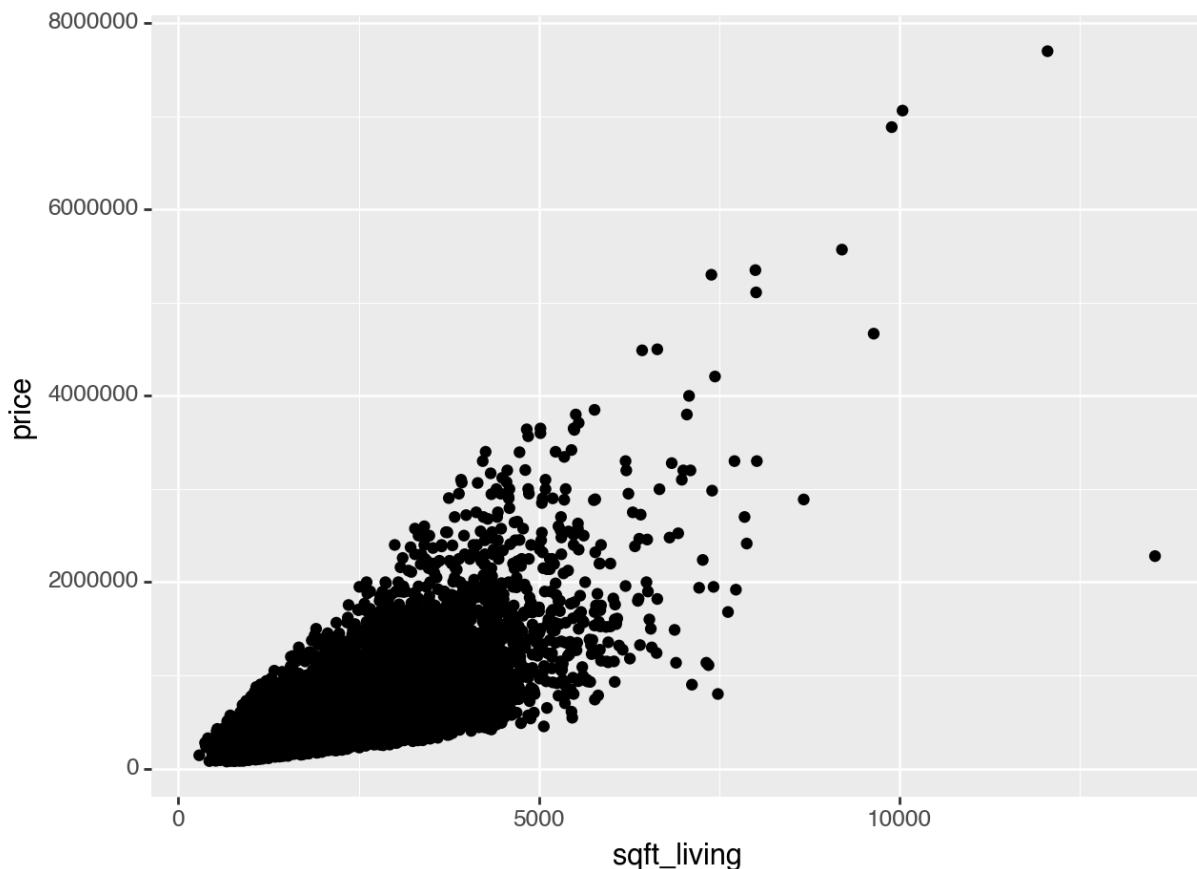


```
In [1]: import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import statsmodels.api as sm
import statsmodels.formula.api as smf
from plotnine import *
```

Examples of Heteroskedasticity

```
In [3]: df = pd.read_csv('../Data/kc_house_data.csv')
```

```
In [4]: (ggplot(df, aes(x = 'sqft_living', y = 'price')) +
    geom_point())
```



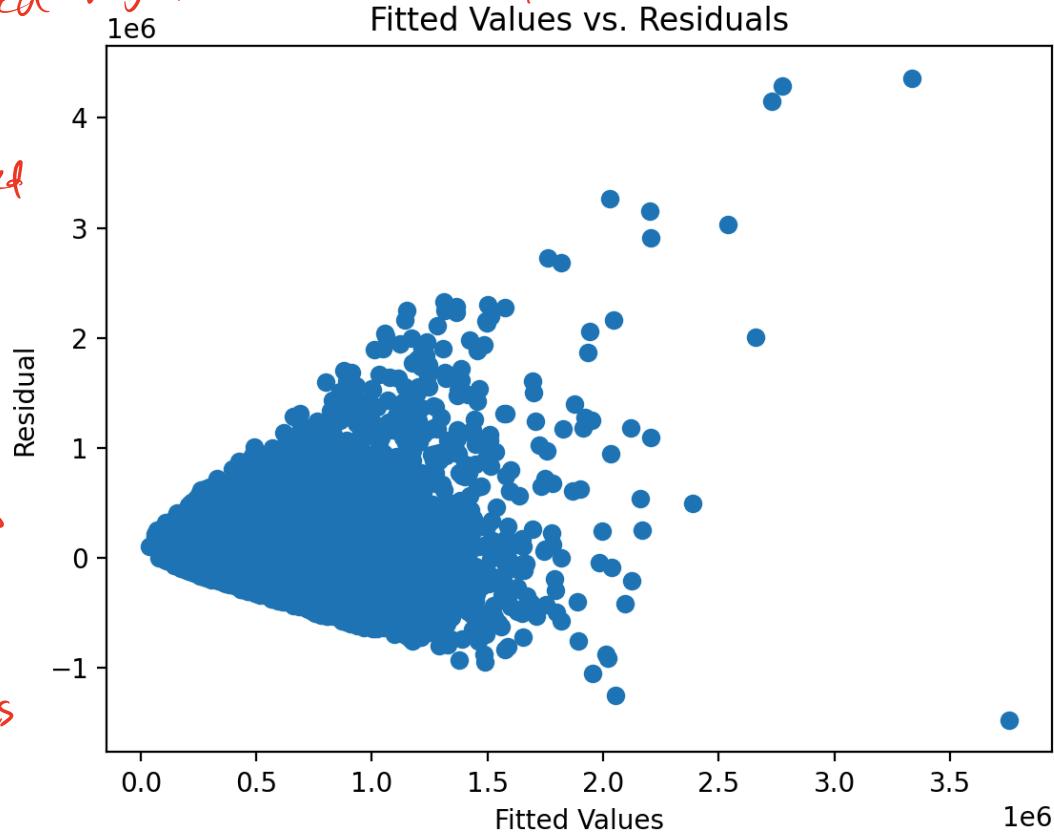
```
Out[4]: <Figure Size: (640 x 480)>
```

```
In [5]: f = 'price~sqft_living'
model = smf.ols(formula=f, data=df).fit()
p = model.fittedvalues
res = model.resid
plt.scatter(p, res)
plt.xlabel("Fitted Values")
plt.ylabel("Residual")
plt.title("Fitted Values vs. Residuals")
```

```
plt.show()
```

Issues caused by heteroskedasticity

- LS estimators are still unbiased but no longer min. variance
- $\text{SE}(\hat{\beta}_k)$'s are incorrect if heteroskedasticity is unchecked
- CI & PI have incorrect widths



