Pre-Smoothed Simulated Linear Pooling

Background

This is a statistical/mathematical approach to combining forecasts from component models, where the underlying distribution of forecasts from each model is unavailable, but rather, is only sparsely represented by a set of discrete points on the underlying quantile function. This is not substantially different from a standard problem whereby one desires access to underlying data but only has access to a smaller set of information from which the underlying data might, under certain conditions, be simulated.

The small number of functions in this package were designed specifically for the problem of combining forecasts from say, multiple models, where the underlying set of original data are not available. Specifically, imagine that we have N models $i, 1, 2, \ldots, N$, and for each, only a summary of the underlying forecasts is available in the form of a finite set of estimates from the quantile function (i.e. for each model i, $Q_i(p) = V$, where V is the number of forecasted outcomes for which p proportion of the i^{th} model estimates fall).

Simulated Linear Pooling

In this package, a continuous $\hat{Q_i}(p)$ is estimated by applying a generalized additive model with penalized cubic splines to the set of (p, V) pairs. Uniform distribution variates are then drawn (in large number) and applied to this curve, to produce a simulated set of forecasts. The simulated forecasts from each component are summed together (i.e. linearly) to produce a single (multi-modal) simulated distribution. This simulated combined distribution of forecasts can be used to provide an “ensemble” estimate of the component models.

Simple Approach / Vincentization

Obviously, a more simple and common approach to combining quantile functions in this situation is to simply take the mean of the values V over the component models, commonly known as *vincentization* (https://en.wikipedia.org/wiki/Vincent_average). Specifically for N models $i = 1, 2, \ldots, N$, each with a finite set of estimates from its corresponding quantile function $Q_i(p)$, the combined quantile function at any point p is

$$Q_{ens}(p) = \frac{1}{N} \sum_{i=1}^{N} Q_i(p)$$

Optionally, weights could be added and/or the median of the component model estimates could be used.

Pre-Smoothing to Reduce Volatility in Longitudinal Forecasts

For either algorithm, we address the situation where each component model is producing a longitudinal series of such forecasts (for example, on D days $d = 1, 2, \ldots, D$). In some cases (i.e. depending on the estimation

problem of interest), the volatility across time at a specific probability level, p, within a model might be considered a nuisance or noise factor to be reduced/removed, rather than a true effort at estimating such volatility. To address this specific need, we provide an optional and additional pre-processing step, prior to combining model estimates. This includes using locally-weighted regression smoothing to reduce volatility in the $Q_i(p)$ levels across different time points for a particular component model. This smoothed version of $Q_i(p)_{sm}$ is then used in the above approach rather than original non-smoothed $Q_i(p)$ from each component.

Demonstration of Simulated Linear Pooling

This vignette provides a very basic example of using the package to:

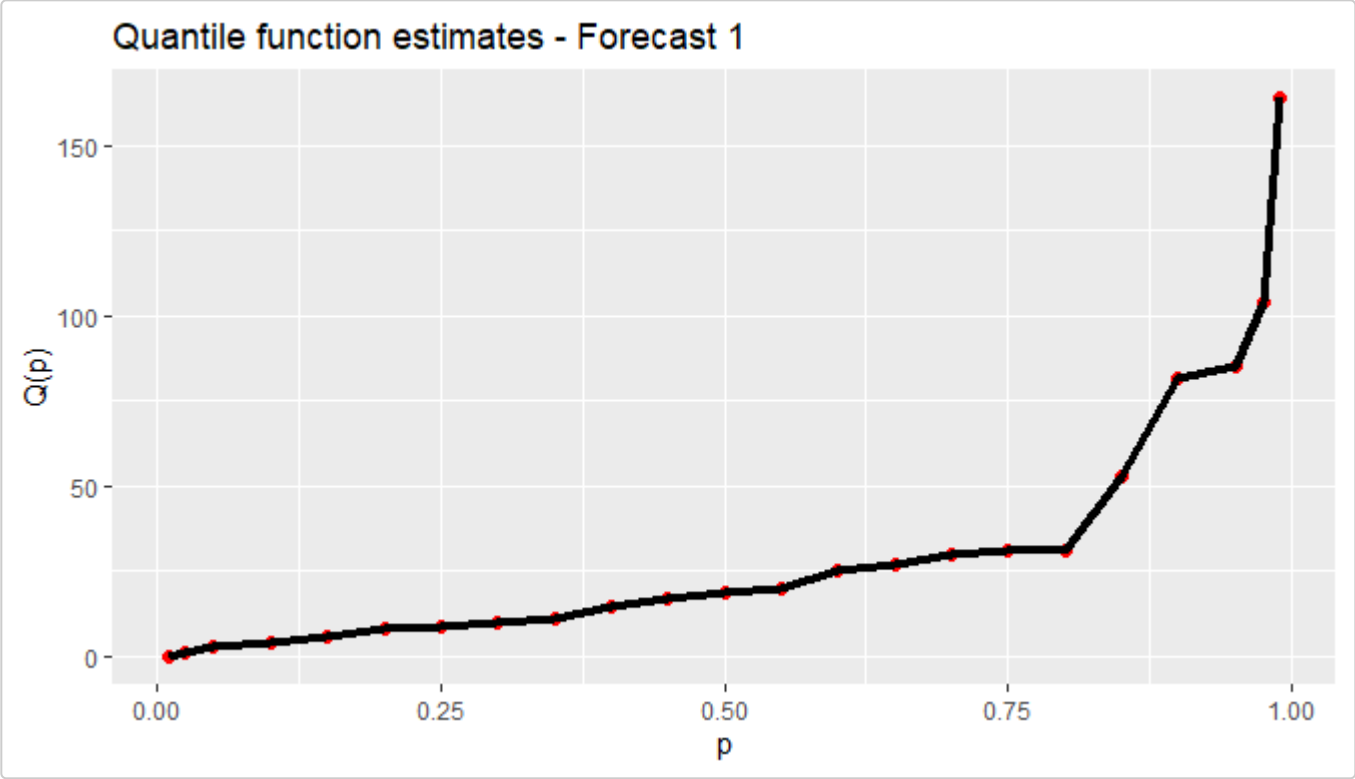
1. Simulate a set of forecasts from a single set of pairs (p, V)
2. Simulate a combined quantile function from multiple quantile functions using:
    - Simulated Linear Pooling
    - Simple Vincentization
3. Pre-Smooth a set of forecasts from a single model

Simulate a set of forecasts from a single set of pairs, (p, V)

