1. Functions.

```
generate.data <- function(n, p, sig.epsilon=1){
  ## This function takes 3 inputs (2 mandatory, 1 optional)
  ## n - the number of observations
  ## p - the number of predictors
  ## sig.epsilon - (optional) the sd of the normal noise (default=1 if omitted)
  X = matrix(rnorm(p*n), ncol=p) ## a matrix of standard normal RVs (n x p)
  epsilon = rnorm(n, sd = sig.epsilon) ## noise ~ N(0, sig.epsilon)
  beta = p:1 # p beta coefficients
  beta.0 = 3 # an intercept
  y = beta.0 + X %*% beta + epsilon # the linear model
  df = data.frame(y, X) # put it all in a data frame
  return(df) # output
}

estimate.and.plot <- function(form, dataframe, plotme = TRUE){
  ## Estimates and (optionally plots some diagnostics for) a linear model
  ## Takes in a formula and data frame
  ## plotme determines whether plots are generated
  mdl = lm(form, data=dataframe) # estimates the model on all predictors
    # assumes the response is named 'y'
  if(plotme){ # do we produce plots?
    preds = labels(terms(form, data=dataframe))
    df = dataframe[preds]
    df$resids = residuals(mdl) # how do you get residuals?
    df$fit = fitted(mdl) # how do you get the fitted values?
    preds.vs.resids = df %>%
      gather(-c(resids,fit), key='predictor', value='value')
    # create a new dataframe for ggplot
    # this concatenates all the predictors into a long vector called 'value'
    # it makes another long vector (a factor) naming the predictors ('predictor')
    # it then replicates resids and fit as two more columns with appropriate
    # length
    p1 <- ggplot(preds.vs.resids, aes(x=value, y=resids)) + geom_point() +
      geom_smooth() + facet_wrap(~predictor, scales = 'free')
      # plot residuals against predictors
      # 1 facet for each predictor, and add a smooth
      # (the scales='free' stuff lets the axis vary
    p2 <- ggplot(df, aes(sample=resids)) + geom_qq() + geom_qq_line()
    # qq plot of the residuals to assess normality
    print(p1) # print out the first plot (wouldn't do this inside a function generally)
    print(p2) # print out the second plot
  }
  return(mdl) # output our fitted model
}
```

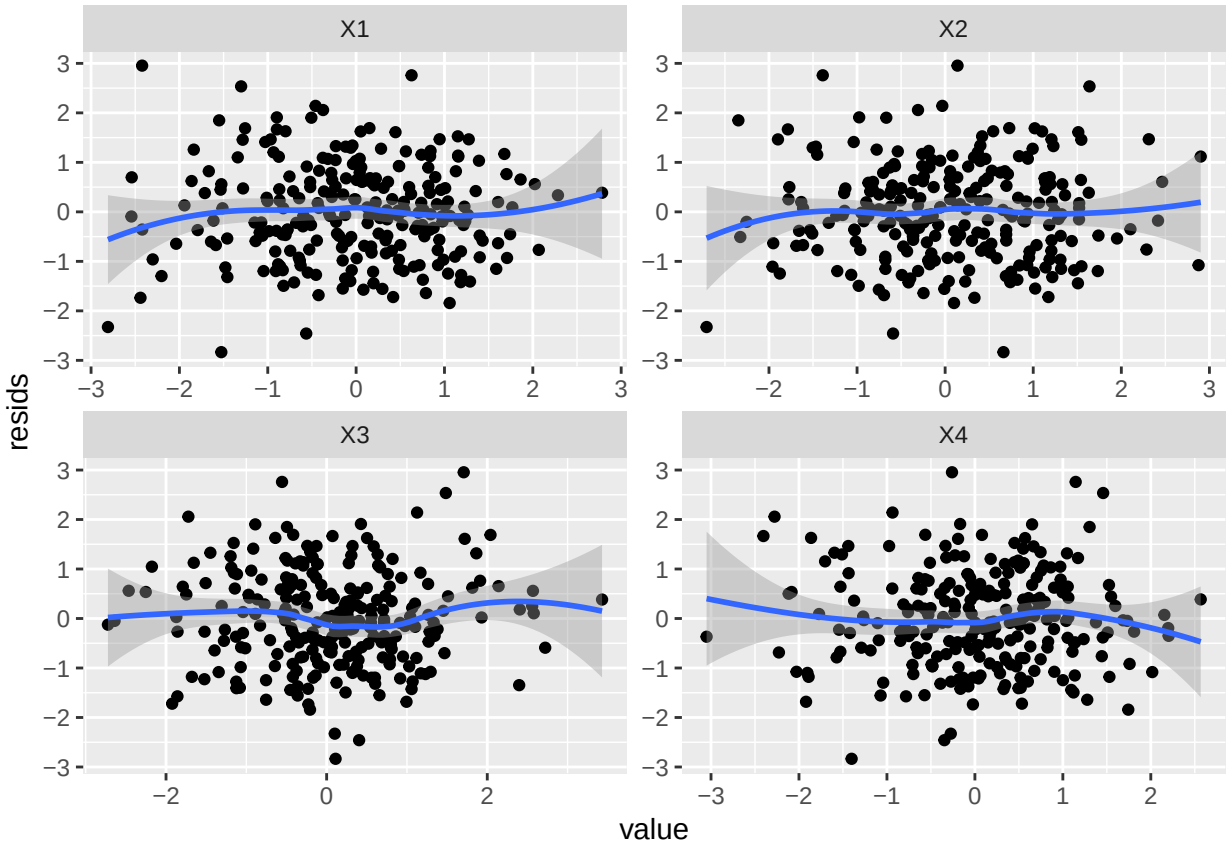
2. Function execution.