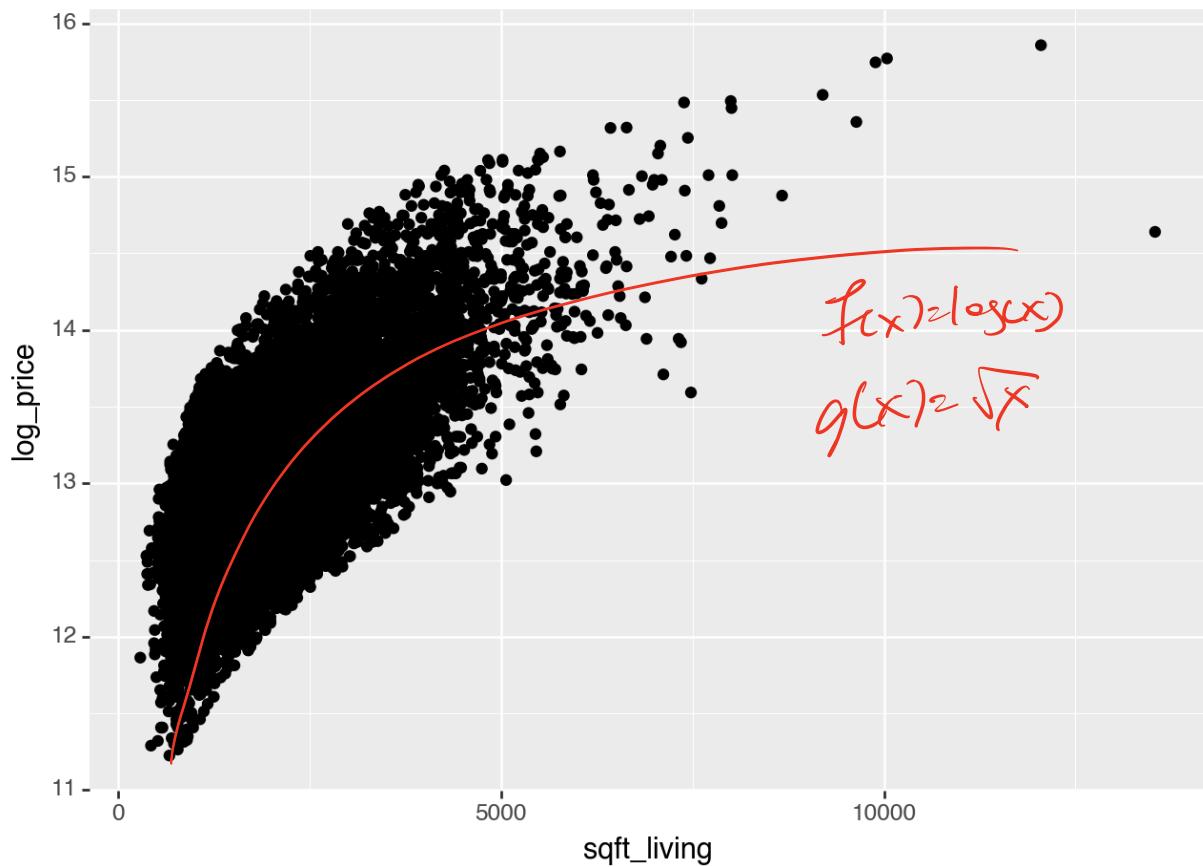
The classic fan shape emblematic of heteroskedasticity

One option: Transforming the Response

Normally when we see the "fan" shape in the residual plot, a log transformation on y helps mitigate heteroskedasticity.

```
In [8]: df['log_price'] = np.log(df['price'])
```

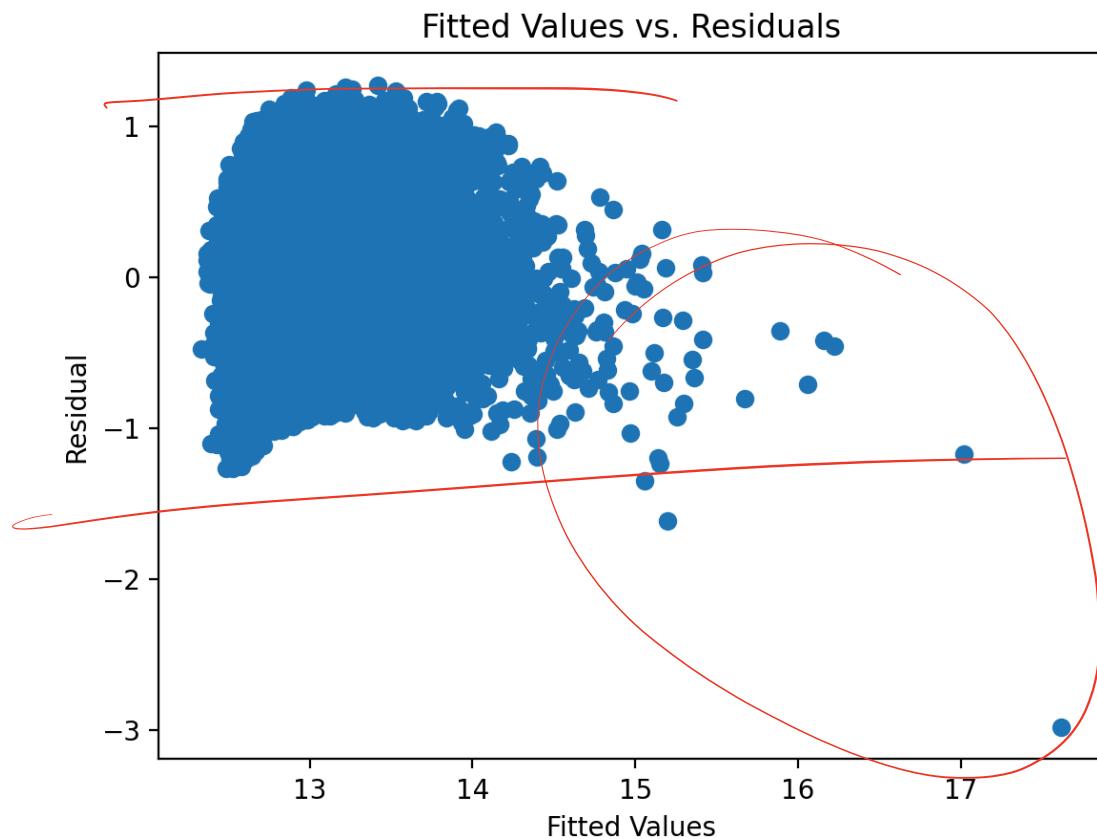
```
In [9]: (ggplot(df, aes(x = 'sqft_living', y = 'log_price')) +  
    geom_point())
```



Out[9]: <Figure Size: (640 x 480)>

```
In [10]: f2 = 'log_price~sqft_living'
model2 = smf.ols(formula=f2, data=df).fit()
p2 = model2.fittedvalues
res2 = model2.resid
plt.scatter(p2,res2)
plt.xlabel("Fitted Values")
plt.ylabel("Residual")
plt.title("Fitted Values vs. Residuals")

plt.show()
```



The heteroskedasticity is improved, but now we see some non-linearity between x and y .

