Homework 3

Jeremy Hubinger, and Tamur Asar

Project Work

Instructions

Goal: Begin an analysis of your dataset to answer your **regression** research question.

Collaboration: Form a team (2-3 members) for the project and this part can be done as a team. Only one team member should submit a Project Work section. Make sure you include the full names of all of the members in your write up.

Data cleaning: If your dataset requires any cleaning (e.g., merging datasets, creation of new variables), first consult the R Resources page (r-resources.html) to see if your questions are answered there. If not, post on the #rcode-questions channel in our Slack workspace to ask for help. *Please ask for help early and regularly* to avoid stressful workloads.

Required Analyses

1. **Initial investigation: ignoring nonlinearity (for now)**
    a. Use ordinary least squares (OLS) by using the `lm` engine and LASSO (`glmnet` engine) to build a series of initial regression models for your quantitative outcome as a function of the predictors of interest. (As part of data cleaning, exclude any variables that you don't want to consider as predictors.)
        - You'll need two model specifications, `lm_spec` and `lm_lasso_spec` (you'll need to tune this one).
    b. For each set of variables, you'll need a `recipe` with the `formula` , `data` , and pre-processing steps
        - You may want to have steps in your recipe that remove variables with near zero variance (`step_nzv()`), remove variables that are highly correlated with other variables (`step_corr()`), normalize all quantitative predictors (`step_normalize(all_numeric_predictors())`) and add indicator variables for any categorical variables (`step_dummy(all_nominal_predictors())`).
        - These models should not include any transformations to deal with nonlinearity. You'll explore this in the next investigation.
    c. Estimate the test performance of the models using CV. Report and interpret (with units) the CV metric estimates along with a measure of uncertainty in the estimate (`std_error` is readily

available when you used `collect_metrics(summarize=TRUE)`).
    - Compare estimated test performance across the models. Which models(s) might you prefer?
d. Use residual plots to evaluate whether some quantitative predictors might be better modeled with nonlinear relationships.
e. Which variables do you think are the most important predictors of your quantitative outcome? Justify your answer. Do the methods you've applied reach consensus on which variables are most important? What insights are expected? Surprising?
    - Note that if some (but not all) of the indicator terms for a categorical predictor are selected in the final models, the whole predictor should be treated as selected.

Your Work

a & b.

```
# library statements
# read in data
library(readr)
library(ggplot2)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(tidymodels)
```

```
## Registered S3 method overwritten by 'tune':
##   method                    from
##   required_pkgs.model_spec parsnip
```

```
## ── Attaching packages ──────────────────────────────────── tidymodels 0.1.3 ──
```

```
## ✓ broom        0.7.9      ✓ rsample      0.1.0
## ✓ dials        0.0.9      ✓ tibble       3.1.4
## ✓ infer        1.0.0      ✓ tidyr        1.1.3
## ✓ modeldata    0.1.1      ✓ tune         0.1.6
## ✓ parsnip      0.1.7      ✓ workflows    0.2.3
## ✓ purrr        0.3.4      ✓ workflowsets 0.1.0
## ✓ recipes      0.1.16     ✓ yardstick    0.0.8
```

```
## ── Conflicts ─────────────────────────────────────── tidymodels_conflicts() ──
## x purrr::discard()  masks scales::discard()
## x dplyr::filter()   masks stats::filter()
## x dplyr::lag()      masks stats::lag()
## x yardstick::spec() masks readr::spec()
## x recipes::step()   masks stats::step()
## • Use tidymodels_prefer() to resolve common conflicts.
```

```
tidymodels_prefer()

fires <- read_csv("forestfires.csv")
```

```
## Rows: 517 Columns: 13
```

```
## ── Column specification ────────────────────────────────────────────────────
## Delimiter: ","
## chr  (2): month, day
## dbl (11): X, Y, FFMC, DMC, DC, ISI, temp, RH, wind, rain, area
```

```
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

The dataset has a variety of variables that might impact the strength and likelihood of fires. These variables include mainly things about weather, like temperature, pressure, wind, rain, ect. The variable we are trying to predict is “area” which is the area burned by fires on that day.

```
# data cleaning
fires <- fires %>%
    # we get rid of day because there isnt a huge reason that it should impact the fi
res, we are more interested in how weather factors impact fires, not day of the week
    select(-day) %>%
   filter(area > 0) %>%
    mutate(area = log(area+.1))
```

```
# creation of cv folds
set.seed(88)
fires_cv10 <- vfold_cv(fires, v = 10)
```

We now fit multiple types of models. One model is fit using simple LM regression with all predictors. The second model is fit using LASSO regression to attempt to provide some amount of filtering to the model predictors. Both models are fit and compared later on.

```
# model spec
lm_spec <-
    linear_reg() %>%
    set_engine(engine = 'lm') %>%
    set_mode('regression')

# this is the LASSO model
lm_lasso_spec <-
  linear_reg() %>%
  set_args(mixture = 1, penalty = tune()) %>%
  set_engine(engine = 'glmnet') %>%
  set_mode('regression')
```

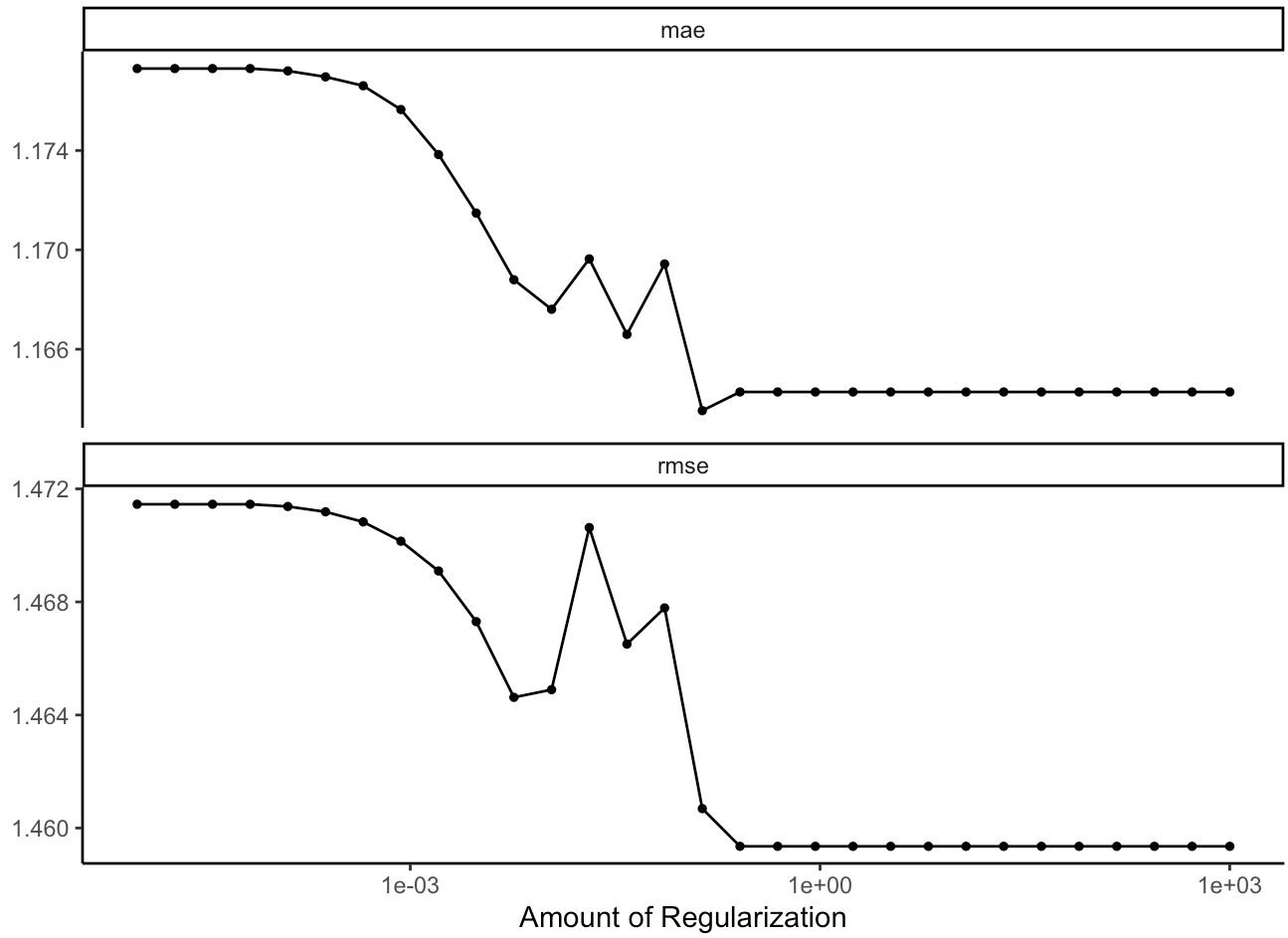
```
# recipes & workflows
# both models are fit with the same recipe because they both start with all predictor
s and all predictors are normalized
full_rec <- recipe(area ~ ., data = fires) %>%
    step_nzv(all_predictors()) %>% # removes variables with the same value
    step_normalize(all_numeric_predictors()) %>% # important standardization step for
LASSO
    step_dummy(all_nominal_predictors())  # creates indicator variables for categoric
al variables