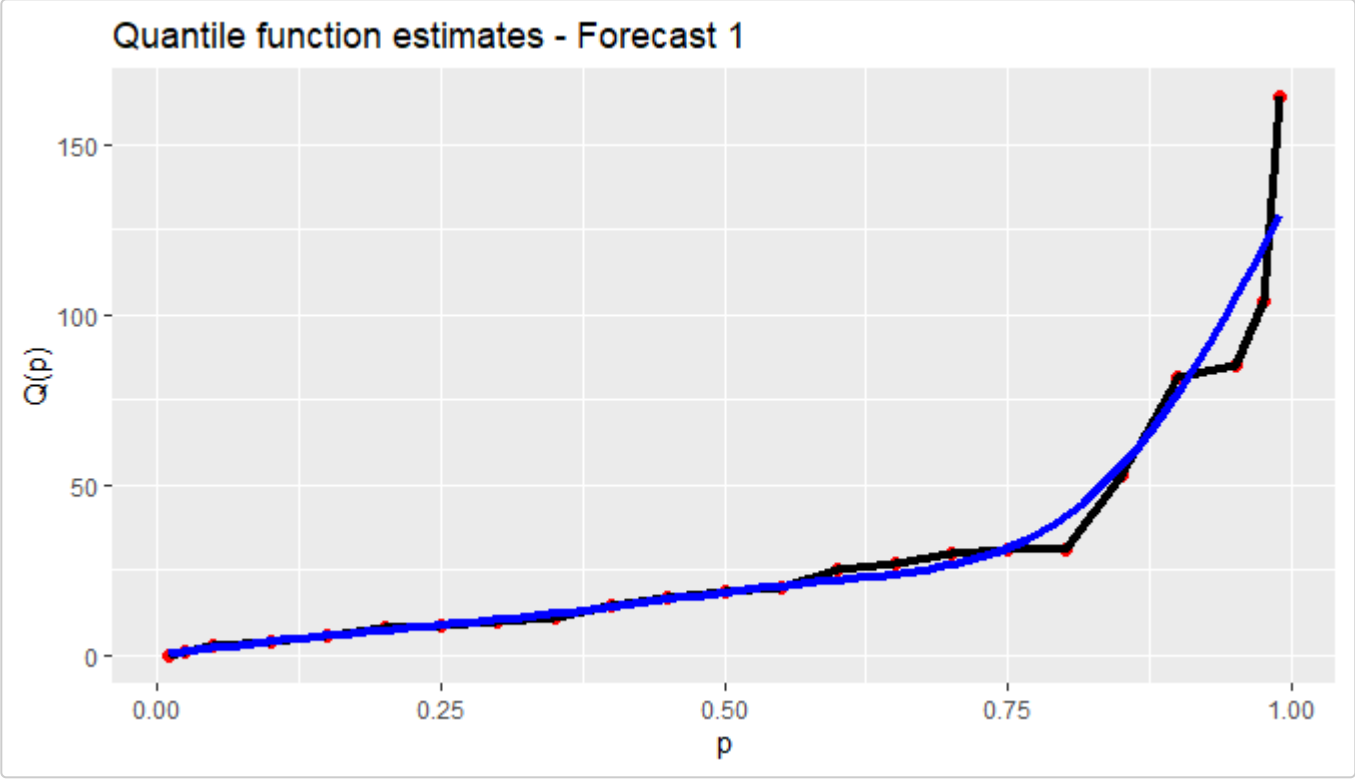
First, we will generate a single hypothetical forecast with 23 quantile levels

```
# Set a sequence of probability levels, p
forecast_1 = data.table::data.table(
  p = c(0.01,0.025,seq(0.05,0.95,0.05),0.975,0.99),
  qp = c(0,1,3,4,6,8,9,10,11,15,17,19,20,25,27,30,31,31,53,82,85,104,164)
)
forecast_1 %>% head()
#>        p qp
#> 1: 0.010  0
#> 2: 0.025  1
#> 3: 0.050  3
#> 4: 0.100  4
#> 5: 0.150  6
#> 6: 0.200  8
```

Next, we plot these data



A smooth curve using the `mgcv::gam()` can be added



The function `generate_slp_predictions()` simulates the underlying distribution from these raw data. See `??generate_slp_predictions` for more information, but the main parameters are as as below:

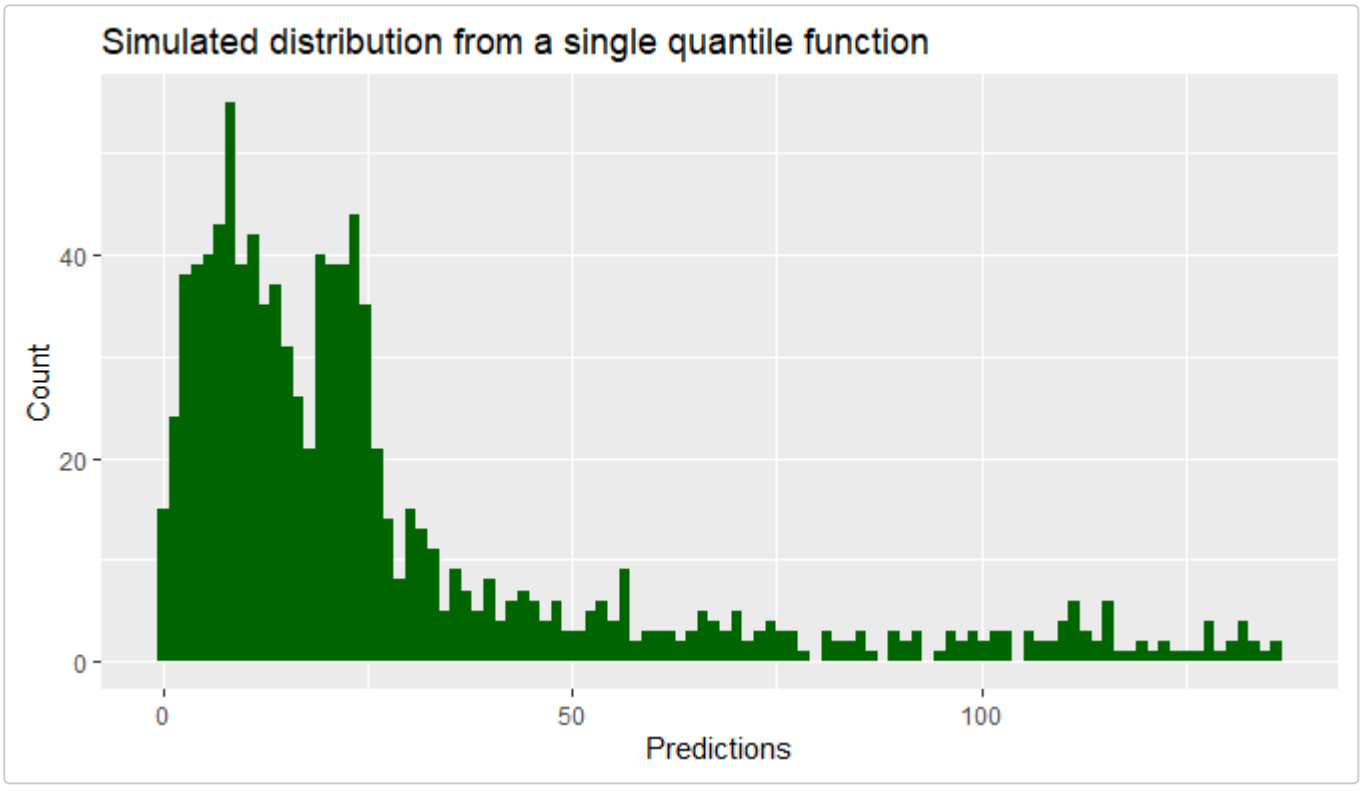
- `x`: input dataframe
- `qp_var`: name of the column holding the $Q(p)$ values
- `p_var`: name of the column holding the p values
- `draw_size`: integer (default 1000) number of simulated values from the underlying distribution

