Probabilistic Wind Power Forecasting with R

What is an Ensemble Forecast?

Numerical weather prediction (NWP) models create deterministic forecasts based on the meteorological observations collected from all around the earth. This observations are also called initial and boundary conditions for the NWP models. Besides, the atmosphere is a chaotic system and there is a high dependency and sensitivity to the initial and boundary conditions. However; it should also be emphasized that the precision of the observation data collected around the earth is unfortunately low; hence, the errors for these observations can be quite high. Therefore; most of the NWP models create various forecasts for the same focused area and time by just slightly changing the observations all around the earth! As a result of that, 50 different model results which are called ensembles from the same NWP models can be obtained for a specific location and time.

Probabilistic Forecasting over Deterministic

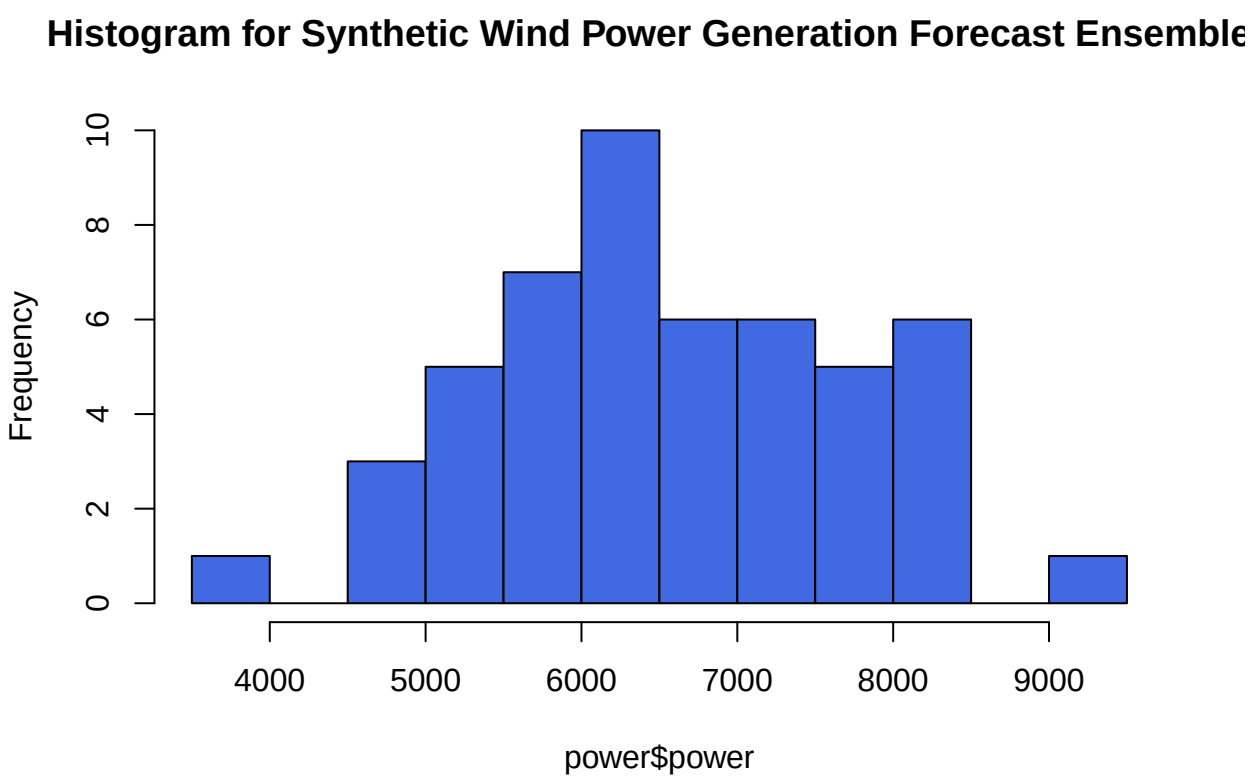
All NWP models also have operational forecasts and most of the websites and meteorological organisations are using these forecasts instead of using ensembles together. Since, these deterministic forecasts can be quite wrong and it can lead to irreversible damages for some sectors such as energy, the probabilities should also have to be given!

Here, I will be showing how to create a probabilistic wind power generation forecasts for a specific wind power plant and a period of time.

Let's Investigate the Ensemble Forecasts!

