr(fit2, 'income')

plt.show()
```

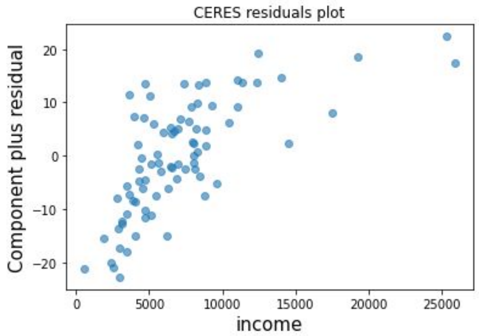


Conditional Expectations and Residuals

- These plots are slightly more more complex to produce and can model stronger linear effects that may go undetected by CCPR plots
- Effects of other predictors are removed by conditioning on the other predictors, but the interpretation is similar
- We can exploit the statsmodels function to get CERES Plots

```
## Ceres
from statsmodels.graphics.regressionplots import plot_ceres_residuals
l = plot_ceres_residuals(fit2, 'income')
plt.gca()
```

```
]: <AxesSubplot:title={'center':'CERES residuals plot'}, xlabel='income', ylabel='Component plus residual'>
```



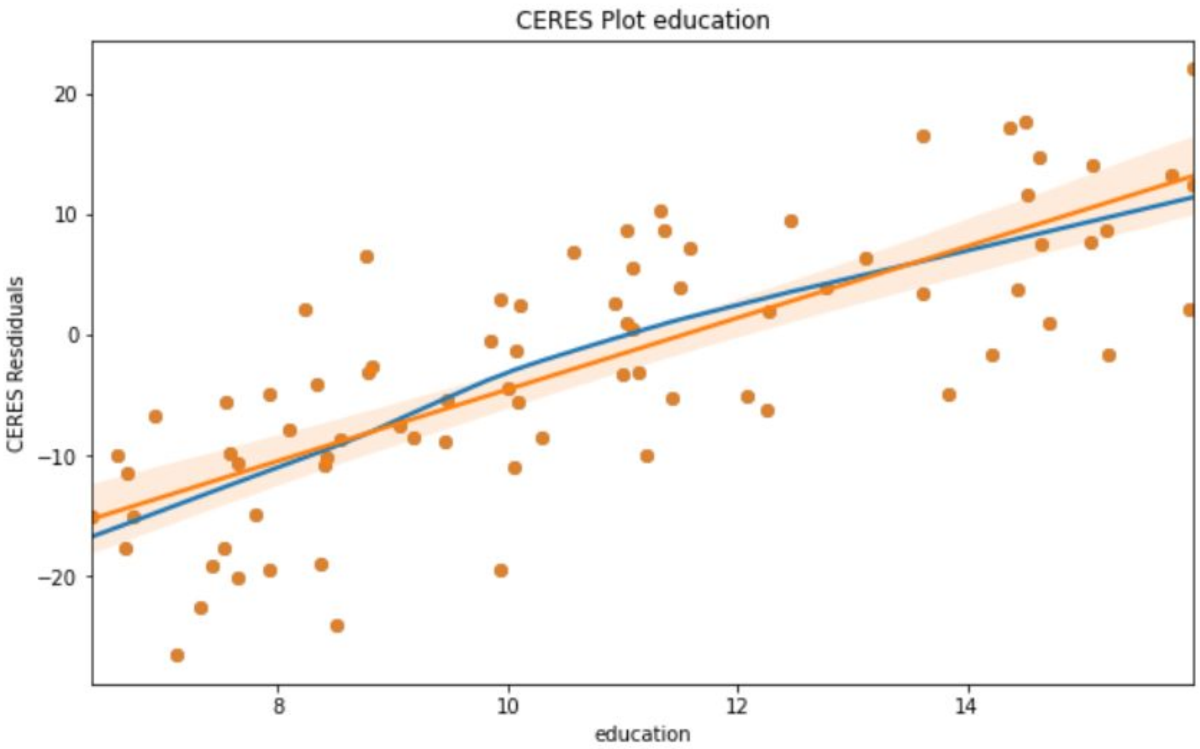
Conditional Expectations and Residuals Function

- To produce a plot similar to the one from statsmodels we can pull the data points from their plot
- Then we can add a regression line and lowess smoother using regplot
- Without doing this we will only get a scatterplot from statsmodels