full_lm_wf <- workflow() %>%
    add_recipe(full_rec) %>%
    add_model(lm_spec)

lasso_wf_tune <- workflow() %>%
  add_recipe(full_rec) %>%
  add_model(lm_lasso_spec)
```

```
# fit & tune models
full_lm_wf <- workflow() %>%
    add_recipe(full_rec) %>%
    add_model(lm_spec)

full_model <- fit(full_lm_wf, data = fires)
```

```
# Tune Model (trying a variety of values of Lambda penalty)
penalty_grid <- grid_regular(
  penalty(range = c(-5, 3)), #log10 transformed
  levels = 30)

tune_output <- tune_grid( # new function for tuning parameters
  lasso_wf_tune, # workflow
  resamples = fires_cv10, # cv folds
  metrics = metric_set(rmse, mae),
  grid = penalty_grid # penalty grid defined above
)
```

```
## ! Fold07: preprocessor 1/1, model 1/1 (predictions): There are new levels in a fa
c...
```

```
autoplot(tune_output) + theme_classic()
```




```
collect_metrics(tune_output) %>%
  filter(.metric == 'rmse') %>% # or choose mae
  select(penalty, rmse = mean)
```

```
## # A tibble: 30 × 2
##      penalty  rmse
##        <dbl> <dbl>
##  1 0.00001    1.47
##  2 0.0000189  1.47
##  3 0.0000356  1.47
##  4 0.0000672  1.47
##  5 0.000127   1.47
##  6 0.000240   1.47
##  7 0.000452   1.47
##  8 0.000853   1.47
##  9 0.00161    1.47
## 10 0.00304    1.47
## # … with 20 more rows
```

```
 best_penalty <- select_best(tune_output, metric = 'rmse')

best_penalty
```

```
## # A tibble: 1 × 2
##   penalty .config
##     <dbl> <chr>
## 1   0.259 Preprocessor1_Model17
```

```
final_wf <- finalize_workflow(lasso_wf_tune, best_penalty)

final_fit <- fit(final_wf, data = fires)

tidy(final_fit)
```

```
## # A tibble: 19 × 3
##    term        estimate penalty
##    <chr>          <dbl>   <dbl>
##  1 (Intercept)     1.89   0.259
##  2 X               0      0.259
##  3 Y               0      0.259
##  4 FFMC            0      0.259
##  5 DMC             0      0.259
##  6 DC              0      0.259
##  7 ISI             0      0.259
##  8 temp            0      0.259
##  9 RH              0      0.259
## 10 wind            0      0.259
## 11 month_aug       0      0.259
## 12 month_dec       0      0.259
## 13 month_feb       0      0.259
## 14 month_jul       0      0.259
## 15 month_jun       0      0.259
## 16 month_mar       0      0.259
## 17 month_may       0      0.259
## 18 month_oct       0      0.259
## 19 month_sep       0      0.259
```

It seems as though all variables were set to 0 with LASSO. This is very interesting because it indicates that none of the variables make a significant effect in the predictive power of the model.

c.

```
#  calculate/collect CV metrics

full_model %>% tidy()
```

```
## # A tibble: 19 × 5
##    term        estimate std.error statistic  p.value
##    <chr>          <dbl>     <dbl>     <dbl>    <dbl>
##  1 (Intercept)   0.694      1.20      0.580 0.562
##  2 X             0.134      0.108     1.24  0.215
##  3 Y            -0.126      0.107    -1.18  0.239
##  4 FFMC          0.0341     0.152     0.224 0.823
##  5 DMC           0.600      0.168     3.58  0.000416
##  6 DC           -1.25       0.457    -2.73  0.00677
##  7 ISI          -0.150      0.145    -1.04  0.300
##  8 temp          0.303      0.195     1.55  0.122
##  9 RH            0.0273     0.136     0.201 0.841
## 10 wind          0.0971     0.105     0.928 0.354
## 11 month_aug     0.894      1.30      0.687 0.493
## 12 month_dec     1.95       1.10      1.77  0.0778
## 13 month_feb    -0.252      0.867    -0.291 0.772
## 14 month_jul     0.124      1.11      0.112 0.911
## 15 month_jun    -0.514      1.05     -0.490 0.624
## 16 month_mar    -0.399      0.812    -0.491 0.624
## 17 month_may     0.991      1.64      0.603 0.547
## 18 month_oct     3.29       1.58      2.08  0.0384
## 19 month_sep     2.17       1.46      1.48  0.139
```

```
best_penalty <- select_best(tune_output, metric = 'rmse')

final_wf <- finalize_workflow(lasso_wf_tune, best_penalty)

final_fit <- fit(final_wf, data = fires)

tidy(final_fit)
```

```
## # A tibble: 19 × 3
##    term        estimate penalty
##    <chr>          <dbl>   <dbl>
##  1 (Intercept)     1.89   0.259
##  2 X               0      0.259
##  3 Y               0      0.259
##  4 FFMC            0      0.259
##  5 DMC             0      0.259
##  6 DC              0      0.259
##  7 ISI             0      0.259
##  8 temp            0      0.259
##  9 RH              0      0.259
## 10 wind            0      0.259
## 11 month_aug       0      0.259
## 12 month_dec       0      0.259
## 13 month_feb       0      0.259
## 14 month_jul       0      0.259
## 15 month_jun       0      0.259
## 16 month_mar       0      0.259
## 17 month_may       0      0.259
## 18 month_oct       0      0.259
## 19 month_sep       0      0.259
```

Based on the above information, using the LASSO model to tell what variables are the most important (which is none). This is much more readable than the LM model which has a lot of importance on a lot of different variables which don't seem to matter, and actually hurt the model's performance on test data.

