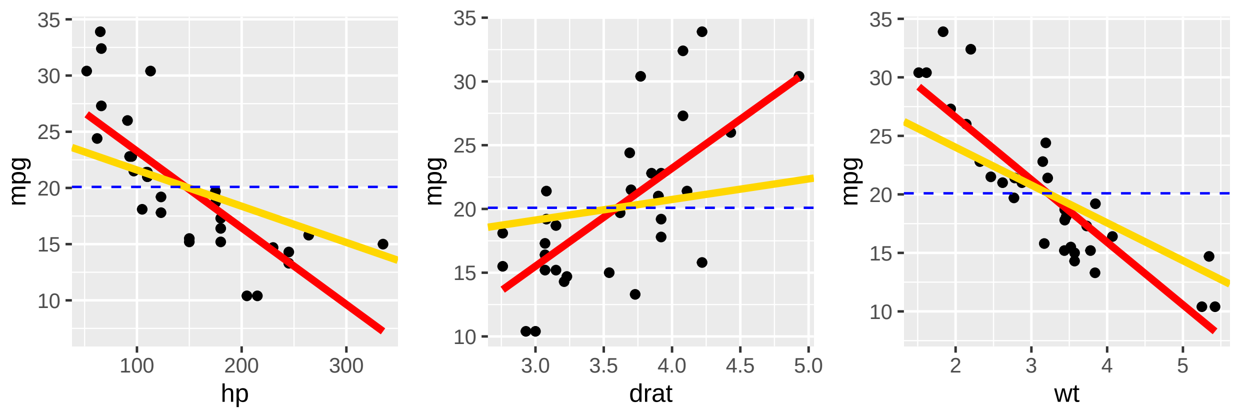
Suppose we have arrived at our final three models (OLS, JHM, GAM). How do we compare them all?

First, we should consider the plots of the individual predictors. Do each of these fits make sense? Are they interpretable? Remember, in any of these models where you have more than one predictor, the fits are with some of the variability in Y explained by other predictors. Second, we should examine all of the residual plots. Is there a pattern left over? Third, we should compare the fits with fit statistics. Between OLS and JHM with the same set of predictors, R^2 and R^2_{adj} is always guaranteed to be largest for the OLS. Additionally, `rfit` does not produce an `AIC` for a JHM model. Also, given the same subset of predictors, R^2 is guaranteed to prefer GAMs over OLS MLRs, and OLS MLRs over JHM rank-based regression fits. Importantly, GAMs are highly subject to overfitting! So how do we compare these models numerically and *fairly*?

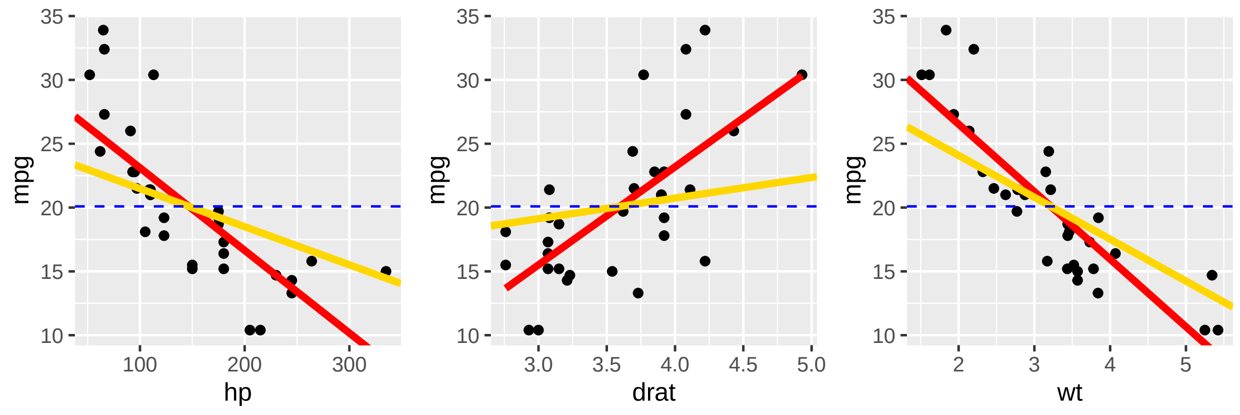
ANSWER: WE USE CROSS-VALIDATION!

Let's consider the following OLS, JHM, and GAM models, which were selected as "best" fits for models of their class. The GAM is the final model from the HW #9 solutions. The red lines are the individual SLR/Theil/Smoother fits, while the yellow lines are the MLR/JHM/GAM fits in the presence of other variables.