```
def ceres_plot(model, data, variable):

    # produce plot
    plot_ceres_residuals(fit2, variable)
    ax = plt.gca()

    # don't show plot in notebook
    plt.close()

    # Pull datapoints from scatterplot from the statsmodels plot
    line = ax.lines[0]
    X = line.get_xdata()
    Y = line.get_ydata()

    # Store the results into format that works with seaborn
    df = pd.DataFrame(np.array([X,Y]).T, columns = [variable, "CERES Resdiduals"])
    plt.figure(figsize = (10, 6))

    # plot the results in a way similar to R
    sns.regplot(x = variable, y = "CERES Resdiduals", data =df, lowess = True)
    sns.regplot(x = variable, y = "CERES Resdiduals", data =df)

    plt.title("CERES Plot "+variable)
```

Conditional Expectations and Residuals Plot

```
fit3 = smf.ols('prestige~income+education+type', prestige).fit()
ceres_plot(fit3, prestige, 'education')
```



Model Misspecification

- A model's functional form will be misspecified if we fail to include relevant interactions and higher order terms in the regression
- This will lead to biased estimators of our coefficients
- For example let's think back to Z2 which was created with a third order term of X

```
results1 = smf.ols('Z2 ~ X + I(X**2) + I(X**3)', synthdata).fit()
results1.summary()
```

OLS Regression Results

Dep. Variable:	Z2	R-squared:	0.496
Model:	OLS	Adj. R-squared:	0.454
Method:	Least Squares	F-statistic:	11.80
Date:	Wed, 26 Oct 2022	Prob (F-statistic):	1.57e-05
Time:	21:15:55	Log-Likelihood:	-124.88
No. Observations:	40	AIC:	257.8
Df Residuals:	36	BIC:	264.5
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-0.6111	1.207	-0.506	0.616	-3.059	1.837
X	1.6622	1.682	0.988	0.330	-1.749	5.074
I(X ** 2)	1.2609	0.760	1.659	0.106	-0.281	2.803
I(X ** 3)	0.8160	0.531	1.537	0.133	-0.261	1.893

```
# this model is misspecified and our estiamte of the coefficient
# on X is biased
results2 = smf.ols('Z2 ~ X', synthdata).fit()
results2.summary()
```

OLS Regression Results

Dep. Variable:	Z2	R-squared:	0.361
Model:	OLS	Adj. R-squared:	0.344
Method:	Least Squares	F-statistic:	21.44
Date:	Wed, 26 Oct 2022	Prob (F-statistic):	4.18e-05
Time:	21:16:05	Log-Likelihood:	-129.62
No. Observations:	40	AIC:	263.2
Df Residuals:	38	BIC:	266.6
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	0.7717	1.016	0.760	0.452	-1.285	2.828
X	4.4788	0.967	4.631	0.000	2.521	6.437

Model Misspecification (RESET)

- The RESET test can help tell us whether we are missing any important nonlinearities
- The algorithm is simple:
  - Generate a regression model
  - Pull out the fitted values
  - Decide what order of polynomial you would like to test (typically up to 3)
  - Generate vectors of fitted values taken to the power of the polynomial up to the desired order
  - Add the vectors in the last step as predictors to the original regression
  - Run an f-test to decide whether the coefficients on the polynomials are significant

```python
# Create our suspect model
results1 = smf.ols('Z2 ~ X', synthdata).fit()
```

```python
# Take the fitted values up to the desired power
synthdata["fitted2"] = results1.fittedvalues**2
synthdata["fitted3"] = results1.fittedvalues**3

# Fit regression on polynomial
ramseyreg = smf.ols('Z2 ~ X + fitted2 + fitted3', synthdata).fit()

# run ftest on polynomial values
hypotheses = ['fitted2 = 0', "fitted3 = 0"]

# we reject the null that the functional form is adequate
ramseyreg.f_test(hypotheses)
```

```
<class 'statsmodels.stats.contrast.ContrastResults'>
<F test: F=4.818604895089585, p=0.013986256771061925, df_denom=36, df_num=2>
```

```python
# statsmodels method
reset_out = smo.reset_ramsey(res = results1, degree = 3)
reset_out
```

