Random Forest Weather Data

Samantha Bothwell

6/11/2019

Random Forest Background

The random forest algorithm is a form of supervised learning. Random Forests are a form of decision trees. In the algorithm, trees are built by choosing a random subset of variables and then the model averages the predicitons of all the trees.

The random forest algorithm is most often used when there is a mixture of numeric and factor variables. Before using the random forest algorithm, here are some pros and cons to consider:

Pros:

- You can obtain a reliable variable importance plot that ranks the variables by how helpful they are in predicting the response.
- The algorithm is effectively able to fill in missing information based on what information is present.
- The estimate for the generalization error progresses as the forest builds, and is unbiased.

Cons:

- This is a black box algorithm - meaning, though it is powerful for predicting, it is difficult to interpret.
- When working with a dataset with significant noise, the algorithm will overfit.
- Variable importance is biased towards categorical variables with more levels.
- Due to overfitting, the algorithm performs poorly with predicting future data if there is any seasonality.

Clean Data

The data we will be using was taken from http://www.atmos.colostate.edu/fccwx/fccwx_latest.php, a record of climate observations for Fort Collins, CO. The data was taken hourly from June 2018 - June 2019. Later we will look at data from June 2014 - June 2019. We will remove the first column that represents the time the data point was taken.

```r
# First we read in the data and clean it
setwd("/Users/sbothwell/Desktop/DATAS/My_material")
dat = read.csv("Weather.csv", stringsAsFactors = FALSE)

# Remove first two columns
dat = dat[,-c(1,2)]

# Rename columns
colnames(dat) = c("Temp_degF","Humidity_Pct","DewPt_degF","Wind_mph",
                  "WindDir_degNorth","Gust_mph","GustDir_degNorth","Pressure_Hg",
                  "Solar_WatPerSqM","Percipitation_in")

# remove any NA's from the dataset
dat = dat[complete.cases(dat),]
```

Analysis - Filling in missing data

Now that our data is clean, let's walk through using the Random Forest Algorithm. First, we will make data sets for testing and training. These are random samples from our original data. We will use the training data to make the model and the testing data to evaluate how effective our model is at making predictions. For this data, I will make my training data 75% of the original data set and testing data the other 25% of the original data set.

```r
set.seed(2019) # set seed for reproducible results

## 75% of the sample size
smp_size <- floor(0.75 * nrow(dat))
train_ind <- sample(seq_len(nrow(dat)), size = smp_size)

train <- dat[train_ind, ]
test <- dat[-train_ind, ]
```

For the model we will try to predict the temperature based on the remaining variables.

```r
# Random Forest algorithm
model = randomForest(`Temp_degF`~., data = train)
```

Visuals

There are many visuals we can make from Random Forest to understand our data better. The visuals I will give examples of are:

- Decision Tree
- Variable Importance Plot
- Partial Plot

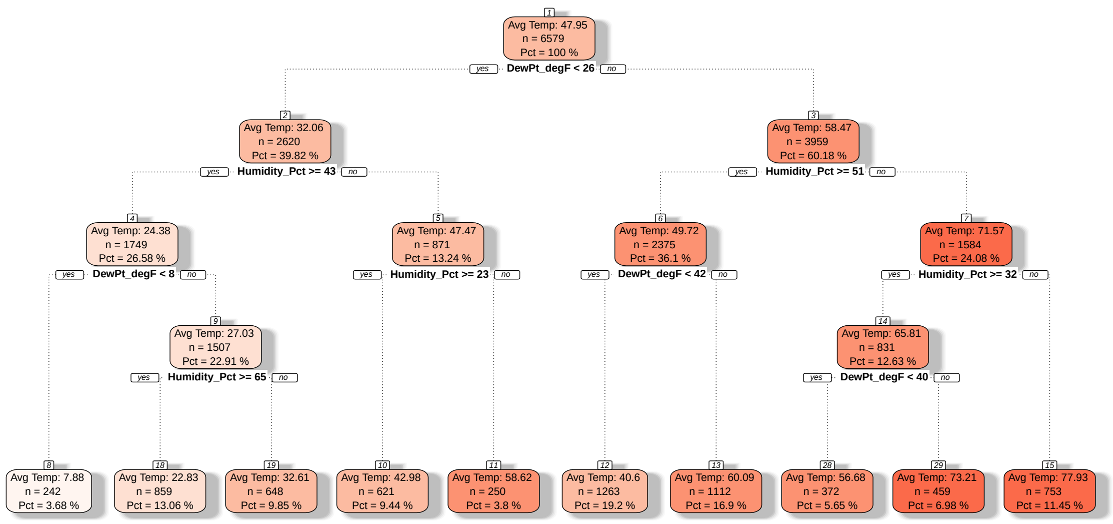
Decision Tree

We can view one of the trees from the Forest.

```r
# make a function to print certain text in tree nodes
nodefun = function(x, labs, digits, varlen){
  paste("Avg Temp:", round(x$frame$yval,2), "\nn =", x$frame$n, "  ",
        "\nPct =", round(x$frame$n/x$frame$n[1]*100,2),"%")
}

# plot decision tree
tree = rpart(model, train)
fancyRpartPlot(tree, palettes=c("Reds"), main="Decision Tree Graph", sub="",
               yesno = 2, node.fun = nodefun)
```

Decision Tree Graph



This tree shows how the algorithm splits observations into categories of temperatures. Each node displays the average temperature of the observations that fall in that category, the number of observations, and the percentage of the observations that fall into that category. First the algorithm looks at the Dew Point and determines that if the observation has a dew point less than 26^oF, it has a lower temperature. Next, the algorithm splits observations based on Humidity Percentage.