- `parallel`: one of “auto”,“on”, or “off”; default is “auto”, will run in parallel on half the available cores, if >1000 gam models
- `threads` = NULL, set an integer value of threads

We’ll start by calling this function with the default parameters, along with specifying `x`, `qp_var`, and `p_var`

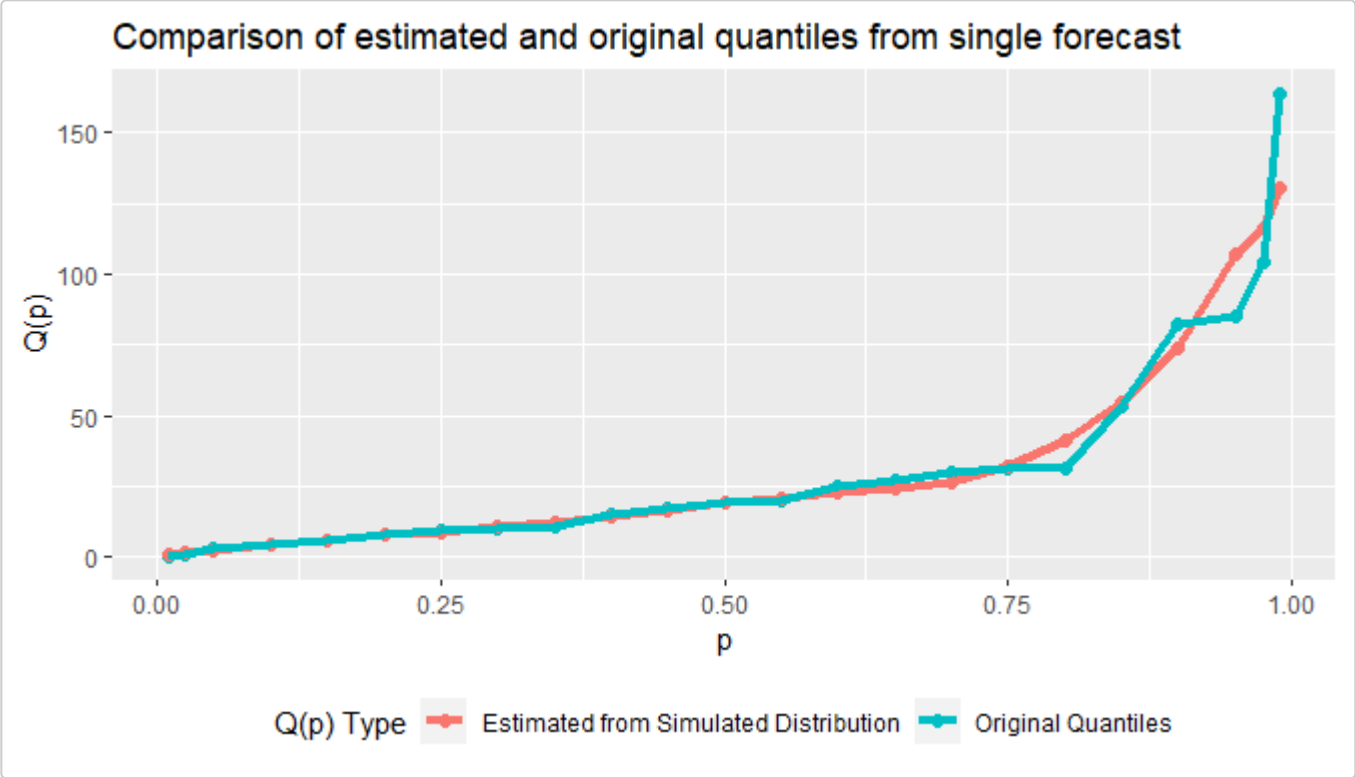
```
# Simulate a 1000 observations from the underlying distribution
simulated_1 = generate_slp_predictions(
  x = forecast_1,
  qp_var = "qp",
  p_var = "p")

# plot a histogram of these
ggplot(simulated_1,aes(x=predictions,stat="identity")) +
  geom_histogram(bins=100,fill="darkgreen") +
  labs(x="Predictions", y="Count") +
  ggtitle("Simulated distribution from a single quantile function")
```



Let’s compare the quantiles from this simulated distribution with the true (i.e. available) quantiles

```
# add simulate quantiles
forecast_1[, sim_qp:= quantile(simulated_1$predictions, probs = p)]
```



Obviously, the simulated quantile looks very similar to the original generalized additive model prediction curve, since the simulated data were derived from this approach when calling `generate_slp_predictions()`

Simulate a combined quantile function from multiple quantile functions

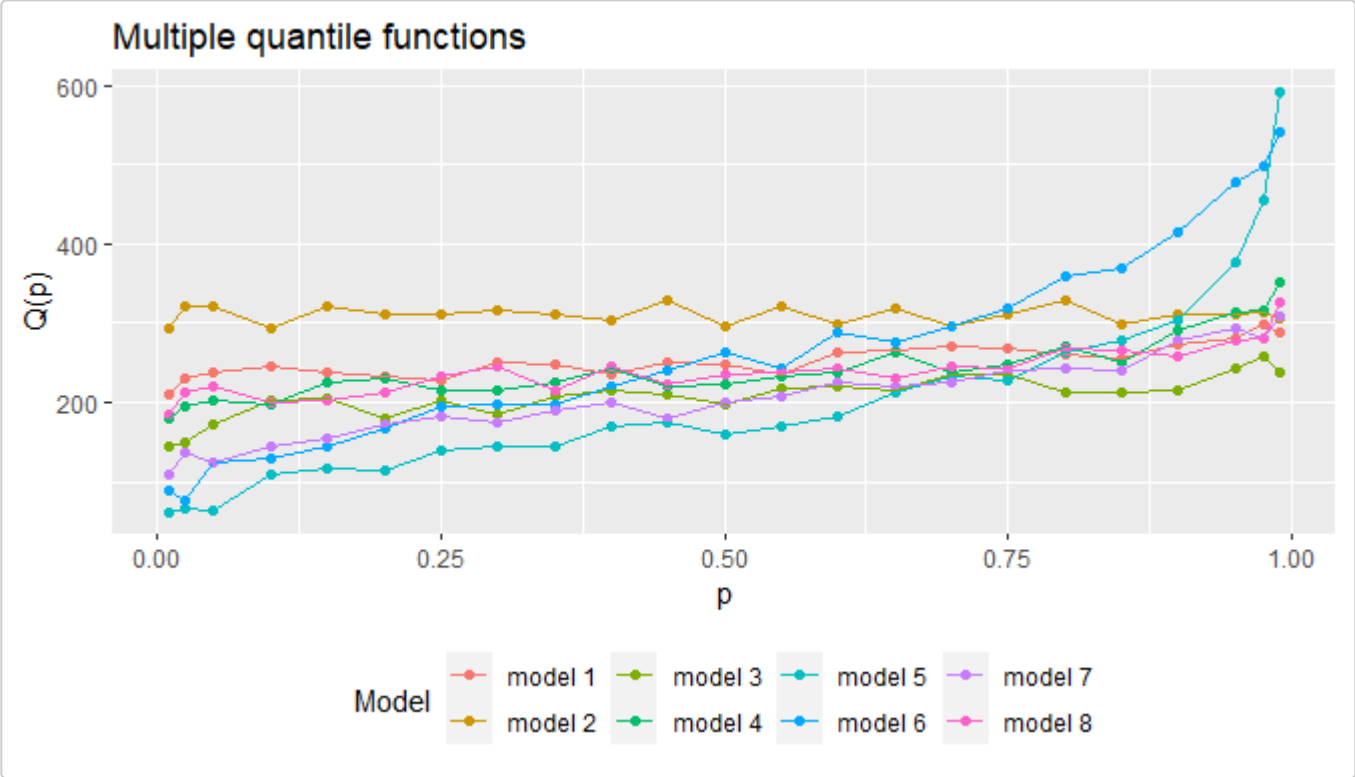
The functionality in this package is best illustrated however in the situation where we have multiple model forecasts, and we want to combine them into a single ensemble forecast. In this next section, we demonstrate how this can be done, using the included example dataset `multiple_models.rda`

```
# Load the data
data("multiple_models")
```

