# Quantile Regression

Rather than make a prediction for the mean and then add a measure of variance to produce a prediction interval (as described in Part 1, A Few Things to Know About Prediction Intervals), quantile regression predicts the intervals directly. In quantile regression, predictions don't correspond with the arithmetic mean but instead with a specified quantile[3]. To create a 90% prediction interval, you just make predictions at the 5th and 95th percentiles – together the two predictions constitute a prediction interval.

The chief advantages over the parametric method described in Part 1, Understanding Prediction Intervals are that quantile regression has…

- fewer and less stringent model assumptions.
- well established approaches for fitting more sophisticated model types than linear regression, e.g. using ensembles of trees.

Advantages over the approach I describe in Simulating Prediction Intervals are…

- computation costs do not get out of control with more sophisticated model types[4].
- can more easily handle heteroskedasticity of errors[5]. This advantage and others are described in Dan Saatrup Nielsen's post on Quantile regression.

I also recommend Dan's post on Quantile regression forests for a description of *how* tree-based methods generate predictions for quantiles (which it turns out is rather intuitive).

For these reasons, quantile regression is often a highly practical choice for many modeling scenarios that require prediction intervals.

# Example

The {parsnip} package does not yet have a `parsnip::linear_reg()` method that supports linear quantile regression[6] (see tidymodels/parsnip#465). Hence I took this as an opportunity to set-up an example for a random forest model using the {ranger} package as the engine in my workflow[7].

When comparing the quality of prediction intervals in this post against those from Part 1 or Part 2 we will not be able to untangle whether differences are due to the difference in model type (linear versus random forest) or the difference in interval estimation technique (parametric or simulated versus quantile regression).

A more apples-to-apples comparison would have been to abandon the {parsnip} framework and gone through an example using the {quantreg} package for quantile regression… maybe in a future post.

## Quantile Regression Forest

Starting libraries and data will be the same as in Part 1, Providing More Than Point Estimates. The code below is sourced and printed from that post's .Rmd file.

Load packages:

```
library(tidyverse)
```

```
library(tidymodels)
library(AmesHousing)
library(gt)

# function copied from here:
# https://github.com/rstudio/gt/issues/613#issuecomment-772072490
# (simpler solution should be implemented in future versions of {gt})
fmt_if_number <- function(..., digits = 2) {
  input <- c(...)
  fmt <- paste0("%.", digits, "f")
  if (is.numeric(input))   return(sprintf(fmt, input))
  return(input)
}
```

Load data:

```
ames <- make_ames() %>%
  mutate(Years_Old = Year_Sold - Year_Built,
         Years_Old = ifelse(Years_Old < 0, 0, Years_Old))

set.seed(4595)
data_split <- initial_split(ames, strata = "Sale_Price", p = 0.75)

ames_train <- training(data_split)
ames_holdout  <- testing(data_split)
```

Unlike in Part 2, Example, the pre-processing and model set-up is not the same as in Part 1. We can remove a few of the transformations that had been important for linear models:

- transformations that don't change the order of observations in a regressor generally don't make a difference for tree-based methods, so we can remove most of the `step_log()`'s
- tree based models are also good at capturing interactions / dependent relationships on their own, hence we can also remove `step_interact()`

```
#RF models require comparably less pre-processing to linear models
rf_recipe <-
  recipe(
    Sale_Price ~ Lot_Area + Neighborhood  + Years_Old + Gr_Liv_Area +
Overall_Qual + Total_Bsmt_SF + Garage_Area,
    data = ames_train
  ) %>%
  step_log(Sale_Price, base = 10) %>%
  step_other(Neighborhood, Overall_Qual, threshold = 50) %>%
  step_novel(Neighborhood, Overall_Qual) %>%
  step_dummy(Neighborhood, Overall_Qual)
```

For our quantile regression example, we are using a random forest model rather than a linear model. Specifying `quantreg = TRUE` tells {ranger} that we will be estimating quantiles rather than averages[8].

```
rf_mod <- rand_forest() %>%
  set_engine("ranger", importance = "impurity", seed = 63233, quantreg
= TRUE) %>%
```

```
  set_mode("regression")

set.seed(63233)
rf_wf <- workflows::workflow() %>%
  add_model(rf_mod) %>%
  add_recipe(rf_recipe) %>%
  fit(ames_train)
```