```
<class 'statsmodels.stats.contrast.ContrastResults'>
<F test: F=4.818604895089885, p=0.01398625677105865, df_denom=36, df_num=2>
```

Exercise

- Import the “hprice1” dataset from the wooldridge module
- Fit the following model:

$$price = \beta_0 + \beta_1 lotsize + \beta_2 sqrft + \beta_3 bdrms + e$$

- Decide whether the model is misspecified
- Plot the component plus residuals plots
- Based on your results decide whether and/or what higher order terms to add to the model (note that this is non-obvious)

Model Selection Metrics (So Far)

- You have already learned about two model selection metrics:
  - R-Squared: A measure of the proportion of the variance in the dependent variable explained by the regression
    - Monotonically increasing measure of fit
  - Adjusted R-Squared: A measure of the proportion of the variance in the dependent variable explained by the regression that is penalized as more predictors are added
    - Doesn’t really have any theoretical basis (is biased)

Dep. Variable:	cumgpa	R-squared:	0.241
Model:	OLS	Adj. R-squared:	0.234
Method:	Least Squares	F-statistic:	38.31
Date:	Fri, 04 Nov 2022	Prob (F-statistic):	1.73e-40
Time:	10:51:52	Log-Likelihood:	-929.74
No. Observations:	732	AIC:	1873.
Df Residuals:	725	BIC:	1906.
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	0.8791	0.298	2.952	0.003	0.294	1.464
sat	0.0009	0.000	3.743	0.000	0.000	0.001
hsperc	-0.0056	0.002	-3.463	0.001	-0.009	-0.002
tothrs	0.0121	0.001	12.941	0.000	0.010	0.014
female	0.1667	0.077	2.164	0.031	0.015	0.318
black	-0.0261	0.192	-0.136	0.892	-0.403	0.351
white	0.0143	0.184	0.078	0.938	-0.347	0.375

AIC and BIC (Manual)

- Two more metrics also exist:
  - Akaike Information Criterion (AIC): Estimates the quality of models being considered for the data, penalizes the addition of new variables

$$AIC = \ln\left(\frac{SSE}{N}\right) + \frac{2K}{N}$$

  - Bayesian Information Criterion (BIC): Similar to AIC, applies a progressively larger penalty to the addition of new variables when compared to AIC

$$BIC = \ln\left(\frac{SSE}{N}\right) + \frac{K\ln(N)}{N}$$

```python
# manual AIC using the formula
def AIC(model, y):

    # get SSE
    SSE = ((model.resid)**2).sum()

    # Get K and N
    k = len(results.params)
    N = len(results.fittedvalues)

    # Calculate and return AIC
    return np.log(SSE/N) + ((2*k)/N)

AIC(results, data.cumgpa)
```

```
-0.2784906693074793
```

```python
def BIC(model, y):

    # get SSE
    SSE = ((model.resid)**2).sum()

    # Get K and N
    k = len(results.params)
    N = len(results.fittedvalues)

    # Calculate and return AIC
    return np.log(SSE/N) + ((k*np.log(N))/N)

BIC(results, data.cumgpa)
```

```
-0.2345419485455542
```

AIC and BIC Statsmodels

- Statsmodels uses a slightly different formula to calculator AIC and BIC
- This is why our implementation and statsmodels are different
- The *ordering* of the models should be the same when ranked according to AIC and BIC

Dep. Variable:	cumgpa	**R-squared:**	0.241
Model:	OLS	**Adj. R-squared:**	0.234
Method:	Least Squares	**F-statistic:**	38.31
Date:	Fri, 04 Nov 2022	**Prob (F-statistic):**	1.73e-40
Time:	10:51:52	**Log-Likelihood:**	-929.74
No. Observations:	732	**AIC:**	1873.
Df Residuals:	725	**BIC:**	1906.
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	0.8791	0.298	2.952	0.003	0.294	1.464
sat	0.0009	0.000	3.743	0.000	0.000	0.001
hsperc	-0.0056	0.002	-3.463	0.001	-0.009	-0.002
tothrs	0.0121	0.001	12.941	0.000	0.010	0.014
female	0.1667	0.077	2.164	0.031	0.015	0.318
black	-0.0261	0.192	-0.136	0.892	-0.403	0.351
white	0.0143	0.184	0.078	0.938	-0.347	0.375