The data contains three columns: `quantile`, `outcomes`,`model`. The first two are simple the (p, V) pairs referred to above, while the third distinguished the set of forecasts. Let's look at the first 6 rows:

```
# Show first 6 rows
multiple_models %>% head()
#>    quantile outcomes   model
#> 1:    0.010 211.1534 model 1
#> 2:    0.025 230.9167 model 1
#> 3:    0.050 239.1890 model 1
#> 4:    0.100 244.8569 model 1
#> 5:    0.150 237.9025 model 1
#> 6:    0.200 233.1660 model 1
```

We can plot these data, and see how different the quantile functions appear:

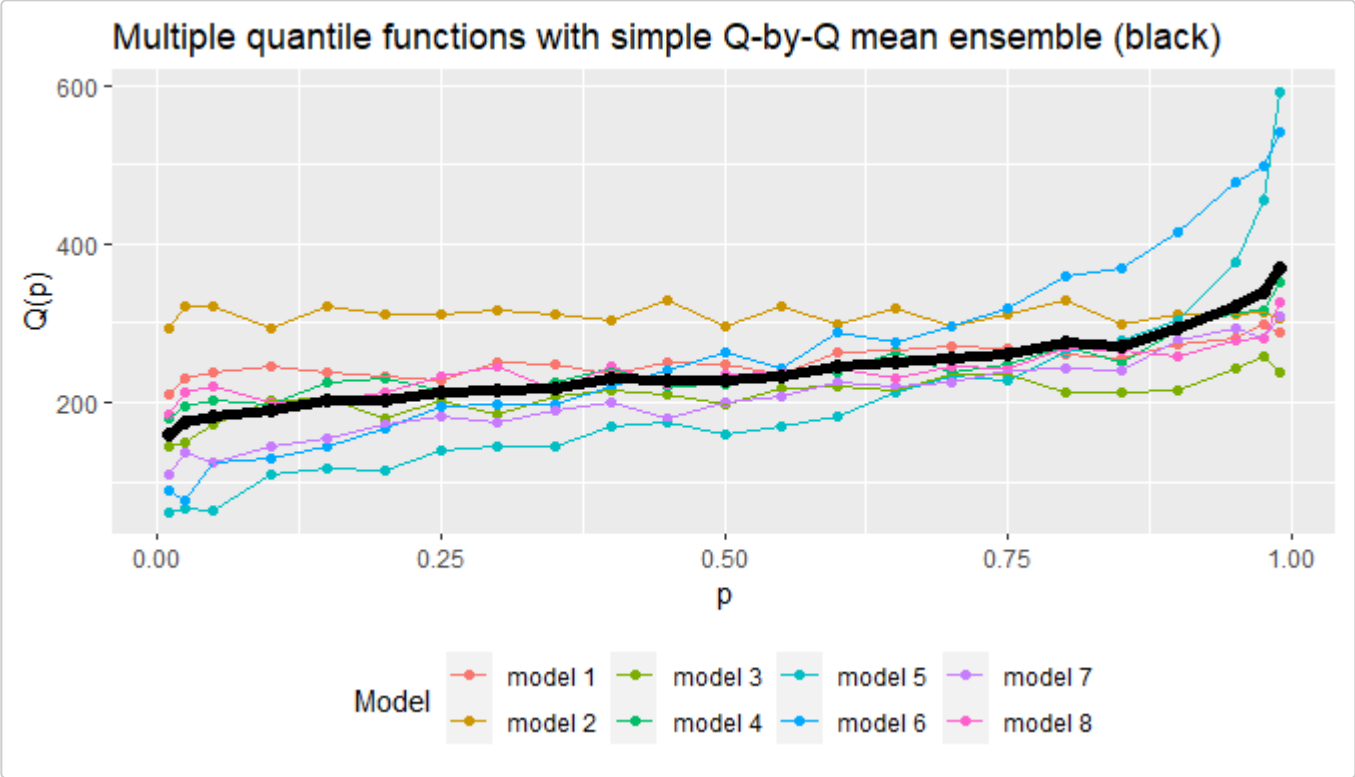


Our task is to combine these individual forecasts. Obviously, the most simple and usually most appropriate is to simply take a (weighted) mean/median of the quantile-by-quantile submissions. This can be implemented simply using `generate_vinc_ensemble()`

```
ens_vinc <- generate_vinc_ensemble(x=multiple_models,qp_var = "outcomes",p_var = "quantile",pooltype
         = "mean")

# view the ensemble
ens_vinc %>% head()
#>    quantile outcomes
#> 1:    0.010 159.1719
#> 2:    0.025 173.7202
#> 3:    0.050 183.6039
#> 4:    0.100 189.6527
#> 5:    0.150 201.4398
#> 6:    0.200 202.4045
```