Variable Importance Plot

With the randomForest package, you can create a variable importance plot with the following command:

```r
varImpPlot(model)
```

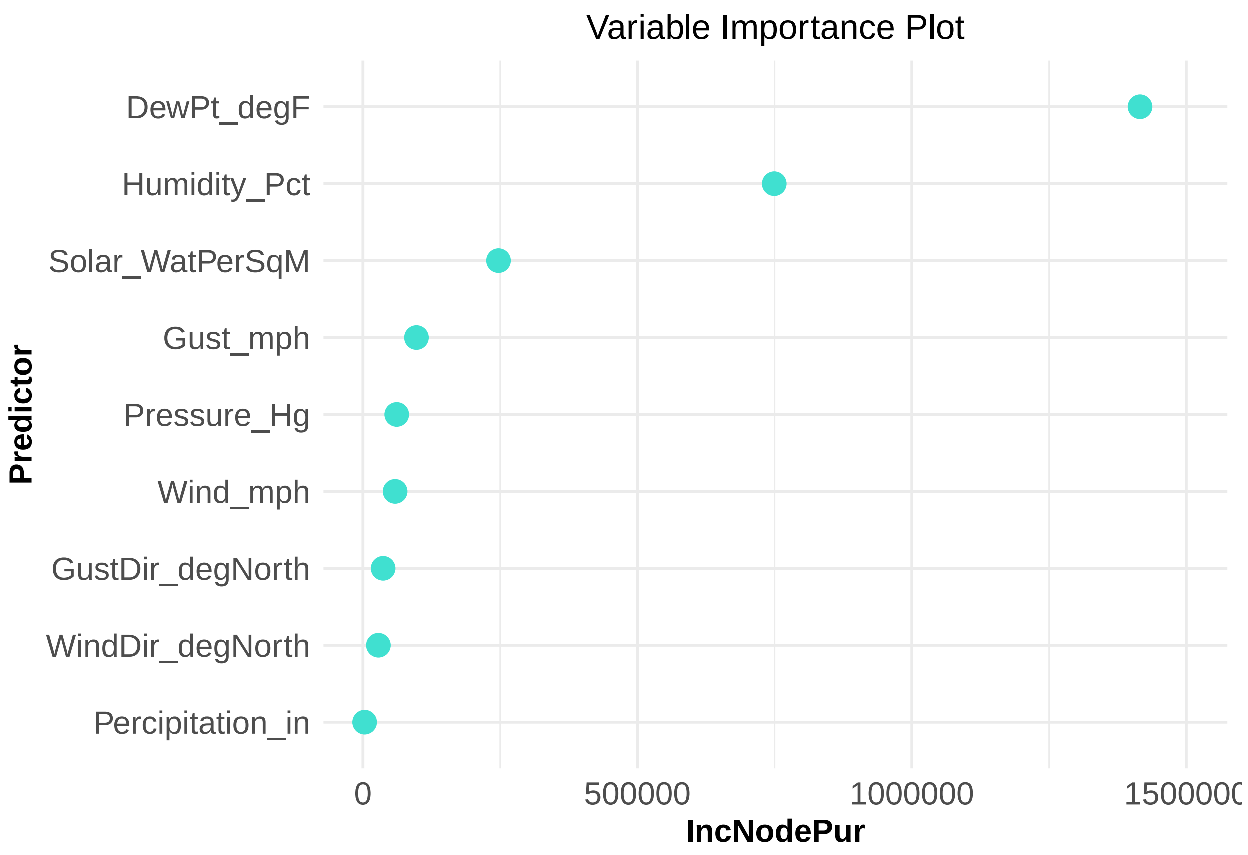
While that is the easiest way to make the plot, I prefer more style to my plots so the following code shows how I generate a variable importance plot:

```r
IncNodePur = model$importance[1:9]
names = rownames(model$importance)

Imp = as.data.frame(cbind(names, IncNodePur))
Imp$IncNodePur = as.numeric(as.character(Imp$IncNodePur))

# Rearrange variables by order of importance
Imp$names = factor(Imp$names, levels = Imp$names[order(Imp$IncNodePur)])
Imp = Imp[order(-IncNodePur),]

p = ggplot(Imp, aes(IncNodePur, names)) +
  geom_point(size = 4, shape = 16, col = "turquoise") +
  xlim(3000, 1500000) + ylab("Predictor") + labs(title = "Variable Importance Plot") +
  theme_minimal() + theme(plot.title = element_text(hjust = 0.5)) +
  theme(axis.text = element_text(size = 12),
        axis.title = element_text(size = 12, face = "bold"))
p
```



Based on the variable importance plot we see that Dew Point (in ^oF) is the most important variable in telling us about what the temperature will be. The next two important variables are the Humidity Perctentage and Solar (W/m^2). This follows with the decision tree that used Dew Point and Humidity Percentage to classify temperatures.

Partial Plots

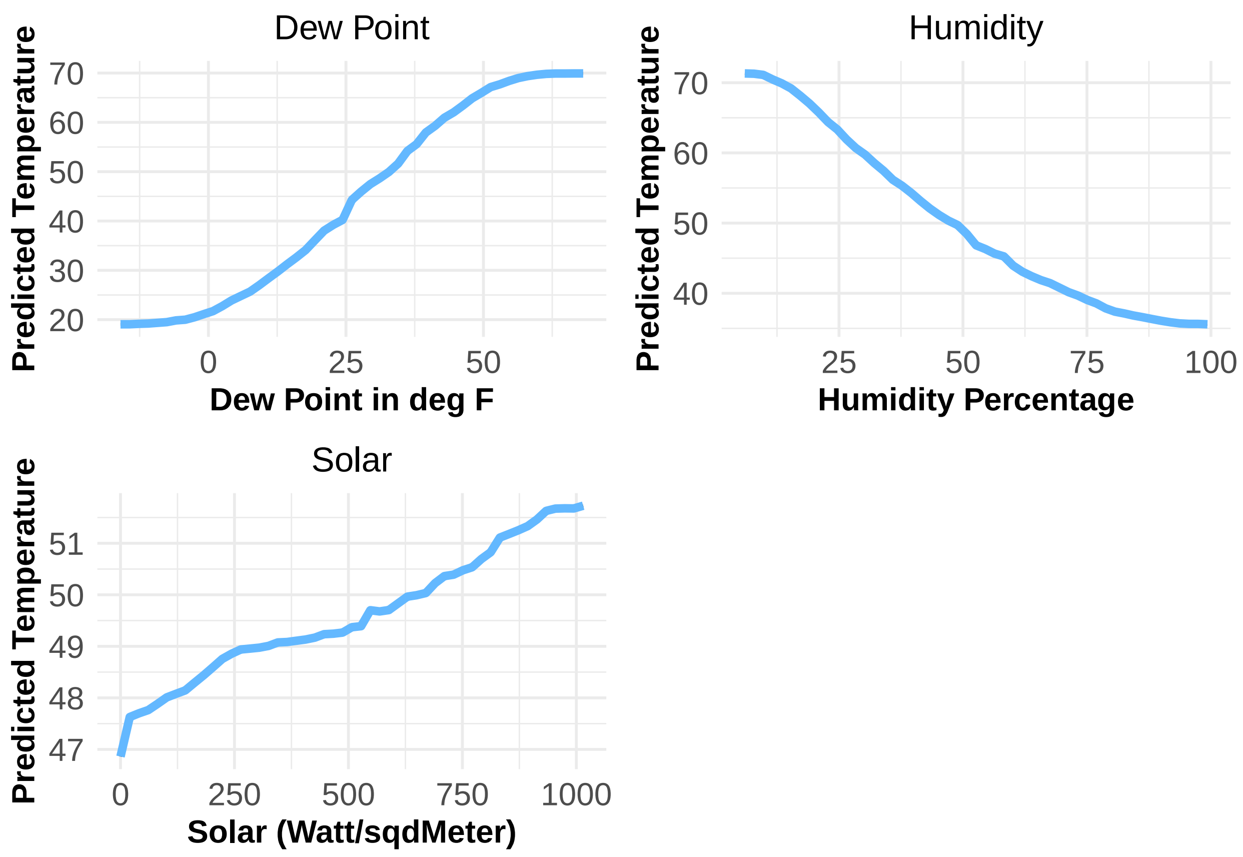
Partial plots show the relationship between each of the variables and the response. Once again, there is a quick way to do this…and my preferred way. The quick way, again using the randomForest package, would be to use the partialPlot command. Alternatively, I use the ggplot and pdp package to create partial plots:

```r
pa <- partial(model, pred.var = "DewPt_degF", plot = TRUE,
        plot.engine = "ggplot2") + theme_minimal() + ylab("Predicted Temperature") +
  xlab("Dew Point in deg F") + labs(title="Dew Point") +
  theme(plot.title = element_text(hjust = 0.5)) +
  theme(axis.text=element_text(size=12),
        axis.title=element_text(size=12,face="bold")) +
  geom_line(color = "steelblue1", lwd = 1.5)

pb <- partial(model, pred.var = "Humidity_Pct", plot = TRUE,
        plot.engine = "ggplot2") + theme_minimal() + ylab("Predicted Temperature") +
  xlab("Humidity Percentage") + labs(title="Humidity") +
  theme(plot.title = element_text(hjust = 0.5)) +
  theme(axis.text=element_text(size=12),
        axis.title=element_text(size=12,face="bold")) +
  geom_line(color = "steelblue1", lwd = 1.5)
```

```r
pc <- partial(model, pred.var = "Solar_WatPerSqM", plot = TRUE,
        plot.engine = "ggplot2") + theme_minimal() + ylab("Predicted Temperature") +
  xlab("Solar (Watt/sqdMeter)") + labs(title="Solar") +
  theme(plot.title = element_text(hjust = 0.5)) +
  theme(axis.text=element_text(size=12),
        axis.title=element_text(size=12,face="bold")) +
  geom_line(color = "steelblue1", lwd = 1.5)

grid.arrange(pa, pb, pc, nrow=2)
```



What these partial plots show is the relationship between the variable and the predicted temperature. Based on the partial plot, as the dew point increases so does the predicted temperature. This is also the case for Solar. On the other hand, as the humidity percentage increases the predicted temperature decreases.

Evaluate Model

Now let's evaluate how effective our model is with mean squared error:

```r
predictions = predict(model, test)

MSE(predictions, test$Temp_degF)
```

```
## [1] 4.713457
```