```
results.aic
```

]: 1873.470842678566

```
results.bic
```

1905.6413062762952

Heteroscedasticity

- The homoscedasticity assumption for multiple linear regression requires that the variance of our error terms is unrelated to the regressors:

$$\mathrm{Var}(u|x_1, \ldots, x_k) = \sigma^2.$$

- If homoscedasticity is violated then the standard errors of our regression and hypothesis tests are no longer valid.

Heteroscedasticity and Spread-Level Plots

- We can check whether the variance of our errors is related to our regressors using a spread-level plot
- This will plot our fitted values against the absolute values of our studentized residuals
- Any detectable pattern in our residuals (increasing, decreasing, or otherwise) tells us that the variance of our errors is somehow related to the function of our predictors
- Can anyone explain why we would use the absolute value of our residuals?

Spread-Level Plots in Python

- There is no simple method for generating a spread-level plot like those from R in python
- Below we have written some code that will *closely* replicate the plots from ECON 430 (with some differences due to software implementations of procedures like rlm and lowess)

```python
def spread_level(model, data):
    df_copy = data.copy()

    # Get the studentized residuals
    df_copy["Absolute_Studentized_Residuals"] = (np.abs(model.get_influence().resid_studentized))
    df_copy["Fitted_Values"] = (model.fittedvalues)

    # run regression to get slope of fitted vs resid, rlm is a robust linear model used by R
    slreg = smf.rlm("np.log(Absolute_Studentized_Residuals) ~ np.log(Fitted_Values)", df_copy).fit()
    slope = slreg.params[1]

    # plot values
    fig, ax = plt.subplots(figsize = (10, 6))
    ax.set_title("Fitted Values vs Studentized Residuals")
    sns.regplot(x = "Fitted_Values", y = "Absolute_Studentized_Residuals", data = df_copy, lowess = True, ax = ax)
    ax.plot(df_copy.Fitted_Values.values, np.exp(slreg.fittedvalues).values)

    # Set to the logarithmic scale
    ax.set_yscale('log')
    ax.set_xscale('log')

    # convert froms scientific notation to scalar notation
    ax.yaxis.set_major_formatter(ScalarFormatter())
    ax.xaxis.set_major_formatter(ScalarFormatter())

    # Resolve overlapping label bug
    ax.minorticks_off()

    # Set tick labels automatically
    ax.set_xticks(np.linspace(df_copy["Fitted_Values"].min(),df_copy["Fitted_Values"].max(), 6))
    ax.set_yticks(np.linspace(df_copy["Absolute_Studentized_Residuals"].min(),
                              df_copy["Absolute_Studentized_Residuals"].max(), 6))

    ax.grid()

    # return a suggested power transform of your y-variable that may correct heteroscedastcity
    # The transform is just one minus the slope of the reegression line of your fitted values vs residuals
    print("Suggested Power Transformation:", 1-slope)
```

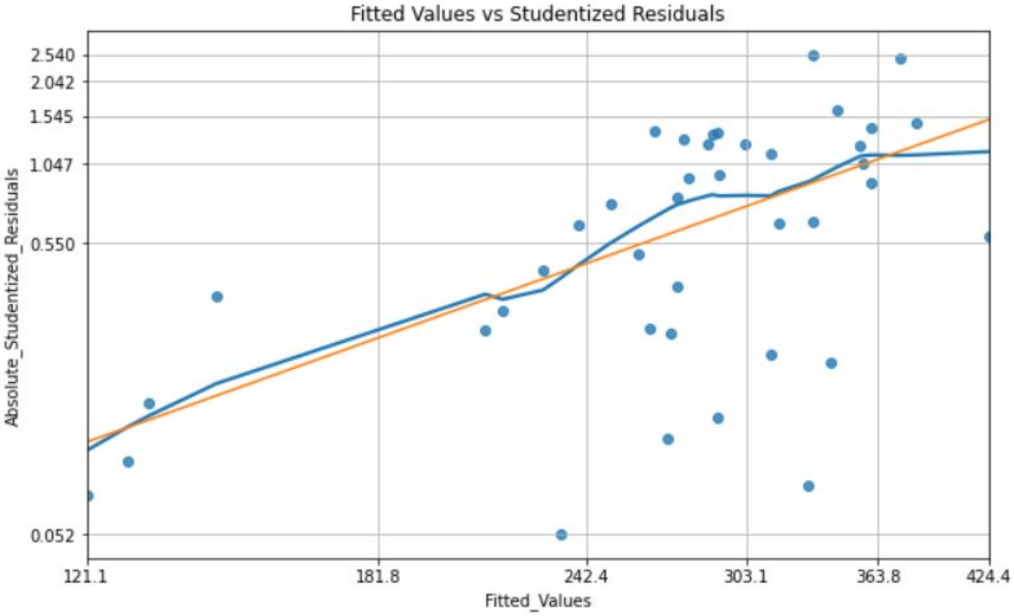
Spread-Level Plot Python Example

- What does this plot tell you about heteroscedasticity in the model?

$$Expenditures\, on\, Food \;=\; \beta_0 + \beta_1 income + \; u$$