```
LASSO_model_cv <- fit_resamples(final_fit,
  resamples = fires_cv10,
  metrics = metric_set(rmse, rsq, mae)
)
```

```
## ! Fold01: internal: A correlation computation is required, but `estimate` is cons
t...
```

```
## ! Fold02: internal: A correlation computation is required, but `estimate` is cons
t...
```

```
## ! Fold03: internal: A correlation computation is required, but `estimate` is cons
t...
```

```
## ! Fold04: internal: A correlation computation is required, but `estimate` is cons
t...
```

```
## ! Fold05: internal: A correlation computation is required, but `estimate` is cons
t...
```

```
## ! Fold06: internal: A correlation computation is required, but `estimate` is cons
t...
```

```
## ! Fold07: preprocessor 1/1, model 1/1 (predictions): There are new levels in a fa
c...
```

```
## ! Fold07: internal: A correlation computation is required, but `estimate` is cons
t...
```

```
## ! Fold08: internal: A correlation computation is required, but `estimate` is cons
t...
```

```
## ! Fold09: internal: A correlation computation is required, but `estimate` is cons
t...
```

```
## ! Fold10: internal: A correlation computation is required, but `estimate` is cons
t...
```

```
LASSO_model_cv %>% collect_metrics(summarize=TRUE)
```

```
## # A tibble: 3 × 6
##   .metric .estimator   mean     n std_err .config
##   <chr>   <chr>       <dbl> <int>   <dbl> <chr>
## 1 mae     standard     1.16    10  0.0697 Preprocessor1_Model1
## 2 rmse    standard     1.46    10  0.0802 Preprocessor1_Model1
## 3 rsq     standard   NaN        0 NA      Preprocessor1_Model1
```

```
LM_model_cv <- fit_resamples(full_model,
  resamples = fires_cv10,
  metrics = metric_set(rmse, rsq, mae)
)
```

```
## ! Fold07: preprocessor 1/1, model 1/1 (predictions): There are new levels in a fa
c...
```

```
LM_model_cv %>% collect_metrics(summarize=TRUE)
```

```
## # A tibble: 3 × 6
##   .metric .estimator   mean     n std_err .config
##   <chr>   <chr>       <dbl> <int>   <dbl> <chr>
## 1 mae     standard   1.18      10  0.0634 Preprocessor1_Model1
## 2 rmse    standard   1.47      10  0.0699 Preprocessor1_Model1
## 3 rsq     standard   0.0441    10  0.0124 Preprocessor1_Model1
```

As we can see in the metrics above, the normal LM model with all the variables performs worse than the LASSO model which is much simpler. Because of the better performance (likely due to overfitting in the LM model's case) and simpler model coefficients we prefer the LASSO model at this point, but better models are yet to come.

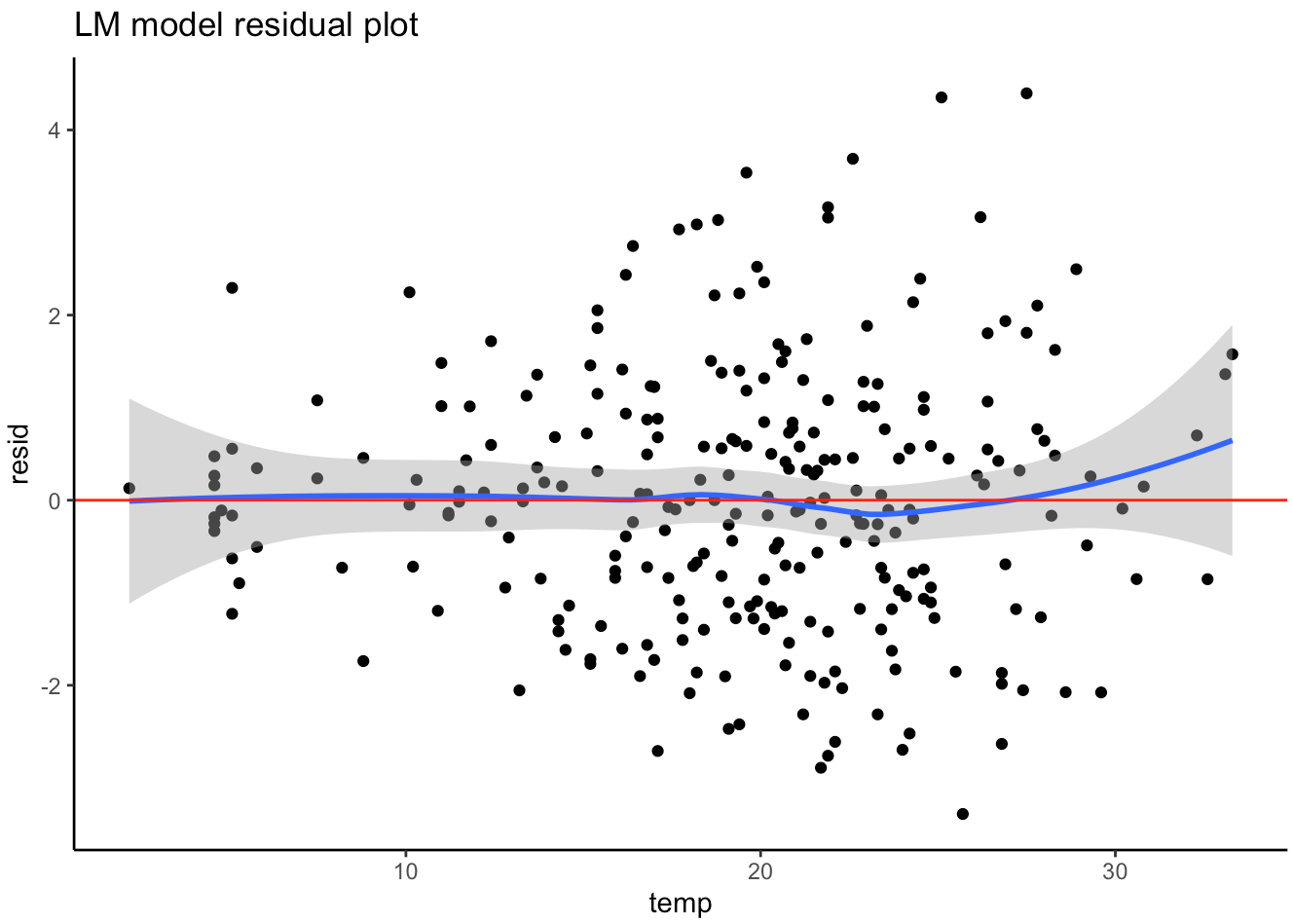
d.