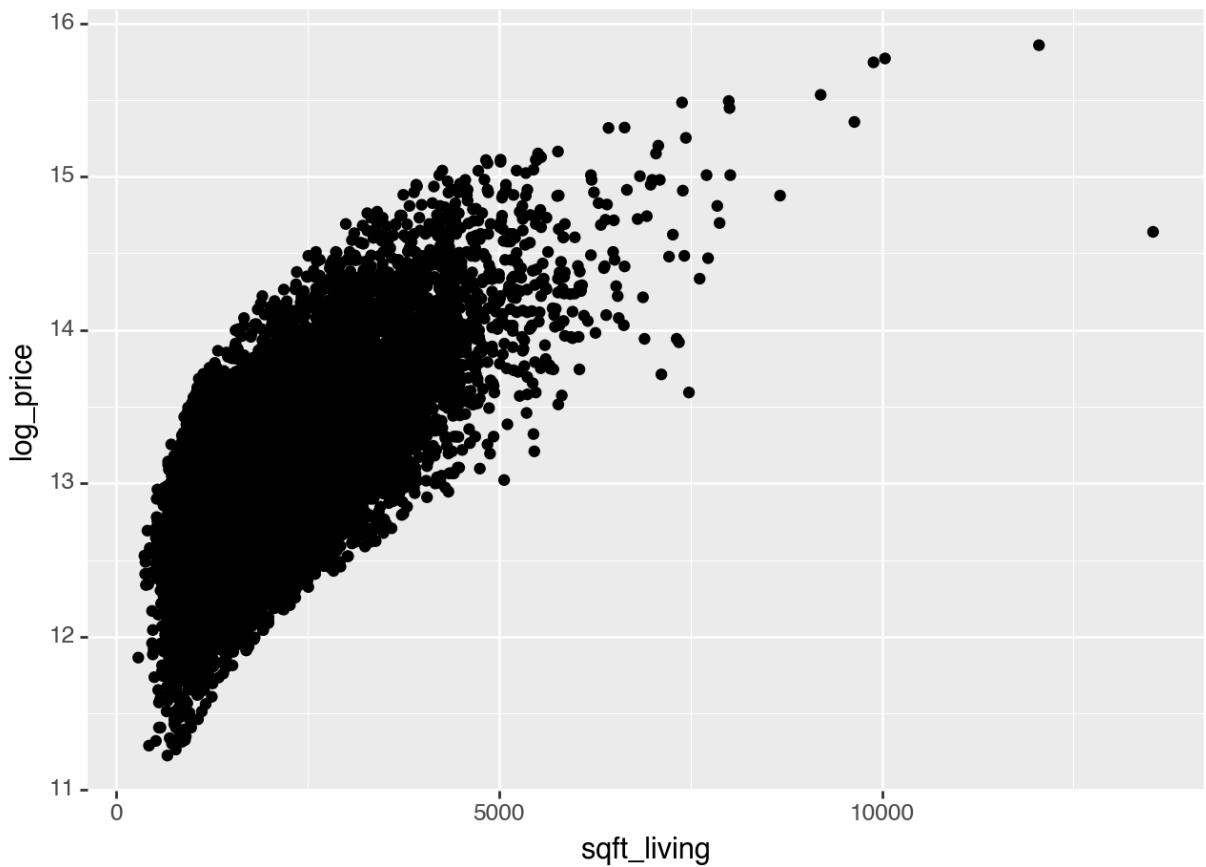
This suggests we may need to transform the predictor too, in order to find a linear relationship between the response and predictor.

Transforming the Predictor

A heuristic for transforming x is just to look at the scatterplot of x vs. the response in question and see what function the scatterplot approximates.

If we are using `sqft_living` to predict `log_price`, the scatterplot looks like $y = \log(x)$ or maybe $y = \sqrt{x}$.

```
In [14]: (ggplot(df, aes(x = 'sqft_living', y = 'log_price')) +
    geom_point())
```

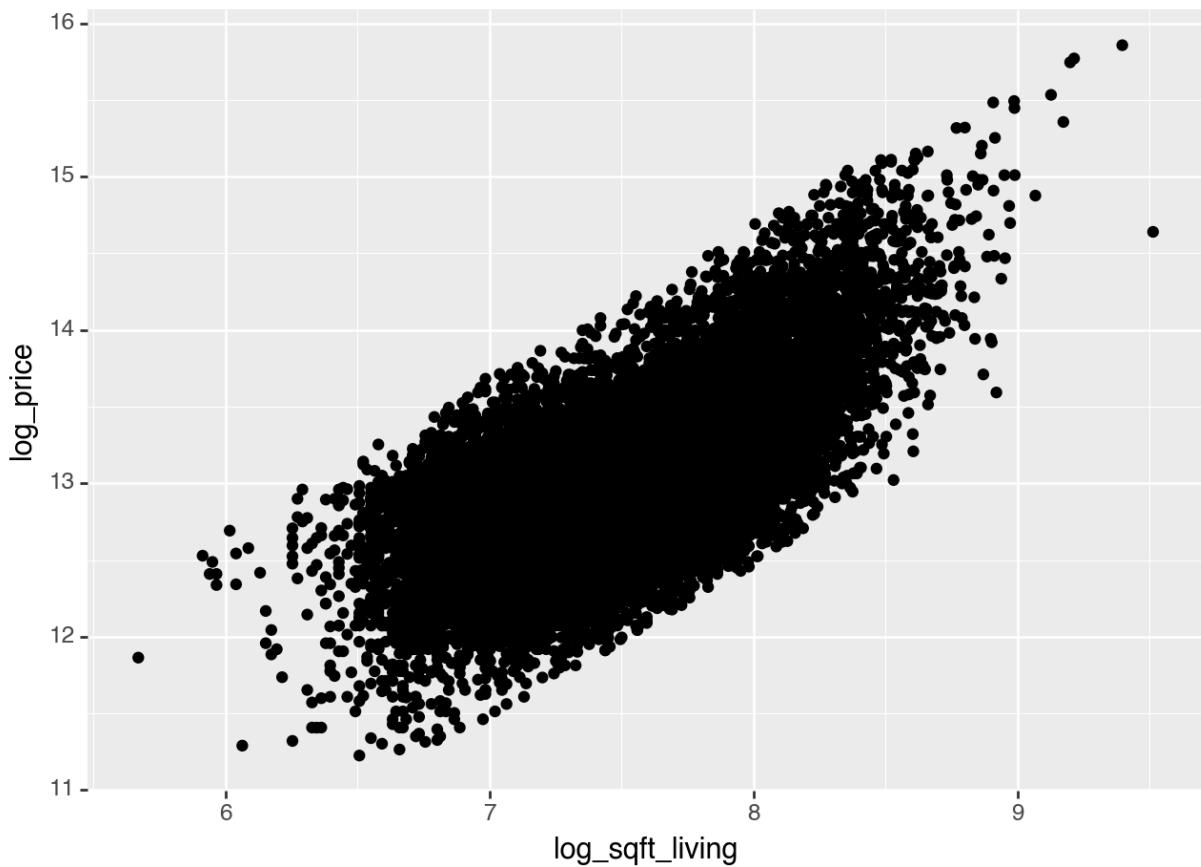


```
Out[14]: <Figure Size: (640 x 480)>
```

We can try both...