```
results2 = smf.ols('food_exp~income', foodata).fit()
spread_level(results2, foodata)
```

```
Suggested Power Transformation: -1.0786406911463162
```

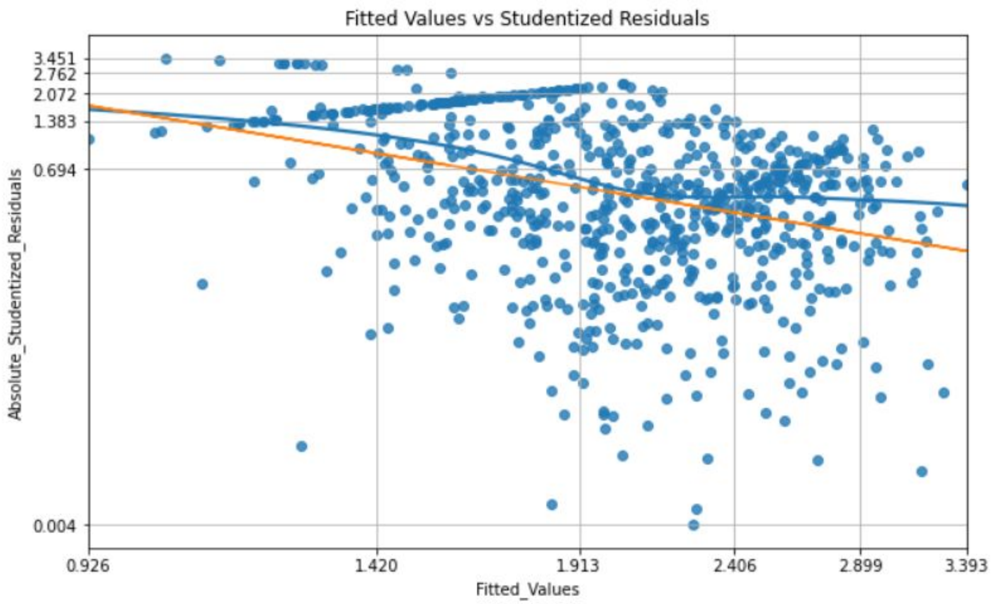


Spread-Level Plot Python Example

- What does this plot tell you about heteroscedasticity in the model?

```
# Fit the model
model = smf.ols('cumgpa ~ sat +hsperc +tothrs +female +black + white', data)
results = model.fit()
spread_level(results, data)
```

```
Suggested Power Transformation: 2.6126484819030966
```



Breusch-Pagan (BP) Test

- Fortunately we do not have to rely solely on visual examination
- The BP test allows us to test whether the residuals of our regression can be predicted as a linear combination of our predictors
- Simply regress the squared residuals from your model on the predictors and run an F or LM test
- The null hypothesis is that all of the betas from the secondary regression are zero (we can’t predict the residuals)

```python
# pull out squared residuals
data["res2"] = results.resid**2

# try to predict the squared residuals using a linear combination of our variables
aux_reg = smf.ols('res2 ~ sat +hsperc +tothrs +female +black + white', data).fit()

# Get the regression f-statistic (f-test version)
f = aux_reg.fvalue
fp = aux_reg.f_pvalue

print("The F-Statistic for the Auxiliary Regression is: "+ str(f) +" and the P-Value is: "+ str(fp))
```