Here we generate a hourly wind power generation forecasts with 50 different ensembles for a wind power plant with 100 MWs of installed power and 40% capacity factor for a 1 week. After generating the synthetic data, a histogram is built with base R.

```
#Let's avoid using scientific notation
options(scipen = 999)

# Create a sample power forecast data with 50 ensembles for weekly total power generation of a wind pow
power = data.frame(power = rnorm(50,mean = 7*24*100*0.4, sd = 1000))
hist(power$power, breaks=12, main = "Histogram for Synthetic Wind Power Generation Forecast Ensembles",
```



Since all ensemble forecasts are created by using rnorm function, it will generate normally distributed random numbers with a mean of 6720 and standard deviation of 1000 MWh. Now let's generate the densities by using R's base function again and evaluate the result.

```
dens = density(power$power)
str(dens)
```

```
## List of 7
##  $ x         : num [1:512] 2385 2401 2417 2433 2449 ...
##  $ y         : num [1:512] 0.000000188 0.000000208 0.000000229 0.000000252 0.000000278 ...
##  $ bw        : num 478
##  $ n         : int 50
##  $ call      : language density.default(x = power$power)
##  $ data.name: chr "power$power"
##  $ has.na    : logi FALSE
##  - attr(*, "class")= chr "density"
```

```
filled = c()

# Regular Grids?
{for (i in 1:(length(dens$x)-1)) {

  filled[i] = diff(dens$x[i:(i+1)])

  }
```

```
a = c()
for (i in 1:(length(filled)-1)) {
  a[i] = diff(filled[i:(i+1)]) <= 0.000001
}
}
print(all(a))
```

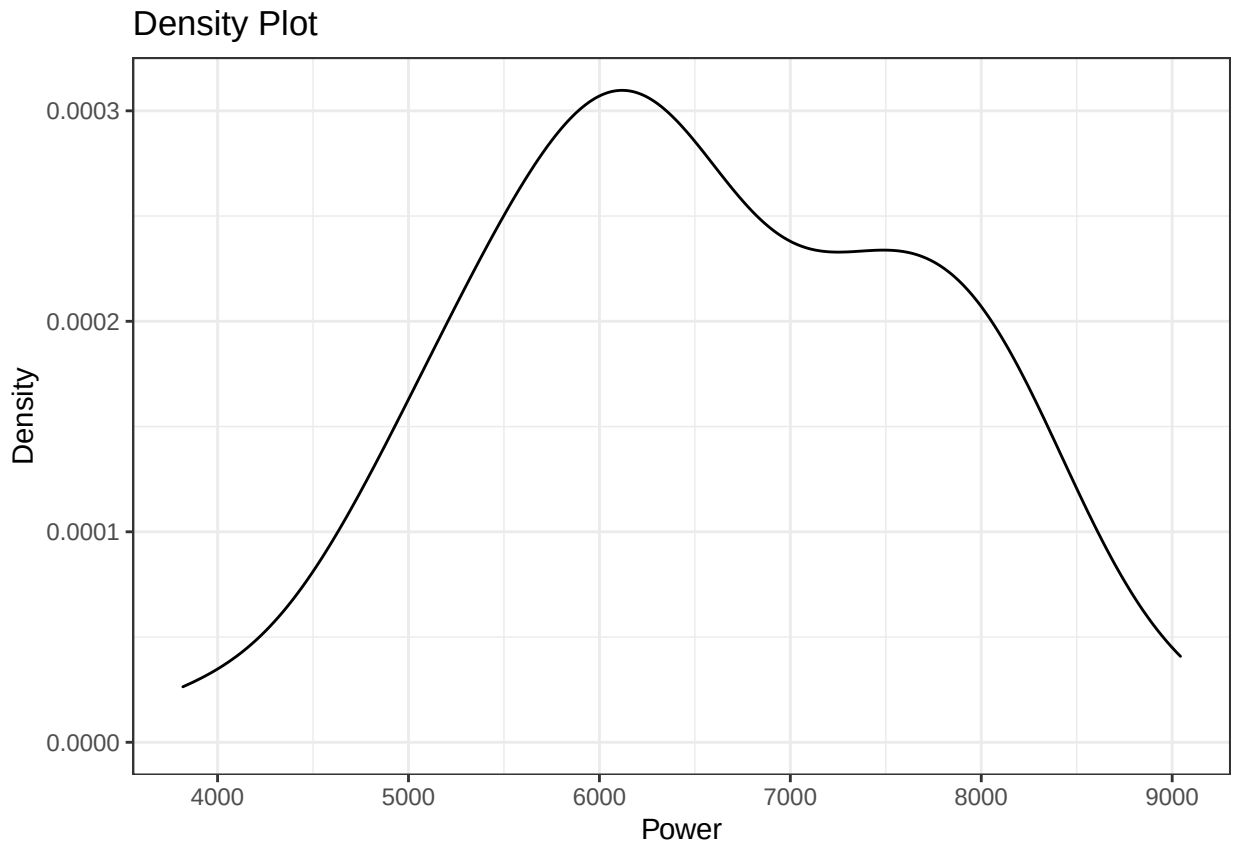
```
## [1] TRUE
```

```
rm(filled, a)
```