Let's overlay the simple $\hat{Q(p)}$ estimated here with the originals

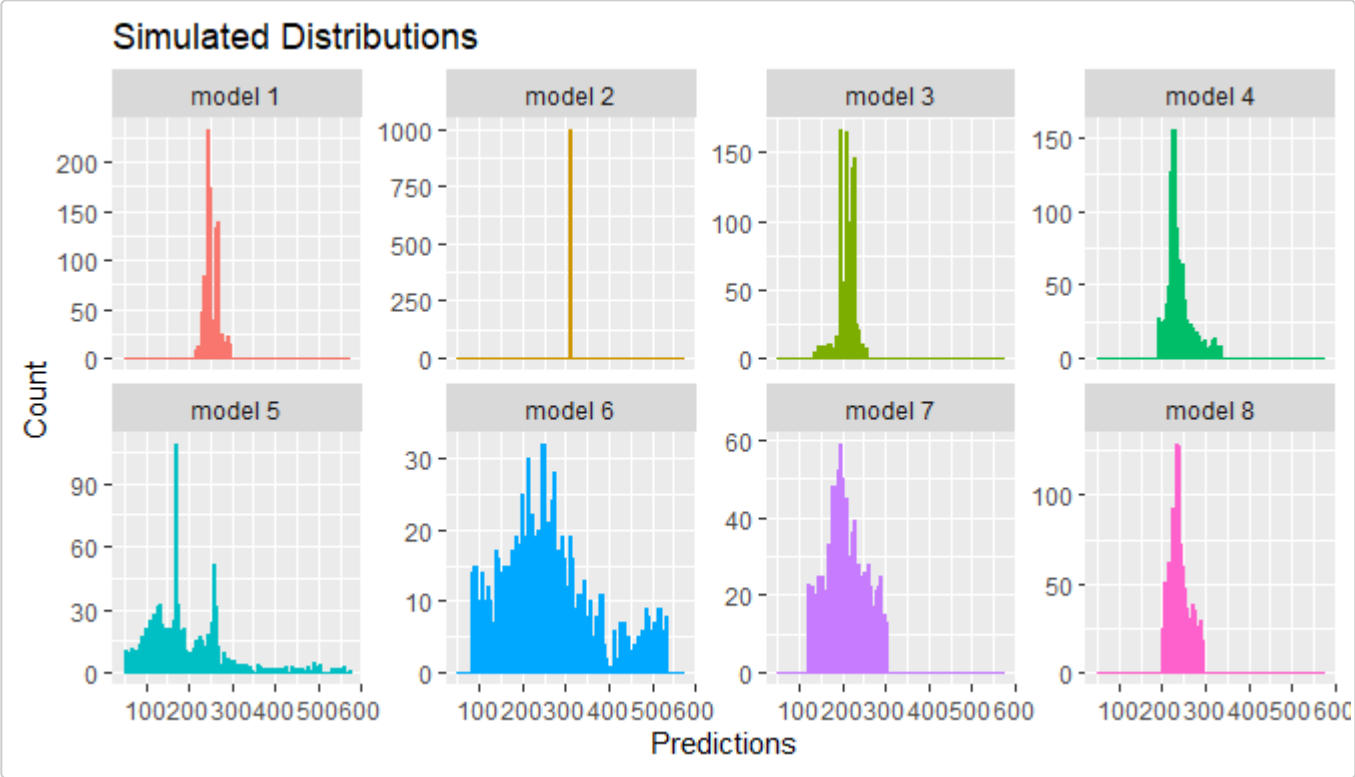


We can simulate the underlying distribution of each model, using `generate_slp_predictions()`, as we did above, but now taking advantage of the `byvars` option. Notice how model 2 has very little variability; the resulting simulated distribution for that model will be centered around a very small interval, relative to the other models

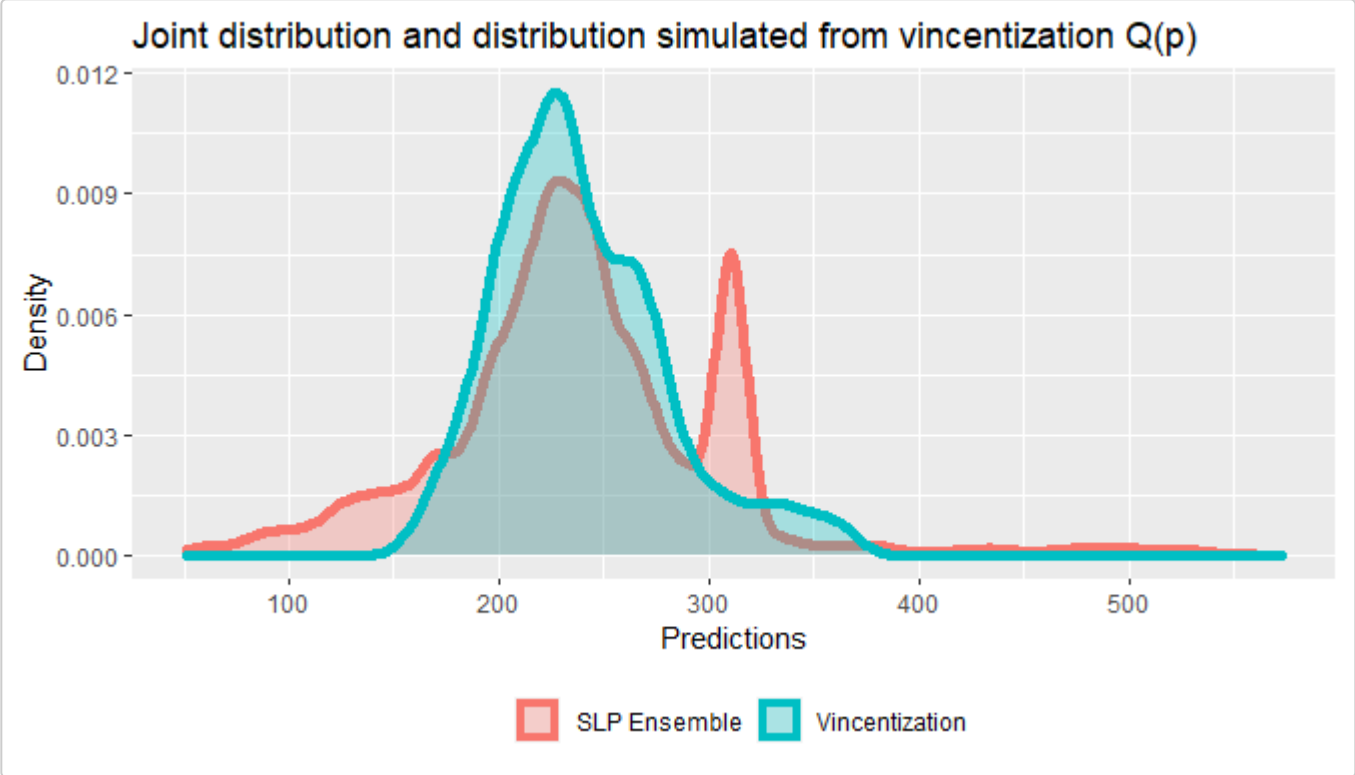
```
# Generate the distributions
sim_distributions <-
         generate_slp_predictions(multiple_models,qp_var="outcomes",p_var="quantile",byvar="model")
#> Running 8 gam models.. will take some time
#> Complete in 0.1 seconds.

# Now, lets look at the result
sim_distributions[,.N,by=model]
#>      model    N
#> 1: model 1 1000
#> 2: model 2 1000
#> 3: model 3 1000
#> 4: model 4 1000
#> 5: model 5 1000
#> 6: model 6 1000
#> 7: model 7 1000
#> 8: model 8 1000
```

We can plot these simulate distributions, as before:

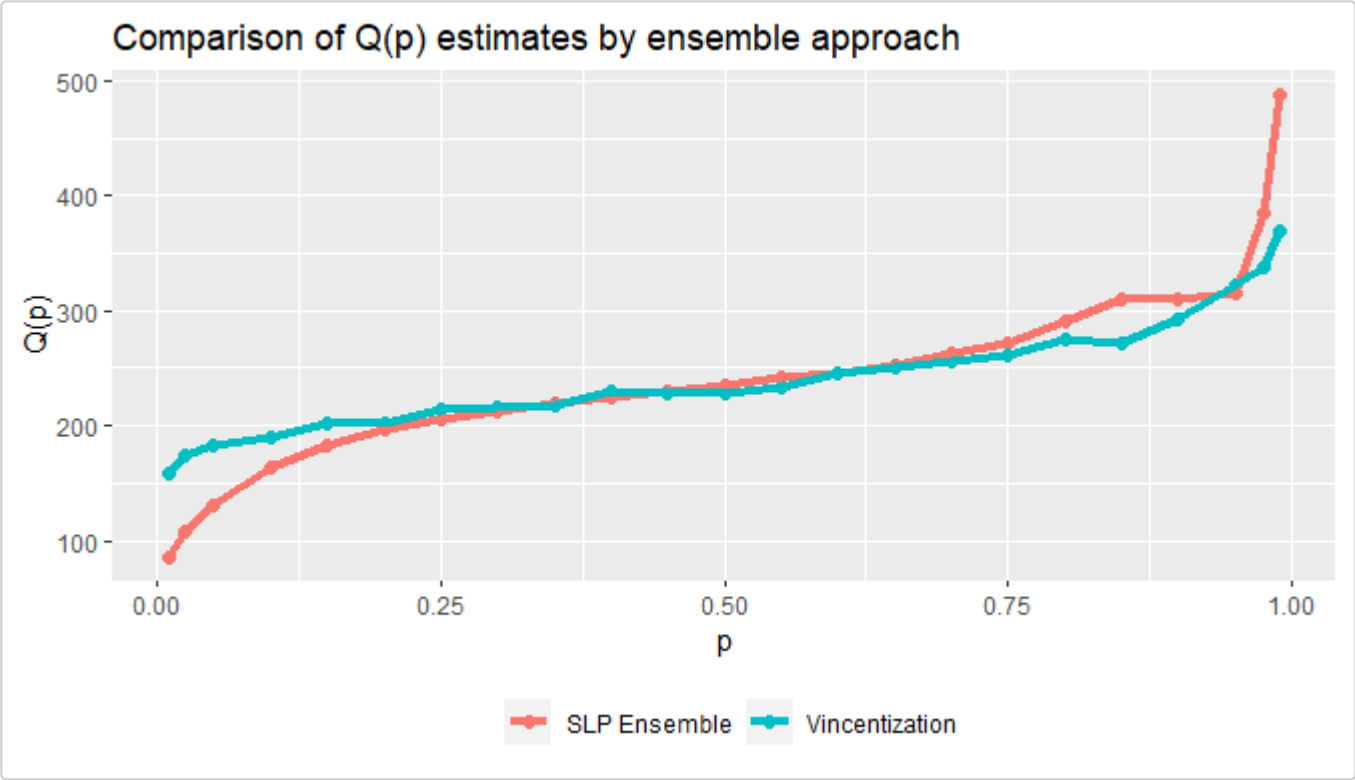


If we simulate a distribution from the vincentization-combined quantile function, and compare to the joint distribution produced by linearly pooling the above 8 separately simulated simulations, we see that the latter is multi-modal, and has broader interval width



We can use the `gen_slp_ensemble()` function immediately following the `gen_slp_predictions()` call to create the $\hat{Q_{slp}}(p)$ and compare to the one estimated through simple averaging of the quantile levels across models (i.e. $\hat{Q_{vinc}}(p)$)

```r
# This line simply estimate the quantiles from the joint (multi-modal) distribution
slp_ensemble <- generate_slp_ensemble(sim_distributions,qp_var="predictions")
```



Notice how the ensemble approach will tend to produce a more disperse distribution.

Pre-Smooth a set of forecasts from a single model

In some situations, we may have a model or set of models that are generating quantile functions longitudinally (i.e. over some time unit). In this situation, we may be interested in generated an ensemble forecast (i.e. $\hat{Q_{ens}}(p)$) for each unit of time. Further, we may want to smooth out day to day volatility in the indivudal model-level quantile functions. In this next section, we demonstrate how this can be done, using the included example dataset `multiple_models_over_time.rda`

```r
# Load the data
data("multiple_models_over_time")
```

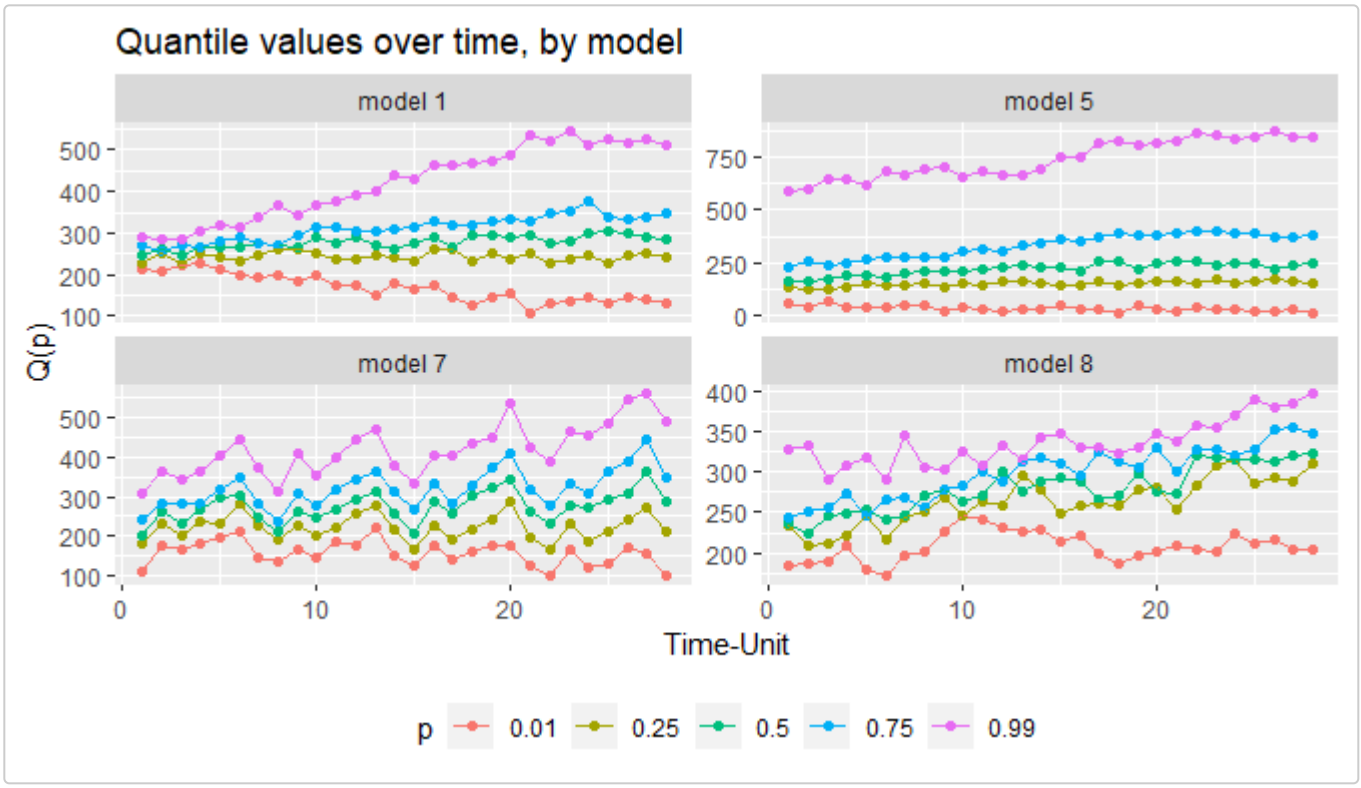
This data set has the same structure as before, except now for each of the models, we have multiple quantile functions, one for each value of the new fourth column `val`. Let's assume this value is an indicator of time (it could be considered days, going from 1 to 28, i.e. 4 weeks).