```
The F-Statistic for the Auxiliary Regression is: 49.18699087724235 and the P-Value is: 9.680220020442915e-51
```

Breusch-Pagan (BP) Test Statsmodels

- The sm.stats.diagnostic submodule contains all of the tests for heteroscedasticity we will use today

```
y, X = pt.dmatrices('cumgpa ~ sat +hsperc +tothrs +female +black + white', data,
                    return_type = 'dataframe')

# Takes in the residuals and our design matrix as arguments
# Order is Lm Test statistic, LM P-value, F-stat, F-Pvalue
sm.stats.diagnostic.het_breuschpagan(results.resid, X)
```

```
(211.76807825368095,
 5.915341448453325e-43,
 49.18699087724235,
 9.680220020442915e-51)
```

```
# LM test statsitic is just n*R2 from the aux regression
LM = len(data)*aux_reg.rsquared

k = results.df_model
```

```
# sf is just 1- cdf (called the survival function)
stats.chi2(k).sf(LM)
```

```
5.915341448453325e-43
```

The White Test for Heteroscedasticity

- The White Test for heteroscedasticity is nearly identical to the BP test
- The white test adds second-order interaction and main effect terms to the auxiliary regression
- This makes it a more robust test for *large sample sizes*, but can also eat up many degrees of freedom
  - The auxiliary regression for this example estimates *28* parameters

```
# Order is Lm Test statistic, LM P-value, F-stat, F-Pvalue
sm.stats.diagnostic.het_white(results.resid, X)
```

```
5]: (373.5469356646l617,
 4.825921983841415e-65,
 32.07881392043317,
 8.692792444556739e-94)
```

Goldfeld-Quandt (GQ) Test

- The Goldfeld-Quandt test is used to compare the variances of different groups within our sample
- The hypotheses for the two-sided GQ test are:

$$H_0 : \hat{\sigma}_1^2 = \hat{\sigma}_0^2 \qquad H_a : \hat{\sigma}_1^2 \neq \hat{\sigma}_0^2$$

- Where the F-statistic is computed as:

$$F = \frac{\hat{\sigma}_1^2}{\hat{\sigma}_0^2}$$

Goldfeld-Quandt (GQ) Test - Manual

- Below we consider the example of comparing the variances of the black and non-black population in our sample
- Here we are only testing the right-hand hypothesis that the variance of sample 1 is greater than the variance of sample 2

```
# manual implementation
data1 = data[data.black == 1]
data0 = data[data.black == 0]

# run regs on different groups
reg1 = smf.ols('cumgpa ~ sat +hsperc +tothrs +female +black + white', data1).fit()
reg0 = smf.ols('cumgpa ~ sat +hsperc +tothrs +female +black + white', data0).fit()

# pull out the residuals of each regression
df1 = reg1.df_resid
df0 = reg0.df_resid

# Get the variance of each regression
sig1squared = reg1.scale
sig0squared = reg0.scale

fstat = sig1squared/sig0squared

# calculate critical calue for right side test
stats.f.ppf(.95, df1, df0)
```

```
1.229602398528648
```

```
fstat
```

```
1.0065563350797615
```

Goldfeld-Quandt (GQ) Test - Statsmodels

- Statsmodels has a simple implementation of this test as well
- Note that we have to provide two indices to run the test on our groups:
  - The index of the column containing the group on which we are making the split
  - The index at which the split is being made within the group
- We also have to provide the design matrix and the vector containing our dependent variable

```python
# I need to provide a split point to the software
# Sprt values in ascending order and reset the index to number from 1 to len(data)
sortedv = data.sort_values(by = "black").copy().reset_index()

# This returns the first index that contains a one
splt = sortedv.black.argmax()

# run regression
gq_reg = smf.ols('cumgpa ~ sat +hsperc +tothrs +female +black + white', sortedv).fit()
```

```python
# get the data for dependent and independent variables
# these are numpy arrays instead of dataframes
y = gq_reg.model.endog
X = gq_reg.model.exog

# Order is f-stat, pvalue, hypothesis
sm.stats.diagnostic.het_goldfeldquandt(y, X, idx = 5, alternative = 'increasing', split= splt)
```

```
(1.0065563350797613, 0.4901417839642259, 'increasing')
```

```python
# get the data for dependent and independent
y = gq_reg.model.endog
X = gq_reg.model.exog