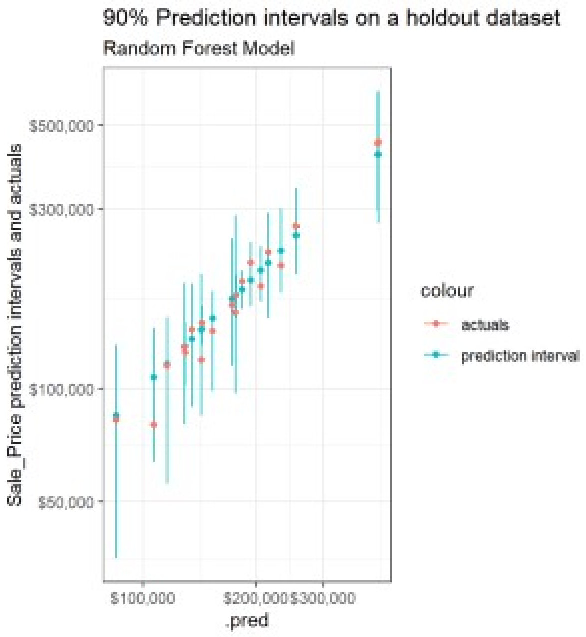
# Review

Tidymodels does not yet have a `predict()` method for extracting quantiles (see issue tidymodels/parsnip#119). Hence in the code below I first extract the {ranger} `fit` object and then use this to make predictions for the quantiles.

```
preds_bind <- function(data_fit, lower = 0.05, upper = 0.95){
  predict(
  rf_wf$fit$fit$fit,
  workflows::pull_workflow_prepped_recipe(rf_wf) %>% bake(data_fit),
  type = "quantiles",
  quantiles = c(lower, upper, 0.50)
  ) %>%
  with(predictions) %>%
  as_tibble() %>%
  set_names(paste0(".pred", c("_lower", "_upper",  ""))) %>%
  mutate(across(contains(".pred"), ~10^.x)) %>%
  bind_cols(data_fit) %>%
  select(contains(".pred"), Sale_Price, Lot_Area, Neighborhood,
Years_Old, Gr_Liv_Area, Overall_Qual, Total_Bsmt_SF, Garage_Area)
}

rf_preds_test <- preds_bind(ames_holdout)
```

Let's review a sample of prediction intervals.

```
set.seed(1234)
rf_preds_test %>%
  mutate(pred_interval = ggplot2::cut_number(Sale_Price, 10)) %>%
  group_by(pred_interval) %>%
  sample_n(2) %>%
  ggplot(aes(x = .pred))+
  geom_point(aes(y = .pred, color = "prediction interval"))+
  geom_errorbar(aes(ymin = .pred_lower, ymax = .pred_upper, color =
"prediction interval"))+
  geom_point(aes(y = Sale_Price, color = "actuals"))+
  scale_x_log10(labels = scales::dollar)+
  scale_y_log10(labels = scales::dollar)+
  labs(title = "90% Prediction intervals on a holdout dataset",
       subtitle = "Random Forest Model",
         y = "Sale_Price prediction intervals and actuals")+
  theme_bw()+
  coord_fixed()
```




If we compare these against similar samples when using the analytic and simulation based approaches for linear regression models, we find that the width of the intervals vary substantially more when built using quantile regression forests.

# Performance

Overall model performance on a holdout dataset is similar (maybe slightly better) for an (untuned) Quantile Regression Forest[9] compared to the linear model (MAPE on holdout dataset of 11% vs. 11.8% with linear model)[10].

As discussed in Part 1, Cautions With Overfitting, we can compare performance on train and holdout datasets to provide an indicator of overfitting:

```
rf_preds_train <- preds_bind(ames_train)

bind_rows(
  yardstick::mape(rf_preds_train, Sale_Price, .pred),
  yardstick::mape(rf_preds_test, Sale_Price, .pred)
) %>%
  mutate(dataset = c("training", "holdout")) %>%
  gt::gt() %>%
  gt::fmt_number(".estimate", decimals = 1)
```

| .metric | .estimator | .estimate | dataset |
|---|---|---|---|
| mape | standard | 4.7 | training |
| mape | standard | 11.0 | holdout |

We see a substantial discrepancy in performance[11]. This puts the validity of the expected coverage of our prediction intervals in question[12]…

# Coverage

Let's check our coverage rates on a holdout dataset:

```
coverage <- function(df, ...){
```

```
  df %>%
    mutate(covered = ifelse(Sale_Price >= .pred_lower & Sale_Price <=
.pred_upper, 1, 0)) %>%
    group_by(...) %>%
    summarise(n = n(),
              n_covered = sum(
                covered
              ),
              stderror = sd(covered) / sqrt(n),
              coverage_prop = n_covered / n)
}

rf_preds_test %>%
  coverage() %>%
  mutate(across(c(coverage_prop, stderror), ~.x * 100)) %>%
  gt::gt() %>%
  gt::fmt_number("stderror", decimals = 2) %>%
  gt::fmt_number("coverage_prop", decimals = 1)
```

| n | n_covered | stderror | coverage_prop |
|---|---|---|---|
| 731 | 706 | 0.67 | 96.6 |

Surprisingly, we see a coverage probability for our prediction intervals of >96%[13] on our holdout dataset[14] – greater than our expected coverage of 90%. This suggests our prediction intervals are, in aggregate, quite conservative[15]. Typically the coverage on the holdout dataset would be the same or *less* than the expected coverage.