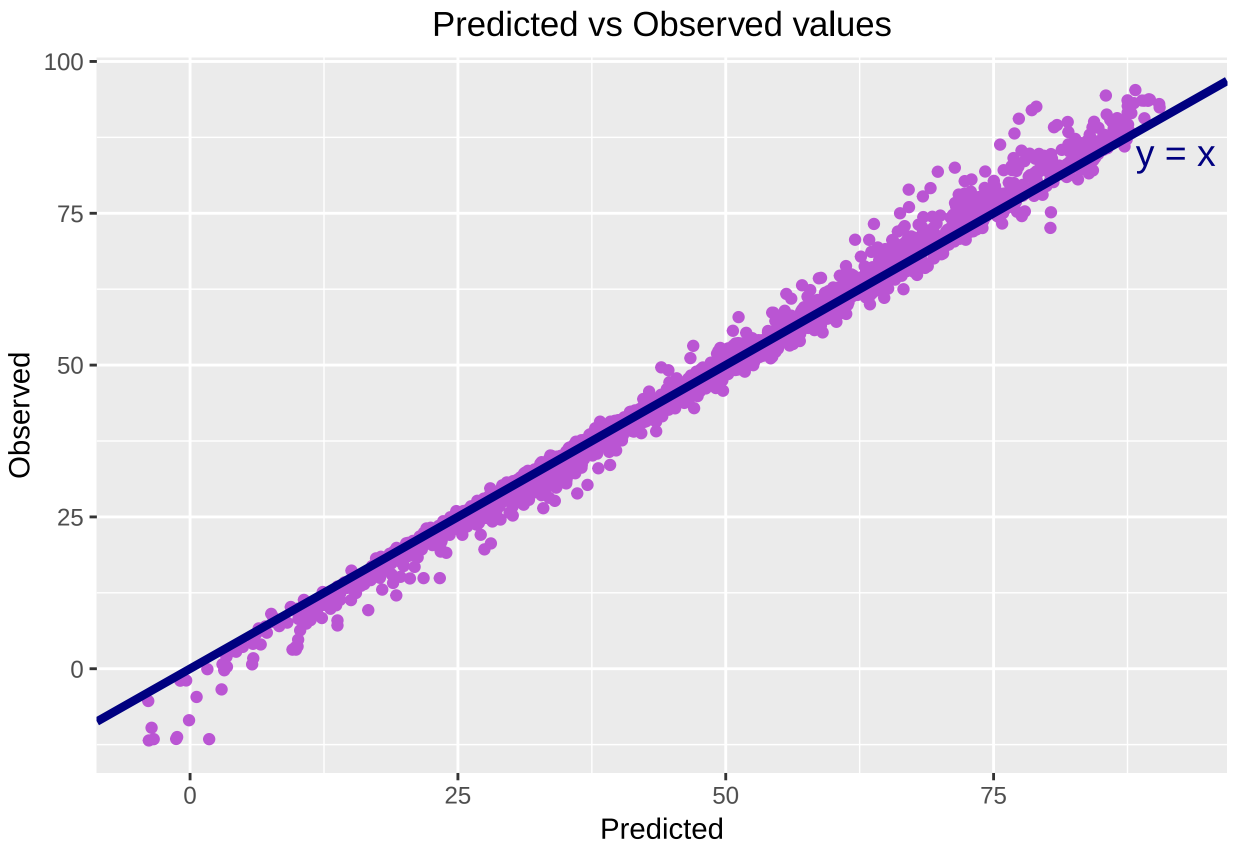
```r
R2 <- 1 - (sum((test$Temp_degF-predictions)^2)/sum((test$Temp_degF-mean(test$Temp_degF))^2))
R2
```

```
## [1] 0.9886718
```

For this model, the mean squared error is **4.71** and R^2 is approximately **0.99** making this model an almost perfect fit for the data!

If the predictions are perfect, the plot of predictions vs actual valus will fit the line y = x.

```
# plot predictions vs observed temperatures
ggplot(data = NULL, aes(x=predictions, y=test$Temp_degF)) +
  geom_point(col = "mediumorchid") +
  xlab("Predicted") + ylab("Observed") + labs(title="Predicted vs Observed values") +
  theme(plot.title = element_text(hjust = 0.5)) +
  geom_abline(intercept = 0, slope = 1, color = "navy", size = 1.5) +
  geom_text(data=data.frame(x=92,y=85), aes(x, y),
            label="y = x", color="navy", size = 5)
```



Analysis - Future Predictions

So we've seen that random forest is great for filling in missing data, but how about future predictions? In general, random forests don't provide a good fit when there's seasonality in time series data. Let's compare using the same process as before, except now our training data will be the first 75% of the data and our testing data will be the final 25%.

```
# make testing and training data sets
N = ceiling(0.75*nrow(dat))
train = dat[1:N-1,] #First 75% of the data
test = dat[N:nrow(dat),] #Final 25% of the data
```

```
# Random Forest algorithm
model = randomForest(`Temp_degF`~., data = train)
```

Evaluate Model

Let's evaulate how effective this model is compared to the previous model:

```
predictions = predict(model, test)

MSE(predictions, test$Temp_degF)
```