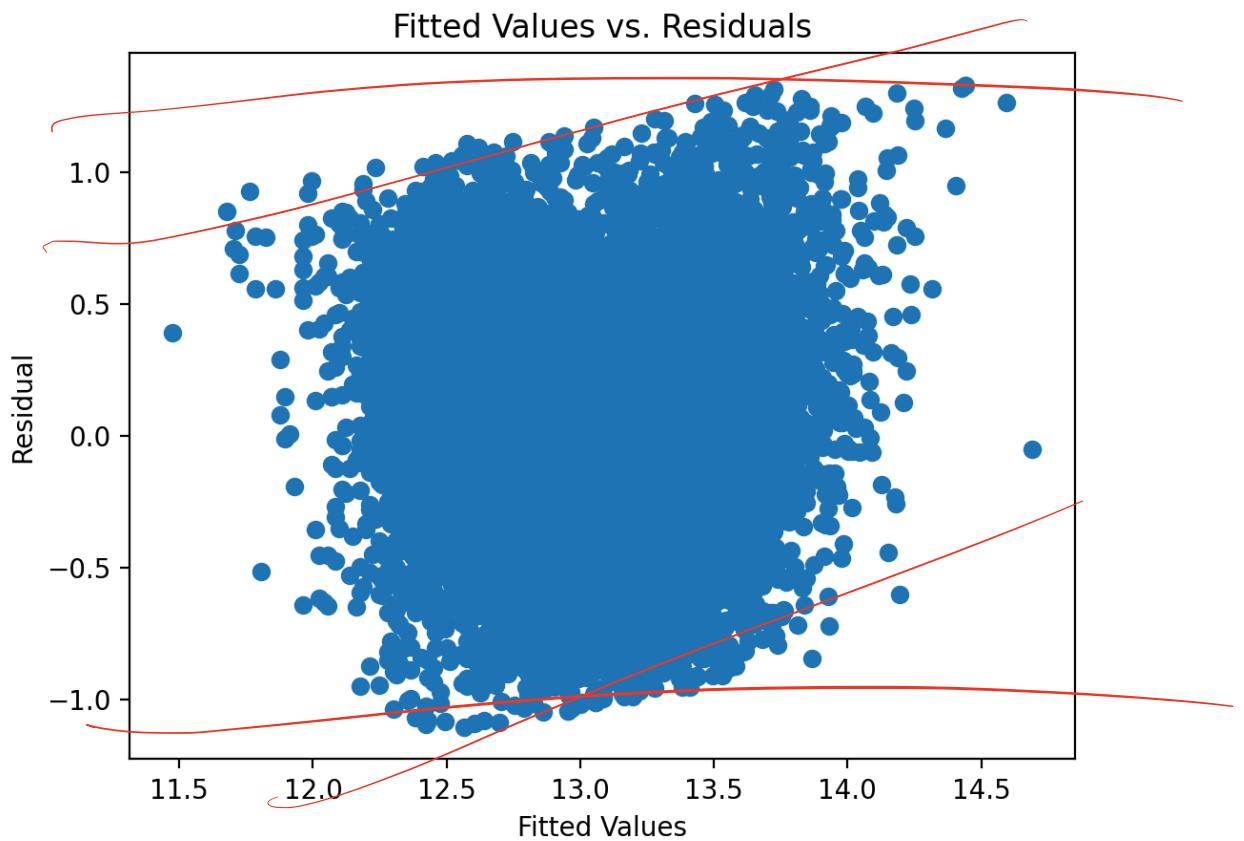
If we use log:

```
In [17]: df['log_sqft_living'] = np.log(df['sqft_living'])  
~~~~~  
(ggplot(df, aes(x = 'log_sqft_living', y = 'log_price')) +  
geom_point())
```



Out[17]: <Figure Size: (640 x 480)>

```
In [18]: f3 = 'log_price~log_sqft_living'
model3 = smf.ols(formula=f3, data=df).fit()
p3 = model3.fittedvalues
res3 = model3.resid
plt.scatter(p3,res3)
plt.xlabel("Fitted Values")
plt.ylabel("Residual")
plt.title("Fitted Values vs. Residuals")
plt.show()
```