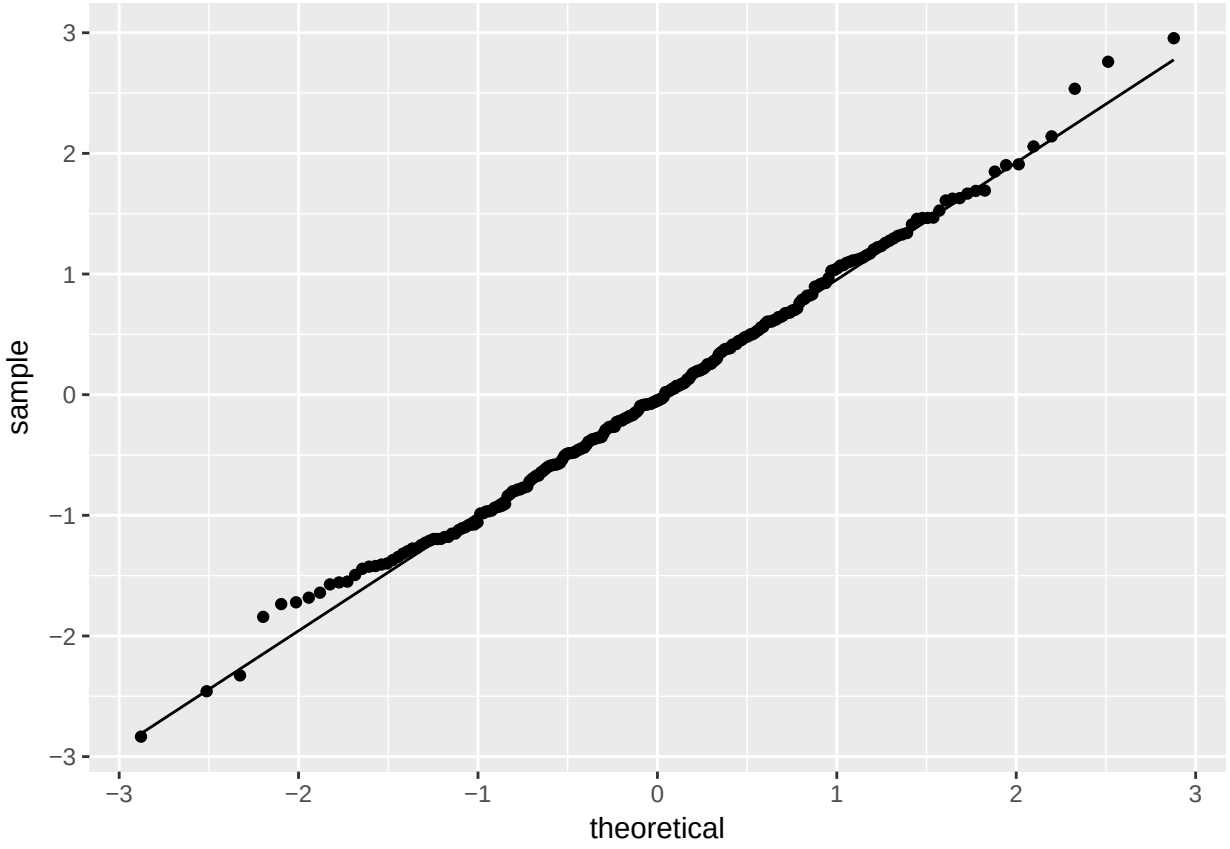
- Generate some data with the first function. Use 4 predictors (you can choose n and the noise SD yourself).

```
df = generate.data(250, 4)
```