This is even more surprising in the context of the Performance indicator of our model for overfitting. See Residual Plots in the Appendix for a few additional figures and notes. In the future I may investigate the reason for this more closely, for now I simply opened a question on Stack Overflow.

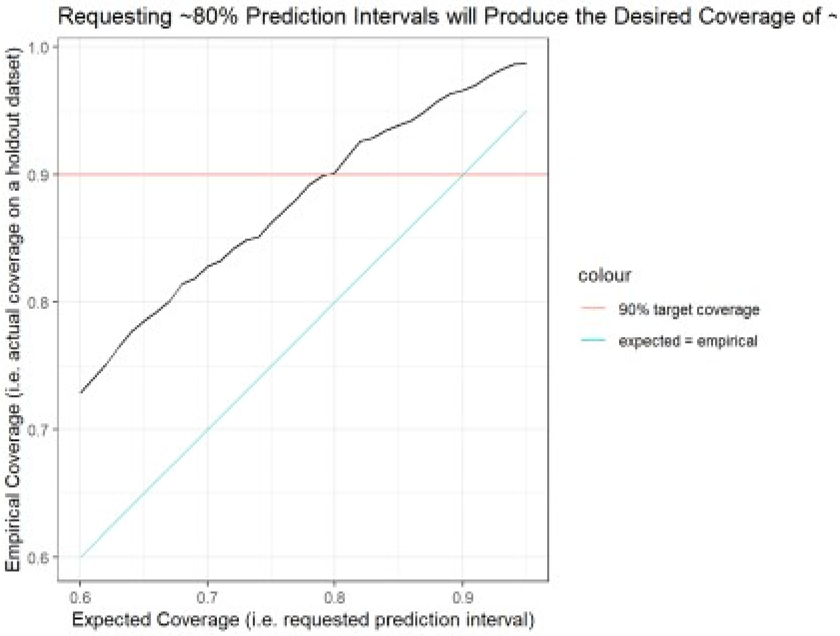
In Part 1, Cautions with overfitting, I described how to tune prediction intervals using coverage rates on holdout data. The code below applies this approach, though due to the surprising finding of a *higher empirical coverage* rate (which is opposite of what we typically observe) I will be identifying a more narrow (rather than broader) *expected coverage*, i.e. prediction interval[16].

```
tune_alpha_coverage <- function(lower, upper){

  preds <- preds_bind(ames_holdout, lower, upper)

  preds %>%
    coverage() %>%
    pull(coverage_prop)
}
```

If we review the expected coverage against the empirical coverage rates, we see the coverage of this model seems, across prediction intervals, to be underestimated[17].

```
coverages <- tibble(lower = seq(0.025, 0.2, by = 0.005)) %>%
  mutate(upper = 1 - lower,
         expected_coverage = upper - lower) %>%
```

```
  mutate(hold_out_coverage = map2_dbl(lower, upper,
tune_alpha_coverage))

coverages %>%
  ggplot()+
  geom_line(aes(x = expected_coverage, y = hold_out_coverage))+
  geom_line(aes(x = expected_coverage, y = expected_coverage, colour =
"expected = empirical"), alpha = 0.5)+
  geom_hline(aes(yintercept = 0.90, colour = "90% target coverage"))+
  coord_fixed()+
  theme_bw()+
  labs(title = "Requesting ~80% Prediction Intervals will Produce the
Desired Coverage of ~90%",
       x = "Expected Coverage (i.e. requested prediction interval)",
       y = "Empirical Coverage (i.e. actual coverage on a holdout
datset)")
```