First, lets look at the raw data for four models, over time:

```r
target_models = c("model 1", "model 5", "model 7", "model 8")
target_quantiles = c(0.01, 0.25, 0.5, 0.75, 0.99)

plt <- ggplot(
```

```
  data = multiple_models_over_time[
    model %in% target_models
    & quantile %in% target_quantiles], aes(val, outcomes, color=factor(quantile))) +
  geom_point() +
  geom_line() +
  facet_wrap(~model, scales="free_y") +
  theme(legend.position="bottom") +
  labs(x="Time-Unit", y="Q(p)", color = "p") +
  ggtitle("Quantile values over time, by model")
plt
```



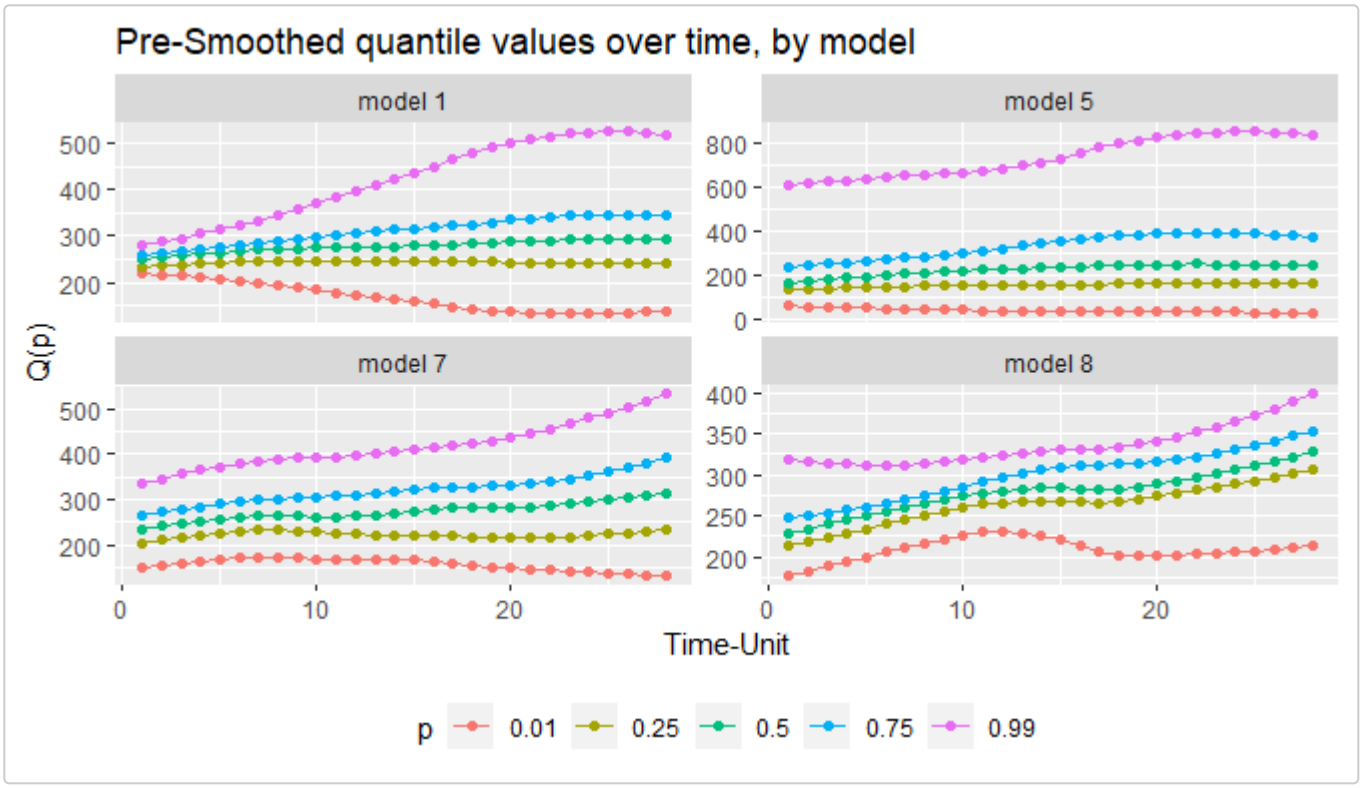
We can use the `pre_smooth_quantile_functions()` function to take set of quantiles and pre-smooth them using locally weighted regression. Here, we set the `byvars` parameter to the separate models. There is an optional parameter, `loess_span` that can help control the level of smoothing; the default is 0.75 (see `??loess` for more information)

```
ps_data <- pre_smooth_quantile_functions(
  x=multiple_models_over_time,
  qp_var="outcomes",
  p_var = "quantile",
  time_var="val",
  byvars="model")
#>
#> Smoothing 184 quantile levels
```

The new dataset returned by this function contains the `qp_var`, the `time_var`, the `p_var`, and `byvars` (in this case, model), and a new variable containing the smoothed estiame of the quantile function. This new variable is by default named with a `_hat` suffixed to the name of the `qp_var`, but this can be changed by providing a string name in the optional `smooth_col_name` parameter.

```
ps_data %>% head()
#>    outcomes val quantile   model outcomes_hat
#> 1: 211.1534   1     0.01 model 1     219.6348
#> 2: 208.5023   2     0.01 model 1     216.9753
#> 3: 223.7004   3     0.01 model 1     214.0260
#> 4: 228.1766   4     0.01 model 1     210.7528
#> 5: 213.7597   5     0.01 model 1     207.1276
#> 6: 197.6504   6     0.01 model 1     203.2041
```

Now, lets' look at the same four models, now smoothed



These pre-smoothed quantile functions can then of course be combined using either the vincentization or the simulated linear pooling method, as shown below.