While the algorithm behind the density function disperses the mass of the empirical distribution function over a regular grid of at least 512 points and then uses the fast Fourier transform to convolve this approximation with a discretized version of the kernel and then uses linear approximation to evaluate the density at the specified points. Here, I would like to focus on dispersing the mass of the edf over a regular grid of at least 512 term. As it is shown, the R codes above show 512 regular spans and for loops prove it since the first for loop extracts the difference of the x axis created by density function, second one shows the difference between them is less than or equal to 0.000001. The all(a) function returns TRUE, therefore we are sure that the distances between x axis values are identical and x axis has regular grid spacing.

Now let us have a look to create a density plot with ggplot since we want to have better graphics.

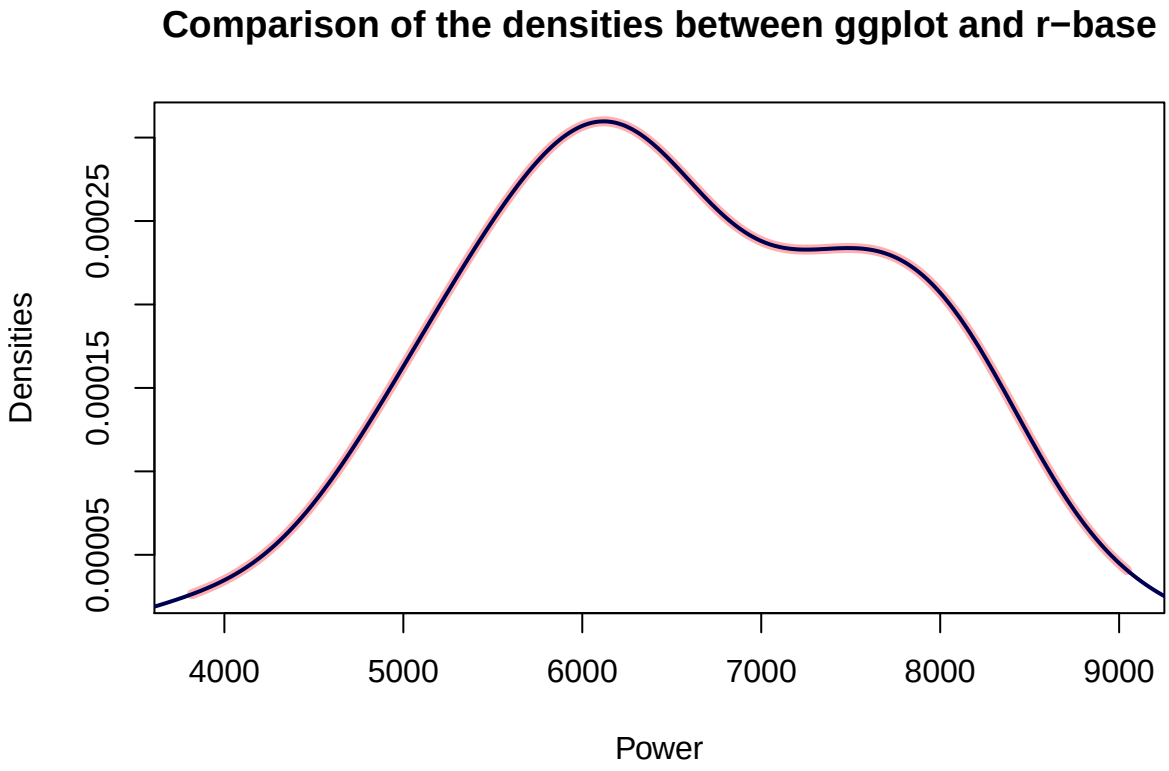
```
library(ggplot2)

# Density plot of the ggplot
(m = ggplot(power, aes(x = power)) +
    geom_density() +
    theme_bw() +
    labs(x = "Power", y = "Density", title = "Density Plot"))
```




```
# Take the data of the plot.
p = ggplot_build(m)
```

This looks better since the ggplot's ability to configure plots is better when it's compared with the base R. Besides, density plot can be created by adding geom_density to ggplot(). It also uses R's base density() function in order to generate it. We can be sure if the calculated densities are same for R and ggplot. ggplot lets you to take any data that is used for the plot by using ggplot_build() function. Let's have a look.

```
# First, ggplot's densities with red thick line then r's base densities with black thinner line.
plot(p$data[[1]]$x,p$data[[1]]$y, type = "l", col="#FF000050",lwd = 5,
     main = "Comparison of the densities between ggplot and r-base",
     xlab = "Power",
     ylab = "Densities")
lines(dens$x, dens$y,col="#000050",lwd =2 )
```



It has been showed that two different functions generates same densities. Now, probability density functions should be generated. Here we can proceed with the outputs of the density() function because this output fulfills the specification of PDF which is area under the PDF should be equal to 1.

```
# Is the total area under the curve is equal to 1?
sum(dens$y)*diff(dens$x[1:2])
```

```
## [1] 1.000926
```

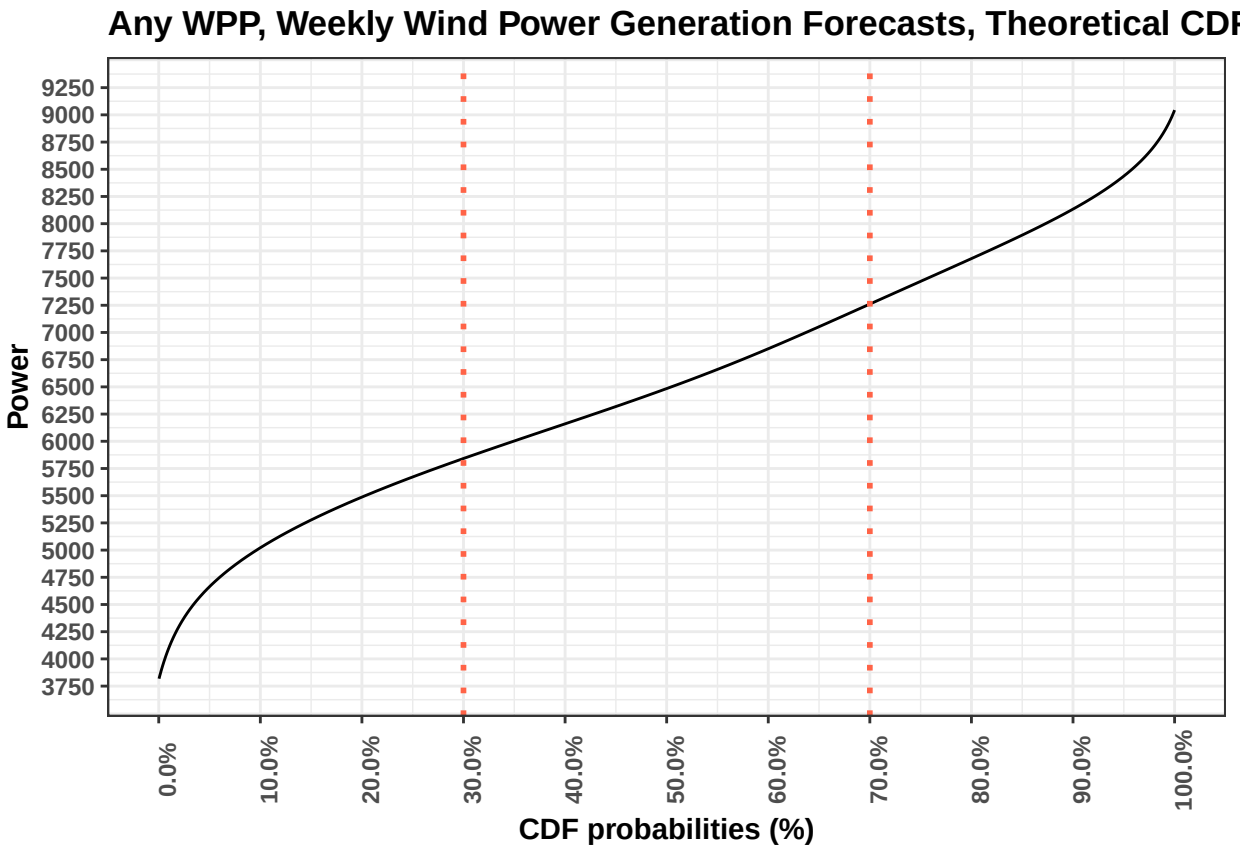
It's very close to 1, hence; outputs of the density(function) can be used as PDF outputs. Since, all the width of the bin of the x axis are identical, theoretical cumulative distribution function (CDF) can also be calculated as cumulative sums of the the mean of each densities in total density.

Note: PDF is the derivative of CDF, so; the area under the PDF curve should be calculated. A better approach is introduced instead of adding all calculated area together.

```
# Extract the data from the output of the ggplot_build()
prob_frcst = p$data[[1]]

# A function to create probabilities for each bin of the density plot.
scaling = function(x) {

  (x)/(sum(x))

}

prob_frcst$cdf = cumsum(scaling(prob_frcst$density))
```

Our CDF outputs are ready as a column inside prob_frcst data and it's time to plot it. It's better to create a basic simple function for generating a plot after obtaining theoretical CDF's. Therefore, after a function that makes configuration for the axis of the plots is introduced, another function that plots the desired output is also introduced.

```
# A function that makes a number rounded towards up or down by a user specified number.
round_it = function(x, roundto, tow = "up") {
  if(tow == "up") {  #Yukar? Yuvarla
    x + (roundto - x %% roundto)
  } else {
    if(tow == "down") {  #Asagi Yuvarla
      x - (x %% roundto)
    }
  }
}

# ggplot function for making easy to visualize the CDFs
plot_cdf =  function(data, prob_1, prob_2, wpp_name) {

  minimum = round_it(min(data$x),250,"down")
  maximum = round_it(max(data$x),250,"up")

  ggplot(data, aes(x = x, y = cdf)) +
    geom_line() +
    scale_x_continuous(name = "Power",
                       limits = c(minimum,maximum),
                       breaks = seq(minimum,maximum,250)) +
    scale_y_continuous(name = "CDF probabilities (%)",
                       limits = c(0,1),
                       breaks = seq(0,1,0.1),
                       labels = scales::percent) +
    ggtitle(paste(wpp_name,"WPP, Weekly Wind Power Generation Forecasts, Theoretical CDF")) +
    theme_bw()+
    theme(text = element_text(face = "bold"),
          axis.text.x = element_text(angle = 90)) +
    geom_hline(yintercept = prob_1, color = "tomato", linetype = "dotted", lwd = 1) +
    geom_hline(yintercept = prob_2, color = "tomato", linetype = "dotted", lwd = 1) +
    coord_flip()

}

plot_cdf(prob_frcst, prob_1 = 0.3, prob_2 = 0.7,wpp_name = "Any")
```



After plotting the theoretical CDF's, let's create a function that extracts the desired probabilities of the CDF. In order to do that, after subtracting the desired probabilities from all CDF probabilities, absolute value of them is taken. Now, extracting the minimum value from that vector provides the power of the desired probabilities.

```
# A method for extracting the closest CDF to 30% and 70%.
(percent30_i = which.min(abs(prob_frcst$cdf-0.3)))
```

```
## [1] 199
```

```
(percent70_i = which.min(abs(prob_frcst$cdf-0.7)))
```

```
## [1] 338
```

```
percent_index_finder = function(data, prob) {

  which.min(abs(data-prob))

}

prob_extraction = data.frame(Probabilities = seq(0.1,0.9,0.1), Value = 0)

for (i in (1:nrow(prob_extraction))) {

  prob_extraction[i,2] = prob_frcst$x[percent_index_finder(prob_frcst$cdf,prob_extraction[i,1])]
```

```
}

prob_extraction
```