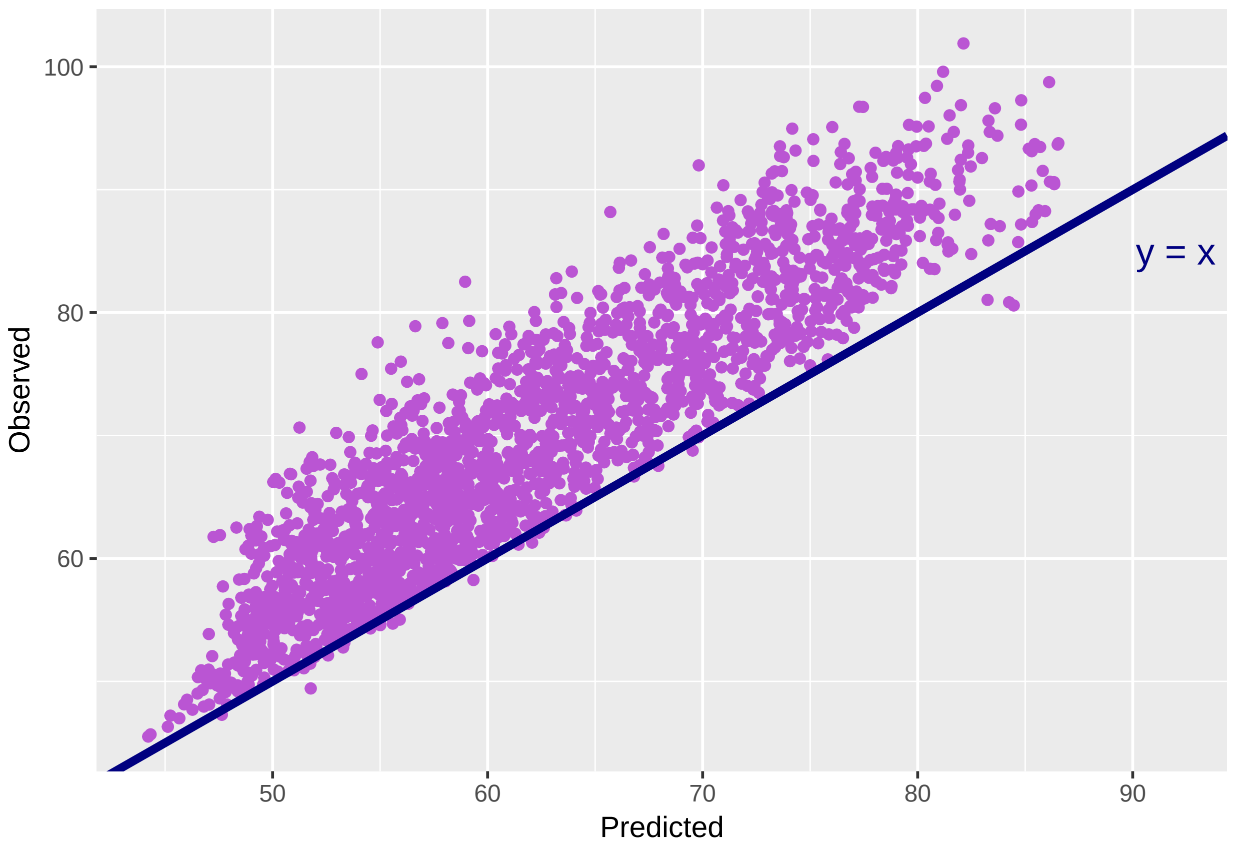
```
## [1] 78.58899
```

```
R2 <- 1 - (sum((test$Temp_degF-predictions)^2)/sum((test$Temp_degF-mean(test$Temp_degF))^2))
R2
```

```
## [1] 0.3759835
```

Our mean squared error in this case is significantly larger than our previous test! In this case our mean squared error is **78.59**, compared to 4.71 from before and now our R^2 is **0.38** - Random Forest is not useful for making prections!

```
# plot predictions vs observed temperatures
ggplot(data = NULL, aes(x=predictions, y=test$Temp_degF)) +
  geom_point(col = "mediumorchid") +
  xlab("Predicted") + ylab("Observed") +
  geom_abline(intercept = 0, slope = 1, color = "navy", size = 1.5) +
  geom_text(data=data.frame(x=92,y=85), aes(x, y), label="y = x", color="navy", size = 5)
```



So the predicted value is consistently higher than the observed value.

In general, the Random Forest algorithm will provide an average result and can not predict future results (extrapolate) becuase it does not understand data with seasonality.