Analytics Report Shell

Dillon Labonte

February 16, 2025

Table of contents

```
# including necessary libraries
library(tidyverse)
library(tidymodels)
library(patchwork)
library(kableExtra)

# styling
options(kable_styling_bootstrap_options = c("hover", "striped"))
theme_set(theme_bw(base_size = 14))

# read in data and clean up column names
data <- read_csv("energy_dataset.csv")
names(data) <- janitor::make_clean_names(names(data))

# adding generation_total column to data set
data <- data |>
  mutate(generation_total = generation_biomass + generation_fossil_brown_coal_lignite + gene

# adding renewable_generation_total column to data set
data <- data |>
  mutate(renewable_generation_total = generation_biomass + generation_geothermal + generatio

# adding nonrenewable_generation_total column to data set
data <- data |>
  mutate(nonrenewable_generation_total = generation_fossil_brown_coal_lignite + generation_f

# adding hour_of_day column to data set
data <- data |>
  mutate(hour_of_day = hour(time))

# adding month_of_year column to data set
```

```r
data <- data |>
  mutate(month_of_year = month(time, label = TRUE, abbr = TRUE))

# calculating RMSE of TSO model for price_day_ahead
tso_rmse <- data |>
  rmse(price_day_ahead, price_actual)

# split data into training, test, and validation sets
set.seed(1234567890)
data_splits <- initial_split(data, prop = 0.75)

train <- training(data_splits)
temp <- testing(data_splits)

set.seed(11111)
test_splits <- initial_split(temp, prop = 0.5)
validation <- training(test_splits)
test <- testing(test_splits)
```

```r
# creating a predictors data set to use for model building
predictors <- train |>
  select(-c(time, generation_hydro_pumped_storage_aggregated, forecast_wind_offshore_eday_ah
```

Statement of Purpose

This project seeks to develop a regression model to predict day ahead electrical power prices by analyzing energy generation, demand, and weather forecasts. These predictions support grid operators, producers, and consumers in maintaining grid stability and optimizing energy use, especially as renewable energy sources grow in popularity.

Executive Summary

In order to assist grid operators, energy producers, and customers in making well-informed decisions, this project aimed to predict electricity prices for the following day. Reliable pricing predictions reduce issues brought on by the fluctuations of renewable energy sources, improve grid stability, and optimize generation and consumption. Using an extensive dataset that included hourly electricity prices, generation, demand, and weather data from 2015 to 2018, the project focused on Spain's electrical grid, with implications in other energy grids around the world.

A number of predictive models were constructed and assessed. The TSO model, a baseline model constructed by ENTSOE, produced a high RMSE of 13.25. We created three models in response: linear regression, random forest, and extreme gradient boosting. Hyper parameter tuning was used to maximize the

predictability of these models. With an RMSE of 3.44 on validation data, the extreme gradient boosting model performed the best, allowing predictions to be within plus or minus $7 of actual prices.

The results of this analysis shows that extreme gradient boosting and other advanced machine learning models provide significant improvements in prediction accuracy. These results highlight how regression can improve grid sustainability and reliability in a market driven by ever increasing renewable energy generation. These results offer useful insights in other global electrical grids.

Introduction

The price of electrical power is a constantly changing variable that profoundly impacts society. Being able to predict electrical power prices a day in advance would allow for increased grid stability which is necessary to maintain a delicate balance of electrical supply and demand that the grid requires. These predictions allow producers of electricity to decide how much power to produce and when, and industrial consumers to decide when to use power for energy intensive processes.

In this day and age renewable energies are becoming more popular and their power production and usage is ever increasing. However, renewable energy sources are variable and often rely on weather conditions. Accurate predictions of day ahead prices will allow grid operators to deploy backup power sources, such as fossil fuels, when needed during renewable energy fluctuations.

This project aims to build a regression model to predict the electrical power prices a day in advance by analyzing generation, demand, and weather forecasts. These predictions support grid operators, producers, and consumers in their decision making, improving the reliability and sustainability of the electrical energy market.

This project focuses on Spain's electrical grid with hourly data over the course of 4 years, from 2015 through 2018. Insights gained from this analysis of Spain's grid will be useful when considering electrical grids of other countries that are also incorporating large scale renewable energy generation.

Exploratory Data Analysis

This data set was obtained from the European Network of Transmission System Operators of Electricity (ENTSOE) [1]. The transmission system operator (TSO) uses models to forecast day ahead electrical load and electricity prices. Let's look at the root mean squared error (RMSE) of the TSO model predictions for day ahead price, compared to the actual price.

```
tso_rmse |>
  kable() |>
  kable_styling()
```

.metric	.estimator	.estimate
rmse	standard	13.24986

This RMSE is quite bad. This indicates that roughly 95% of the true day ahead prices will fall within 2 RMSEs, or about $26, of the true value. This is on the same order of magnitude as the electrical prices, often in the $50 or $60 range.