```
In [19]: model3.summary()
```

Out[19]:

OLS Regression Results

Dep. Variable:	log_price	R-squared:	0.456			
Model:	OLS	Adj. R-squared:	0.455			
Method:	Least Squares	F-statistic:	1.808e+04			
Date:	Tue, 24 Sep 2024	Prob (F-statistic):	0.00			
Time:	08:57:41	Log-Likelihood:	-10240.			
No. Observations:	21613	AIC:	2.048e+04			
Df Residuals:	21611	BIC:	2.050e+04			
Df Model:	1					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
Intercept	6.7299	0.047	143.001	0.000	6.638	6.822
log_sqft_living	0.8368	0.006	134.459	0.000	0.825	0.849
Omnibus:	123.344	Durbin-Watson:		1.978		
Prob(Omnibus):	0.000	Jarque-Bera (JB):		113.759		
Skew:	0.142	Prob(JB):		1.98e-25		
Kurtosis:	2.787	Cond. No.		137.		

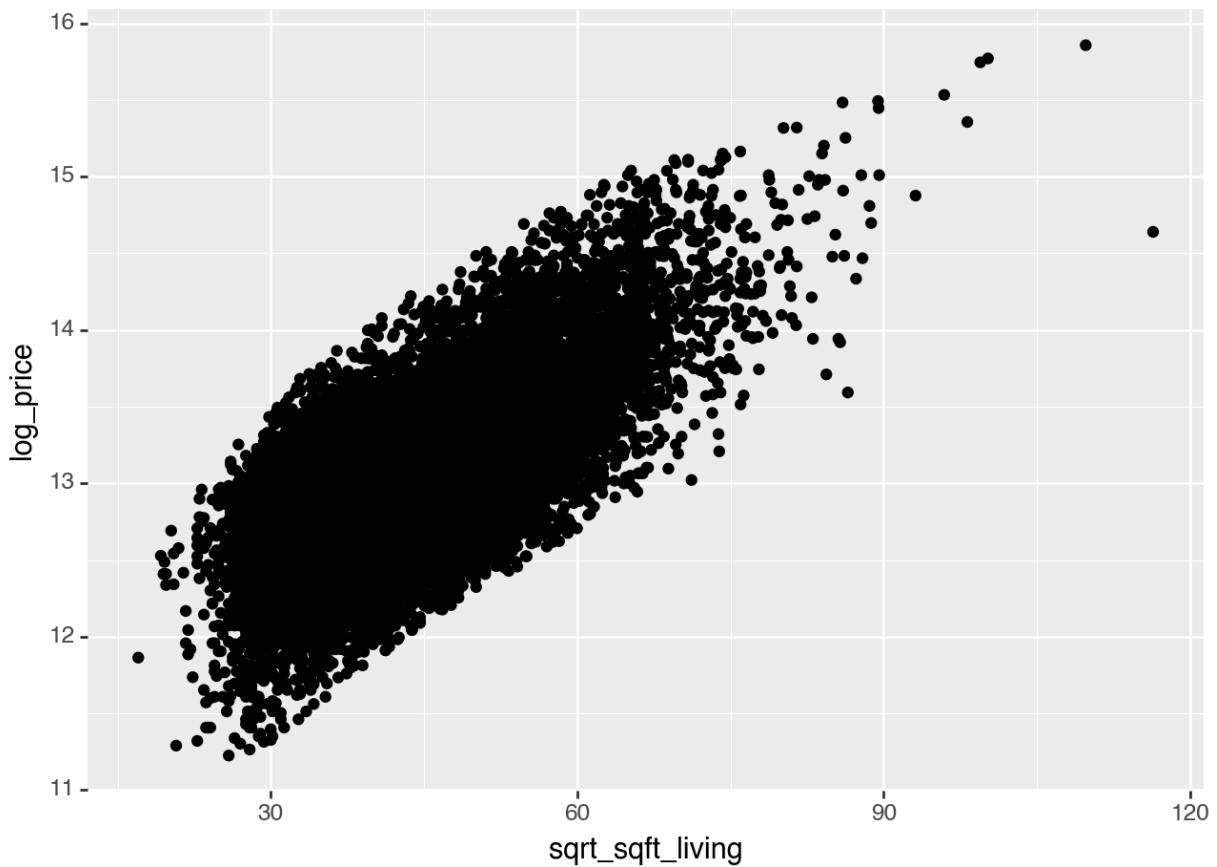
Notes:

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

If we use sqrt:

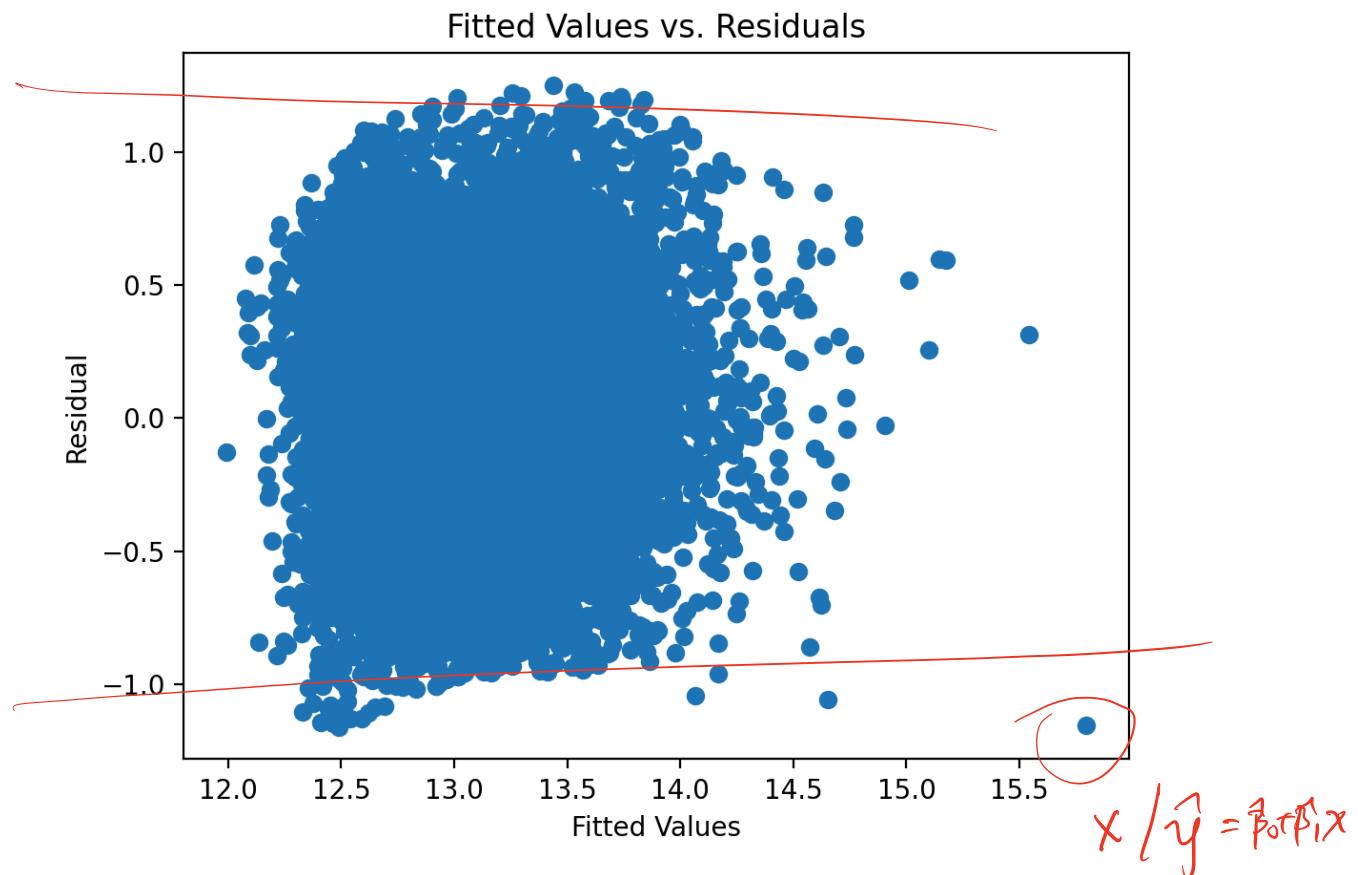
```
In [21]: df['sqrt_sqft_living'] = np.sqrt(df['sqft_living'])

(ggplot(df, aes(x = 'sqrt_sqft_living', y = 'log_price')) +
    geom_point())
```



Out[21]: <Figure Size: (640 x 480)>

```
In [22]: f4 = 'log_price~sqrt_sqft_living'
model4 = smf.ols(formula=f4, data=df).fit()
p4 = model4.fittedvalues
res4 = model4.resid
plt.scatter(p4,res4)
plt.xlabel("Fitted Values")
plt.ylabel("Residual")
plt.title("Fitted Values vs. Residuals")
plt.show()
```



Either works – I think log price ~ log sqft looks a little bit nicer in terms of the residual plot.

Note: Transformation can be applicable even when we don't have heteroskedasticity -- for example if there's a nonlinear relationship between x and y.

A general strategy to identify which transformations are most appropriate is to use the Box-Cox transformation method. This is helpful, for example, if I couldn't guess that log or sqrt were reasonable functions to use based off of a visual inspection. See, e.g.,
https://www.css.cornell.edu/faculty/dgr2/_static/files/R_html/Transformations

Another solution to address heteroskedasticity: Robust standard errors

It is common to use "robust standard errors" in econometric applications. We can access the robust standard errors, or Heteroskedasticity-Consistent Standard Errors (HCE) using `statsmodels` methods:

The robust estimated variance matrix is calculated as:

$$Var_{HC}(\hat{\beta}) = (X^T X)^{-1} X^T \text{diag}(e_i^2) X (X^T X)^{-1}$$

$$\downarrow (X^T X)^{-1} X^T \text{diag}(e_i^2) X (X^T X)^{-1}$$

$$\check{r}^2 \quad (\check{X}^T \check{X})^{-1} \check{X}^T \check{Y} \quad (\check{X}^T \check{X})^{-1}$$

in line with White (1980).

This adjustment tries to downweigh the impact of increasing residuals in the variance estimation step.

Notice that if $e_i^2 \approx \sigma^2$ for all i, the formula reduces to the usual variance matrix for beta hat:

$$Var(\hat{\beta}) = \sigma^2(X^T X)^{-1}.$$

```
In [27]: credit = pd.read_csv('Data/Credit.csv')

#original model that we know having heteroskedasticity problem
model = smf.ols('Balance ~ Income + Rating + Cards + Age + Education \
+ Gender + Student + Married + Ethnicity', data=credit).fit()

#extract HCE variance-covariance matrix
var_HC = model.cov_HC0
```

Var(β_0)