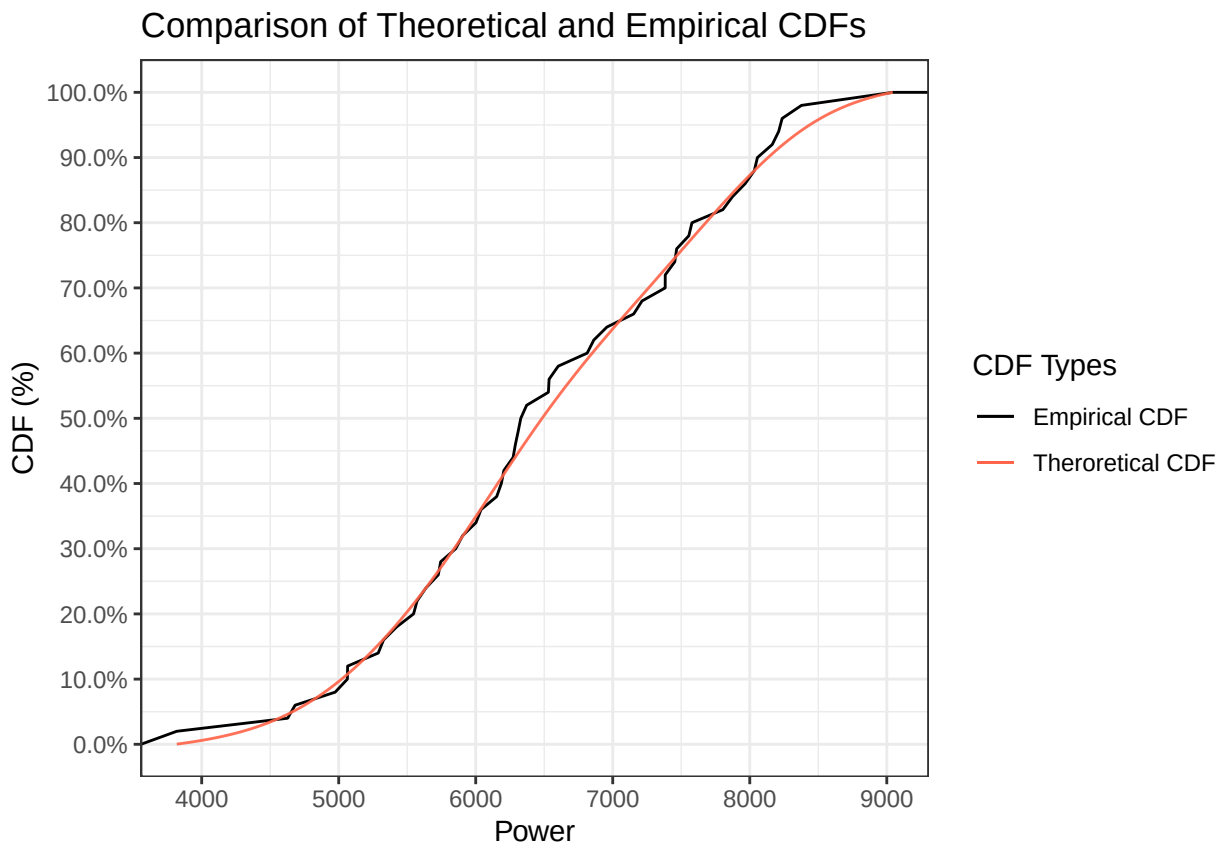
```
##   Probabilities    Value
## 1           0.1 5024.821
## 2           0.2 5484.991
## 3           0.3 5842.901
## 4           0.4 6159.907
## 5           0.5 6487.139
## 6           0.6 6855.275
## 7           0.7 7264.316
## 8           0.8 7683.582
## 9           0.9 8133.526
```

A table which includes probabilities of the theoretical CDF and values have been created. It clearly shows that the probability of less than or equal to for a given value of power!

On the other hand; this creates a theoretical CDF as described, however; since we are dealing with real continuous data, empirical CDF can also be generated by using ecdf function of base R or stat_ecdf() function of the ggplot. Now let us compare the results of the theoretical and empirical CDFs.

Remember that the cdf variable inside prob_frcst is theoretical and created by us, above. An empirical CDF is also shown on plot for comparison.

```
# Comparison plot for the eCDF and theoretical CDFs
ggplot(power, aes(power)) +
  stat_ecdf(geom = "line", aes(color = "Empirical CDF")) +
  geom_line(data = prob_frcst, aes(x = x, y = cdf, color = "Theroretical CDF"), alpha = 0.9) +
  scale_x_continuous(name = "Power")+
  scale_y_continuous(name = "CDF (%)", labels = scales::percent, breaks = seq(0,1,0.1))+
  ggtitle("Comparison of Theoretical and Empirical CDFs") +
  scale_color_manual(name = "CDF Types", values = c("Empirical CDF" = "black","Theroretical CDF"="tomat
  theme_bw()
```



Even though this study does not cover the usage of eCDF's, it also has been plotted and the differences have been shown. If you would like to use eCDF outputs, ggplot_build() function can be used for extracting the data of the eCDF plot. Besides, ecdf() function of the base R also gives a function which generates the probability of less than or equal to for a given value of power for the eCDF's!