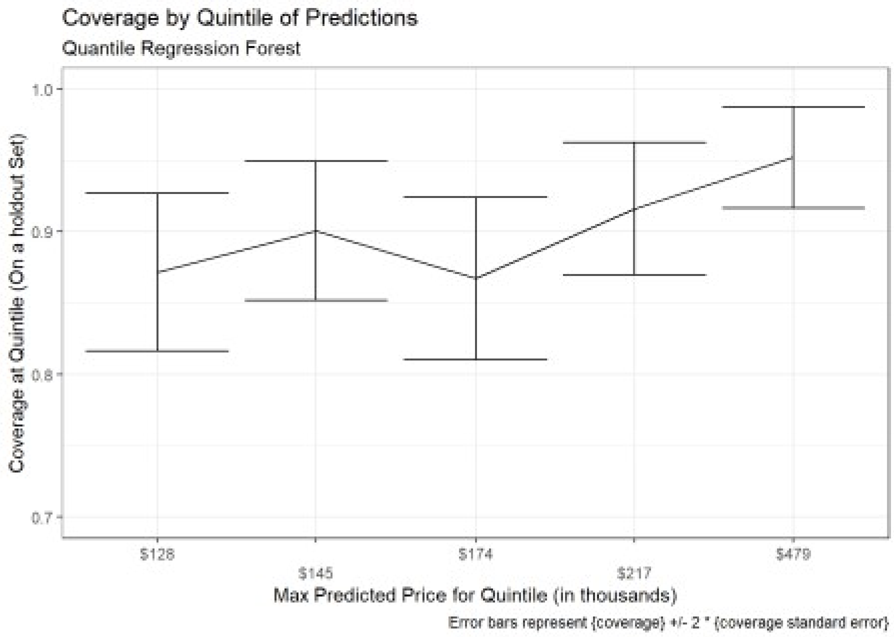



The figure above suggests that an expected prediction interval of 80% will produce an interval with actual coverage of about 90%. For the remainder of the body of this post I will use the 80% expected prediction intervals (90% empirical prediction intervals) from our quantile regression forest model. (See Other Charts in the Appendix for side-by-side comparisons of measures between the 80% and 90% expected prediction intervals.)

### Coverage Across Deciles

```
separate_cut <- function(df, group_var = price_grouped){
  df %>%
    mutate(x_tmp = str_sub({{ group_var }}, 2, -2)) %>%
    separate(x_tmp, c("min", "max"), sep = ",") %>%
    mutate(across(c(min, max), as.double))
}
rf_preds_test_80 <- preds_bind(ames_holdout, lower = 0.10, upper =
0.90)

coverage_80 <- rf_preds_test_80 %>%
```

```
  mutate(price_grouped = ggplot2::cut_number(.pred, 5)) %>%
  coverage(price_grouped) %>%
  separate_cut() %>%
  mutate(expected_coverage = "80%")
coverage_80 %>%
  ggplot(aes(x = forcats::fct_reorder(scales::dollar(max, scale =
1/1000), max), y = coverage_prop))+
  geom_line(aes(group = expected_coverage))+
  geom_errorbar(aes(ymin = coverage_prop - 2 * stderror, ymax =
ifelse(coverage_prop + 2 * stderror > 1, 1, coverage_prop + 2 *
stderror)))+
  coord_cartesian(ylim = c(0.70, 1))+
  scale_x_discrete(guide = guide_axis(n.dodge = 2))+
  # facet_wrap(~expected_coverage)+
  labs(x = "Max Predicted Price for Quintile (in thousands)",
       y = "Coverage at Quintile (On a holdout Set)",
       title = "Coverage by Quintile of Predictions",
       subtitle = "Quantile Regression Forest",
       caption = "Error bars represent {coverage} +/- 2 * {coverage
standard error}")+
  theme_bw()
```




There *appears* to be slightly lower empirical coverage rates for smaller predicted prices, however a statistical test suggests any difference is not significant:

*Chi-squared test of association between {covered} ~ {predicted price group}:*

```
rf_preds_test_80 %>%
  mutate(price_grouped = ggplot2::cut_number(.pred, 5)) %>%
  mutate(covered = ifelse(Sale_Price >= .pred_lower & Sale_Price <=
.pred_upper, 1, 0)) %>%
  with(chisq.test(price_grouped, covered)) %>%
  pander::pander()
```

Pearson's Chi-squared test:
`price_grouped` and `covered`

| Test statistic | df | P value |
|---|---|---|
| 7.936 | 4 | 0.09394 |

# Interval Width

**In aggregate:**

```
get_interval_width <- function(df, ...){
  df %>%
    mutate(interval_width = .pred_upper - .pred_lower,
           interval_pred_ratio = interval_width / .pred) %>%
    group_by(...) %>%
    summarise(n = n(),
              mean_interval_width_percentage =
mean(interval_pred_ratio),
              stdev = sd(interval_pred_ratio),
              stderror = sd(interval_pred_ratio) / sqrt(n))
}

rf_preds_test_80 %>%
  get_interval_width() %>%
  mutate(across(c(mean_interval_width_percentage, stdev, stderror),
~.x*100)) %>%
  gt::gt() %>%
  gt::fmt_number(c("stdev", "stderror"), decimals = 2) %>%
  gt::fmt_number("mean_interval_width_percentage", decimals = 1)
```

| n | mean_interval_width_percentage | stdev | stderror |
|---|---|---|---|
| 731 | 44.1 | 19.19 | 0.71 |