```
Out[27]: array([[ 1.28979518e+03, -2.85353537e+01, -1.93737996e+01,
   -1.91773618e+01, -1.21102507e+02, -1.31108552e+02,
   2.15189155e+00, -9.87186508e-01, -2.83436775e+01,
   -4.19545031e+00, -4.02601264e+01],
  [-2.85353537e+01,  1.04540615e+02, -1.76173734e+01,
   -9.13676854e+00,  3.60555120e+00, -2.99505322e+00,
   1.95705491e-01, -3.60836602e-02,  1.37253190e+00,
   -1.93788690e-01, -5.71573694e-01],
  [-1.93737996e+01, -1.76173734e+01,  2.60173417e+02,
   1.19331093e+01, -1.82455379e+01, -6.48625401e+00,
   -7.32623436e-01,  1.69110939e-01, -1.18638848e+00,
   -2.39572478e-02, -1.69790633e+00],
  [-1.91773618e+01, -9.13676854e+00,  1.19331093e+01,
   1.09805461e+02, -2.66797583e+01, -1.78231341e+01,
   2.22313056e-01, -5.20147833e-02,  1.69160764e+00,
   -1.79079061e-01, -9.93094775e-01],
  [-1.21102507e+02,  3.60555120e+00, -1.82455379e+01,
   -2.66797583e+01,  2.11832833e+02,  1.03655948e+02,
   1.16961731e-01,  1.30085214e-03, -5.00674435e+00,
   6.48215640e-01,  4.77855921e-01],
  [-1.31108552e+02, -2.99505322e+00, -6.48625401e+00,
   -1.78231341e+01,  1.03655948e+02,  1.54647620e+02,
   3.56773320e-01, -4.99088021e-02, -2.83152775e-01,
   5.56792020e-01,  1.05395794e+00],
  [ 2.15189155e+00,  1.95705491e-01, -7.32623436e-01,
   2.22313056e-01,  1.16961731e-01,  3.56773320e-01,
   5.46879998e-02, -1.16917379e-02,  1.03503126e-01,
   -5.85418584e-03, -2.66999700e-02],
  [-9.87186508e-01, -3.60836602e-02,  1.69110939e-01,
   -5.20147833e-02,  1.30085214e-03, -4.99088021e-02,
   -1.16917379e-02,  4.09522670e-03, -2.38449033e-02,
   -1.79848498e-03,  9.53202836e-03],
  [-2.83436775e+01,  1.37253190e+00, -1.18638848e+00,
   1.69160764e+00, -5.00674435e+00, -2.83152775e-01,
   1.03503126e-01, -2.38449033e-02,  1.14035045e+01,
   -1.15145126e-01,  4.51416457e-01],
  [-4.19545031e+00, -1.93788690e-01, -2.39572478e-02,
   -1.79079061e-01,  6.48215640e-01,  5.56792020e-01,
   -5.85418584e-03, -1.79848498e-03, -1.15145126e-01,
   9.20241639e-02,  1.93567461e-02],
  [-4.02601264e+01, -5.71573694e-01, -1.69790633e+00,
   -9.93094775e-01,  4.77855921e-01,  1.05395794e+00,
   -2.66999700e-02,  9.53202836e-03,  4.51416457e-01,
   1.93567461e-02,  2.64627072e+00]])
```

In [28]: #sometimes we may not care to see the corrected SE values themselves,
#we can extract the corrected t test and p values directly if we just care about the statistics

```
print(np.round(model.get_robustcov_results(cov_type = "HC0").summary2().tables[0].iloc[1, 1:-1]))
```

MLR_heteroskedasticity_and_transformations

	Coef.	Std.Err.	t	P> t	[0.025 \
Intercept	-549.31402	35.91372	-15.29538	0.00000	-619.92330
Gender[T.Female]	-10.71056	10.22451	-1.04754	0.29550	-30.81278
Student[T.Yes]	416.43756	16.12989	25.81775	0.00000	384.72489
Married[T.Yes]	-15.10961	10.47881	-1.44192	0.15013	-35.71180
Ethnicity[T.Asian]	21.76158	14.55448	1.49518	0.13568	-6.85370
Ethnicity[T.Caucasian]	10.64919	12.43574	0.85634	0.39234	-13.80048
Income	-7.77460	0.23385	-33.24542	0.00000	-8.23437
Rating	3.97896	0.06399	62.17709	0.00000	3.85314
Cards	3.96537	3.37691	1.17426	0.24101	-2.67390
Age	-0.64159	0.30335	-2.11500	0.03507	-1.23801
Education	-0.37986	1.62674	-0.23351	0.81549	-3.57815
	0.975]				
Intercept	-478.70473				
Gender[T.Female]	9.39166				
Student[T.Yes]	448.15024				
Married[T.Yes]	5.49258				
Ethnicity[T.Asian]	50.37687				
Ethnicity[T.Caucasian]	35.09886				
Income	-7.31482				
Rating	4.10478				
Cards	10.60465				
Age	-0.04517				
Education	2.81844				

In [29]: `#Compare to the orginal model summary, numbers are slightly changed.
model.summary()`

Out[29]:

OLS Regression Results

Dep. Variable:	Balance	R-squared:	0.951			
Model:	OLS	Adj. R-squared:	0.950			
Method:	Least Squares	F-statistic:	757.8			
Date:	Tue, 24 Sep 2024	Prob (F-statistic):	4.46e-248			
Time:	08:57:42	Log-Likelihood:	-2415.4			
No. Observations:	400	AIC:	4853.			
Df Residuals:	389	BIC:	4897.			
Df Model:	10					
Covariance Type:	nonrobust					
	coef	std err	t	P> t 	[0.025	0.975]
Intercept	-549.3140	35.085	-15.657	0.000	-618.293	-480.335
Gender[T.Female]	-10.7106	10.325	-1.037	0.300	-31.010	9.589
Student[T.Yes]	416.4376	17.336	24.021	0.000	382.353	450.522
Married[T.Yes]	-15.1096	10.728	-1.408	0.160	-36.202	5.983
Ethnicity[T.Asian]	21.7616	14.678	1.483	0.139	-7.096	50.619
Ethnicity[T.Caucasian]	10.6492	12.716	0.837	0.403	-14.351	35.649
Income	-7.7746	0.244	-31.878	0.000	-8.254	-7.295
Rating	3.9790	0.055	72.332	0.000	3.871	4.087
Cards	3.9654	3.793	1.045	0.296	-3.492	11.422
Age	-0.6416	0.306	-2.096	0.037	-1.243	-0.040
Education	-0.3799	1.659	-0.229	0.819	-3.642	2.882
Omnibus:	15.651	Durbin-Watson:	1.987			
Prob(Omnibus):	0.000	Jarque-Bera (JB):	16.769			
Skew:	0.490	Prob(JB):	0.000228			
Kurtosis:	2.789	Cond. No.	2.73e+03			

Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The condition number is large, 2.73e+03. This might indicate that there are strong multicollinearity or other numerical problems.

Another solution: Weighted least squares

In []:

$$\text{original: } \gamma \sim N(X\beta, \sigma^2 I) \quad \text{vs.} \quad \gamma \sim N(X\beta, \Sigma)$$

In [31]:

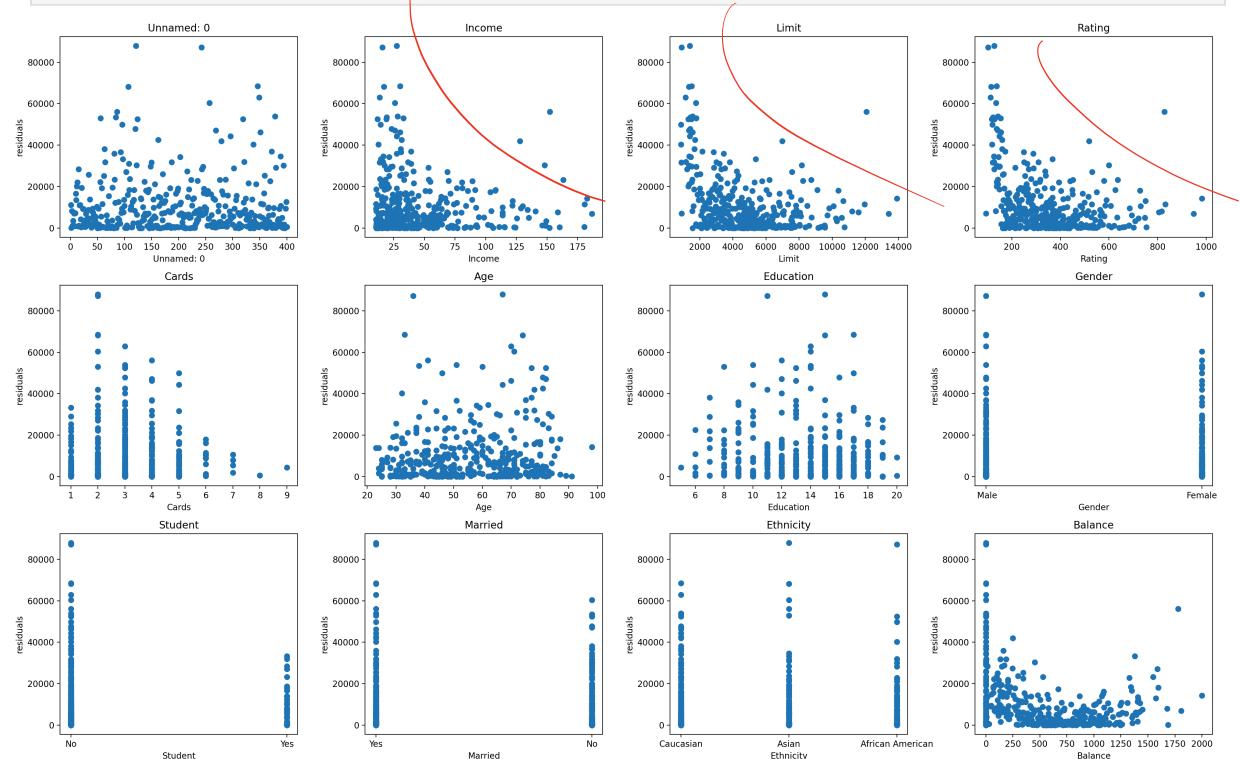
```
plt.figure(figsize=(20, 16))