# Order is f-stat, pvalue, hypothesis
sm.stats.diagnostic.het_goldfeldquandt(y, X, idx = 5, alternative = 'two-sided', split= splt)
```

```
(1.0065563350797613, 0.9396730744525191, 'two-sided')
```

Correcting Heteroscedasticity

- There are several methods for correcting for heteroskedasticity within our models
- The first (and most simple way) is by using robust standard errors
- These new standard errors will give us a valid basis for running all of our hypothesis tests
- Statsmodels allows us to simply include a flag within the fit() method that specifies the type of standard errors we want to use
- Your textbook states that the results will be similar for each type used

- `reg.fit(cov_type='nonrobust')` or `reg.fit()` for the default homoscedasticity-based standard errors.
- `reg.fit(cov_type='HC0')` for the classical version of White's robust variance-covariance matrix presented by Wooldridge (2019, Equation 8.4 in Section 8.2).
- `reg.fit(cov_type='HC1')` for a version of White's robust variance-covariance matrix corrected by degrees of freedom.
- `reg.fit(cov_type='HC2')` for a version with a small sample correction. This is the default behavior of Stata.
- `reg.fit(cov_type='HC3')` for the refined version of White's robust variance-covariance matrix.

Python - Robust Standard Errors

```
robust_reg = smf.ols('cumgpa ~ sat +hsperc +tothrs +female +black + white', data).fit(cov_type = 'HC0')
robust_reg.summary()
```

OLS Regression Results

Dep. Variable:	cumgpa	**R-squared:**	0.241
Model:	OLS	**Adj. R-squared:**	0.234
Method:	Least Squares	**F-statistic:**	30.67
Date:	Thu, 03 Nov 2022	**Prob (F-statistic):**	6.76e-33
Time:	13:15:41	**Log-Likelihood:**	-929.74
No. Observations:	732	**AIC:**	1873.
Df Residuals:	725	**BIC:**	1906.
Df Model:	6		
Covariance Type:	HC0		

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	0.8791	0.302	2.915	0.004	0.288	1.470
sat	0.0009	0.000	3.660	0.000	0.000	0.001
hsperc	-0.0056	0.002	-3.391	0.001	-0.009	-0.002
tothrs	0.0121	0.001	10.713	0.000	0.010	0.014
female	0.1667	0.079	2.123	0.034	0.013	0.321
black	-0.0261	0.181	-0.144	0.885	-0.381	0.329
white	0.0143	0.167	0.085	0.932	-0.314	0.342

```
results.summary()
```

OLS Regression Results

Dep. Variable:	cumgpa	**R-squared:**	0.241
Model:	OLS	**Adj. R-squared:**	0.234
Method:	Least Squares	**F-statistic:**	38.31
Date:	Thu, 03 Nov 2022	**Prob (F-statistic):**	1.73e-40
Time:	13:15:42	**Log-Likelihood:**	-929.74
No. Observations:	732	**AIC:**	1873.
Df Residuals:	725	**BIC:**	1906.
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	0.8791	0.298	2.952	0.003	0.294	1.464
sat	0.0009	0.000	3.743	0.000	0.000	0.001
hsperc	-0.0056	0.002	-3.463	0.001	-0.009	-0.002
tothrs	0.0121	0.001	12.941	0.000	0.010	0.014
female	0.1667	0.077	2.164	0.031	0.015	0.318
black	-0.0261	0.192	-0.136	0.892	-0.403	0.351
white	0.0143	0.184	0.078	0.938	-0.347	0.375

Weighted and Generalized Least Squares

- Robust standard errors acknowledge that the standard errors around your coefficients may be incorrect and adjusts them accordingly
- Given a few (admittedly important) assumptions, we may be able to do better and adjust the coefficients *directly*
- This requires us to assume a functional form for the variance of our errors
- Each side of the regression equation is then divided by the function in order to cancel out the heteroscedasticity in our model
- For example if we assume that our errors for a simple linear model:

$$y_i = \beta_0 + \beta_1 x_i + e_i$$

- Scale directly with x such that:

$$var(e_i) = \sigma^2 x_i$$

- Then we can divide both sides the initial model by sqrt(x_i) to get our final BLUE model

WLS Python Example

- Fortunately WLS is simple to implement in python
- We simply feed in the vector of weights as an argument and python will automatically take the square root and make the estimates