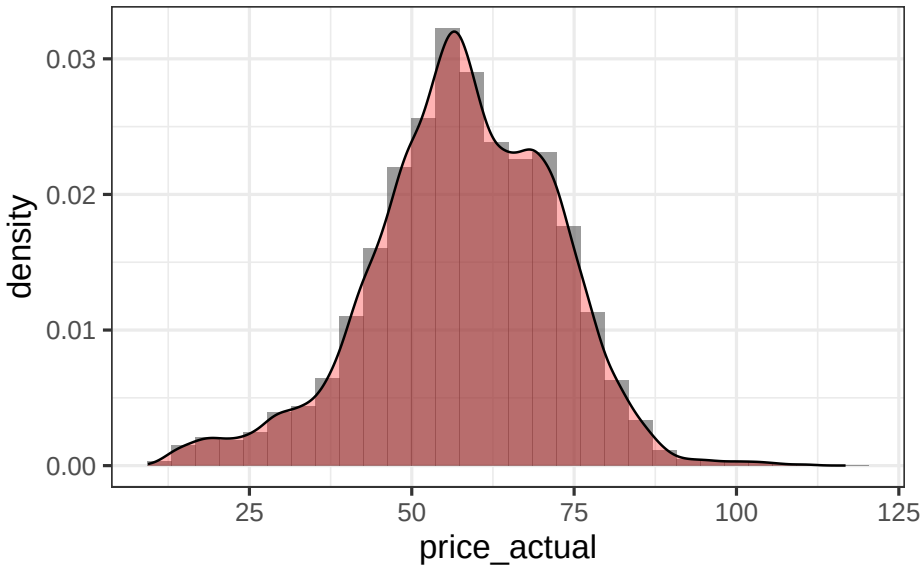
Let's look at the first 6 rows of our dataset to get an idea of what we're dealing with before making models.

```
train |>
  head() |>
  kable() |>
  kable_styling()
```

time	generation_biomass	generation_fossil_brown_coal_lignite	generation_fossil_coa
2017-11-19 21:00:00	336	656	
2015-12-26 05:00:00	370	0	
2018-03-12 20:00:00	397	0	
2015-04-30 12:00:00	376	964	
2018-08-07 11:00:00	367	419	
2018-07-19 03:00:00	361	660	

Let's look at the distribution of our actual day ahead prices.

```
priceDayAheadDistribution <- train |>
  ggplot() +
  geom_histogram(aes(x = price_actual, y = after_stat(density)), alpha = 0.6) +
  geom_density(aes(x = price_actual), fill = 'red', alpha = 0.3)

priceDayAheadDistribution
```



Our response variable is normally distributed, so we can work with this response variable without making any changes.

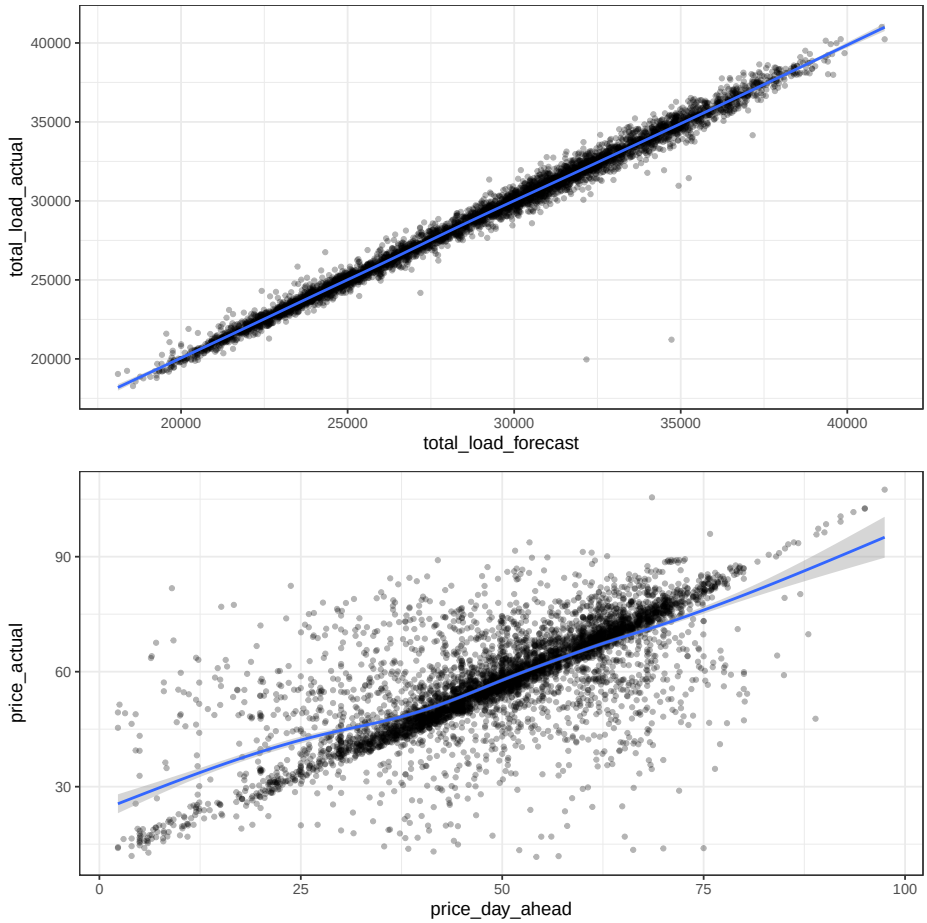
Let's build some plots and see if we can spot any visual trends.

```
plot1 <- train |>
  sample_frac(.2) |>
  ggplot() +
  geom_point(aes(x = generation_total, y = price_actual), alpha = 0.3) +
  geom_smooth(aes(x = generation_total, y = price_actual))

plot2 <- train |>
  sample_frac(.2) |>
  ggplot() +
  geom_point(aes(x = total_load_actual, y = price_actual), alpha = 0.3) +
  geom_smooth(aes(x = total_load_actual, y = price_actual))

plot3 <- train |>
  sample_frac(.2) |>
  ggplot() +
  geom_point(aes(x = total_load_forecast, y = price_actual), alpha = 0.3) +
  geom_smooth(aes(x = total_load_forecast, y = price_actual))

plot4 <- train |>
  sample_frac(.2) |>
  ggplot() +
  geom_point(aes(x = total_load_forecast, y = total_load_actual), alpha = 0.3) +
  geom_smooth(aes(x = total_load_forecast, y = total_load_actual))
```

```
plot5 <- train |>
  sample_frac(.2) |>
  ggplot() +
  geom_point(aes(x = price_day_ahead, y = price_actual), alpha = 0.3) +
  geom_smooth(aes(x = price_day_ahead, y = price_actual))

plot6 <- train |>
  sample_frac(.2) |>
  ggplot() +
  geom_point(aes(x = renewable_generation_total, y = price_actual), alpha = 0.3) +
  geom_smooth(aes(x = renewable_generation_total, y = price_actual))

plot7 <- train |>
  sample_frac(.2) |>
  ggplot() +
  geom_point(aes(x = nonrenewable_generation_total, y = price_actual), alpha = 0.3) +
  geom_smooth(aes(x = nonrenewable_generation_total, y = price_actual))

plot8 <- train|>
  sample_frac(.2) |>
  ggplot() +
  geom_boxplot(aes(group = hour_of_day, x = hour_of_day, y = price_actual)) +
  geom_smooth(aes(x = hour_of_day, y = price_actual))

plot9 <- train|>
  sample_frac(.2) |>
  ggplot() +
  geom_boxplot(aes(group = month_of_year, x = month_of_year, y = price_actual)) +
  geom_smooth(aes(x = month_of_year, y = price_actual))
```