# i: index
for i, col in enumerate(credit.columns):
    plt.subplot(4, 4, i+1)
    x = credit[col]
    y = model.resid**2
    plt.plot(x, y, 'o')

    plt.tight_layout()
    plt.title(col)
    plt.xlabel(col)
    plt.ylabel('residuals')

# looks like residuals are proportional to Income, Limit, and Rating

plt.show()
```



One method of performing weighted least squares is to regress each observation's squared residual onto the predictors causing

$$Y = X\beta + \varepsilon$$

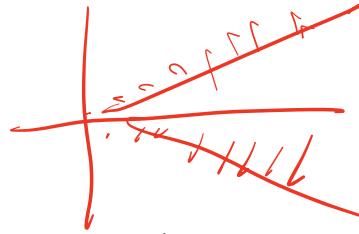
$$\text{Var}(\varepsilon) = \Sigma$$

Consider the transformation

$$\tilde{Y} = \Sigma^{-1/2} Y$$

$$\tilde{X} = \Sigma^{-1/2} X$$

$$\tilde{\varepsilon} = \Sigma^{-1/2} \varepsilon$$



$$\Sigma = \begin{pmatrix} \sigma_1^2 & & \\ & \ddots & \\ & & \sigma_n^2 \end{pmatrix}$$

$$W = \Sigma^{-1/2} = \begin{pmatrix} 1/\sigma_1 & & \\ & \ddots & \\ & & 1/\sigma_n \end{pmatrix}$$

$$\Rightarrow \Sigma^{-1/2} Y = \Sigma^{-1/2} X \beta + \Sigma^{-1/2} \varepsilon$$

$$\tilde{Y} = \tilde{X} \beta + \tilde{\varepsilon} \leftarrow \text{now I have uncorr var: } \text{Var}(\tilde{\varepsilon}) = I$$

(Motivation)

$$\text{Var}(\tilde{\varepsilon}) = \text{Var}(\Sigma^{-1/2} \varepsilon) = \Sigma^{-1/2} \text{Var}(\varepsilon) \Sigma^{-1/2}$$

$$= \Sigma^{-1/2} \Sigma \Sigma^{-1/2} = I$$

$$= \Sigma^{-1/2} \Sigma^{1/2} \Sigma^{1/2} \Sigma^{-1/2} = I$$

This is the OLS problem!

$$\hat{\beta} = (\tilde{X}^T \tilde{X})^{-1} \tilde{X}^T \tilde{Y}$$

plus in

$$\begin{aligned}
 \hat{\beta} &= ((\Sigma^{-1/2} X)^T (\Sigma^{-1/2} X))^{-1} (\Sigma^{-1/2} X)^T (\Sigma^{-1/2} Y) \\
 &= (\underbrace{X^T \Sigma^{-1/2} \Sigma^{-1/2} X}_{=})^{-1} (\underbrace{X^T \Sigma^{-1/2} \Sigma^{-1/2} Y}_{=}) \\
 &= (\underbrace{X^T \Sigma^{\hat{\wedge}-1} X}_{=})^{-1} (\underbrace{X^T \Sigma^{\hat{\wedge}-1} Y}_{=})
 \end{aligned}$$

If we don't have heteroskedasticity

$$\Sigma = \sigma^2 I$$

$$\begin{aligned}
 \hat{\beta} &= (X^T (\sigma^2 I)^{-1} X)^{-1} (X^T (\sigma^2 I)^{-1} Y) \\