- Estimate the model with the second function. And produce the plots.

```
mdl = estimate.and.plot(formula('y~.'), df)
```





- Create a table which shows the coefficients, their standard errors, and p-values. You must use the `knitr::kable` function to do this. Print only 2 significant digits. Hint: there is a way to extract all of this information easily from the `lm` output; you can do this in 1 line.

```
knitr::kable(summary(mdl)$coef, digits = 2)
```

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	3.07	0.06	49	0
X1	3.89	0.06	62	0
X2	3.03	0.06	50	0
X3	2.05	0.06	33	0
X4	0.95	0.07	14	0

3. Engage.

You will now attempt to re-engage last semester's brain cells by doing things you should already know how to do in possibly new ways. Consider the "properties.txt" dataset from HW 9 and 10 (optional) in S431 (described in exercise 6.18 of that text). Recall that it has an outcome (rental rates) and four predictors (age, operating expenses + taxes, vacancy rates, square footage). The goal is to predict rental rates using these four variables.

1. Use the `lm` function to estimate the linear model of rental rates on all four predictors. Produce a table summarizing the output.

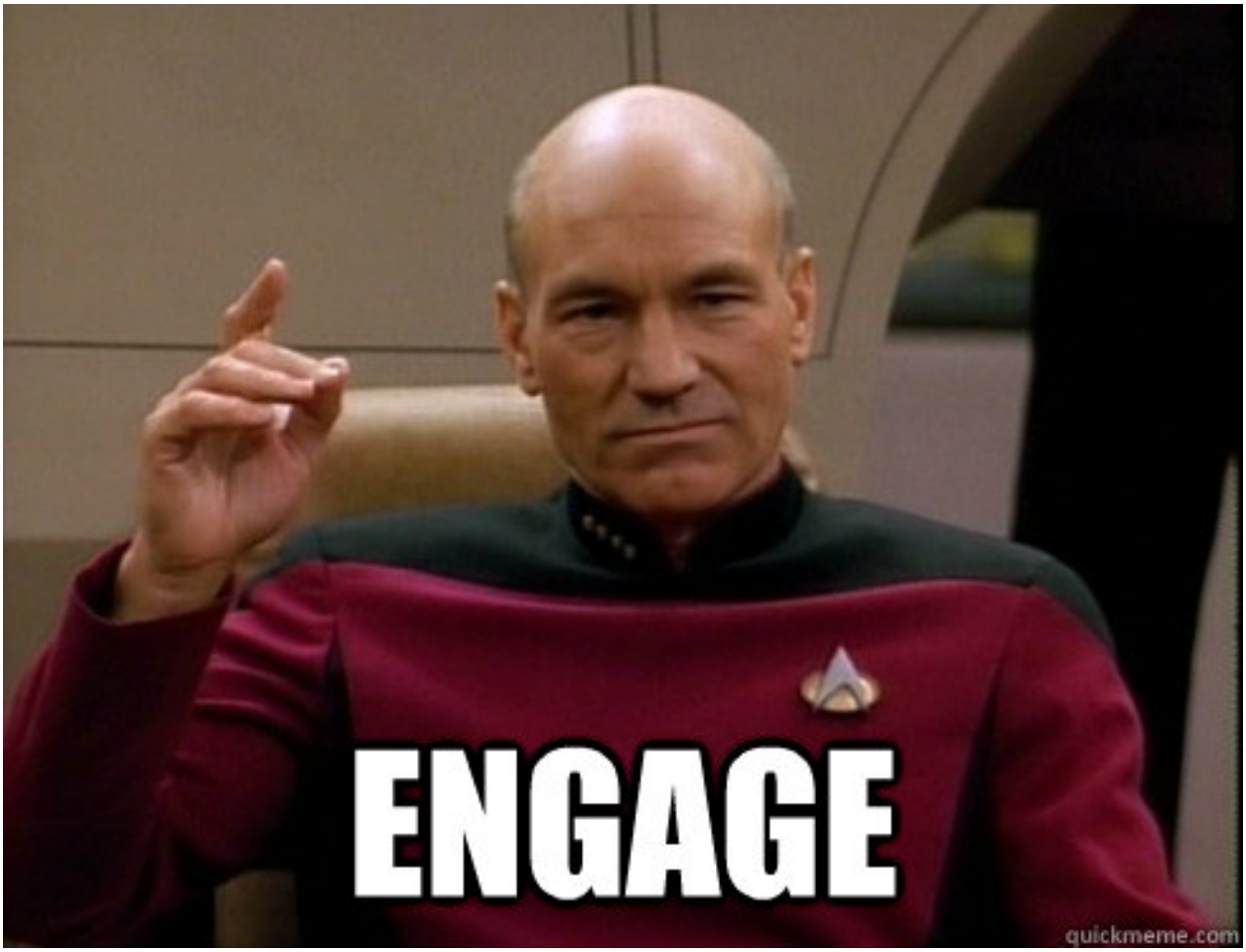


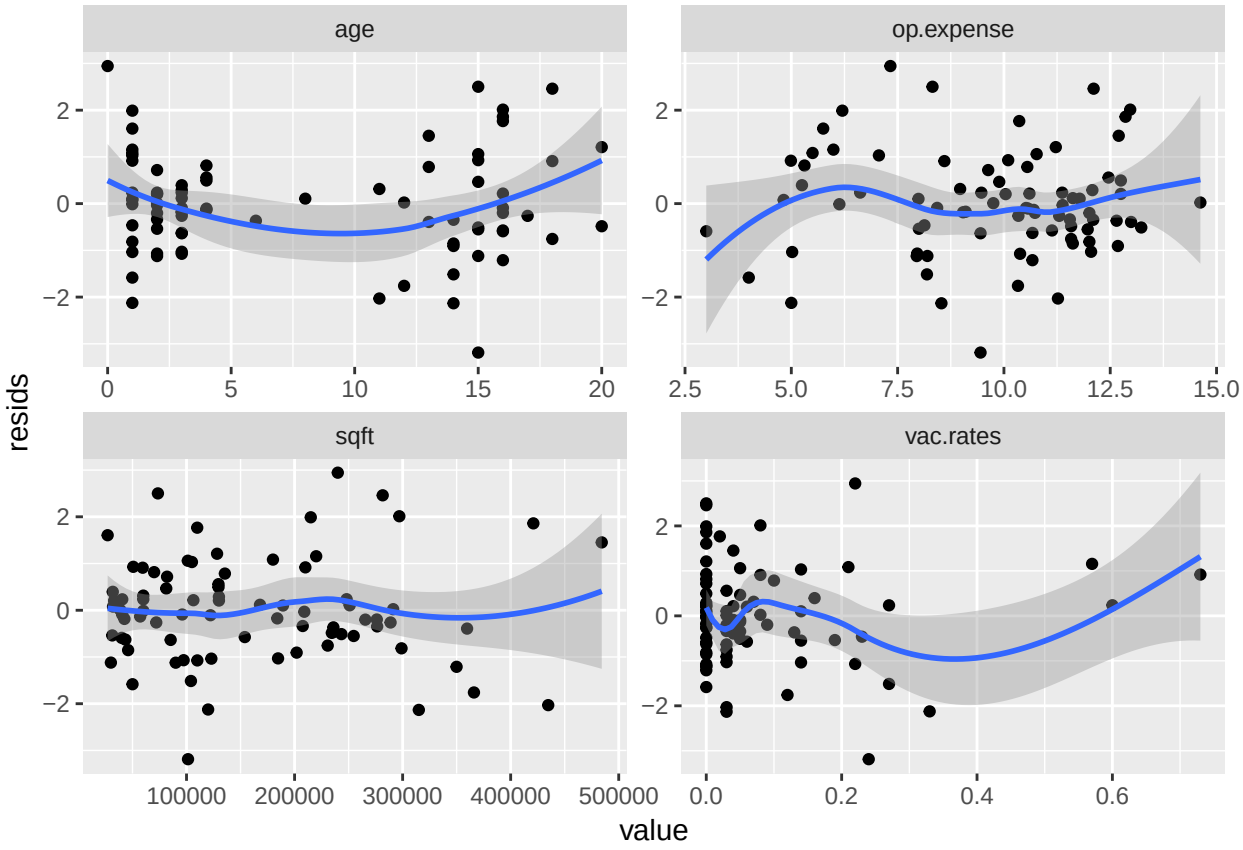
Figure 1

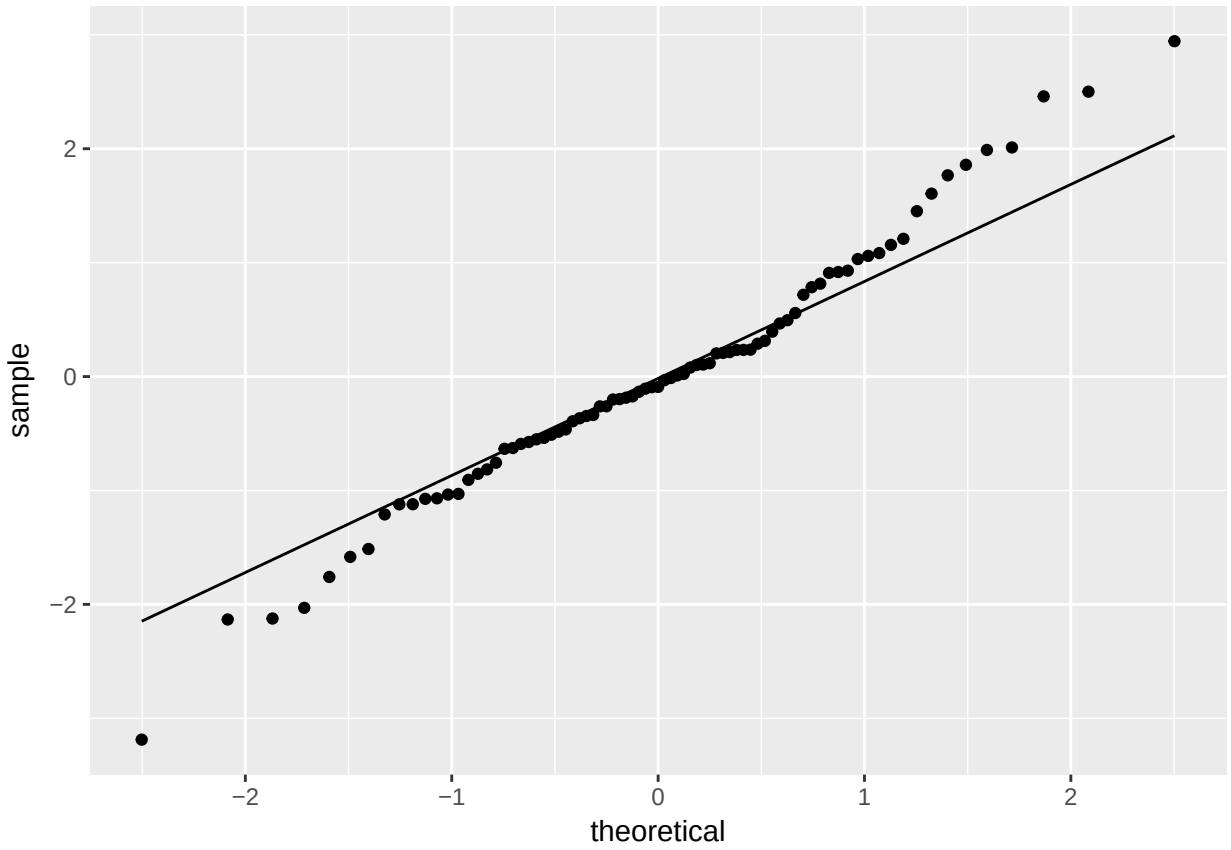
```
form1 = formula('rent.rates~.')
prop.lin.mod = lm(form1, data=properties)
knitr::kable(summary(prop.lin.mod)$coef,digits = 2)
```