The first two plots explore the TSO model predictions of total load and price, compared to their true values.

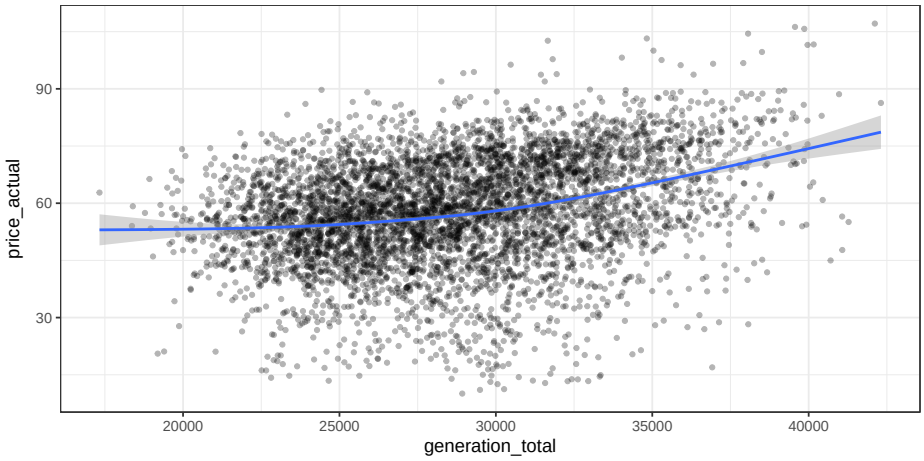
```
plot4 / plot5
```

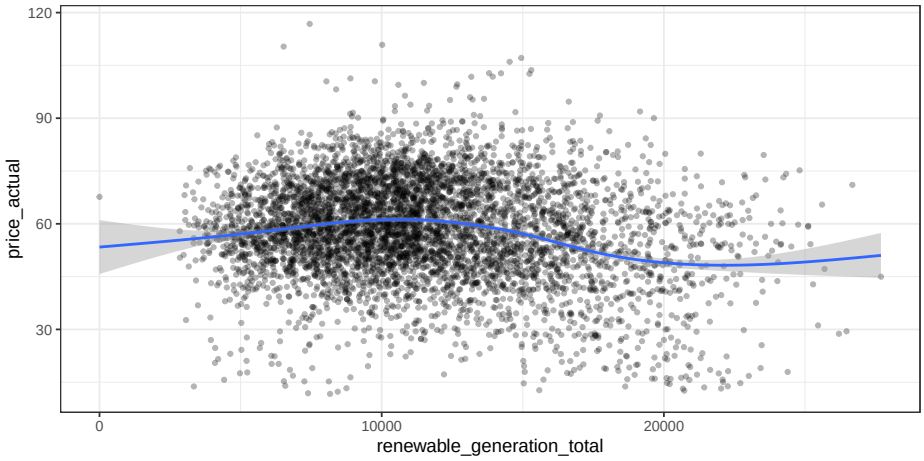


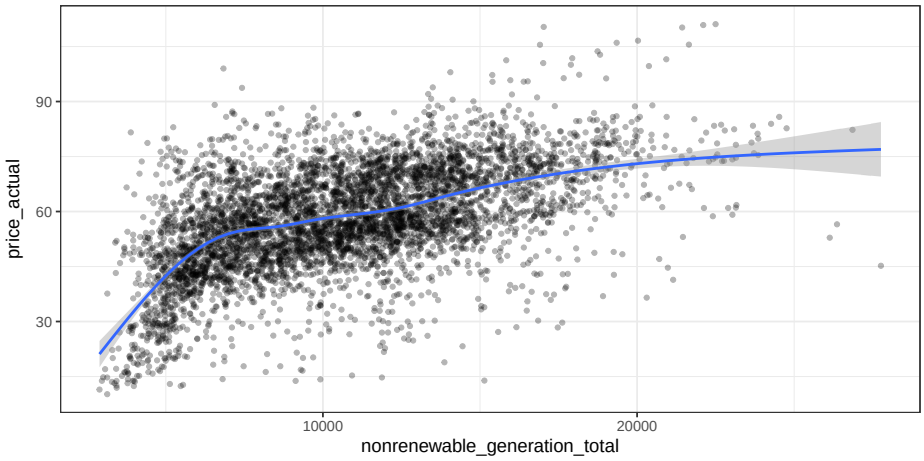
In the first plot, the predictions are very accurate and hug the line $y = x$, indicating that the model used to predict the total load is accurate and reliable. However, in the second plot we see that the predicted prices for the next day are much less accurate. It is clear they follow some positive relationship, however it is not necessarily linear. We will try to improve upon this with our models.