```
# visual residuals
final_fit %>% tidy() %>% filter(estimate != 0)
```

```
## # A tibble: 1 × 3
##   term        estimate penalty
##   <chr>          <dbl>   <dbl>
## 1 (Intercept)     1.89   0.259
```

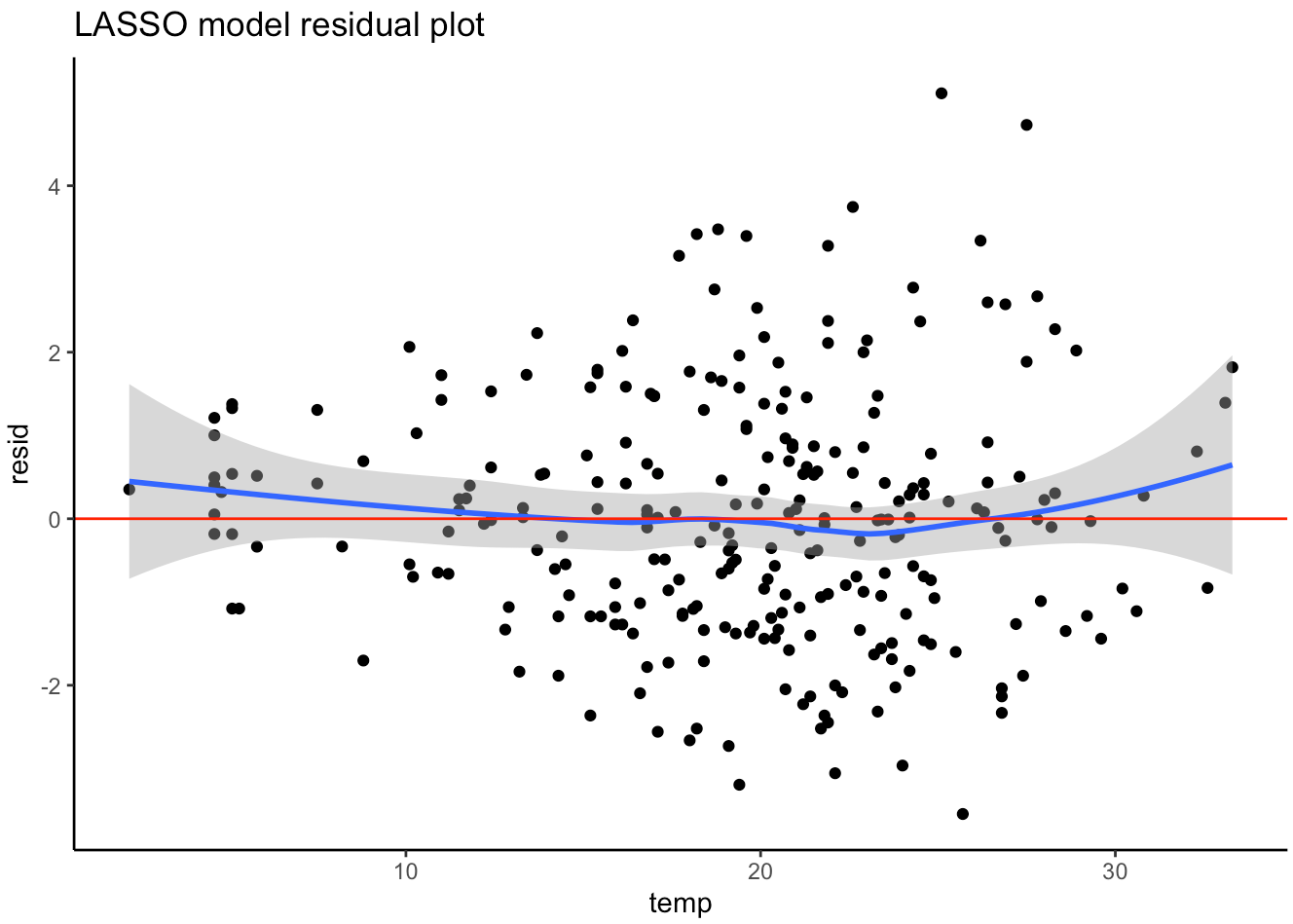
```
lasso_mod_out <- final_fit %>%
    predict(new_data = fires) %>%
    bind_cols(fires) %>%
    mutate(resid = area - .pred)

lm_mod_out <- full_model %>%
    predict(new_data = fires) %>%
    bind_cols(fires) %>%
    mutate(resid = area - .pred)

ggplot(lm_mod_out, aes(x = temp, y = resid)) +
    geom_point() +
    geom_smooth() +
    geom_hline(yintercept = 0, color = "red") +
    theme_classic() +
    ggtitle("LM model residual plot")
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```




```
ggplot(lasso_mod_out, aes(x = temp, y = resid)) +
    geom_point() +
    geom_smooth() +
    geom_hline(yintercept = 0, color = "red") +
    theme_classic() +
    ggtitle("LASSO model residual plot")
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



e.

As mentioned earlier, the variables don't seem to be impactful on the performance of the model, Which is interesting because it means that the predictors are fairly useless when trying to predict fires, this just goes to show the unpredictive nature of wildfires. We can also see that in the residual plots above, given that the LM model and the LASSO model (which has all coefficients set to 0) because the residual plots don't look very different.

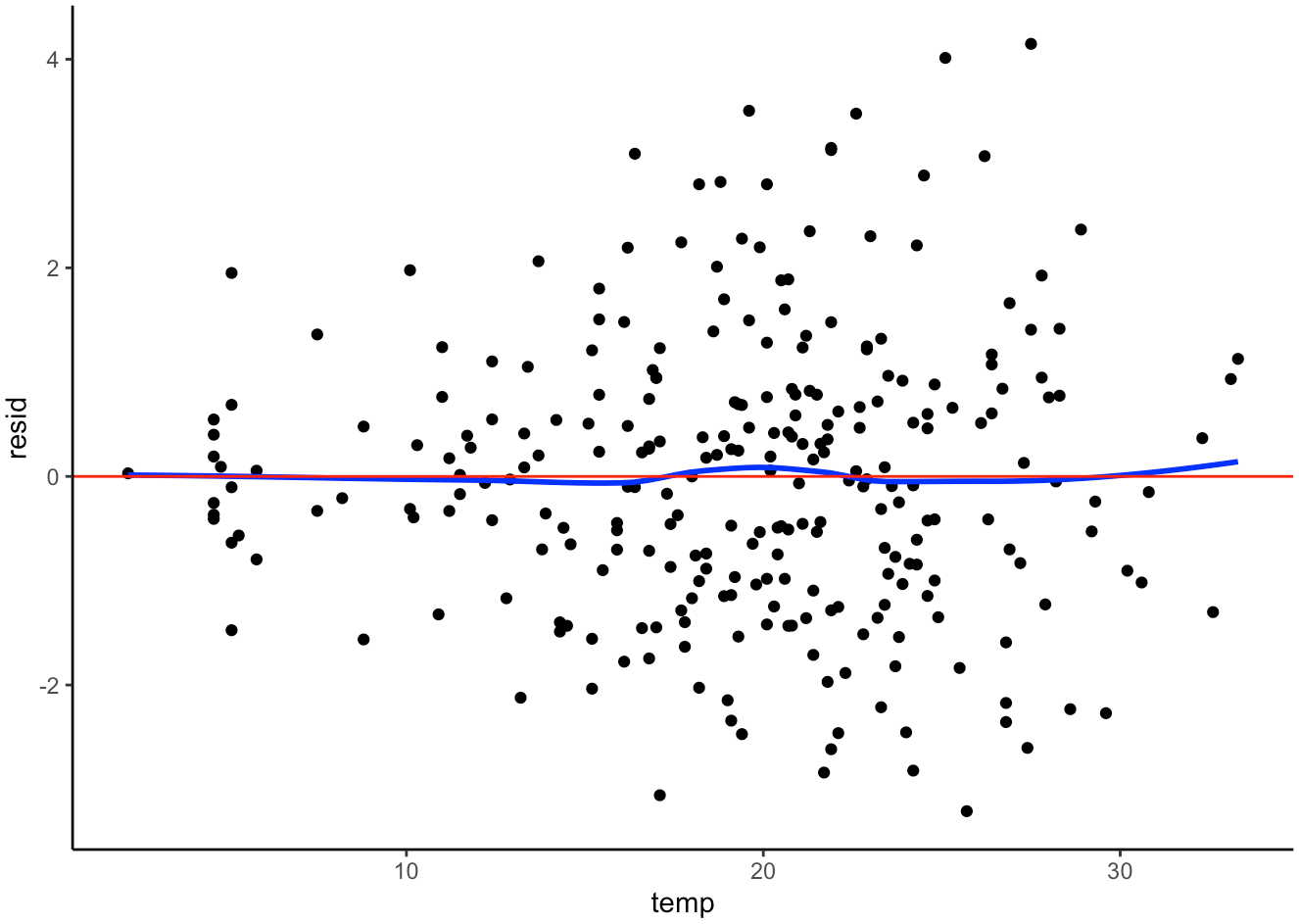
Part 2:

```
# Natural Spline Recipe
ns2_rec <- full_rec %>%
  step_ns(temp, deg_free = 3) %>%
  step_ns(RH, deg_free = 3) %>%
  step_ns(DMC, deg_free = 3) %>%
  step_ns(FFMC, deg_free = 3) %>%
  step_ns(DC, deg_free = 3) %>%
  step_ns(ISI, deg_free = 3) %>%
  step_ns(wind, deg_free = 3) # natural cubic spline (higher deg_free means more knot
s)

# Workflow (Recipe + Model)
wf <- workflow() %>%
    add_recipe(ns2_rec) %>%
    add_model(lm_spec)

# CV to Evaluate
cv_output <- fit_resamples(
  wf, # workflow
  resamples = fires_cv10, # cv folds
  metrics = metric_set(rsq, mae, rmse)
)
```

```
## ! Fold07: preprocessor 1/1, model 1/1 (predictions): There are new levels in a fa
c...
```

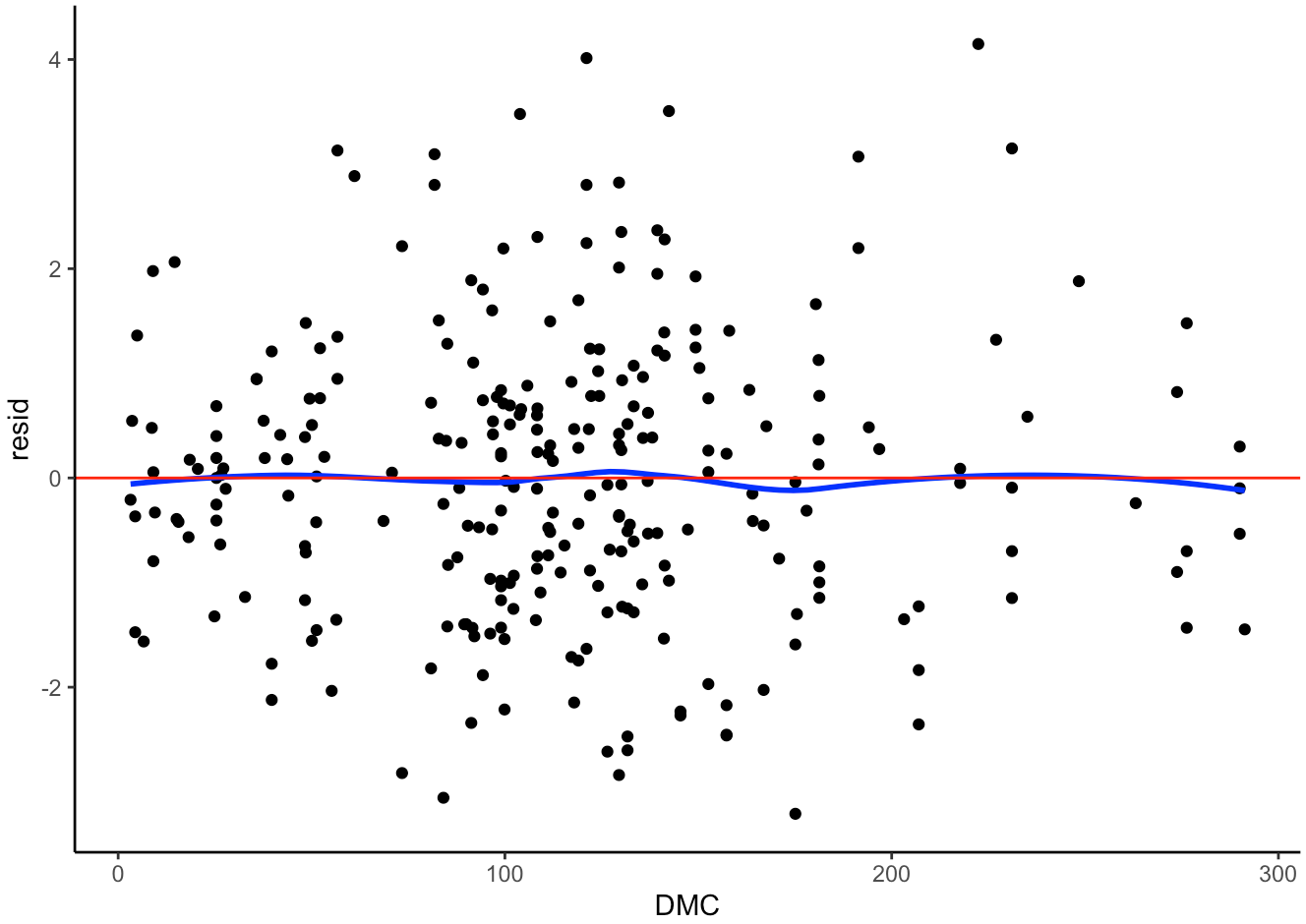
```
# Fit with all data
ns_mod <- fit(
  wf, #workflow
  data = fires
)

ns_mod_output <- fires %>%
  bind_cols(predict(ns_mod, new_data = fires)) %>%
    mutate(resid = area - .pred)

ggplot(ns_mod_output, aes(y=resid,x=temp)) +
    geom_point() +
    geom_smooth(color = "blue", se = FALSE) +
    geom_hline(yintercept = 0, color = "red") +
    theme_classic()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