 &= \sigma^2 (X^T X)^{-1} \left(\frac{1}{\sigma^2} (X^T Y) \right) = (X^T X)^{-1} X^T Y
 \end{aligned}$$

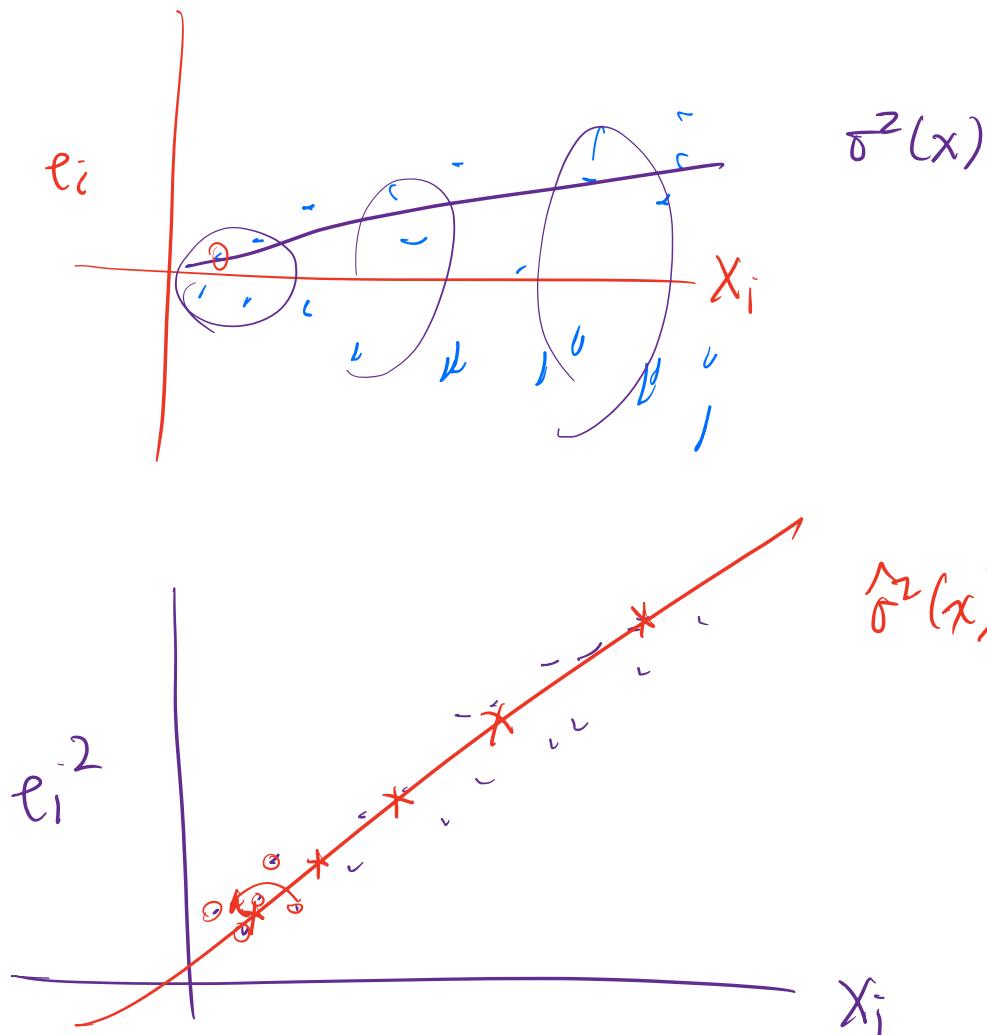
heteroskedasticity, and then use the fitted value's absolute value to weigh each observation:

$$e_i^2 \sim \text{Rating}$$

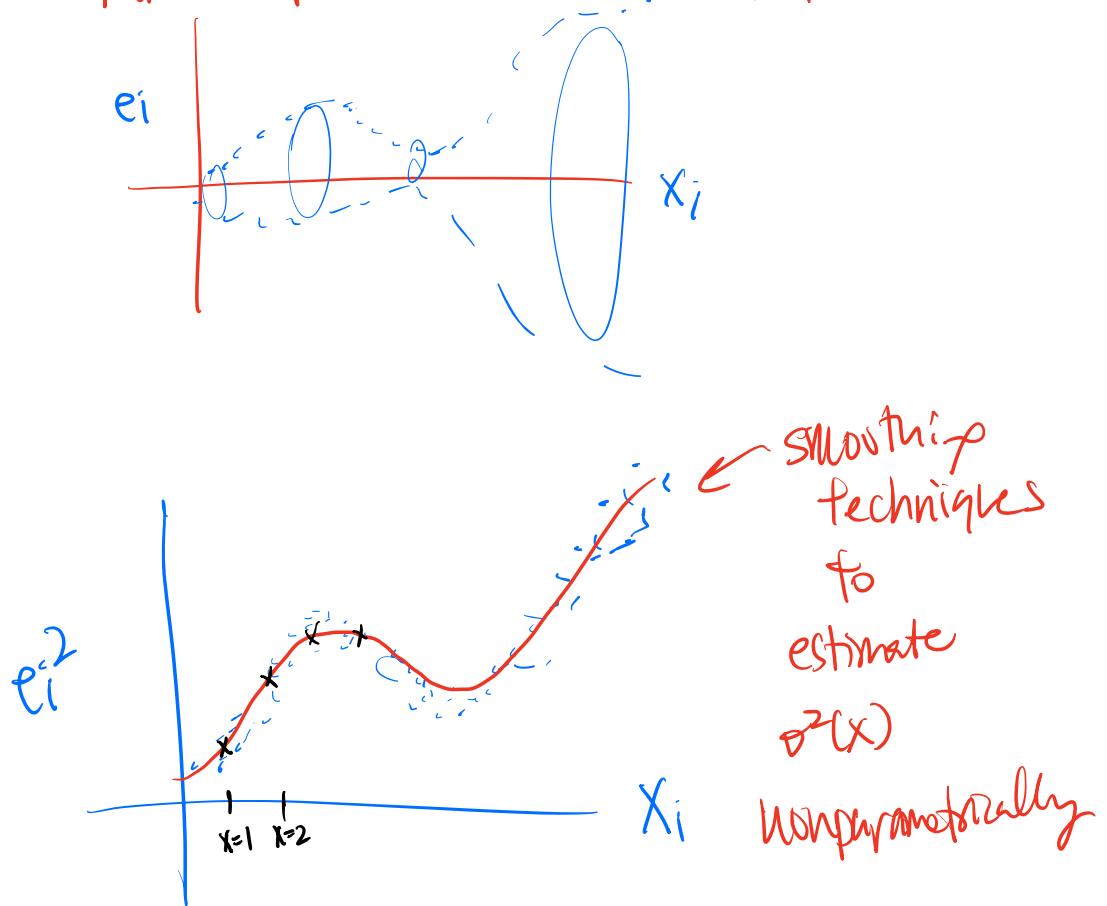
```
In [33]: credit["res_sq"] = model.resid**2
model_res = smf.ols("res_sq ~ Rating", data=credit).fit() #Here we don't al
weight = model_res.fittedvalues  $\hat{\beta}_1^2$ 
weight = abs(weight)  $\hat{\beta}_2^2$ 
weight = weight**-1/2
credit['weight'] = weight  $\hat{\beta}_n^2$ 
```

```
In [34]: y=credit.Balance
x=credit[['Income', 'Rating', 'Cards', 'Age', 'Education']] #or some other comb
```

```
In [35]: mod_wls = sm.WLS(y,x.astype(float), credit['weight'])
mod_res = mod_wls.fit()
print(mod_res.summary())
```



More Complicated Heteroskedasticity



WLS Regression Results

Dep. Variable:	Balance	R-squared (uncentered):				
0.975						
Model:	WLS	Adj. R-squared (uncentered):				
0.975						
Method:	Least Squares	F-statistic:				
3064.						
Date:	Tue, 24 Sep 2024	Prob (F-statistic):				
2.25e-313						
Time:	08:57:43	Log-Likelihood:				
-2773.1						
No. Observations:	400	AIC:				
5556.						
Df Residuals:	395	BIC:				
5576.						
Df Model:	5					
Covariance Type:	nonrobust					
<hr/>						
<hr/>						
5]	coef	std err	t	P> t	[0.025	0.97
<hr/>						
Income	-6.4211	0.419	-15.341	0.000	-7.244	-5.5
98						
Rating	3.7216	0.104	35.802	0.000	3.517	3.9
26						
Cards	-31.4870	6.676	-4.716	0.000	-44.613	-18.3
62						
Age	-4.2615	0.458	-9.297	0.000	-5.163	-3.3
60						
Education	-9.2931	2.494	-3.726	0.000	-14.197	-4.3
90						
<hr/>						
<hr/>						
Omnibus:	368.200	Durbin-Watson:			1.8	
03						
Prob(Omnibus):	0.000	Jarque-Bera (JB):			25125.4	
76						
Skew:	3.514	Prob(JB):			0.	
00						
Kurtosis:	41.185	Cond. No.			44	
5.						
<hr/>						
<hr/>						

Notes:

- [1] R^2 is computed without centering (uncentered) since the model does not contain a constant.
- [2] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In []: