Contents

```r
# Set working directory to where your CSVs live
setwd("C:/Users/Tilak Heble/OneDrive/Desktop/Seismic Noise")

# 1. Load required libraries
# Helper function to install and load packages
load_if_needed <- function(pkg) {
  if (!require(pkg, character.only = TRUE)) {
    install.packages(pkg, dependencies = TRUE)
    library(pkg, character.only = TRUE)
  }
}

# List of required packages
packages <- c(
  "tidyverse",   # Core data science tools: dplyr, ggplot2, readr, etc.
  "readxl",      # Read Excel files (.xls and .xlsx)
  "lubridate",   # Date/time manipulation: ymd(), hour(), etc.
  "mgcv",        # Fit GAM (Generalized Additive Models)
  "gratia",      # GAM model visualization and diagnostics
  "dplyr",       # Data manipulation (part of tidyverse)
  "ggplot2",     # Data visualization (part of tidyverse)
  "scales",      # Custom scales (e.g., scientific notation in plots)
  "openair",     # Air quality data tools and plotting (from UK AURN)
  "RColorBrewer",# Color palettes for plots
  "patchwork"    # Combine multiple ggplots into one layout
)

# Install and load each package
invisible(lapply(packages, load_if_needed))
```

```
## Loading required package: tidyverse

## Warning: package 'tidyverse' was built under R version 4.4.3

## Warning: package 'ggplot2' was built under R version 4.4.3

## Warning: package 'tidyr' was built under R version 4.4.3

## Warning: package 'forcats' was built under R version 4.4.3

## Warning: package 'lubridate' was built under R version 4.4.3

## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.2     v tibble    3.2.1
## v lubridate 1.9.4     v tidyr     1.3.1
## v purrr     1.0.4
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
## Loading required package: readxl

## Warning: package 'readxl' was built under R version 4.4.3

## Loading required package: mgcv

## Warning: package 'mgcv' was built under R version 4.4.3

## Loading required package: nlme
##
## Attaching package: 'nlme'
##
## The following object is masked from 'package:dplyr':
##
##     collapse
##
## This is mgcv 1.9-3. For overview type 'help("mgcv-package")'.
## Loading required package: gratia

## Warning: package 'gratia' was built under R version 4.4.3

##
## Attaching package: 'gratia'
##
## The following object is masked from 'package:stringr':
##
##     boundary
##
## Loading required package: scales

## Warning: package 'scales' was built under R version 4.4.3

##
## Attaching package: 'scales'
##
## The following object is masked from 'package:purrr':
##
##     discard
##
## The following object is masked from 'package:readr':
##
##     col_factor
##
## Loading required package: openair

## Warning: package 'openair' was built under R version 4.4.3
```

```
## Loading required package: RColorBrewer
## Loading required package: patchwork

## Warning: package 'patchwork' was built under R version 4.4.3
```

```r
# 2. Define file paths

# Frequency-domain PSD @ WS12
path_bg_fd  <- "Background_Frequency_Domain_WS12.csv"
path_op_fd  <- "Operational_Frequency_Domain_WS12.csv"

# Frequency-distance weighting function
path_fdwf   <- "FDWF_Data.xlsx"

# 3. Import data
# 3.1 Background PSD
freq_bg <- read_csv(path_bg_fd)
```

```
## New names:
## Rows: 73728 Columns: 12
## -- Column specification
## -------------------------------------------------------- Delimiter: "," chr
## (1): Hour dbl (10): ...1, Frequency..Hz., PSD.Displacement..m.2.Hz.,
## Frequency.Depend... date (1): Date
## i Use `spec()` to retrieve the full column specification for this data. i
## Specify the column types or set `show_col_types = FALSE` to quiet this message.
## * `` -> `...1`
```

```r
# 3.2 Operational PSD
freq_op <- read_csv(path_op_fd)
```

```
## New names:
## Rows: 311296 Columns: 12
## -- Column specification
## -------------------------------------------------------- Delimiter: "," chr
## (1): Hour dbl (10): ...1, Frequency..Hz., PSD.Displacement..m.2.Hz.,
## Frequency.Depend... date (1): Date
## i Use `spec()` to retrieve the full column specification for this data. i
## Specify the column types or set `show_col_types = FALSE` to quiet this message.
## * `` -> `...1`
```

```r
# 3.3 Frequency-Distance Weighting Function
fdwf    <- read_excel(path_fdwf, sheet = 1)

# 3.4 Quick check
glimpse(freq_bg)
```

```
## Rows: 73,728
## Columns: 12
## $ ...1                                     <dbl> 1, 2, 3, 4, 5, 6, 7, 8, ~
## $ Frequency..Hz.                           <dbl> 0.006103516, 0.012207031~
```

```
## $ PSD.Displacement..m.2.Hz.                     <dbl> 6.830252e-11, 4.254498e-~
## $ Frequency.Dependent.PSD.Displacement..m.2.Hz. <dbl> 2.667622e-24, 1.646402e-~
## $ Date                                          <date> 2020-03-28, 2020-03-28,~
## $ Hour                                          <chr> "1510", "1510", "1510", ~
## $ Wind.speed                                    <dbl> 12.02112, 12.02112, 12.0~
## $ Rounded.Wind.speed                            <dbl> 12, 12, 12, 12, 12, 12, ~
## $ Wind.direction.degrees                        <dbl> 37.08, 37.08, 37.08, 37.~
## $ Wind.direction.radians                        <dbl> 0.6471681, 0.6471681, 0.~
## $ No.of.samples                                 <dbl> 100, 100, 100, 100, 100,~
## $ Operational                                   <dbl> 0, 0, 0, 0, 0, 0, 0, 0, ~
```

```
glimpse(freq_op)
```

```
## Rows: 311,296
## Columns: 12
## $ ...1                                          <dbl> 1, 2, 3, 4, 5, 6, 7, 8, ~
## $ Frequency..Hz.                                <dbl> 0.006103516, 0.012207031~
## $ PSD.Displacement..m.2.Hz.                     <dbl> 8.324401e-11, 3.025114e-~
## $ Frequency.Dependent.PSD.Displacement..m.2.Hz. <dbl> 3.251176e-24, 1.170656e-~
## $ Date                                          <date> 2021-09-22, 2021-09-22,~
## $ Hour                                          <chr> "1130", "1130", "1130", ~
## $ Wind.speed                                    <dbl> 11.58977, 11.58977, 11.5~
## $ Rounded.Wind.speed                            <dbl> 12, 12, 12, 12, 12, 12, ~
## $ Wind.direction.degrees                        <dbl> 216.7858, 216.7858, 216.~
## $ Wind.direction.radians                        <dbl> 3.783626, 3.783626, 3.78~
## $ No.of.samples                                 <dbl> 100, 100, 100, 100, 100,~
## $ Operational                                   <dbl> 1, 1, 1, 1, 1, 1, 1, 1, ~
```

```
glimpse(fdwf)
```

```
## Rows: 8,193
## Columns: 2
## $ Frequency_Hz                          <dbl> 0.000000000, 0.006103516, 0.0122~
## $ Frequency_Distance_Weighting_Function <dbl> 3.941719e-14, 3.905598e-14, 3.86~
```

```
# 4. Data Preprocessing