The following plots show the affect of generation, and variations of generation, and how they affect the day ahead price.

```
plot1 / plot6 / plot7
```





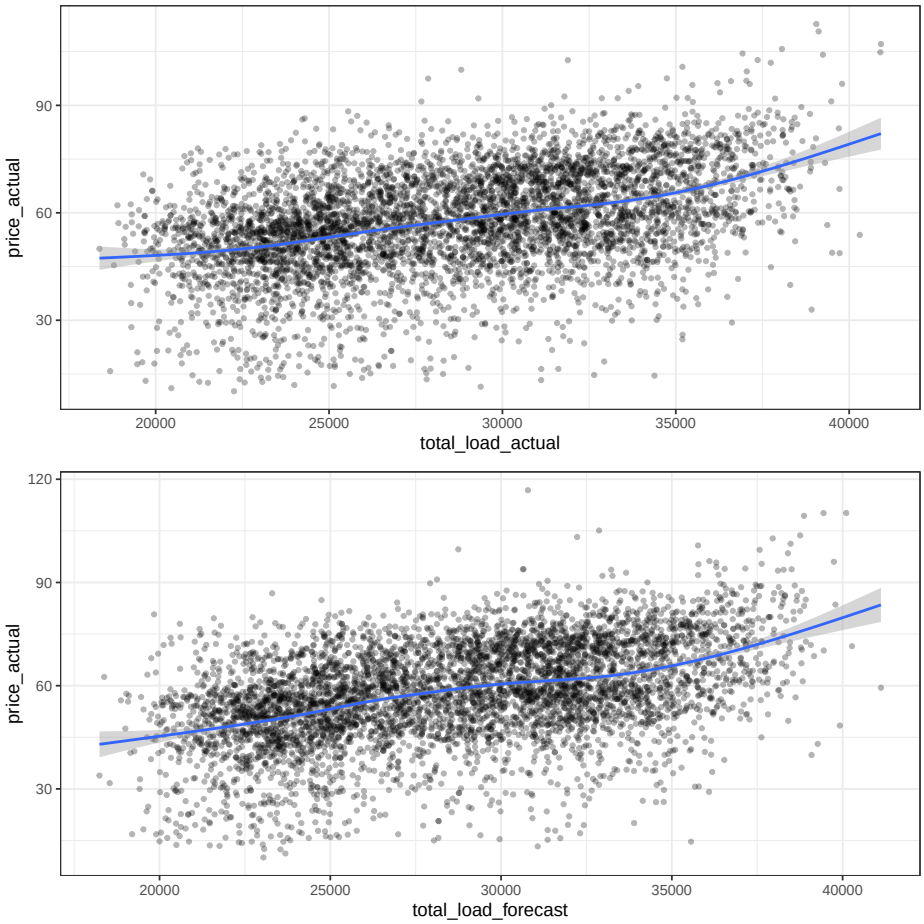


We see some interesting relationships in these plots. The `generation_total` and

the `nonrenewable_generation_total` appear to have a positive relationship, whereas the `renewable_generation_total` doesn't appear to have a strong positive or negative relationship.

The following plots show the affect of `total_load_actual` and `total_load_forecast` on the `price_actual` response. We assume that the TSO model to predict prices has access to the load forecast model, so we will include this predictor in our model construction.