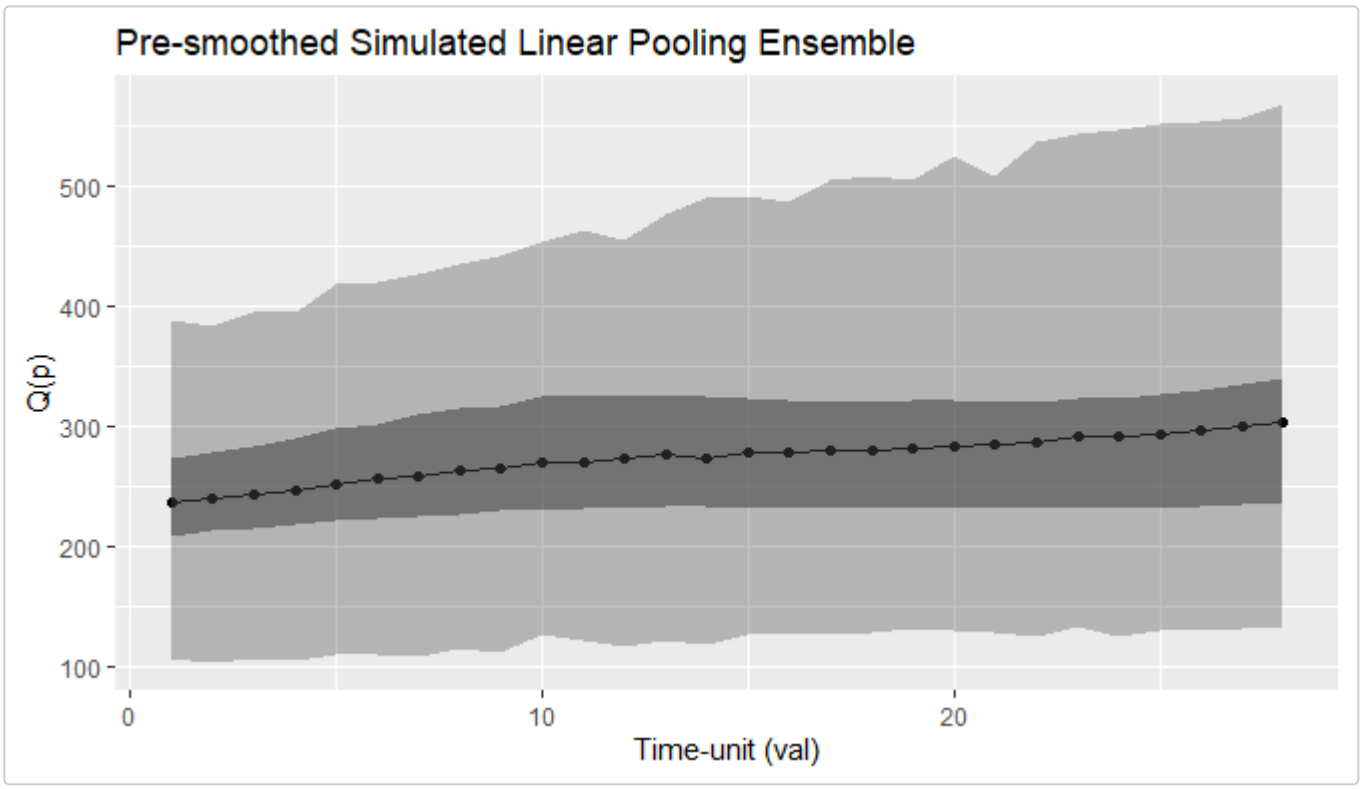
```
# Get simulate linear pooling of the pre-smoothed quantile functions (by model and time value)
ps_slp <- generate_slp_ensemble(
  generate_slp_predictions(
    x = ps_data,
    qp_var="outcomes_hat",
    p_var = "quantile",
    byvars = c("model","val")
  ),
  byvars = "val"
)
#> Running 224 gam models.. will take some time
#> Complete in 2.22 seconds.
```

We now have a an estimate of the combined quantile function across models, for each of the time_points. We can plot these over time, with desired prediction intervals. To do this, we manipulate the dataset to swing it wide, so that `geom_ribbon()` from the `ggplot2` package can be used to show these intervals

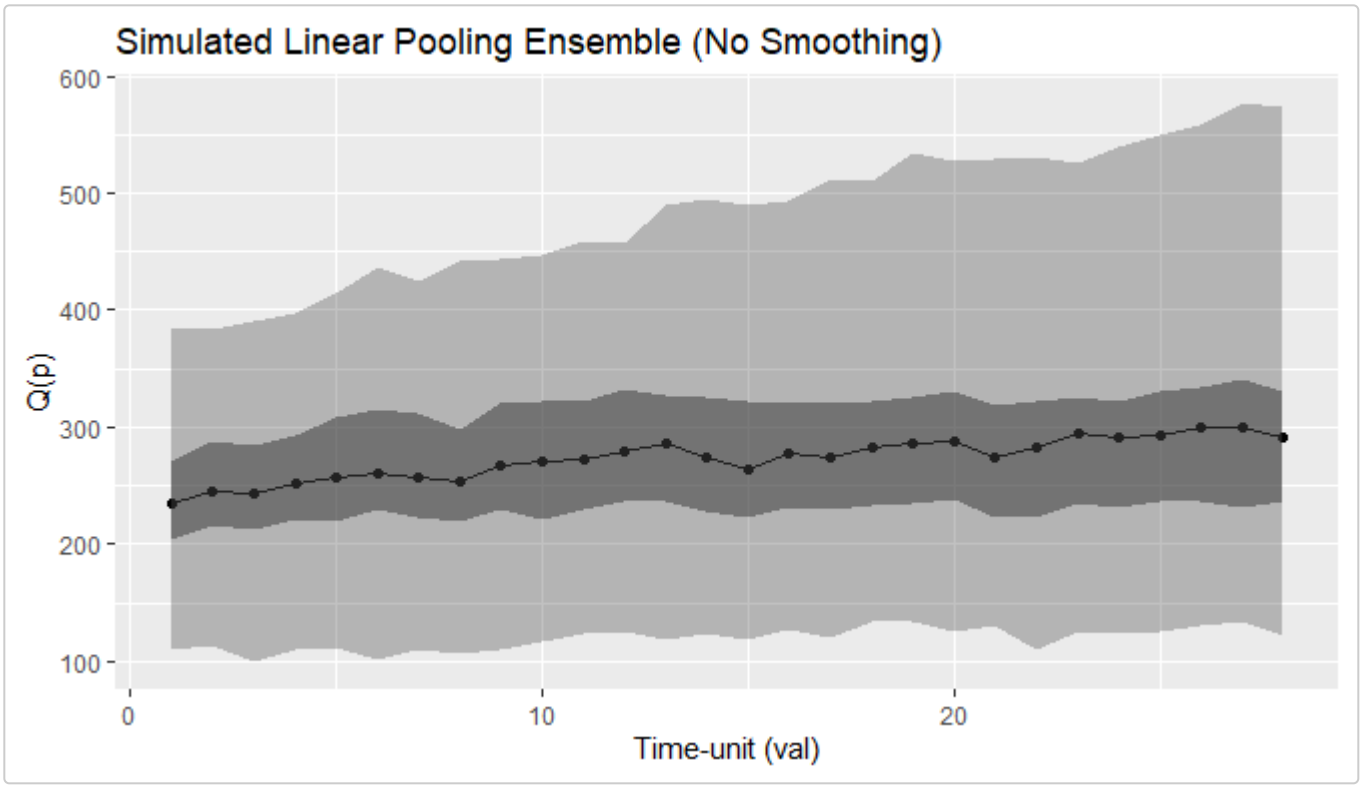
```
ps_slp_wide <- dcast(
  data = ps_slp[as.character(quantile) %in% c("0.025","0.25","0.5","0.75","0.975")],
  formula = val~quantile,
  value.var = "predictions"
)

ps_slp_wide %>% head()
#>    val    0.025     0.25      0.5     0.75    0.975
#> 1:   1 107.6871 209.2302 236.3742 273.9179 388.4930
#> 2:   2 103.8197 213.0240 239.9957 278.7834 383.3562
#> 3:   3 106.3881 215.9800 243.1379 283.4129 395.1320
#> 4:   4 105.8158 219.2031 247.7912 289.8374 394.9425
#> 5:   5 111.1180 222.2215 252.6983 298.0638 419.3160
#> 6:   6 109.8713 223.4168 256.3217 302.8456 421.1236
```

We then plot the ensemble over time



Notice the difference between the ensemble pre-smoothed above, and if we had not pre-smoothed:


Simulated Linear Pooling Ensemble (No Smoothing)
600
500
400
300
200
100
Q(p)
0
10
20
Time-unit (val)