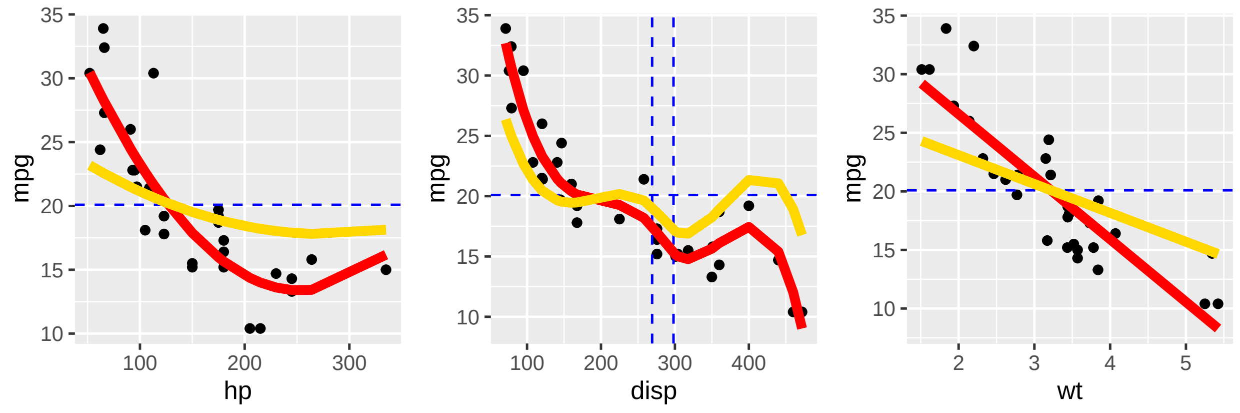
10.1 OLS MLR Fit



10.2 JHM Fit

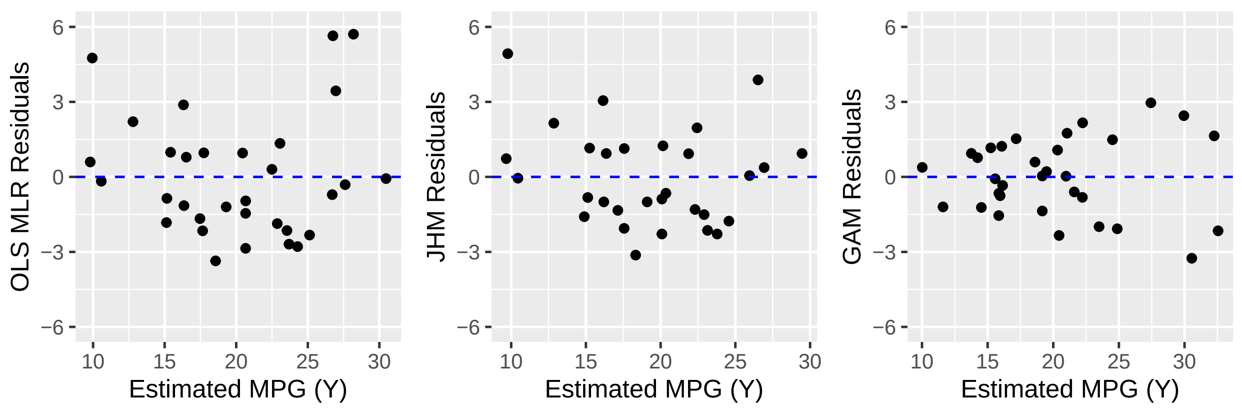


10.3 GAM Fit



10.4 Visually Assessing Model Fit

Here, we have all three residual plots on the same vertical axis (i.e, $Y_i - \hat{Y}_i \in (-6, 6)$). We can see that the linear fits (OLS and JHM) leave a u-shaped pattern in the residual plot, while the GAM leaves a residual plot with no pattern and approximately equal variance at each level of $\hat{Y}$.



10.5 Assessing Model Fit

Copy and paste the following R code chunk after your setup chunk. These functions find $R^2 = 1 - \frac{SSE}{SST}$, $R^2_{\text{adj}} = 1 - (1 - R^2)\frac{n-1}{n-df_{\text{model}}}$, and $L1_{\text{prop}} = 1 - \frac{\sum_{i=1}^{n} |Y_i - \hat{Y}_i|}{\sum_{i=1}^{n} |Y_i - \bar{Y}|}$. $L1_{\text{prop}}$ is the absolute value analogue to R^2, and can be interpreted in a similar way, i.e., "The proportion of error in Y that can be explained by the model".