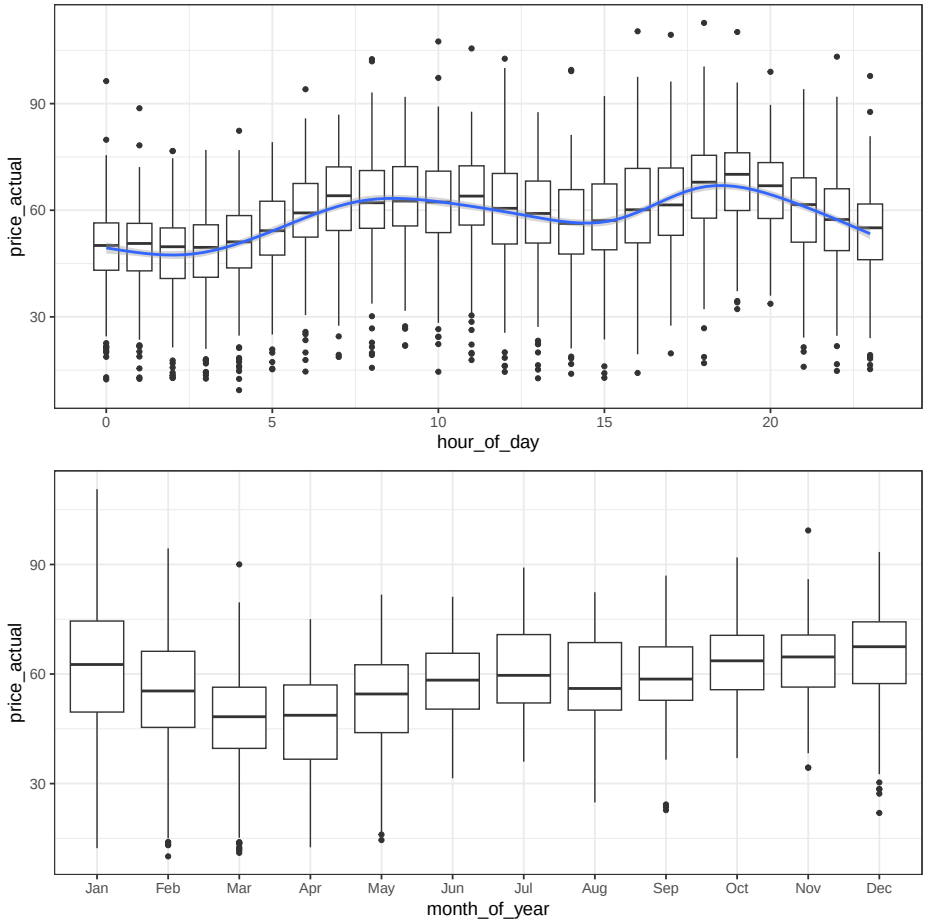
```
plot2 / plot3
```



Both of these plots seem to show a positive, somewhat linear relationship.

The following plots are quite interesting, and are built from extracting data from the `time` predictor in the dataset. By extracting `hour_of_day` and `month_of_year`, we can see how electricity prices change throughout the day and throughout the year.

```
plot8 / plot9
```



These plots show some really clear relationships with curvature. These strong relationships will certainly improve our model accuracy.

Model Construction

We will use cross validation throughout our model construction to improve our ability to make accurate decisions.

```
train_folds <- vfold_cv(train, v = 5)
```

Linear Regression Model

```
lin_reg_spec <- linear_reg(penalty = tune(), mixture = tune()) %>%
  set_engine("glmnet")
```

```
lin_reg_rec <- recipe(price_actual ~ ., data = predictors) %>%
  step_dummy(all_nominal_predictors()) |>
  step_other(all_nominal_predictors()) %>%
  step_impute_median(all_numeric_predictors())

lin_reg_wf <- workflow() %>%
  add_model(lin_reg_spec) %>%
  add_recipe(lin_reg_rec)

lin_reg_grid <- grid_regular(
  penalty(),
  mixture(),+
  levels = 10
)

n_cores <- parallel::detectCores()
cluster <- parallel::makeCluster(n_cores - 1, type = "PSOCK")
doParallel::registerDoParallel(cluster)

tictoc::tic()

lin_reg_tune_results <- lin_reg_wf %>%
  tune_grid(
    grid = lin_reg_grid,
    resamples = train_folds
  )

parallel::stopCluster(cluster)

tictoc::toc()
```

After using hyper paramater tuning to build a linear regression model, we look at the 10 best models it built in order of ascending RMSE.

```
#save(lin_reg_tune_results, lin_reg_wf, file = "lin_reg_tune_results.RData")
load("lin_reg_tune_results.RData")

lin_reg_tune_results %>%
  show_best(n = 10, metric = "rmse") |>
  kable() |>
  kable_styling()
```

penalty	mixture	.metric	.estimator	mean	n	std_err	.config
0.0059948	1.0000000	rmse	standard	10.29642	5	0.0478108	Preprocessor1_Model098
0.0059948	0.8888889	rmse	standard	10.29647	5	0.0478723	Preprocessor1_Model088

0.0059948	0.7777778	rmse	standard	10.29658	5	0.0479133	Preprocessor1_Model078
0.0000000	0.1111111	rmse	standard	10.29662	5	0.0481989	Preprocessor1_Model011
0.0000000	0.1111111	rmse	standard	10.29662	5	0.0481989	Preprocessor1_Model012
0.0000000	0.1111111	rmse	standard	10.29662	5	0.0481989	Preprocessor1_Model013
0.0000002	0.1111111	rmse	standard	10.29662	5	0.0481989	Preprocessor1_Model014
0.0000028	0.1111111	rmse	standard	10.29662	5	0.0481989	Preprocessor1_Model015
0.0000359	0.1111111	rmse	standard	10.29662	5	0.0481989	Preprocessor1_Model016
0.0004642	0.1111111	rmse	standard	10.29662	5	0.0481989	Preprocessor1_Model017

```
best_lin_reg <- lin_reg_tune_results %>%
  select_best(metric = "rmse")

lin_reg_wf_final <- lin_reg_wf %>%
  finalize_workflow(best_lin_reg)

lin_reg_fit <- lin_reg_wf_final %>%
  fit(train)
```

We fit the best model to our training data, and calculate the RMSE.

```
#save(lin_reg_fit, file = "lin_reg_fit.RData")
load("lin_reg_fit.RData")

lin_reg_predictions_train <- lin_reg_fit |>
  augment(train) |>
  select(.pred, price_actual) |>
  rename(.pred_lin_reg_train = .pred)

lin_reg_predictions_train |>
  rmse(.pred_lin_reg_train, price_actual) |>
  kable() |>
  kable_styling()
```

.metric	.estimator	.estimate
rmse	standard	10.28414

On the training data, we see that this linear regression model has an RMSE of 10.28, which is already outperforming the TSO RMSE of 13.25.