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	12.20	0.58	21.11	0.00
age	-0.14	0.02	-6.65	0.00
op.expense	0.28	0.06	4.46	0.00
vac.rates	0.62	1.09	0.57	0.57
sqft	0.00	0.00	5.72	0.00

2. Make plots of the residuals against each predictor. Make a qq-plot of the residuals. Discuss what you see. Does the assumption of "normally distributed residuals" appear to be satisfied?

```
prop.lin.mod = estimate.and.plot(form1, properties) #same as before
```





The normality assumption seems a bit inaccurate especially at the tails. The residuals are generally centered around zero against each of the predictors without too much trend. Vacancy rates as a few large outliers that we should perhaps worry about, and age is either small or large without much data in the 5-10 year range.

3. Interpret the estimated coefficient on vacancy rates. Find and interpret a 90% confidence interval for $\beta_{vacancy}$. Test, with $\alpha = 0.05$, whether or not $\beta_{vacancy} = 0$. State your conclusion in the context of the problem.

The coefficient on vacancy rates is 0.62. This means that for every 1% increase in vacancy rate, predicted rental rates increase by \$0.62. (I'm assuming here that vacancy rates are in decimals not percentage and that rental rates are in units of \$100. Note that the range of vacancy rates is 0, 0.73.)

The predicted increase is fairly unexpected, but the p-value is only 0.57 and the 90% confidence interval is -1.19, 2.43. This is not a significant predictor. (It's most likely driven by the skewness of the distribution: there are a few large outliers which coincide with increased rental rates. Perhaps expensive new construction?)

4. Someone suggests including an interaction for age and vacancy rates. Add this interaction to the model reinterpret the effect of vacancy rates on rental rates.

```
prop.lin.mod.interaction = lm(rent.rates~.+age*vac.rates, data=properties)
```