**By quintiles of predictions:**

```
interval_width_80 <- rf_preds_test_80 %>%
  mutate(price_grouped = ggplot2::cut_number(.pred, 5)) %>%
  get_interval_width(price_grouped) %>%
  separate_cut() %>%
  select(-price_grouped) %>%
  mutate(expected_coverage = "80%")

interval_width_80 %>%
  mutate(across(c(mean_interval_width_percentage, stdev, stderror),
~.x*100)) %>%
  gt::gt() %>%
  gt::fmt_number(c("stdev", "stderror"), decimals = 2) %>%
  gt::fmt_number("mean_interval_width_percentage", decimals = 1)
```

| n | mean_interval_width_percentage | stdev | stderror | min | max | expected_coverage |
|---|---|---|---|---|---|---|
| 148 | 53.3 | 19.86 | 1.63 | 63900 | 128000 | 80% |
| 151 | 40.8 | 18.30 | 1.49 | 128000 | 145000 | 80% |

| n | mean_interval_width_percentage | stdev | stderror | min | max | expected_coverage |
|---|---|---|---|---|---|---|
| 143 | 42.4 | 17.31 | 1.45 | 145000 | 174000 | 80% |
| 143 | 37.1 | 17.35 | 1.45 | 174000 | 217000 | 80% |
| 146 | 46.7 | 19.04 | 1.58 | 217000 | 479000 | 80% |

Compared to the intervals created with linear regression analytically in Part 1 and simulated in Part 2, the intervals from our quantile regression forests are…

- a bit more narrow (~44% of `.pred` compared to >51% with prior methods)
- vary more in interval widths between observations (see `stdev`)
- more variable across quintiles[18] – there is a range of more than 16 percentage points in mean interval widths between deciles, roughly 3x what was seen even in Part 2[19].

This suggests that quantile regression forests are better able to differentiate measures of uncertainty by observations compared to the linear models from the previous posts[20].

# Closing Notes

This post walked through an example using quantile regression forests within {tidymodels} to build prediction intervals. Such an approach is relatively simple and computationally efficient to implement and flexible in its ability to vary interval width according to the uncertainty associated with an observation. The section on Coverage suggests that additional review may be required of prediction intervals and that alpha levels may need to be tuned according to coverage rates on holdout data.

*Advantages of Quantile Regression for Building Prediction Intervals:*

- Quantile regression methods are generally more robust to model assumptions (e.g. heteroskedasticity of errors).
- For random forests and other tree-based methods, estimation techniques allow a single model to produce predictions at all quantiles[21].
- While higher in computation costs than analytic methods, costs are still low compared to simulation based approaches[22].

*Downsides:*

- Are not immune to overfitting and related issues (though these also plague parametric methods and can sometimes be improved by tuning).
- Models are more commonly designed to predict the mean rather than a quantile, so there may be fewer model classes or packages available that are ready to use out-of-the-box. These may require editing the objective function so that the model is optimized on the quantile loss, for example[23].

# Appendix

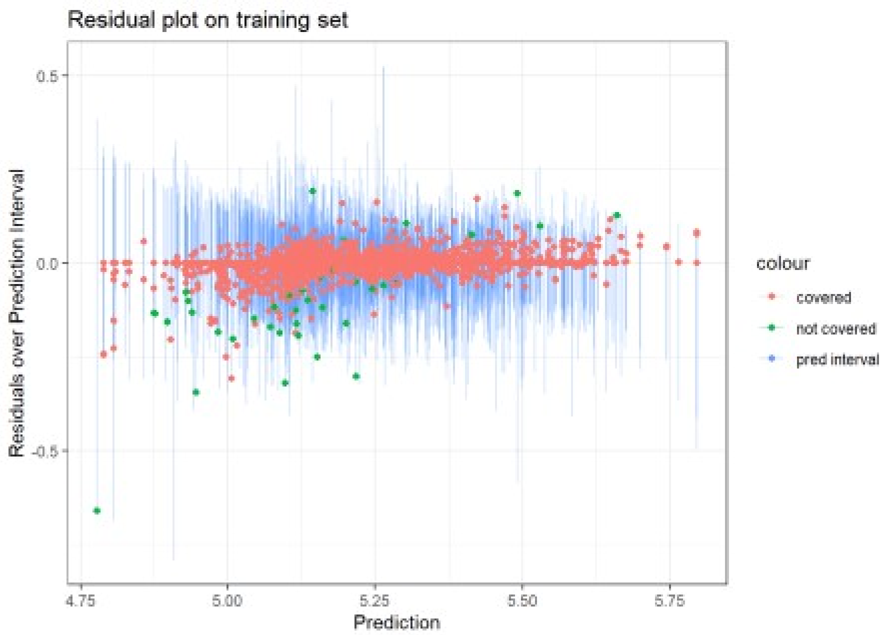
## Residual Plots

Remember that these are on a `log(10)` scale.