Let's make predictions on the `validation` data set.

```
lin_reg_predictions_validation <- lin_reg_fit |>
  augment(validation) |>
  select(.pred, price_actual) |>
  rename(.pred_lin_reg_validation = .pred)
```

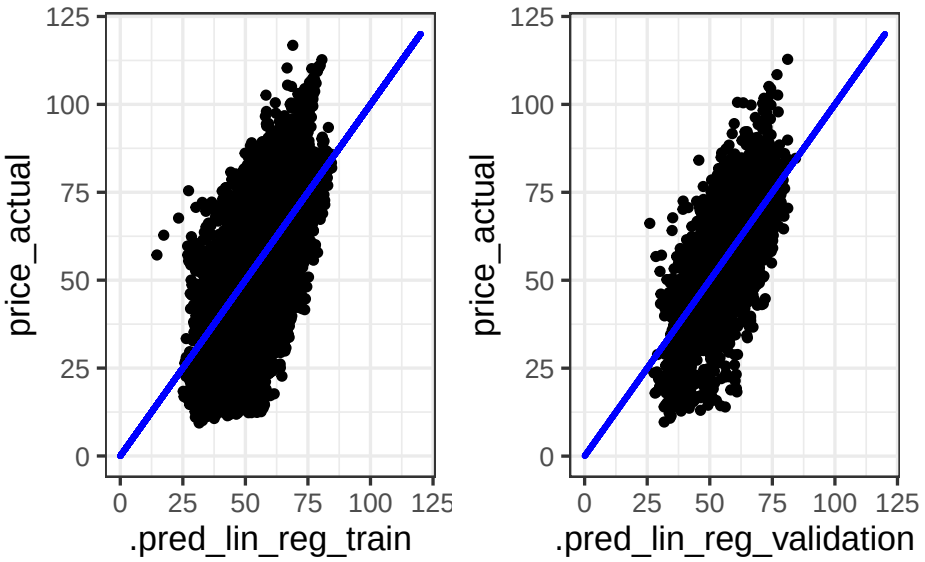
```
lin_reg_predictions_validation |>
  rmse(.pred_lin_reg_validation, price_actual) |>
  kable() |>
  kable_styling()
```

.metric	.estimator	.estimate
rmse	standard	10.21945

We see that our RMSE on the `validation` data is 10.22, which interestingly is better than the training data.

Let's plot our predictions against the true values.

```
xVals1 <- runif(26298, min = 0, max = 120)
yVals1 <- xVals1
#linedata <- tibble(x = xVals, y = yVals)

xVals2 <- runif(4383, min = 0, max = 120)
yVals2 <- xVals2

lin_reg_plot_train <- lin_reg_predictions_train |>
  mutate(xVals1, yVals1) |>
  ggplot() +
  geom_point(aes(x = .pred_lin_reg_train, y = price_actual)) +
  geom_point(aes(x = xVals1, y = yVals1), color = 'blue', size = 0.5)

lin_reg_plot_validation <- lin_reg_predictions_validation |>
  mutate(xVals2, yVals2) |>
  ggplot() +
  geom_point(aes(x = .pred_lin_reg_validation, y = price_actual)) +
  geom_point(aes(x = xVals2, y = yVals2), color = 'blue', size = 0.5)

lin_reg_plot_train + lin_reg_plot_validation
```



From these plots, we can see that this model is outperforming the TSO model, but still can be improved.

Random Forest Model

```
rf_spec <- rand_forest(mtry = tune(), trees = tune()) %>%
  set_engine("ranger") %>%
  set_mode("regression")

rf_rec <- recipe(price_actual ~ ., data = predictors) %>%
  step_dummy(all_nominal_predictors()) |>
  step_other(all_nominal_predictors()) %>%
  step_impute_median(all_numeric_predictors())

rf_wf <- workflow() %>%
  add_model(rf_spec) %>%
  add_recipe(rf_rec)

rf_grid <- grid_regular(
  mtry(range = c(1, 10)),
  trees(),
  levels = 5
)

n_cores <- parallel::detectCores()
cluster <- parallel::makeCluster(n_cores - 1, type = "PSOCK")
doParallel::registerDoParallel(cluster)
```

```
tictoc::tic()

rf_tune_results <- rf_wf %>%
  tune_grid(
    grid = rf_grid,
    resamples = train_folds
  )

parallel::stopCluster(cluster)

tictoc::toc()
```

After using hyperparamater tuning to build a random forest model, we show the best 10 models in order of ascending RMSE.

```
#save(rf_tune_results, rf_wf, file = "rf_tune_results.RData")
load("rf_tune_results.RData")

rf_tune_results %>%
  show_best(n = 10, metric = "rmse") |>
  kable() |>
  kable_styling()
```

mtry	trees	.metric	.estimator	mean	n	std_err	.config
10	2000	rmse	standard	4.570574	5	0.0472913	Preprocessor1_Model25
10	1500	rmse	standard	4.571607	5	0.0477593	Preprocessor1_Model20
10	1000	rmse	standard	4.572664	5	0.0472862	Preprocessor1_Model15
10	500	rmse	standard	4.588664	5	0.0468897	Preprocessor1_Model10
7	2000	rmse	standard	4.633177	5	0.0488629	Preprocessor1_Model24
7	1500	rmse	standard	4.635829	5	0.0483290	Preprocessor1_Model19
7	1000	rmse	standard	4.638231	5	0.0453621	Preprocessor1_Model14
7	500	rmse	standard	4.641607	5	0.0442504	Preprocessor1_Model09
5	1500	rmse	standard	4.782919	5	0.0448245	Preprocessor1_Model18
5	2000	rmse	standard	4.783133	5	0.0454290	Preprocessor1_Model23

```
best_rf <- rf_tune_results %>%
  select_best(metric = "rmse")

rf_wf_final <- rf_wf %>%
  finalize_workflow(best_rf)

rf_fit <- rf_wf_final %>%
  fit(train)
```

We fit our best model to the training data, and calculate the RMSE.

```
#save(rf_fit, file = "rf_fit.RData")
load("rf_fit.RData")

rf_predictions_train <- rf_fit |>
  augment(train) |>
  select(.pred, price_actual) |>
  rename(.pred_rf_train = .pred)

rf_predictions_train |>
  rmse(.pred_rf_train, price_actual) |>
  kable() |>
  kable_styling()
```

.metric	.estimator	.estimate
rmse	standard	1.772513

On the training data, we see a RMSE of 1.77! This is quite a bit better than the linear regression model and the TSO model.