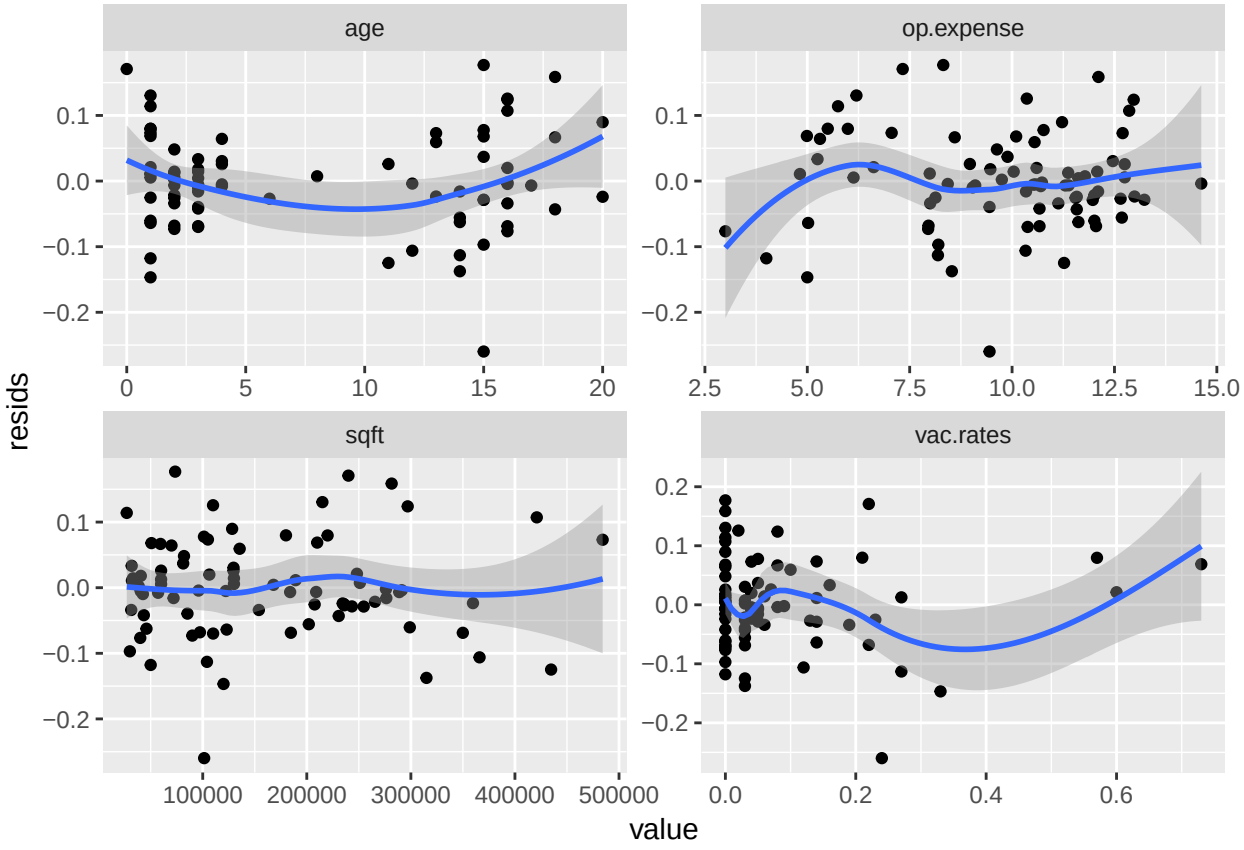
With the addition of the interaction, we a change in vacancy rates will affect predicted rental rates through two sources: it's own marginal contribution and the interacted contribution with age. Holding age constant (non-zero), therefore, we have