*Residual plot on training data:*

```
rf_preds_train %>%
```

```
  mutate(covered = ifelse(
    Sale_Price >= .pred_lower & Sale_Price <= .pred_upper,
    "covered",
    "not covered")
    ) %>%
  mutate(across(c("Sale_Price", contains(".pred")), ~log(.x, 10))) %>%
  mutate(.resid = Sale_Price - .pred) %>%
  mutate(pred_original = .pred) %>%
  mutate(across(contains(".pred"), ~(.x - .pred))) %>%
  ggplot(aes(x = pred_original,))+
  geom_errorbar(aes(ymin = .pred_lower, ymax = .pred_upper, colour =
"pred interval"), alpha = 0.3)+
  geom_point(aes(y = .resid, colour = covered))+
  theme_bw()+
  labs(x = "Prediction",
       y = "Residuals over Prediction Interval",
       title = "Residual plot on training set")
```



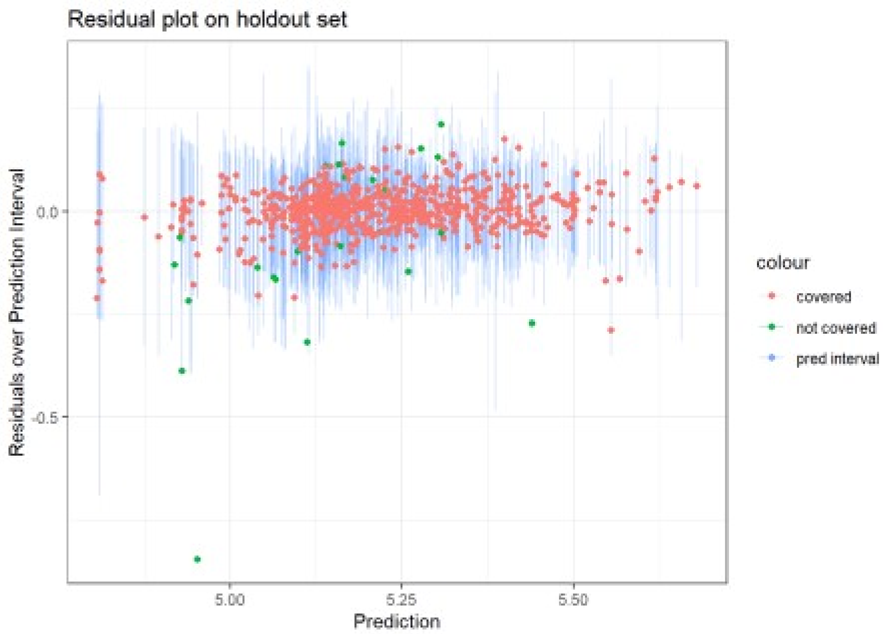

- The several points with residuals of zero on the training data made me think that {ranger} might be set-up such that if there are many residuals of 0, it may assume there is overfitting going on and then default to some kind of conservative or alternative approach to computing the prediction intervals. However this proved false as when I tried higher values of `min_n` – which would reduce any overfitting – I still had similar results regarding a higher than expected coverage rate.
- I also wondered whether some of the extreme points (e.g. the one with a residual of -0.5) may be contributing to the highly conservative intervals. When I removed outliers I found a *slight* narrowing of the prediction intervals, but not much… so that also didn't seem to explain things.

Hopefully people on Stack Overflow know more…

*Residual plot on holdout data:*

```
rf_preds_test %>%
  mutate(covered = ifelse(
```

```
    Sale_Price >= .pred_lower & Sale_Price <= .pred_upper,
    "covered",
    "not covered")
  ) %>%
  mutate(across(c("Sale_Price", contains(".pred")), ~log(.x, 10))) %>%
  mutate(.resid = Sale_Price - .pred) %>%
  mutate(pred_original = .pred) %>%
  mutate(across(contains(".pred"), ~(.x - .pred))) %>%
  ggplot(aes(x = pred_original,))+
  geom_errorbar(aes(ymin = .pred_lower, ymax = .pred_upper, colour =
"pred interval"), alpha = 0.3)+
  geom_point(aes(y = .resid, colour = covered))+
  theme_bw()+
  labs(x = "Prediction",
       y = "Residuals over Prediction Interval",
       title = "Residual plot on holdout set")
```