Let's make predictions for our `validation` dataset.

```
rf_predictions_validation <- rf_fit |>
  augment(validation) |>
  select(.pred, price_actual) |>
  rename(.pred_rf_validation = .pred)

rf_predictions_validation |>
  rmse(.pred_rf_validation, price_actual) |>
  kable() |>
  kable_styling()
```

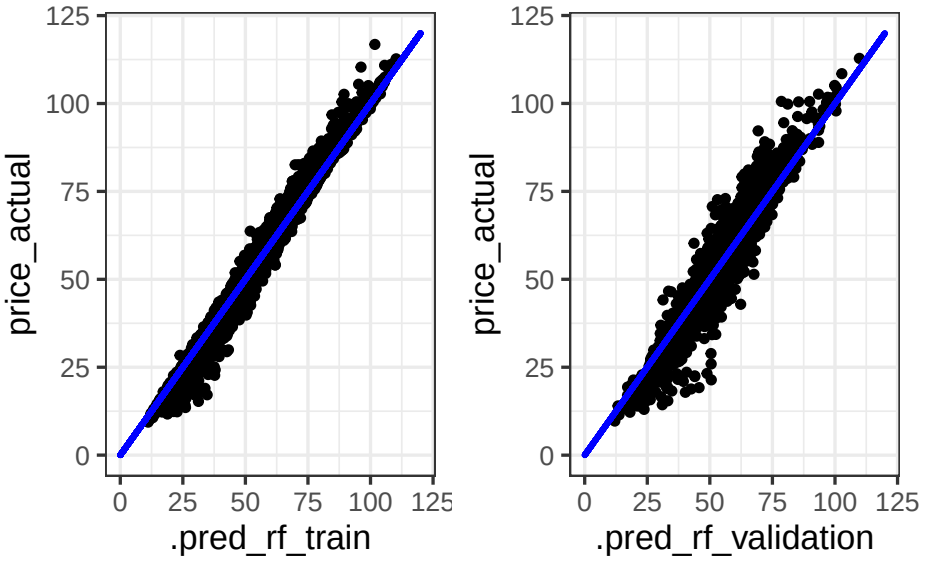
.metric	.estimator	.estimate
rmse	standard	4.118898

The RMSE for our `validation` dataset was 4.12, which is worse than the RMSE on the training data, however it is still significantly better than the RMSE of the linear regression and the TSO models.

Let's plot our predictions compared to the true day ahead prices.

```
xVals1 <- runif(26298, min = 0, max = 120)
yVals1 <- xVals1
#linedata <- tibble(x = xVals, y = yVals)
```

```r
xVals2 <- runif(4383, min = 0, max = 120)
yVals2 <- xVals2

rf_plot_train <- rf_predictions_train |>
  mutate(xVals1, yVals1) |>
  ggplot() +
  geom_point(aes(x = .pred_rf_train, y = price_actual)) +
  geom_point(aes(x = xVals1, y = yVals1), color = 'blue', size = 0.5)

rf_plot_validation <- rf_predictions_validation |>
  mutate(xVals2, yVals2) |>
  ggplot() +
  geom_point(aes(x = .pred_rf_validation, y = price_actual)) +
  geom_point(aes(x = xVals2, y = yVals2), color = 'blue', size = 0.5)

rf_plot_train + rf_plot_validation
```



We see that the predictions from the random forest model are significantly better, and are getting closer and closer to the line $y = x$, the ideal model. We will see if we can improve upon this in our last model class, extreme gradient boosting.

Extreme Gradient Boosting Model

```r
xgb_spec <- boost_tree(mtry = tune(), trees = tune(), learn_rate = tune()) %>%
  set_engine("xgboost") %>%
  set_mode("regression")
```

```
xgb_rec <- recipe(price_actual ~ ., data = predictors) %>%
  step_dummy(all_nominal_predictors()) |>
  step_other(all_nominal_predictors()) %>%
  step_impute_median(all_numeric_predictors())

xgb_wf <- workflow() %>%
  add_model(xgb_spec) %>%
  add_recipe(xgb_rec)

xgb_grid <- grid_regular(
  mtry(range = c(1, 10)),
  trees(),
  learn_rate(),
  levels = 5
)

n_cores <- parallel::detectCores()
cluster <- parallel::makeCluster(n_cores - 1, type = "PSOCK")
doParallel::registerDoParallel(cluster)

tictoc::tic()

xgb_tune_results <- xgb_wf %>%
  tune_grid(
    grid = xgb_grid,
    resamples = train_folds
  )

parallel::stopCluster(cluster)

tictoc::toc()
```

After using hyperparamater tuning to build an extreme gradient boosting model, we show the best 10 models in order of ascending RMSE.

```
#save(xgb_tune_results, xgb_wf, file = "xgb_tune_results.RData")
load("xgb_tune_results.RData")

xgb_tune_results %>%
  show_best(n = 10, metric = "rmse") |>
  kable() |>
  kable_styling()
```

mtry	trees	learn_rate	.metric	.estimator	mean	n	std_err	.config
7	2000	0.1	rmse	standard	3.790247	5	0.0350662	Preprocessor1_Model100