# 4.1 Standardize column names for PSD tables (background and operational)
# Rename columns to a uniform format so they can be processed identically downstream.
# Frequency and PSD column names are reformatted, while other fields are retained.
freq_bg <- freq_bg %>%
  rename(
    Frequency_Hz = `Frequency..Hz.`,                # frequency in Hz
    PSD          = `PSD.Displacement..m.2.Hz.`,     # raw PSD (displacement units)
    Date         = Date,
    Hour         = Hour,
    Wind.speed   = Wind.speed,                      # in m/s
    Wind.dir.deg = `Wind.direction.degrees`         # wind direction in degrees
  )

freq_op <- freq_op %>%
  rename(
```

```r
    Frequency_Hz = `Frequency..Hz.`,
    PSD          = `PSD.Displacement..m.2.Hz.`,
    Date         = Date,
    Hour         = Hour,
    Wind.speed   = Wind.speed,
    Wind.dir.deg = `Wind.direction.degrees`
  )

# 4.2 Standardize FDWF table (Frequency-Distance Weighting Function)
# Rename the columns in the FDWF table to standard names: frequency and associated weight.
fdwf <- fdwf %>%
  rename(
    Frequency_Hz = Frequency_Hz,                              # frequency
    weight       = Frequency_Distance_Weighting_Function      # FDWF value
  )

# 4.3 Merge PSD tables with FDWF and compute weighted PSD
# Multiply raw PSD values by FDWF weights to get frequency-weighted PSD.
# Add a new column 'Operational' to indicate turbine state (0 = background, 1 = operational).
freq_bg <- freq_bg %>%
  inner_join(fdwf, by = "Frequency_Hz") %>%                   # merge with weights
  mutate(
    PSD_wtd     = PSD * weight,                               # weighted PSD
    Operational = 0                                           # turbine off
  )

freq_op <- freq_op %>%
  inner_join(fdwf, by = "Frequency_Hz") %>%
  mutate(
    PSD_wtd     = PSD * weight,
    Operational = 1                                           # turbine on
  )

# 4.4 Combine PSD tables and tag turbine state (Type)
# Merge background and operational PSD data into one table.
# Re-label wind direction and speed columns, drop missing data, and label the 'Type'.
psd_all <- bind_rows(freq_bg, freq_op) %>%                    # merge datasets
  rename(
    Wind.speed = Wind.speed,                                  # rename for clarity
    Wind.dir   = Wind.dir.deg
  ) %>%
  drop_na(Wind.speed, PSD_wtd) %>%                            # remove rows with missing critical value
  mutate(
    Type = if_else(Operational == 1, "Operational", "Background")  # human-readable label
  )

# 4.5 Integrate PSD over 0.5-8 Hz to compute Energy (E)
# Use trapezoidal rule to numerically integrate weighted PSD into total energy in band.
# Compute covariates like timestamp, log-energy, and date-based variables.
energy <- psd_all %>%
  filter(between(Frequency_Hz, 0.5, 8)) %>%                   # filter to frequency band of interest
  arrange(Date, Hour, Frequency_Hz) %>%                       # ensure correct order for integration
  group_by(Date, Hour, Wind.speed, Wind.dir, Operational, Type) %>%
```

```r
  summarize(
    E = sum((PSD_wtd + lead(PSD_wtd, default = last(PSD_wtd))) / 2 *
            (lead(Frequency_Hz, default = last(Frequency_Hz)) - Frequency_Hz)),  # trapezoidal integrat
    .groups = "drop"
  ) %>%
  mutate(
    Datetime = ymd(Date) + hours(Hour),                             # full timestamp
    logE     = log(E + 1e-9),                                       # log-transform energy, offset to avoid -
    DateF    = factor(Date),                                        # factor date for random effect in GAM
    Hour     = hour(Datetime),                                      # hour of day (0-23)
    Month    = month(Datetime),                                     # month (1-12)
    Weekday  = as.integer(format(Datetime, "%u"))                   # day of week (1=Mon, ..., 7=Sun)
  ) %>%
  drop_na(Wind.speed, E)                                            # remove rows with missing energy or wind

# Final structure: `energy` contains all cleaned, engineered variables required for modeling.
# Key fields include:
#   - E, logE (integrated energy and its log)
#   - Operational (0/1) and Type ("Background" / "Operational")
#   - Wind.speed, Wind.dir
#   - Datetime, Hour, Month, Weekday (temporal)
#   - DateF (factor for day-level random effects)

glimpse(energy)  # quick check of structure
```

```
## Rows: 47
## Columns: 12
## $ Date        <date> 2020-03-28, 2020-03-29, 2020-06-27, 2020-06-27, 2020-08-0~
## $ Hour        <int> 22, 14, 20, 4, 12, 2, 22, 10, 2, 2, 12, 22, 0, 10, 6, 14, ~
## $ Wind.speed  <dbl> 12.02112, 12.27036, 11.54181, 11.65684, 11.82940, 12.26078~
## $ Wind.dir    <dbl> 37.0800, 44.5100, 208.4000, 203.0000, 229.2000, 171.2000, ~
## $ Operational <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
## $ Type        <chr> "Background", "Background", "Background", "Background", "B~
## $ E           <dbl> 3.345300e-21, 3.430003e-21, 2.697738e-21, 3.476958e-21, 2.~
## $ Datetime    <dttm> 2020-05-29 22:00:00, 2020-04-07 14:00:00, 2020-09-15 20:0~
## $ logE        <dbl> -20.72327, -20.72327, -20.72327, -20.72327, -20.72327, -20~
## $ DateF       <fct> 2020-03-28, 2020-03-29, 2020-06-27, 2020-06-27, 2020-08-05~
## $ Month       <dbl> 5, 4, 9, 9, 8, 10, 11, 10, 2, 11, 11, 11, 11, 11, 11, 12, ~
## $ Weekday     <int> 5, 2, 2, 6, 6, 6, 5, 3, 7, 1, 1, 1, 4, 4, 5, 5, 6, 6, 6, 7~
```

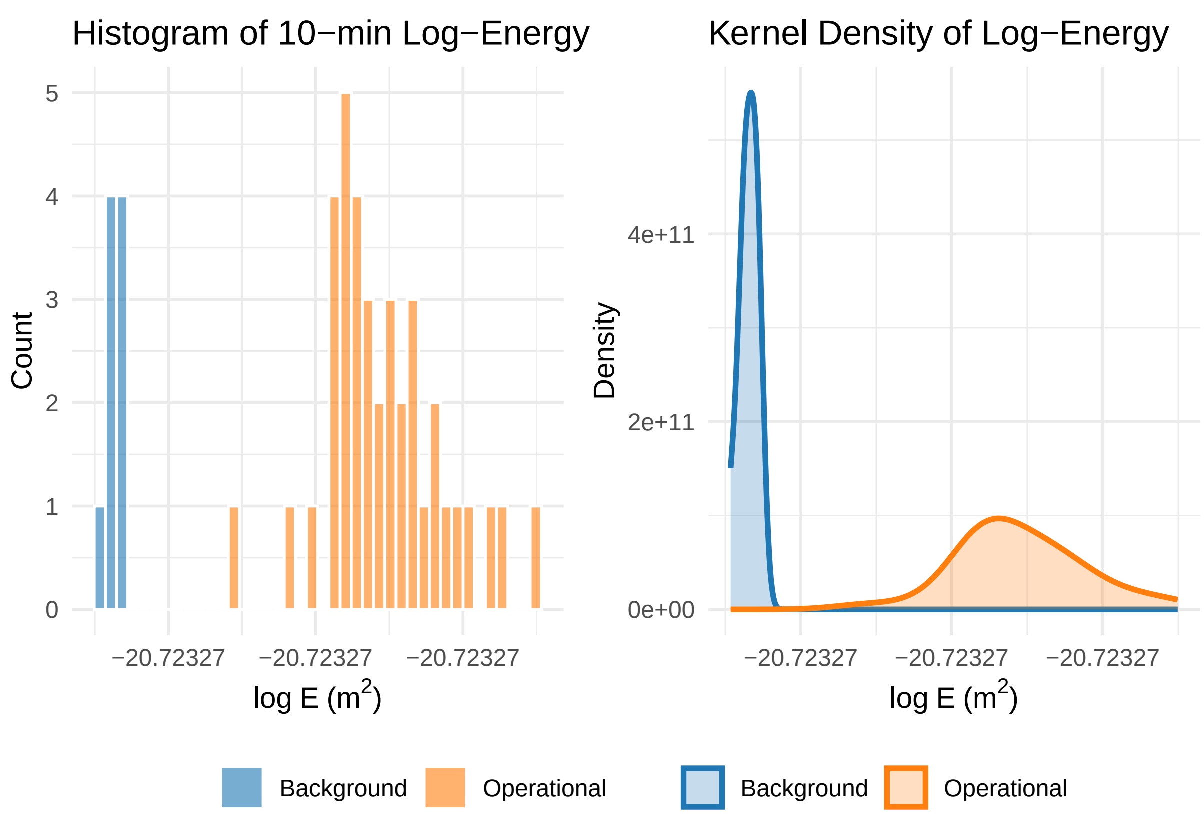
EDA & DATA VISUALIZATION

```r
#Set up for Plotting
limit_nm <- 0.336                # compliance line for later plots
energy   <- energy %>%
  mutate(RMS_nm = sqrt(E) * 1e9) # m² → metres → nm
```

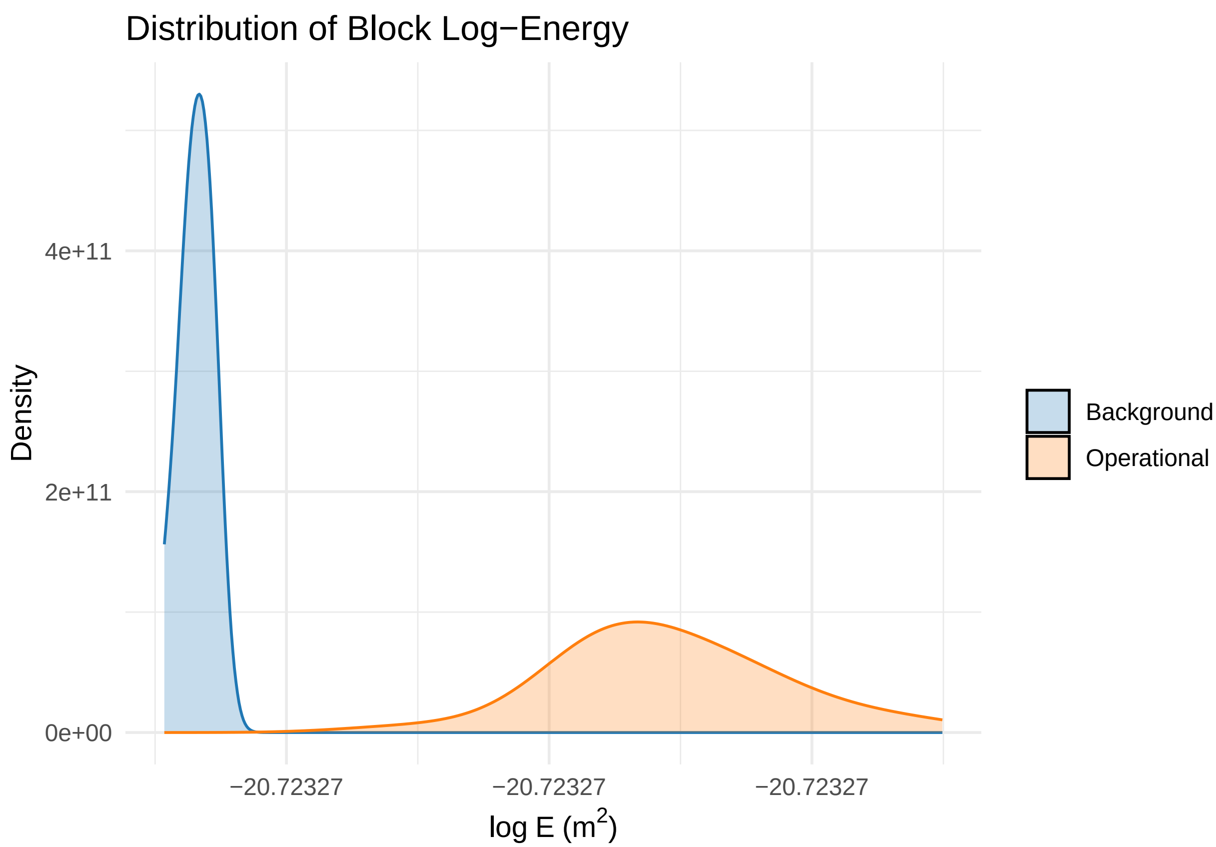
1. Distribution & Summary Plots

```r
### Plot 1 - Distribution of 10-minute Log-Energy (Histogram and Density)

# 1. Histogram
```

```
p_hist <- ggplot(energy, aes(logE, fill = Type)) +             # Set up histogram for log-energy by
  geom_histogram(bins = 40, alpha = 0.6, position = "identity",     # Histogram with semi-transparency a
                 colour = "white") +
  scale_fill_manual(values = c("#1f77b4", "#ff7f0e"), name = "") + # Custom fill colours for Type
  labs(title = "Histogram of 10-min Log-Energy",                 # Title and axis labels
       x = expression(log~E~"(m"^2*")"), y = "Count") +
  theme_minimal() +                                              # Clean minimal theme
  theme(legend.position = "none")       # legend will be shared later

# 2. Density curve
p_dens <- ggplot(energy, aes(logE, colour = Type, fill = Type)) +  # Set up density plot for log-energy
  geom_density(alpha = .25, adjust = 1.3, linewidth = 1) +         # Smoothed density with partial fill
  scale_colour_manual(values = c("#1f77b4", "#ff7f0e"), name = "") + # Custom line colours
  scale_fill_manual(values = c("#1f77b4", "#ff7f0e"), name = "") +   # Custom fill colours
  labs(title = "Kernel Density of Log-Energy",                   # Title and axis labels
       x = expression(log~E~"(m"^2*")"), y = "Density") +
  theme_minimal() +                                              # Clean minimal theme
  theme(legend.position = "none")                                # Hide legend (to be shared in combi

# 3. Combine and add a shared legend
p_combo <- (p_hist | p_dens) +                                   # Combine histogram and density side
           plot_layout(guides = "collect") & theme(legend.position = "bottom")  # Share and place legen

print(p_combo)                                                   # Display the combined plot
```

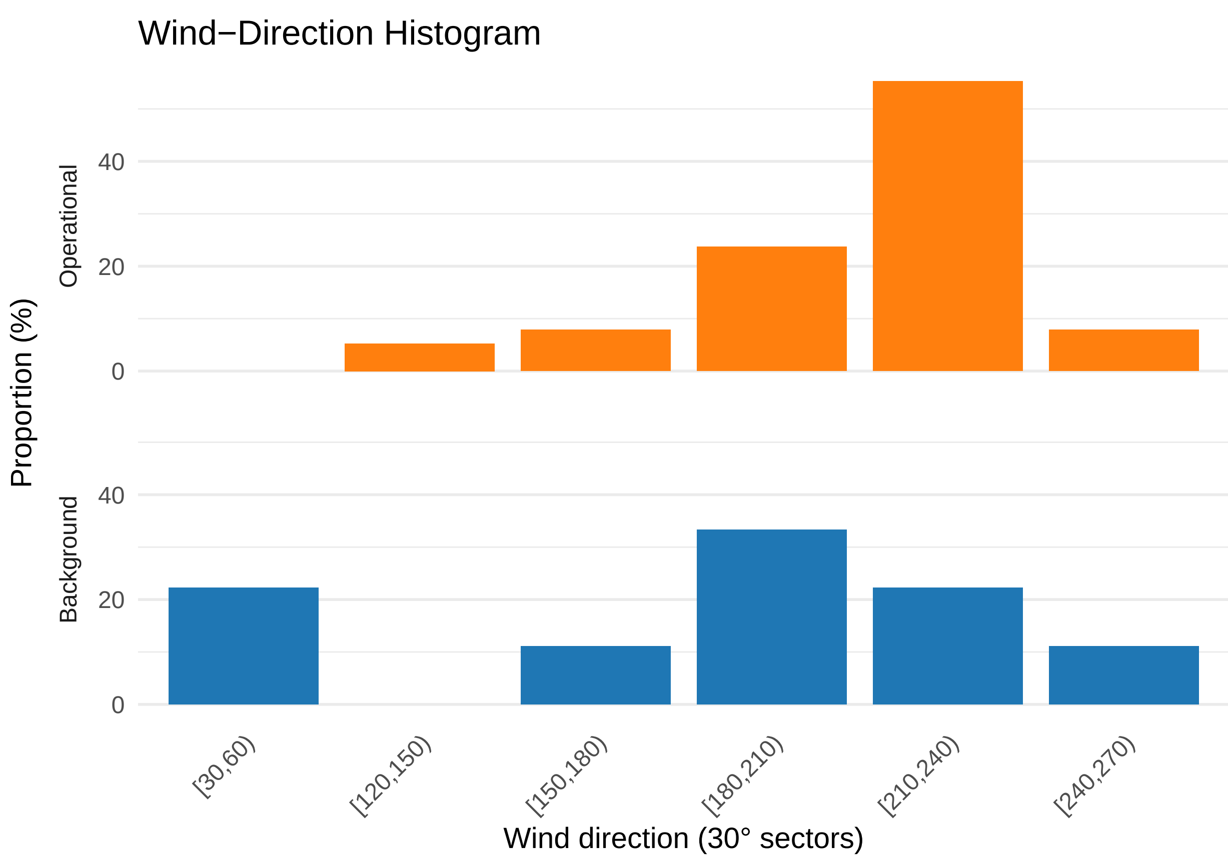


```r
### Plot 2 - Distribution of Block Log-Energy (Kernel Density)

p_dens <- ggplot(energy,
                 aes(x = logE,
                     fill = factor(Operational),
                     colour = factor(Operational))) +

  # Kernel density estimation with light transparency
  geom_density(alpha = 0.25, adjust = 1.5) +

  # Fill color: blue for background, orange for operational
  scale_fill_manual(values = c("0" = "#1f77b4", "1" = "#ff7f0e"),
                    labels = c("Background", "Operational"),
                    name   = "") +

  # Line color to match fill, but no legend for outlines
  scale_colour_manual(values = c("0" = "#1f77b4", "1" = "#ff7f0e"),
                      guide = "none") +

  # Axis labels and plot title
  labs(title = "Distribution of Block Log-Energy",
       x = expression(log~E~"(m"^2*")"),
       y = "Density") +

  # Clean visual style
  theme_minimal()

# Display the plot
print(p_dens)
```



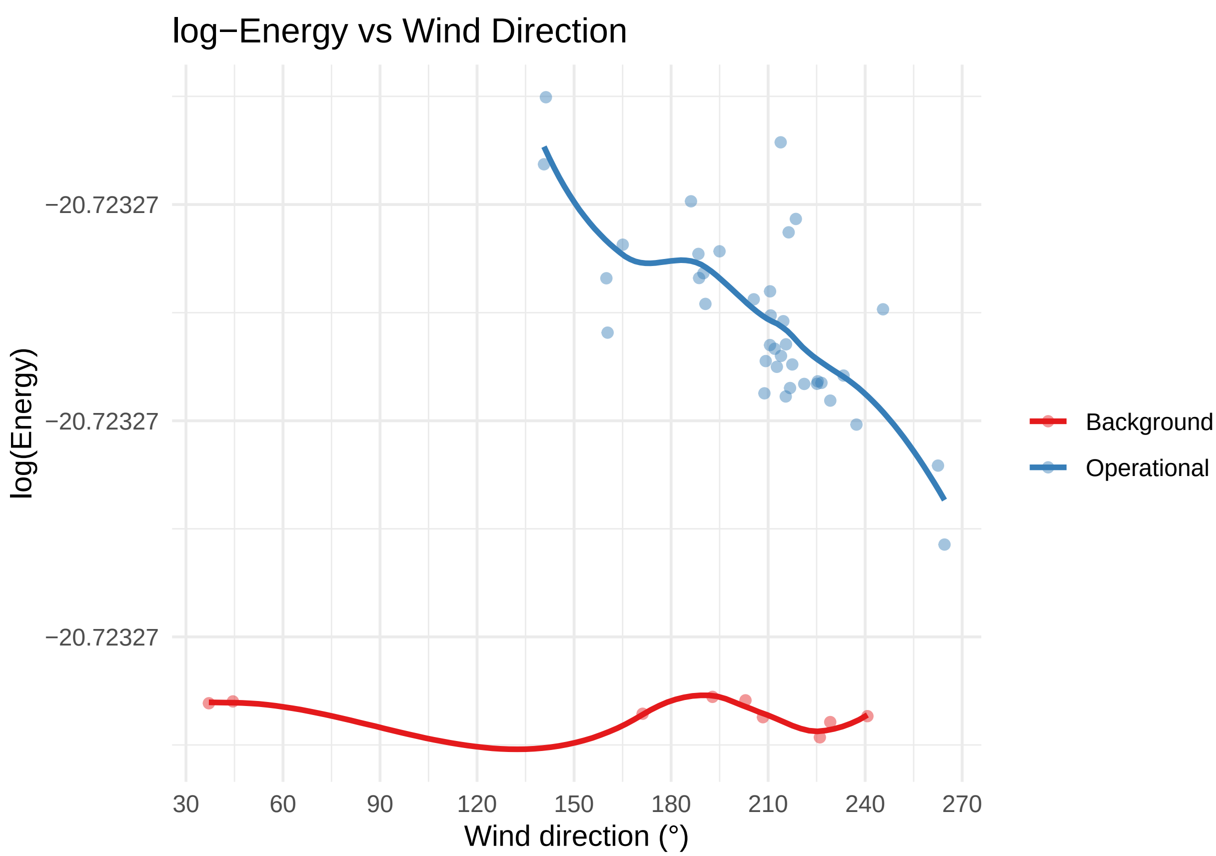

```r
### Plot 3 - Wind Direction Histogram (Faceted by State)

# Bin wind direction into 30° sectors
dir_df <- energy %>%
  mutate(
    WD_bin = cut(Wind.dir,                               # bin wind direction into intervals
                 breaks = seq(0, 360, 30),               # 12 bins: [0,30), [30,60), ...
                 include.lowest = TRUE, right = FALSE),

    # Set factor levels so "Operational" is plotted above "Background" in the facet layout
    Type   = factor(Type, levels = c("Operational", "Background"))
  ) %>%
  count(Type, WD_bin) %>%                                # count observations per bin and state
  group_by(Type) %>%                                     # group by turbine state
  mutate(prop = n / sum(n) * 100) %>%                    # convert counts to percentage per state
  ungroup()

# Define colours for each turbine state
cols <- c("Background"  = "#1f77b4",                     # blue
          "Operational" = "#ff7f0e")                     # orange

# Plot wind direction histogram as vertically stacked bar plots per state
p_dir_hist <- ggplot(dir_df,
                     aes(x = WD_bin, y = prop, fill = Type)) +
  geom_col(width = 0.85, show.legend = FALSE) +          # draw proportional bars
  facet_grid(Type ~ ., switch = "y") +                   # vertically stack Background & Operational
```

```
  scale_fill_manual(values = cols) +                        # apply custom fill colours
  labs(title = "Wind-Direction Histogram",                  # plot title and axis labels
       x = "Wind direction (30° sectors)", y = "Proportion (%)") +
  theme_minimal(base_size = 11) +                           # use minimal theme
  theme(
    strip.placement = "outside",                            # place facet labels outside the panel
    strip.text.y     = element_text(angle = 0, hjust = .5),  # rotate facet labels
    axis.text.x      = element_text(angle = 45, hjust = 1),  # slant x-axis labels for readability
    panel.grid.major.x = element_blank()                    # remove vertical grid lines
  )

print(p_dir_hist)
```



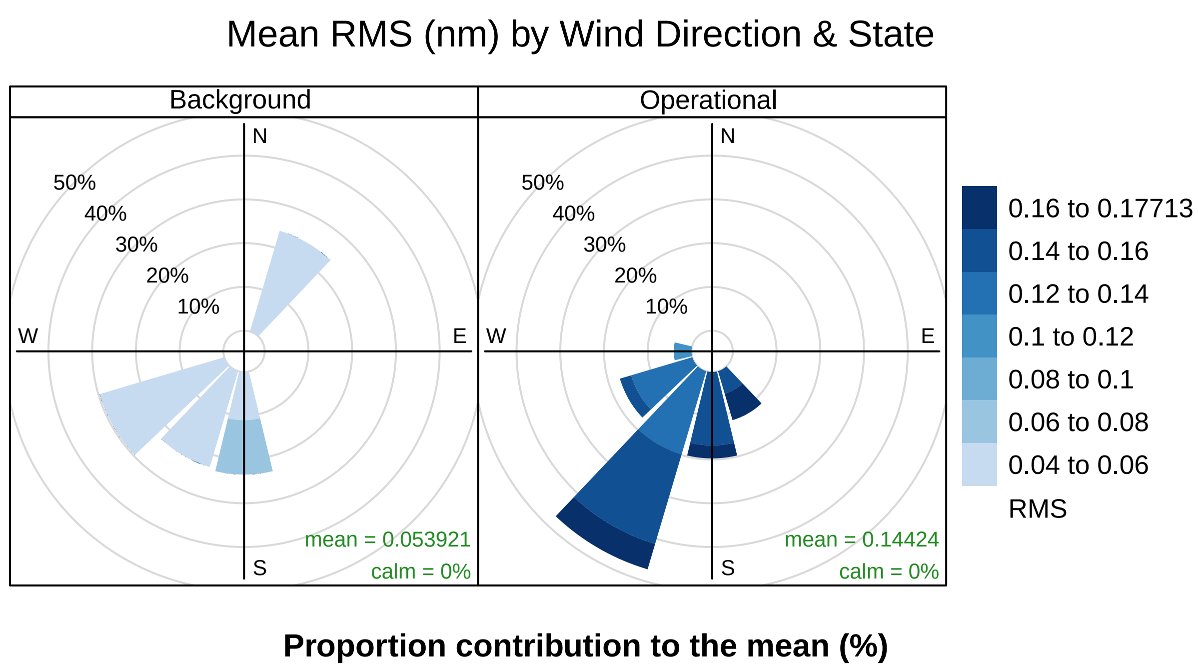

```
### Plot 4 - Relationship Between Wind Direction and log-Energy by Turbine State

# Define custom colors for Background and Operational types
cols <- c("Background"="#e41a1c",
          "Operational"="#377eb8")

# Create scatter plot of log-energy vs wind direction with LOESS smoothing
p_dir <- ggplot(energy, aes(Wind.dir, logE, colour = Type)) +
  geom_point(alpha = .45) +                    # Add semi-transparent points
  geom_smooth(method = "loess",                # Add LOESS smooth line for trend
              formula = y ~ x,                 # Explicitly define smoothing formula
              se = FALSE, linewidth = 1) +     # Remove shaded CI, set line width
```

```r
  scale_colour_manual(values = cols, name = "") +                 # Apply custom colors
  scale_x_continuous(breaks = seq(0,360,30)) +                    # Set x-axis breaks every 30°
  labs(title = "log-Energy vs Wind Direction",                     # Add axis and title labels
       x = "Wind direction (°)", y = "log(Energy)") +
  theme_minimal()                                                 # Use minimal theme

# Display plot
print(p_dir)
```



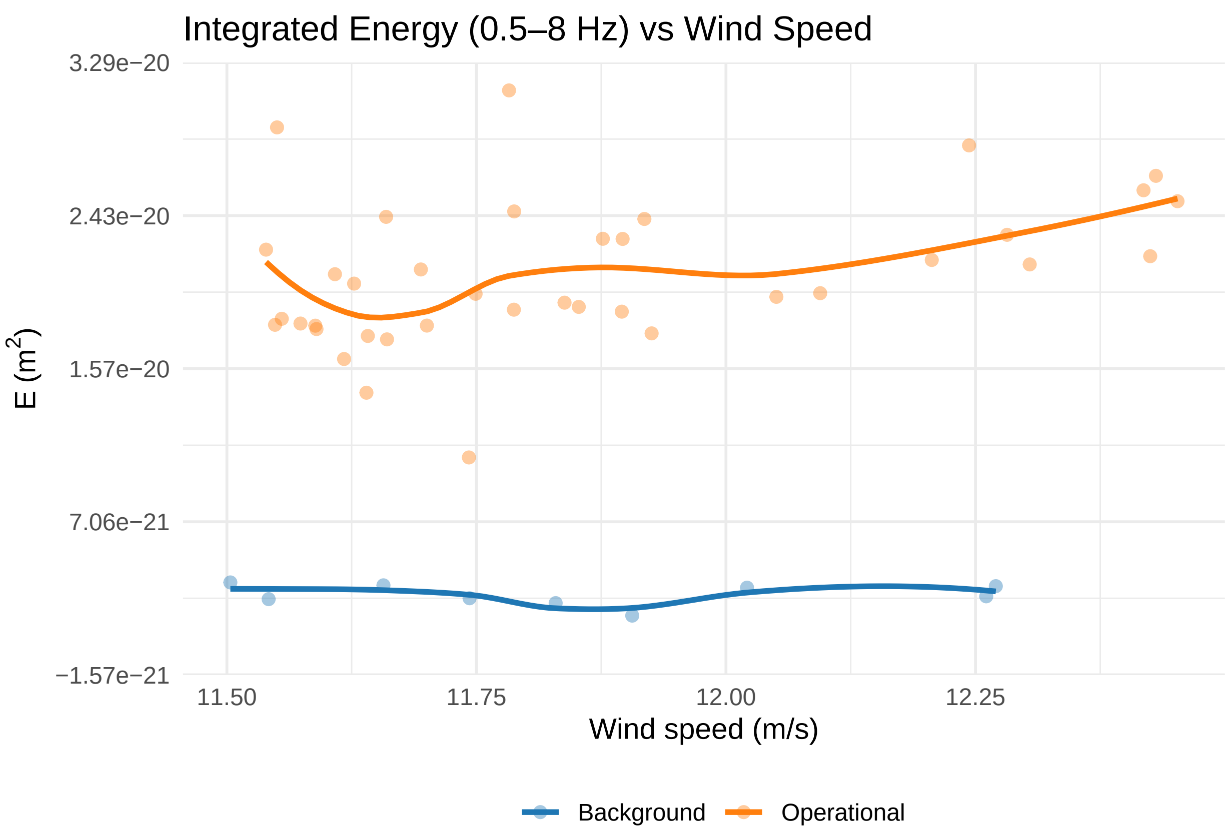

```r
### Plot 5 - Wind Directional Dependence of RMS Displacement (Polar Plot)

# 1. Define a custom 5-shade blue gradient from light to dark
blue_grad <- colorRampPalette(brewer.pal(9, "Blues")[3:9])(5)

# 2. Prepare the input data for the rose diagram:
#    - Rename columns to standard names expected by pollutionRose()
#    - Extract datetime, wind speed, wind direction, RMS displacement, and turbine status
rose_df <- energy %>% transmute(
  date = Datetime,          # time stamp
  ws   = Wind.speed,        # wind speed
  wd   = Wind.dir,          # wind direction
  RMS  = RMS_nm,            # RMS displacement in nanometers
  Type = Type               # turbine operational status
)
```

```r
# 3. Create a pollution rose plot of RMS vs wind direction:
#     - Use mean RMS per direction bin
#     - Split by turbine Type
#     - Apply custom blue color gradient
#     - Disable the paddle-style legend
#     - Position the legend to the right
#     - Set main title for the plot
pollutionRose(
  mydata       = rose_df,
  pollutant    = "RMS",             # variable to plot
  statistic    = "prop.mean",       # use mean proportion in each bin
  type         = "Type",            # split by turbine status
  cols         = blue_grad,         # color palette
  paddle       = FALSE,             # rectangular legend
  key.position= "right",            # place legend to the right
  main         = "Mean RMS (nm) by Wind Direction & State"
)
```




```r
### Plot 6 - Integrated Energy vs Wind Speed by Turbine State

# Define custom colours for turbine states
cols <- c("Background"  = "#1f77b4",
          "Operational" = "#ff7f0e")

# Scatter plot with loess smoothing: Energy vs Wind Speed
```

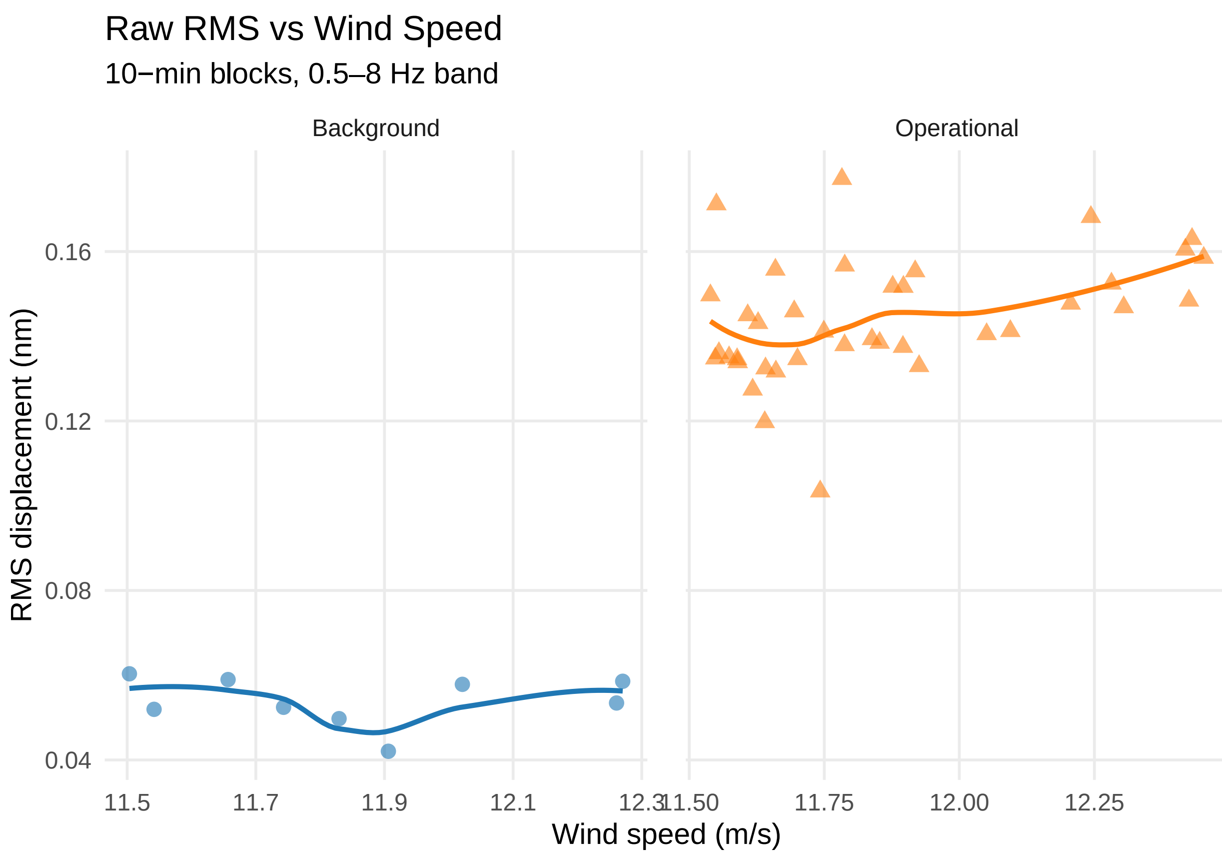
```r
p_ws_E <- ggplot(energy, aes(Wind.speed, E, colour = Type)) +
  geom_point(alpha = .4, size = 1.8) +                   # semi-transparent points
  geom_smooth(method  = "loess",                         # non-parametric smoother
              formula = y ~ x,                           # specify formula explicitly
              se      = FALSE,                           # don't show confidence band
              linewidth = 1) +
  scale_y_continuous(labels = scientific,               # scientific notation for y-axis
                     limits = c(0, NA)) +                # lower limit at 0, upper auto
  scale_colour_manual(values = cols, name = "") +        # apply custom colours
  labs(title = "Integrated Energy (0.5-8 Hz) vs Wind Speed",  # plot title and labels
       x = "Wind speed (m/s)",
       y = expression(E~"(m"^2*")")) +
  theme_minimal(base_size = 11) +                        # clean theme
  theme(legend.position = "bottom")                      # place legend below plot

print(p_ws_E)  # display the plot
```




```r
### Plot 7 - Relationship Between Wind Speed and RMS Displacement (Faceted by State)

p_scatter <- ggplot(energy, aes(Wind.speed, RMS_nm)) +
  # Scatter plot of RMS vs Wind Speed, colored and shaped by Type (Background/Operational)
  geom_point(aes(colour = Type, shape = Type),
             size = 2.5, alpha = 0.6, stroke = .3) +

  # Add a LOESS smoothed trend line per Type, with no confidence interval (se = FALSE)
```

```r
  geom_smooth(aes(colour = Type),
              method = "loess", span = .8,
              se = FALSE, linewidth = 0.9, alpha = .5) +

  # Create separate facet panels for each Type (faceted by Type)
  facet_wrap(~ Type, ncol = 2, scales = "free_x") +

  # Set custom colors for each Type and hide legend (guide = "none")
  scale_colour_manual(values = c("#1f77b4", "#ff7f0e"), guide = "none") +

  # Set custom shapes for points and hide legend (guide = "none")
  scale_shape_manual(values = c(16, 17), guide = "none") +

  # Titles and axis labels
  labs(title    = "Raw RMS vs Wind Speed",
       subtitle = "10-min blocks, 0.5-8 Hz band",
       x        = "Wind speed (m/s)",
       y        = "RMS displacement (nm)") +

  # Minimal theme with adjusted base font size
  theme_minimal(base_size = 11) +
  theme(
    panel.grid.minor = element_blank(),      # Remove minor gridlines for clarity
    panel.spacing    = unit(1, "lines")      # Increase spacing between facet panels
  )

# Display the plot
print(p_scatter)
```

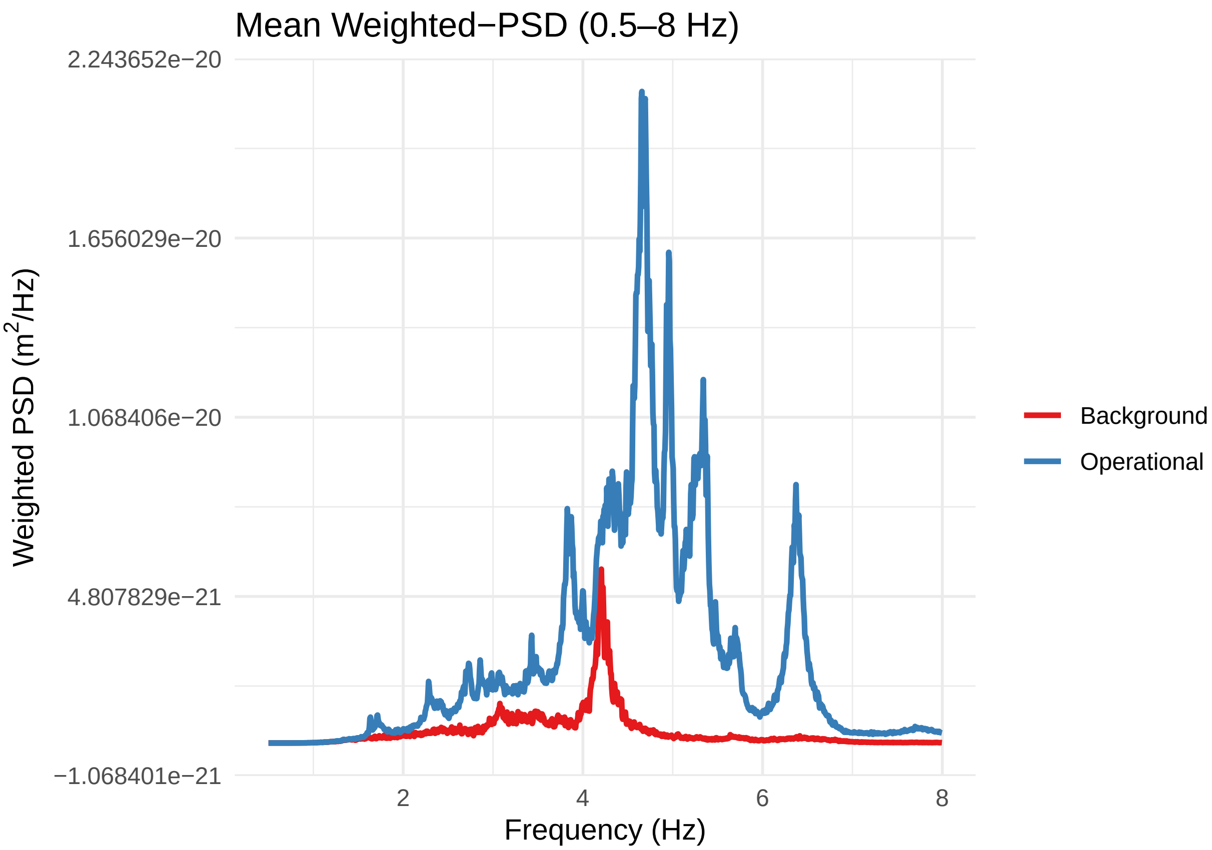
```
## `geom_smooth()` using formula = 'y ~ x'
```




```r
### Plot 8 - Mean Frequency-Weighted PSD (0.5-8 Hz) by Turbine State
cols <- c("Background"="#e41a1c", "Operational"="#377eb8")  # Define custom color palette for each stat