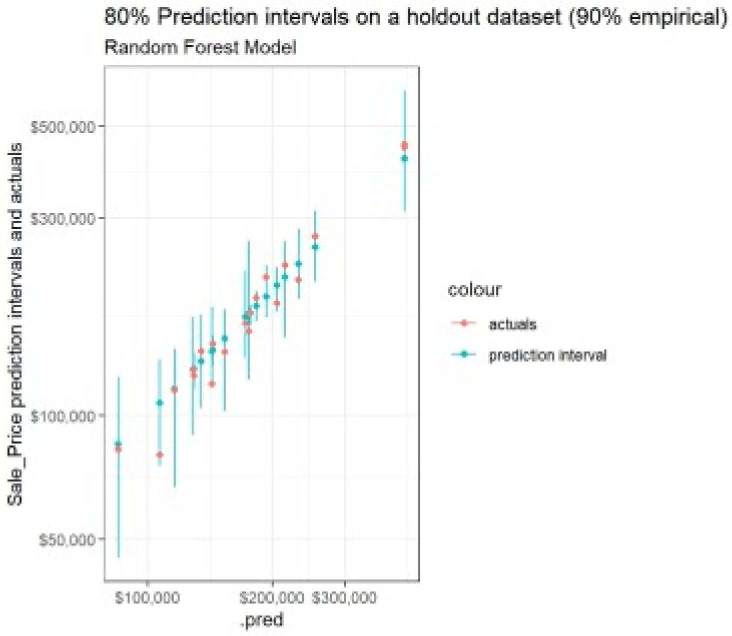



# Other Charts

**Sample of observations, but now using 80% Prediction Intervals:**

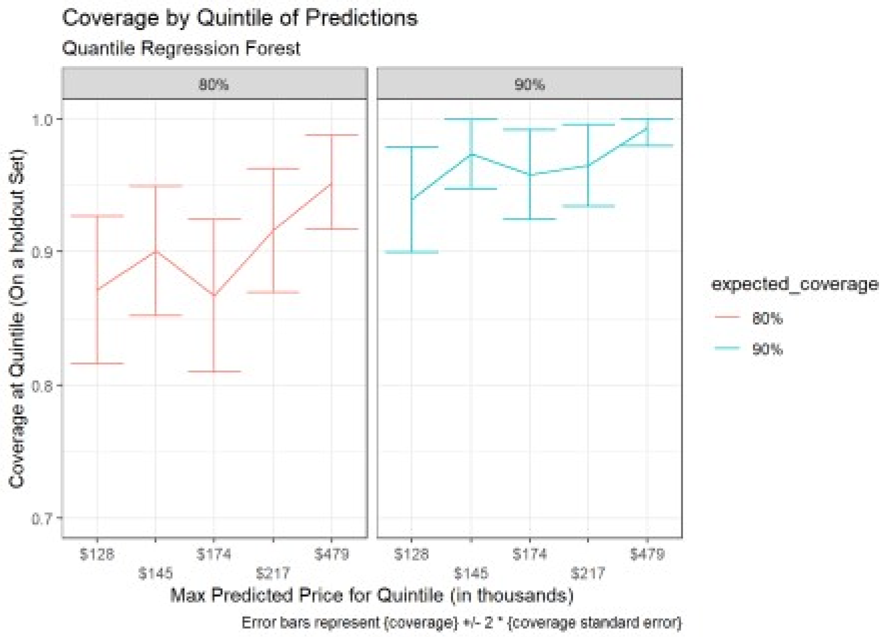
```
set.seed(1234)
rf_preds_test_80 %>%
  mutate(pred_interval = ggplot2::cut_number(Sale_Price, 10)) %>%
  group_by(pred_interval) %>%
  sample_n(2) %>%
  ggplot(aes(x = .pred))+
  geom_point(aes(y = .pred, color = "prediction interval"))+
  geom_errorbar(aes(ymin = .pred_lower, ymax = .pred_upper, color =
"prediction interval"))+
  geom_point(aes(y = Sale_Price, color = "actuals"))+
  scale_x_log10(labels = scales::dollar)+
  scale_y_log10(labels = scales::dollar)+
  labs(title = "80% Prediction intervals on a holdout dataset (90%
empirical)",
```

```
       subtitle = "Random Forest Model",
         y = "Sale_Price prediction intervals and actuals")+
  theme_bw()+
  coord_fixed()
```