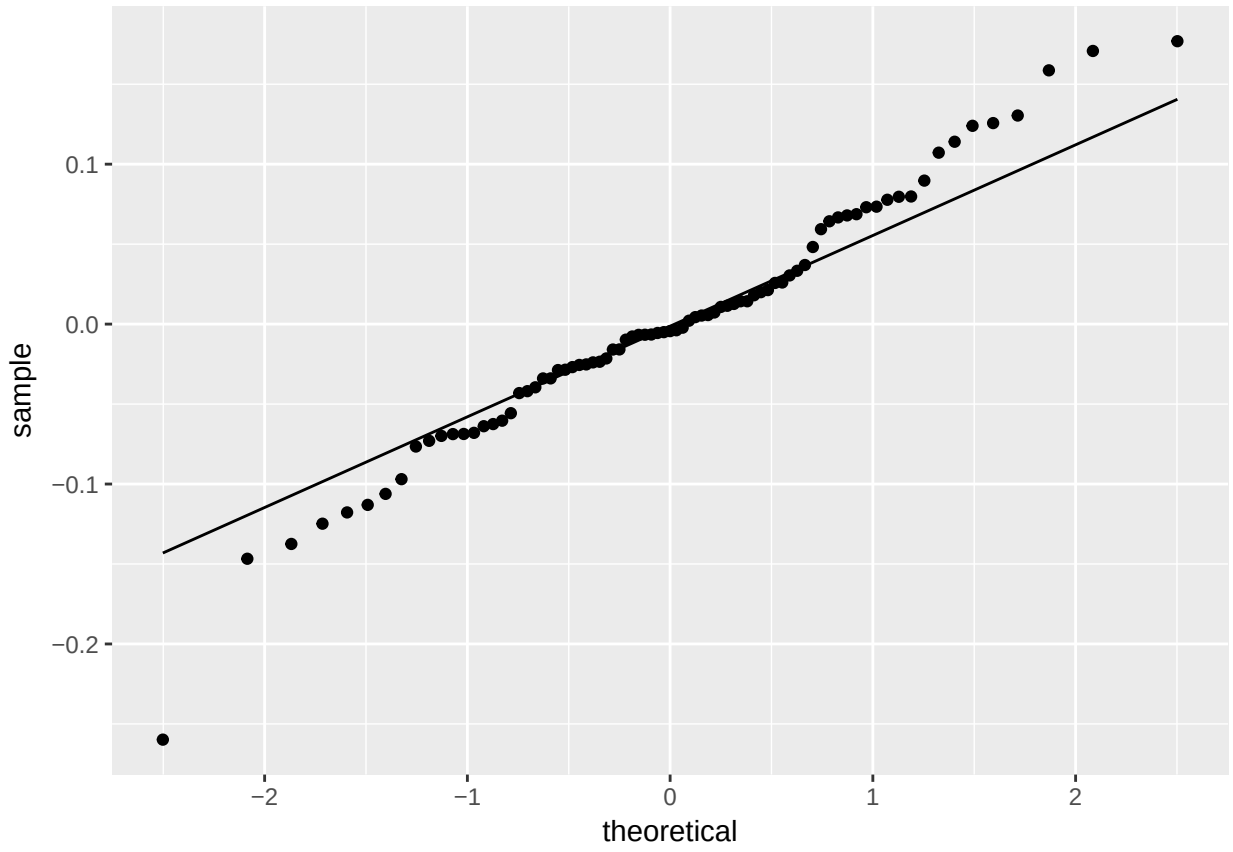
$$\Delta\widehat{\text{rent.rates}} = \widehat{\beta}_{\text{vac.rates}} + \widehat{\beta}_{\text{vac.rates}\times\text{age}}.$$

So, in this case, for every 1% increase in vacancy rate, predicted rental rates increase by \$2.76 + \$-0.75 = \$2.01

5. Someone suggests that it would be better to use the log of rental rates as the outcome. Repeat steps 1 to 3 with this change.

```
form2 = formula('log(rent.rates)~.')
prop.log.lin.mod = estimate.and.plot(form2, properties)
```






```
knitr::kable(summary(prop.log.lin.mod)$coef,digits = 2)
```

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	2.51	0.04	63.63	0.00
age	-0.01	0.00	-6.84	0.00
op.expense	0.02	0.00	4.76	0.00
vac.rates	0.04	0.07	0.56	0.58
sqft	0.00	0.00	5.40	0.00

The coefficient on vacancy rates is now 0.04. This means that for every 1% increase in vacancy rate, **log** predicted rental rates increase by 0.04. There isn't much change in the diagnostic plots. This hasn't really fixed the departure from normality.

It is not so easy to convert this to predicted rental rates because

$$\frac{\partial}{\partial x_j} \exp\left(\sum_{i=1}^{p} \widehat{\beta}_i x_i\right) = \widehat{\beta}_j \exp\left(\sum_{i=1}^{p} \widehat{\beta}_i x_i\right) \neq \exp\left(\widehat{\beta}_j\right).$$

One way to convert is to hold all the predictors constant at their average values. Thus, we examine the effect of a 1% increase in vacancy rates from its average where everything else is also average (you could use any meaningful starting point that you like). So we predict at the average, exponentiate, and then multiply by the estimated coefficient:

```
avg.pred = predict(prop.log.lin.mod, newdata = summarise_all(properties, mean))
chg.from.average = coef(prop.log.lin.mod)['vac.rates']*avg.pred
```

Now we have, that predicted rental rates increase (from average) by \$0.11 if vacancy rates increase by 1% from their average. The p-value is 0.58 and the 90% confidence interval is -0.08, 0.16. Again, this is not a significant predictor.