5	2000	0.1	rmse	standard	3.795045	5	0.0429371	Preprocessor1_Model075
10	2000	0.1	rmse	standard	3.803137	5	0.0324896	Preprocessor1_Model125
3	2000	0.1	rmse	standard	3.820950	5	0.0446594	Preprocessor1_Model050
7	1500	0.1	rmse	standard	3.858999	5	0.0386719	Preprocessor1_Model099
10	1500	0.1	rmse	standard	3.862962	5	0.0337291	Preprocessor1_Model124
5	1500	0.1	rmse	standard	3.876833	5	0.0429419	Preprocessor1_Model074
3	1500	0.1	rmse	standard	3.924880	5	0.0451926	Preprocessor1_Model049
10	1000	0.1	rmse	standard	3.986085	5	0.0354970	Preprocessor1_Model123
7	1000	0.1	rmse	standard	4.006600	5	0.0403066	Preprocessor1_Model098

```
best_xgb <- xgb_tune_results %>%
  select_best(metric = "rmse")

xgb_wf_final <- xgb_wf %>%
  finalize_workflow(best_xgb)

xgb_fit <- xgb_wf_final %>%
  fit(train)
```

We fit our best model to the training data, and calculate the RMSE.

```
#save(xgb_fit, file = "xgb_fit.RData")
load("xgb_fit.RData")

xgb_predictions_train <- xgb_fit |>
  augment(train) |>
  select(.pred, price_actual) |>
  rename(.pred_xgb_train = .pred)

xgb_predictions_train |>
  rmse(.pred_xgb_train, price_actual) |>
  kable() |>
  kable_styling()
```

.metric	.estimator	.estimate
rmse	standard	0.988913

On our training data, we see an RMSE of 0.99!! This is significantly better than any of the previously built models, including the TSO model.

Let's look at the RMSE for the `validation` dataset.

```
xgb_predictions_validation <- xgb_fit |>
  augment(validation) |>
  select(.pred, price_actual) |>
```

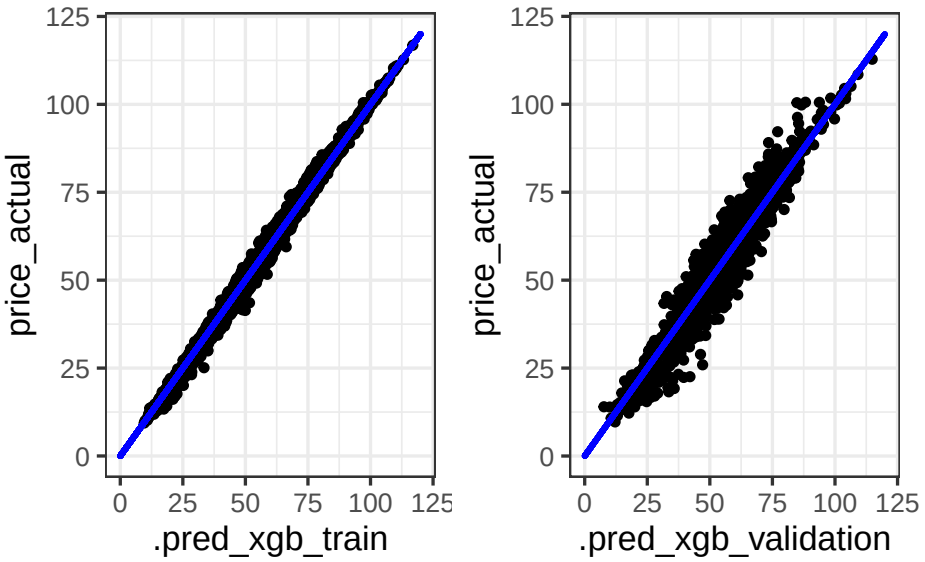
```
  rename(.pred_xgb_validation = .pred)

xgb_predictions_validation |>
  rmse(.pred_xgb_validation, price_actual) |>
  kable() |>
  kable_styling()
```

.metric	.estimator	.estimate
rmse	standard	3.439827

On our `validation` data we see a RMSE of 3.44. This is definitely worse than the RMSE on the training data, however it is still significantly better than the RMSE of the other models tested on the `validation` set.

Let's plot our predictions compared to the true day ahead prices.

```
xgb_plot_train <- xgb_predictions_train |>
  mutate(xVals1, yVals1) |>
  ggplot() +
  geom_point(aes(x = .pred_xgb_train, y = price_actual)) +
  geom_point(aes(x = xVals1, y = yVals1), color = 'blue', size = 0.5)

xgb_plot_validation <- xgb_predictions_validation |>
  mutate(xVals2, yVals2) |>
  ggplot() +
  geom_point(aes(x = .pred_xgb_validation, y = price_actual)) +
  geom_point(aes(x = xVals2, y = yVals2), color = 'blue', size = 0.5)

xgb_plot_train + xgb_plot_validation
```



This model is even better than the random forest model, and this can be seen in the plot. Our predictions are getting closer and closer to the line $y = x$ and our RMSE has improved drastically.

Model Assessment

TSO Model

The TSO model created by ENTSOE was used as a baseline for comparison with our constructed models. This model had an RMSE of 13.25 on the complete data set, indicating that 95% of our predictions would fall within plus or minus 2 RMSEs, about $26. This is quite bad as the electricity prices were generally in the $50 or $60 range. We greatly improve upon this in our constructed models.

Linear Regression Model

The linear regression model was built using hyper parameter tuning. The best model had a mixture of 1.0 and a penalty of 0.0059948. This model had an RMSE of 10.28 on the training data, and 10.22 on the validation data. This RMSE is already quite a bit better than the TSO model, but plots of our predictions compared to the true values indicate that the model can still be improved. While linear regression models might not give us the most accurate predictions for such complex relationships in the real world, their interpretability is unmatched by other model classes.

Let's look at our estimates for our linear regression model.

```
lin_reg_fit |>
  tidy() |>
```