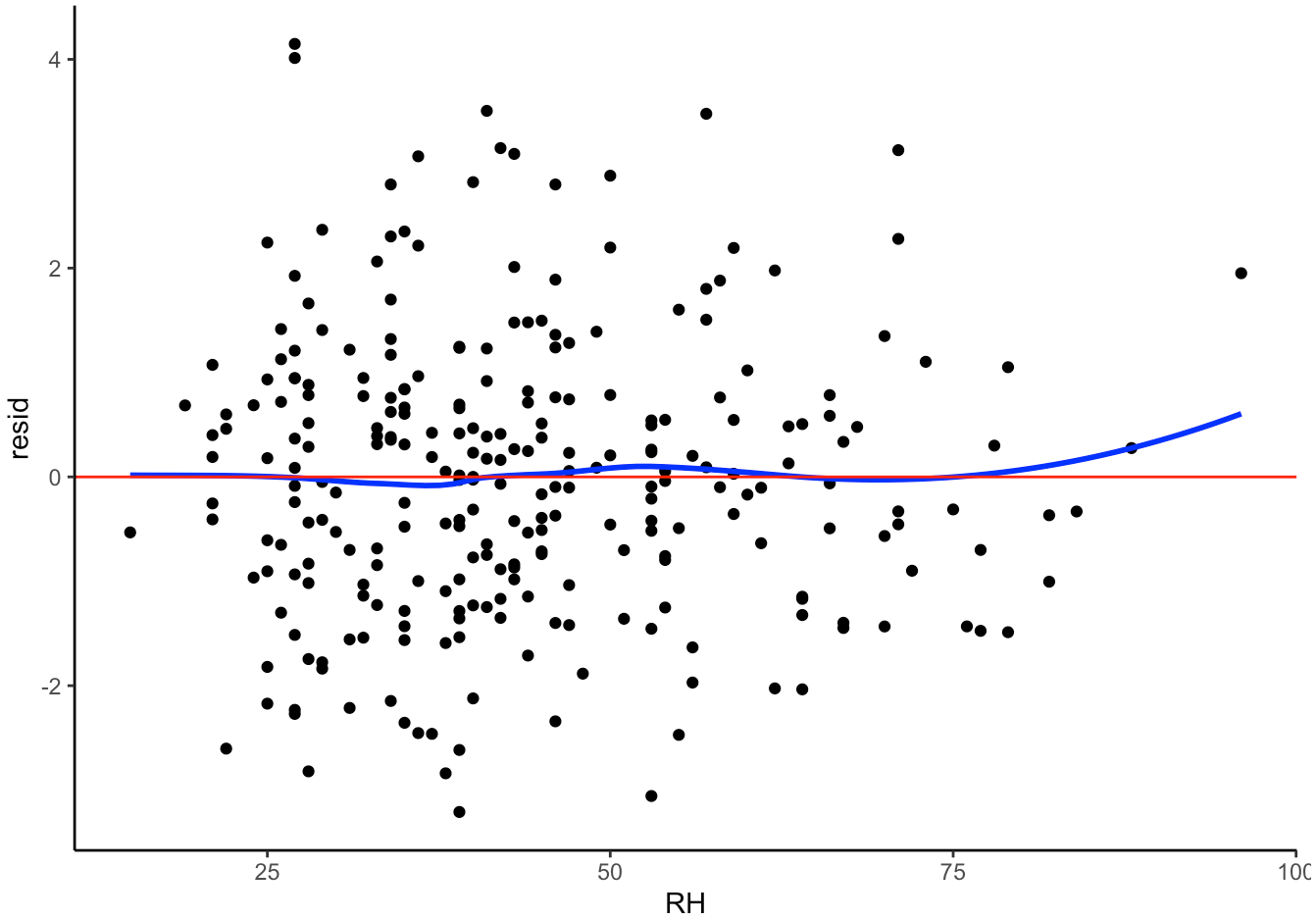

```
ggplot(ns_mod_output, aes(y=resid,x=DMC)) +
    geom_point() +
    geom_smooth(color = "blue", se = FALSE) +
    geom_hline(yintercept = 0, color = "red") +
    theme_classic()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



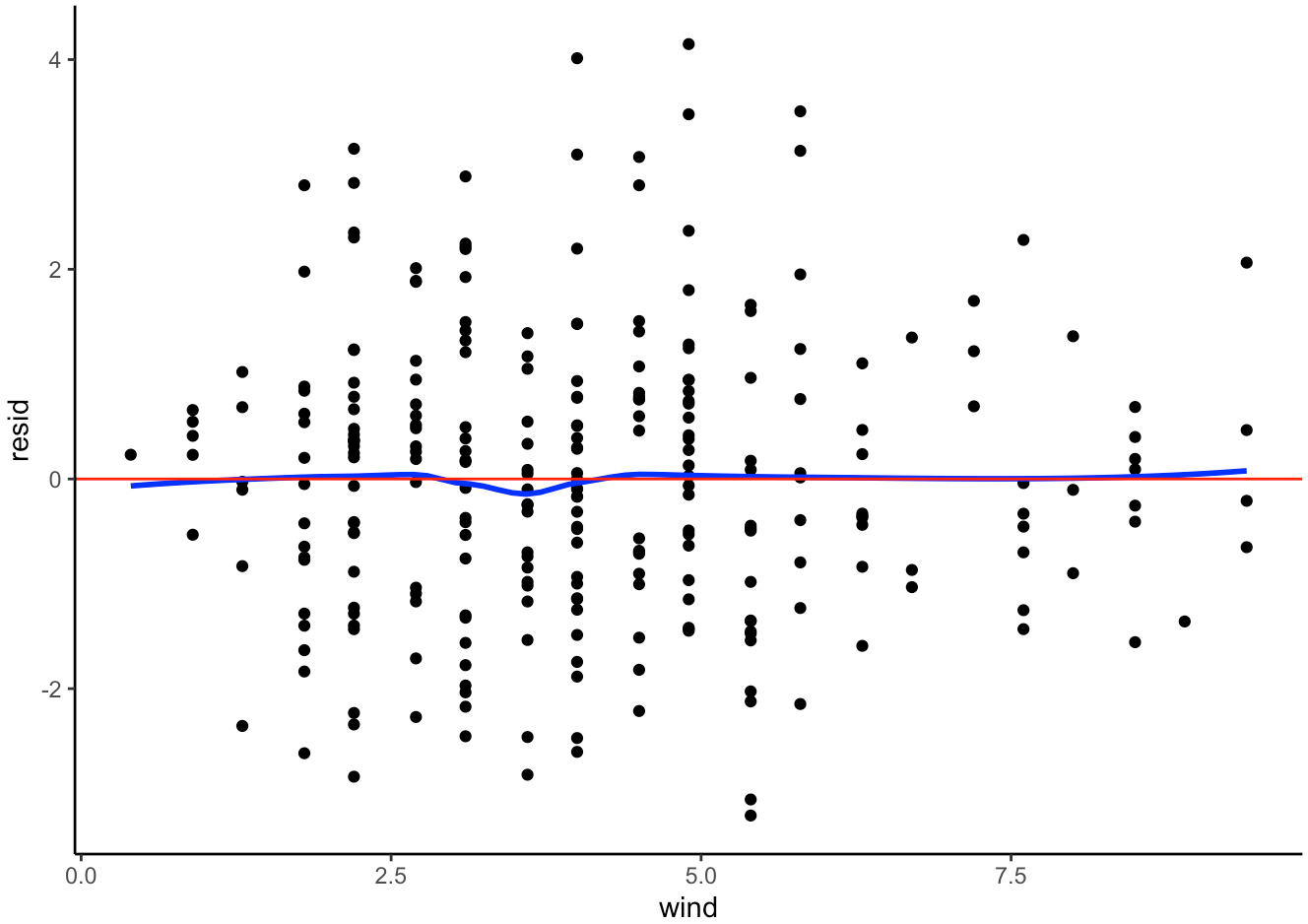