Note that ordinary least squares multiple linear regression (OLS MLR) seeks to minimize SSE, and so is guaranteed to have a higher R^2 and R^2_{adj} than a Jaeckel-Hettmansperger-McKean rank-based regression (JHM) if both are modeled on the same subset of predictors. Additionally, a generalized additive model (GAM) should have at least the same R^2 as an OLS MLR if built on the same subset of predictors. Remember that a component of a GAM can always reduce to a linear fit if that is what it takes to minimize SSE.

```r
#Fit statistics for OLS
fit_ols = function(model) {
    yy = model$residuals + model$fitted.values
    rsq = 1 - sum(model$residuals^2)/sum((yy - mean(yy))^2)
    nn = length(yy)
    adjrsq = 1 - (1 - rsq)*((nn - 1)/(nn - length(model$coefficients)))
    propL1 = 1 - sum(abs(model$residuals))/sum(abs(yy - mean(yy)))
return(cbind(rsq = rsq, adjrsq = adjrsq, propL1 = propL1))
}

#Fit statistics for JHM
fit_jhm = function(model) {
    rsq = 1 - sum(model$residuals^2)/sum((model$y - mean(model$y))^2)
    nn = length(model$y)
    adjrsq = 1 - (1 - rsq)*((nn - 1)/(nn - length(model$coefficients)))
    propL1 = 1 - sum(abs(model$residuals))/sum(abs(model$y - mean(model$y)))
return(cbind(rsq = rsq, adjrsq = adjrsq, propL1 = propL1))
}

#Fit statistics for GAM
fit_gam = function(model) {
    rsq = 1 - model$deviance/model$null.deviance
    adjrsq = 1 - (1 - rsq)*(model$df.null/model$df.residual)
    propL1 = 1 - sum(abs(model$residuals))/sum(abs(model$y - mean(model$y)))
return(cbind(rsq = rsq, adjrsq = adjrsq, propL1 = propL1))
}
```

10.6 *k*-fold Cross-Validation

Copy and paste the following R code chunk after your setup chunk. ALL of your `library` and `require` calls should be at the top of your .RMD in the `setup` chunk.

```r
#General fit statistics
fit_gen = function(y, res, df){
    rsq = 1 - sum(res^2)/sum((y - mean(y))^2)
    nn = length(y)
    adjrsq = 1 - (1 - rsq)*((nn - 1)/(nn - df))
    propL1 = 1 - sum(abs(res))/sum(abs(y - mean(y)))
return(cbind(rsq = rsq, adjrsq = adjrsq, propL1 = propL1))
}

#My cross-validation function for this project
cv_rmc = function(dat, ols_mod, jhm_mod, gam_mod, k = 5, m = 10){
#(Some) error checking
if(class(ols_mod) != "lm") stop('ols_mod should come from the lm() function')
if(class(jhm_mod) != "rfit") stop('jhm_mod should come from the rfit() function')
if(class(gam_mod)[1] != "Gam") stop('gam_mod should come from the gam() function')

#Create model call character strings with subsetted data; uses stringr f()s
    ols_call = paste0("lm(formula = ", ols_mod$terms[[2]], " ~ ",
       capture.output(ols_mod$terms[[3]]), ", data = ", "dat[-part[[i]], ])")
    jhm_call = capture.output(jhm_mod$call)
    dat.name = paste0("data = ", deparse(substitute(dat)))
    jhm_call = str_replace(jhm_call, dat.name, "data = dat[-part[[i]], ]")
    gam_call = paste(str_trim(capture.output(gam_mod$call)), sep="", collapse="")
    gam_call = str_replace(gam_call, dat.name, "data = dat[-part[[i]], ]")

#Set up objects
    ols_fit = matrix(nrow = m, ncol = 3)
    jhm_fit = ols_fit; gam_fit = ols_fit
    yy = jhm_mod$y
    nn = dim(as.data.frame(dat))[1]
    oos_lmres = vector(length = nn)
    oos_jhres = oos_lmres; oos_gares = oos_lmres
    df_ols = length(ols_mod$coefficients)
    df_jhm = length(jhm_mod$coefficients)
    df_gam = nn - gam_mod$df.residual

#Repeat k-fold cross-validation m times
    for(j in 1:m) {
#Split data into k equal-ish parts, with random indices
        part = split(sample(nn), 1:k)
#Execute model calls for all k folds; %*% is matrix multiplication
        for(i in 1:k){
            lm_mod = eval(parse(text = ols_call))
            pred = predict(object = lm_mod, newdata = dat[part[[i]],])
            oos_lmres[part[[i]]] = yy[part[[i]]] - pred

            jh_mod = eval(parse(text = jhm_call))
            subdat = select(.data = dat, colnames(jh_mod$x)[-1])[part[[i]],]
            subdat = cbind(1, as.matrix.data.frame(subdat))
            pred = subdat %*% jh_mod$coefficients
            oos_jhres[part[[i]]] = yy[part[[i]]] - pred

            ga_mod = eval(parse(text = gam_call))
            pred = predict(object = ga_mod, newdata = dat[part[[i]],])
            oos_gares[part[[i]]] = yy[part[[i]]] - pred
        }
        ols_fit[j, ] = fit_gen(y = yy, res = oos_lmres, df = df_ols)
        jhm_fit[j, ] = fit_gen(y = yy, res = oos_jhres, df = df_jhm)
        gam_fit[j, ] = fit_gen(y = yy, res = oos_gares, df = df_gam)
    }

#Manage output -- average fit statistics
    outtie = rbind(colMeans(ols_fit), colMeans(jhm_fit), colMeans(gam_fit))
    colnames(outtie) = paste0("cv.", colnames(fit_ols(lm_mod)))
    row.names(outtie) = c("OLS", "JHM", "GAM")
return(outtie)
}
```