```python
# maybe I believe the heteroscedastic relationship is due to sat
w = 1/data.sat

# run a weighted regression and provide weights
# note we can use WLS and robust standrad errors
wls_known = smf.wls('cumgpa ~ sat +hsperc +tothrs +female +black + white', weights = w, data = data).fit()
wls_known.summary()
```

WLS Regression Results

Dep. Variable:	cumgpa	R-squared:	0.213
Model:	WLS	Adj. R-squared:	0.206
Method:	Least Squares	F-statistic:	32.66
Date:	Thu, 03 Nov 2022	Prob (F-statistic):	6.67e-35
Time:	13:48:42	Log-Likelihood:	-928.52
No. Observations:	732	AIC:	1871.
Df Residuals:	725	BIC:	1903.
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	1.0094	0.290	3.483	0.001	0.441	1.578
sat	0.0008	0.000	3.316	0.001	0.000	0.001
hsperc	-0.0057	0.002	-3.537	0.000	-0.009	-0.003
tothrs	0.0108	0.001	11.870	0.000	0.009	0.013
female	0.1358	0.077	1.762	0.078	-0.015	0.287
black	0.0075	0.181	0.041	0.967	-0.348	0.363
white	0.0495	0.174	0.284	0.776	-0.293	0.392

Feasible Generalized Least Squares

- Since we usually don't know the functional form of our variance, Feasible Generalized Least Squares (FGLS) is a procedure used to estimate an unknown variance function
- We assume a flexible general functional form for our variance of:

$$var(e \mid x) = \sigma^2 \exp\left(\delta_0 + \delta_1 x_1 + \ldots + \delta_k x_k\right)$$

- Then use the following steps:
  - Estimate the initial model and extract the squared residuals
  - Regress the log of the squared residuals on the original regressors
  - Plug in the fitted values from this secondary regression as weights in wls

Feasible Generalized Least Squares

```
results2.summary()
```

[3]:

OLS Regression Results

Dep. Variable:	food_exp	**R-squared:**	0.385
Model:	OLS	**Adj. R-squared:**	0.369
Method:	Least Squares	**F-statistic:**	23.79
Date:	Fri, 04 Nov 2022	**Prob (F-statistic):**	1.95e-05
Time:	12:17:44	**Log-Likelihood:**	-235.51
No. Observations:	40	**AIC:**	475.0
Df Residuals:	38	**BIC:**	478.4
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	83.4160	43.410	1.922	0.062	-4.463	171.295
income	10.2096	2.093	4.877	0.000	5.972	14.447

Omnibus:	0.277	**Durbin-Watson:**	1.894
Prob(Omnibus):	0.870	**Jarque-Bera (JB):**	0.063
Skew:	-0.097	**Prob(JB):**	0.969
Kurtosis:	2.989	**Cond. No.**	63.7

```
foodata["ehatsq"] = results2.resid**2

# estimate weights
w_est = smf.ols('np.log(ehatsq) ~ income', data = foodata).fit()

vari = np.exp(w_est.fittedvalues) #estimated variances
w = 1/vari**2

fgls =smf.wls('food_exp ~ income', foodata, weights = w).fit()

fgls.summary()
```

:

WLS Regression Results

Dep. Variable:	food_exp	**R-squared:**	0.772
Model:	WLS	**Adj. R-squared:**	0.766
Method:	Least Squares	**F-statistic:**	128.6
Date:	Fri, 04 Nov 2022	**Prob (F-statistic):**	9.21e-14
Time:	12:41:46	**Log-Likelihood:**	-234.12
No. Observations:	40	**AIC:**	472.2
Df Residuals:	38	**BIC:**	475.6
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	73.0257	6.206	11.766	0.000	60.462	85.590
income	11.0233	0.972	11.342	0.000	9.056	12.991

Exercise 2

- Use the *hprice1* dataset from the wooldridge module and fit a regression of the form:

  $$price = \beta_0 + \beta_1 lotsize + \beta_2 sqrft + \beta_3 bdrms + e$$

  Use robust standard errors (HCO) and run a separate regression without them

- Use the BP or White test to check for heteroscedasticity in both models. What do you notice? Why do you think this happens?
- Use the FGLS procedure to estimate reestimate the model from above