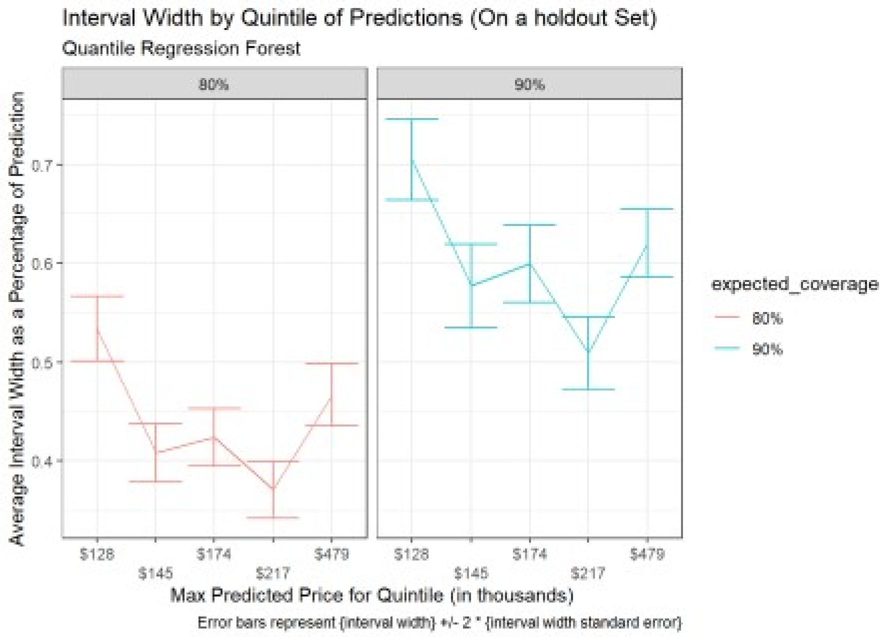

**Coverage rates across quintiles for expected coverage of 80% and 90%:**

```
coverage_90 <- rf_preds_test %>%
  mutate(price_grouped = ggplot2::cut_number(.pred, 5)) %>%
  coverage(price_grouped) %>%
  separate_cut() %>%
  mutate(expected_coverage = "90%")

bind_rows(coverage_80, coverage_90) %>%
  ggplot(aes(x = forcats::fct_reorder(scales::dollar(max, scale =
1/1000), max), y = coverage_prop, colour = expected_coverage))+
  geom_line(aes(group = expected_coverage))+
  geom_errorbar(aes(ymin = coverage_prop - 2 * stderror, ymax =
ifelse(coverage_prop + 2 * stderror > 1, 1, coverage_prop + 2 *
stderror)))+
  coord_cartesian(ylim = c(0.70, 1))+
  scale_x_discrete(guide = guide_axis(n.dodge = 2))+
  facet_wrap(~expected_coverage)+
  labs(x = "Max Predicted Price for Quintile (in thousands)",
       y = "Coverage at Quintile (On a holdout Set)",
       title = "Coverage by Quintile of Predictions",
       subtitle = "Quantile Regression Forest",
       caption = "Error bars represent {coverage} +/- 2 * {coverage
standard error}")+
  theme_bw()
```




**Interval Widths across quantiles for expected coverage of 80% and 90%:**

```
interval_width_90 <- rf_preds_test %>%
  mutate(price_grouped = ggplot2::cut_number(.pred, 5)) %>%
  get_interval_width(price_grouped) %>%
  separate_cut() %>%
  select(-price_grouped) %>%
  mutate(expected_coverage = "90%")


bind_rows(interval_width_80, interval_width_90) %>%
  ggplot(aes(x = forcats::fct_reorder(scales::dollar(max, scale =
1/1000), max), y = mean_interval_width_percentage, colour =
expected_coverage))+
  geom_line(aes(group = expected_coverage))+
  geom_errorbar(aes(ymin = mean_interval_width_percentage - 2 *
stderror, ymax = mean_interval_width_percentage + 2 * stderror))+
  # coord_cartesian(ylim = c(0.70, 1.01))+
  scale_x_discrete(guide = guide_axis(n.dodge = 2))+
  facet_wrap(~expected_coverage)+
  labs(x = "Max Predicted Price for Quintile (in thousands)",
     y = "Average Interval Width as a Percentage of Prediction",
     title = "Interval Width by Quintile of Predictions (On a holdout
Set)",
     subtitle = "Quantile Regression Forest",
     caption = "Error bars represent {interval width} +/- 2 * {interval
width standard error}")+
  theme_bw()
```


Interval Width by Quintile of Predictions (On a holdout Set)
Quantile Regression Forest
80%
90%
Average Interval Width as a Percentage of Prediction
0.4
0.5
0.6
0.7
$128
$145
$174
$217
$479
Max Predicted Price for Quintile (in thousands)
Error bars represent {interval width} +/- 2 * {interval width standard error}
expected_coverage
80%
90%