10.7 Example usage of fit statistics

The following are the fit statistics for the model, as you fit it (i.e., on the entire dataset).

```r
fit_final = rbind(fit_ols(ls2), fit_jhm(jhm3), fit_gam(gam4))
rownames(fit_final) = c("OLS", "JHM", "GAM")
kable(round(fit_final, 4)) %>% kable_styling(position = "center")
```

	rsq	adjrsq	propL1
OLS	0.8369	0.8194	0.5947
JHM	0.8307	0.8126	0.6019
GAM	0.9342	0.9113	0.7296

Note that these values of $R^2, R^2_{\text{adj}},$ and $L1_{\text{prop}}$ are fairly large, and by each measure, GAM $>$ OLS $>$ JHM. This makes sense as there are nonlinear patterns and there are no outliers that unduly affect the OLS fit.

10.8 Example usage of k-fold cross-validation

Here is an example call of the k-fold cross-validation function I wrote, using the models I fit in the first three sections. Note that each of your models needed to be built with the same dataset (i.e., the complete dataset, but with no missing values for any observation). Additionally, the function presumes you used the `lm`, `rfit`, and `gam` functions to fit your model. If you did not, you may need to re-run your models using the preceding functions. If you find that is not possible, you will want to rewrite the `cv_rmc` function to meet your needs.

Additionally, `cv_rmc` takes `k` and `m` parameters. `k` is the number of parts ("folds") you are breaking your dataset into for cross-validation, and `m` is the number of times you are repeating cross-validation. `k = 5` and `m = 10` are the defaults, and I do not recommend fiddling with them.

Remember that in k-fold cross-validation, we remove (hold out) approximately $\frac{n}{k}$ randomly selected observations, fit the models as we did on n observations but with the remaining $\frac{n-k}{k}$ observations, and then use the resulting model to predict Y for the holdout observations. We repeat this process for each of the remaining $k-1$ holdout groups (folds), which are randomly selected without replacement so that no observations appear in multiple folds. Usually, this is repeated some number of times (m). We can then compare the errors aggregated in some way. I have chosen to calculate the same fit statistics we did in **10.7** for each iteration of k-fold cross-validation, and then find the average of those fit statistics over the m iterations.

```r
out10 = cv_rmc(dat = mt_dat, ols_mod = ls2, jhm_mod = jhm3, gam_mod = gam4)
kable(round(out10,4)) %>% kable_styling(position = "center")
```

	cv.rsq	cv.adjrsq	cv.propL1
OLS	0.7646	0.7394	0.5148
JHM	0.7556	0.7294	0.5115
GAM	0.8109	0.7451	0.5592

Here, we see that the cross-validated fit statistics are much lower. This is due in part to having fewer observations to fit with, but also, these models were likely overfit (especially the GAM – see the plot for `disp`). Still, the GAM that I fit at the end of HW #9 (in GAM2), which is the GAM represented here, performs best across all fit statistics.