```
ggplot(ns_mod_output, aes(y=resid,x=RH)) +
    geom_point() +
    geom_smooth(color = "blue", se = FALSE) +
    geom_hline(yintercept = 0, color = "red") +
    theme_classic()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```




```
ggplot(ns_mod_output, aes(y=resid,x=wind)) +
    geom_point() +
    geom_smooth(color = "blue", se = FALSE) +
    geom_hline(yintercept = 0, color = "red") +
    theme_classic()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```




```
cv_output %>% collect_metrics(summarize = TRUE) # splines
```

```
## # A tibble: 3 × 6
##   .metric .estimator   mean     n std_err .config
##   <chr>   <chr>       <dbl> <int>   <dbl> <chr>
## 1 mae     standard   1.22      10  0.0663 Preprocessor1_Model1
## 2 rmse    standard   1.53      10  0.0802 Preprocessor1_Model1
## 3 rsq     standard   0.0340    10  0.0117 Preprocessor1_Model1
```

```
LM_model_cv %>% collect_metrics(summarize=TRUE)
```

```
## # A tibble: 3 × 6
##   .metric .estimator   mean     n std_err .config
##   <chr>   <chr>       <dbl> <int>   <dbl> <chr>
## 1 mae     standard   1.18      10  0.0634 Preprocessor1_Model1
## 2 rmse    standard   1.47      10  0.0699 Preprocessor1_Model1
## 3 rsq     standard   0.0441    10  0.0124 Preprocessor1_Model1
```

```
LASSO_model_cv %>% collect_metrics(summarize=TRUE)
```

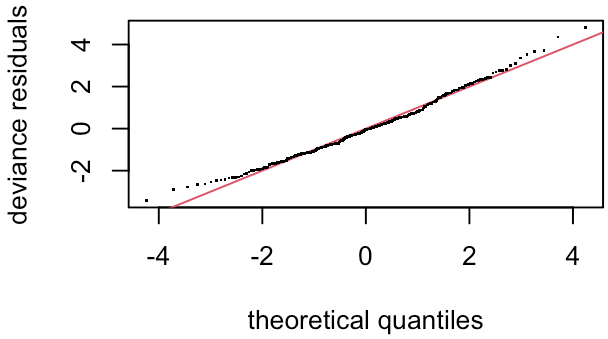
```
## # A tibble: 3 × 6
##   .metric .estimator   mean     n std_err .config
##   <chr>   <chr>       <dbl> <int>   <dbl> <chr>
## 1 mae     standard     1.16    10  0.0697 Preprocessor1_Model1
## 2 rmse    standard     1.46    10  0.0802 Preprocessor1_Model1
## 3 rsq     standard   NaN        0 NA      Preprocessor1_Model1
```

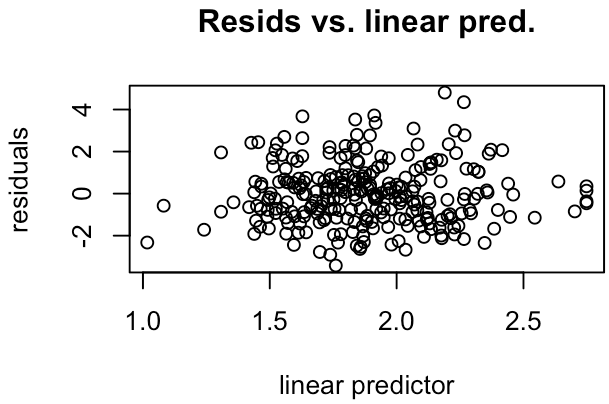
Above we can see that the splines did not improve our model that much. This can be largely explained by the lack of non-linearity in our predictors (as we can see in the plots above), as well as the general overall un-importance of our predictors (as we can see the LASSO model with all coefficients at 0 still performs the best).

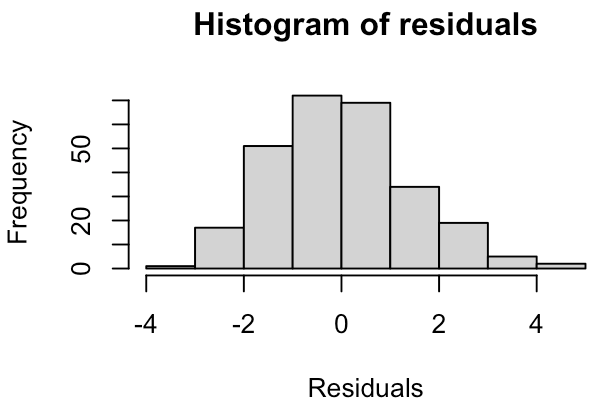
GAM WITH SPLINES

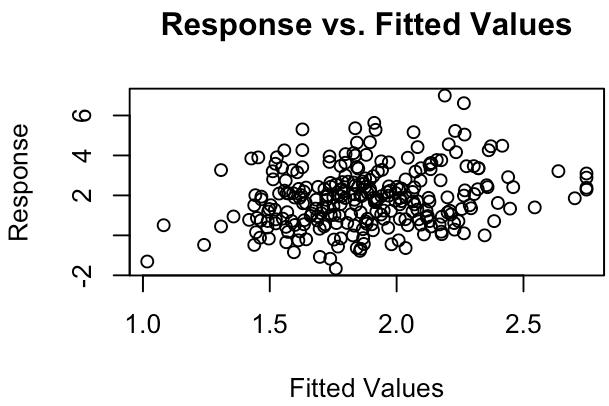
```
gam_spec <-
  gen_additive_mod() %>%
  set_engine(engine = 'mgcv') %>%
  set_mode('regression')

gam_mod <- fit(gam_spec,
    area ~ s(DMC) + s(temp) + s(RH) + s(wind) + s(ISI) + s(DC) + s(FFMC),
    data = fires
)
```

```
par(mfrow=c(2,2))
gam_mod %>% pluck('fit') %>% mgcv::gam.check()
```









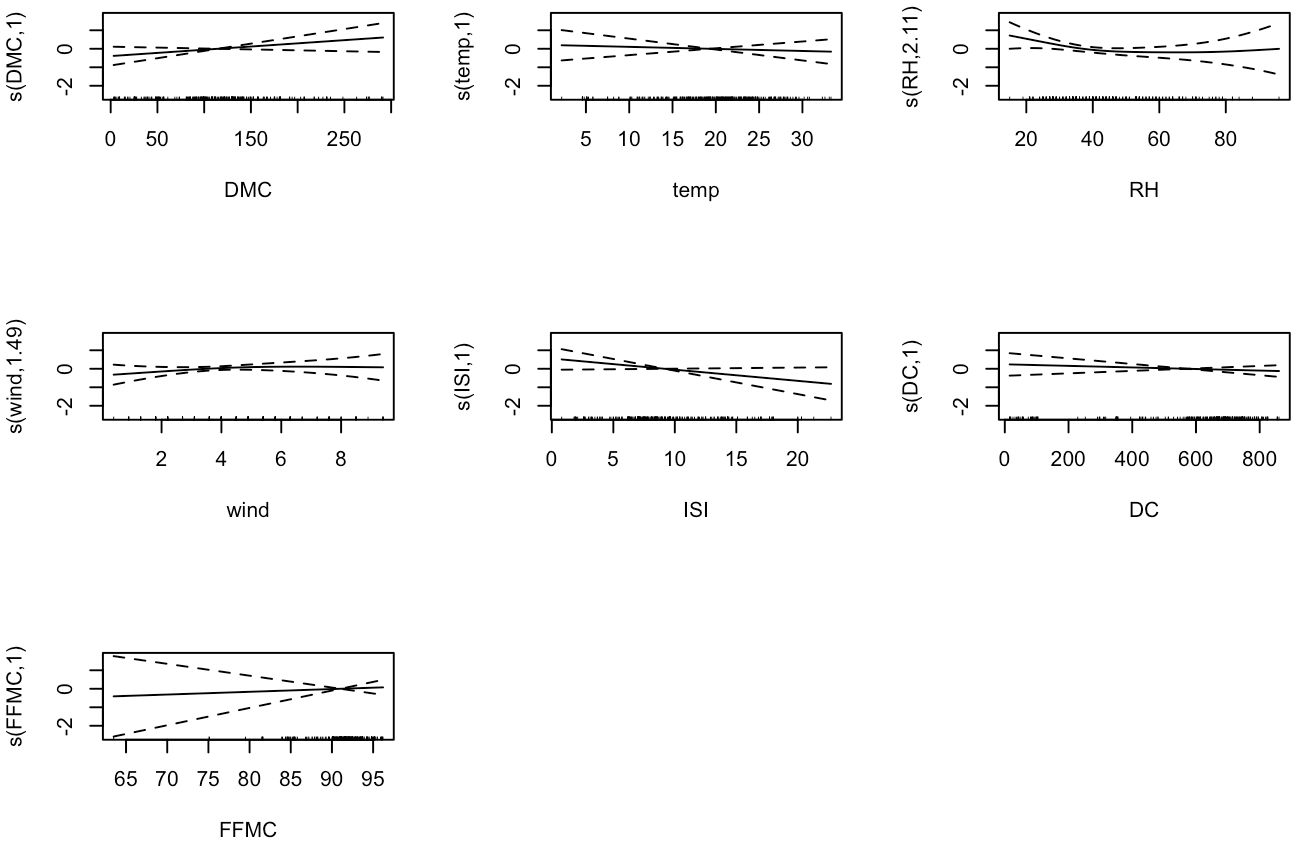

```
##
## Method: GCV   Optimizer: magic
## Smoothing parameter selection converged after 17 iterations.
## The RMS GCV score gradient at convergence was 5.418353e-08 .
## The Hessian was positive definite.
## Model rank =  64 / 64
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##           k'  edf k-index p-value
## s(DMC)  9.00 1.00    0.86    0.01 **
## s(temp) 9.00 1.00    0.99    0.43
## s(RH)   9.00 2.11    0.98    0.35
## s(wind) 9.00 1.49    0.81  <2e-16 ***
## s(ISI)  9.00 1.00    0.80  <2e-16 ***
## s(DC)   9.00 1.00    0.83  <2e-16 ***
## s(FFMC) 9.00 1.00    0.80  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
gam_mod %>% pluck('fit') %>% summary()
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## area ~ s(DMC) + s(temp) + s(RH) + s(wind) + s(ISI) + s(DC) +
##     s(FFMC)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.88540    0.08896   21.19   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##           edf Ref.df     F p-value
## s(DMC)  1.000  1.000 2.421  0.1210
## s(temp) 1.000  1.000 0.215  0.6430
## s(RH)   2.106  2.668 1.985  0.1624
## s(wind) 1.489  1.832 0.591  0.4394
## s(ISI)  1.000  1.000 3.312  0.0699 .
## s(DC)   1.000  1.000 0.603  0.4381
## s(FFMC) 1.000  1.000 0.141  0.7075
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.0217   Deviance explained =  5.3%
## GCV = 2.2154  Scale est. = 2.1367    n = 270
```

In the above table you can see that the r-squared is 0.2 and that the model only explains 5.3% of the variance in the outcome. This is quite bad (especially when compared to our other models) and again comes down to the lack of non-linearity in the data (which can also be observed below) and the overall lack of variable importance for any variable in the dataset.

```
gam_mod %>% pluck('fit') %>% plot( all.terms = TRUE, pages = 1)
```



From the plots we can see that most of our data is linear (hovering around 1 edf with the default range) while RH is the most nonlinear plot (with an edf of 2). As GAM regression is primarily to generate models explaining nonlinearity in multiple predictors, GAMs are most likely not the ideal model to be using in order to perform predictions on the data.

2. **Summarize investigations**
    - Decide on an overall best model based on your investigations so far. To do this, make clear your analysis goals. Predictive accuracy? Interpretability? A combination of both? A mix of interpretability and predictive accuracy is best. The interpradability of our model is important because if someone actually wants to predict a fire using our model, they should only have to take and interpret the measurements that matter the most. The predictive power of the model obviously should not be significantly sacrificed for this goal because predicting fires is the outcome that matters. Taking these factors into account, we do not belive that any of our models are best. That is to say that none of our models have sufficient predictive power to be used to predict fires. We can see this in the LASSO model and specifically the fact that all of the coefficients in the model were set to 0, indicating that none of them are important. We can also see this through the error metrics for our models. The LASSO model had the best error metrics when preforming corss validation, which means that all the other models (splines, normal LM, and our GAM models) were all overfit to training data.

3. **Societal impact**

- Are there any harms that may come from your analyses and/or how the data were collected?
- What cautions do you want to keep in mind when communicating your work? If these models are taken to be accurate, there could be real harms that occur. Specifically, that our models do not predict fires very well at all. That is the take away that should be taken from this analysis, that fires are very unpredictable. This is actually a very important take-away and something that would certainly be useful for fire experts to know. Another possible harm is if the area variable is interpreted as pure area, without consideration for the transformation that occurred prior to analysis. The area was transformed with an ln(x+1) function. Therefore, all predictions should be interpreted through the inverse of this transform to be interpreted in any meaningful way. That being said, the predictions from the models should not be interpreted at all, because, as mentioned above, all predictinos are quite bad.