psd_mean <- psd_all %>%                          # Use full PSD dataset
  filter(between(Frequency_Hz, 0.5, 8)) %>%     # Filter for frequency band of interest (0.5-8 Hz)
  group_by(Type, Frequency_Hz) %>%              # Group by turbine state and frequency
  summarise(meanPSD = mean(PSD_wtd),            # Compute mean weighted PSD per group
            .groups="drop")                     # Drop grouping structure after summarizing

p_psd <- ggplot(psd_mean, aes(Frequency_Hz, meanPSD, colour = Type)) +
  geom_line(size = 1) +                         # Line plot of mean PSD across frequencies
  scale_colour_manual(values = cols, name = "") +  # Apply manual color scale with no legend title
  labs(title = "Mean Weighted-PSD (0.5-8 Hz)",      # Add plot title and axis labels
       x = "Frequency (Hz)",
       y = expression("Weighted PSD (m"^2*"/Hz)")) +
  theme_minimal() +                             # Use minimal theme for clean look
  theme(legend.position = "right")              # Position legend on the right
```

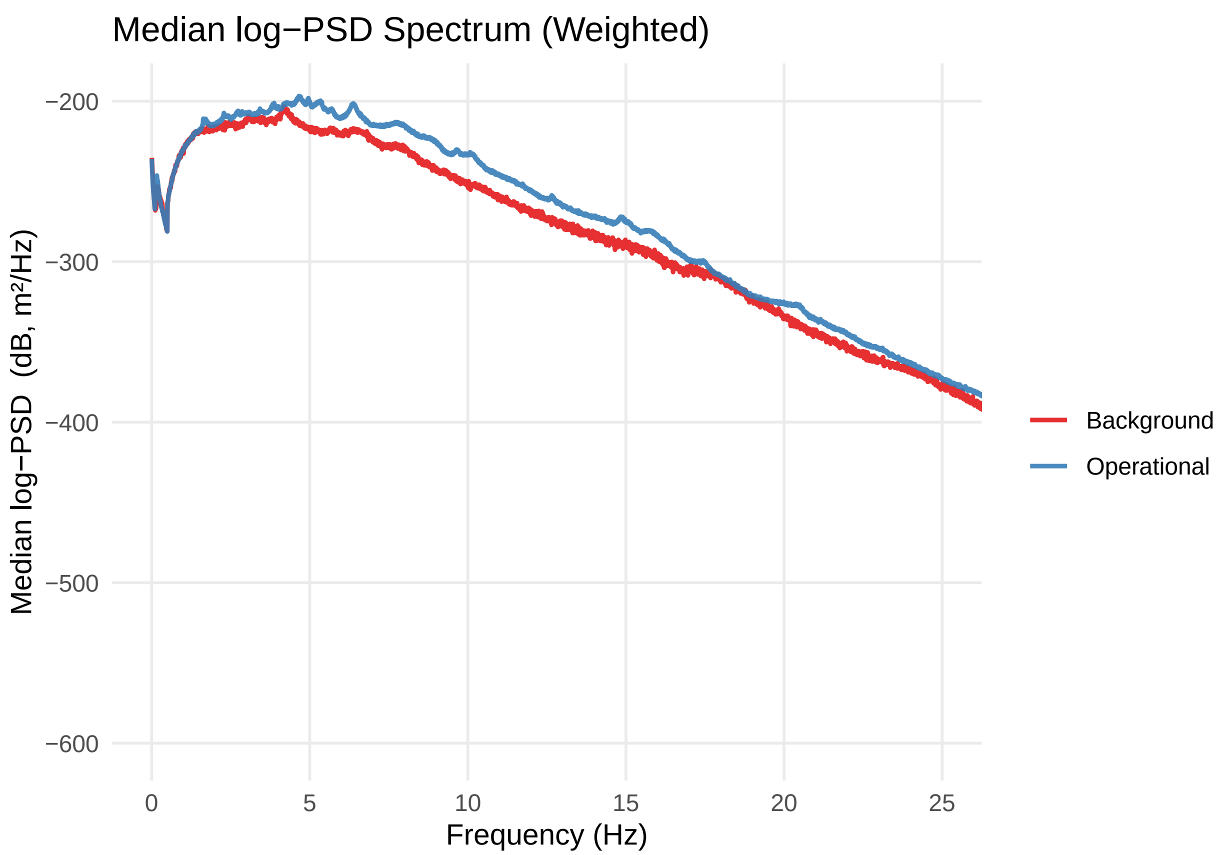
```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```r
print(p_psd)                                          # Display the plot
```

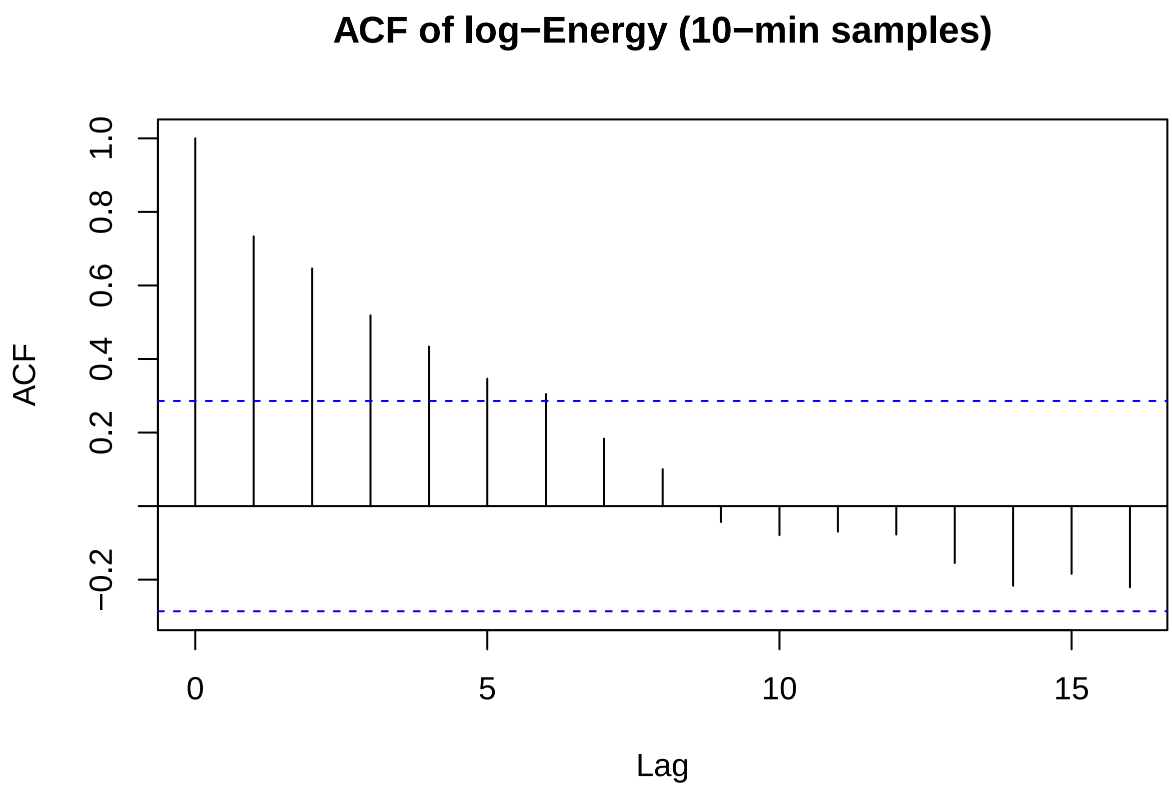



```r
### Plot 9 - Median log-PSD Spectrum (Weighted)

# 1.  Summary: median log-PSD per frequency & state
spec_df <- psd_all %>%                              # Start with PSD data (must include PSD_wtd)
  mutate(logPSD = 10 * log10(PSD_wtd)) %>%          # Convert PSD values to decibels (log10 scale)
  group_by(Operational, Frequency_Hz) %>%           # Group by turbine state and frequency
  summarise(med_logPSD = median(logPSD, na.rm = TRUE), .groups = "drop")  # Compute median log-PSD per 

# 2.  Plot
cols <- c("0" = "#e41a1c",     # red  = Background
          "1" = "#377eb8")     # blue = Operational

p_spec <- ggplot(spec_df, aes(Frequency_Hz, med_logPSD,        # Set x = frequency, y = median log-PSD
                              colour = factor(Operational))) + # Colour by turbine state
  geom_line(size = 0.8, alpha = .9) +                           # Line plot with moderate thickness
  scale_colour_manual(values = cols,                           # Manual colour assignment
                      labels = c("Background", "Operational"),# Legend labels
                      name   = "") +                            # No legend title
  coord_cartesian(xlim = c(0, 25)) +                           # Zoom in to 0-25 Hz band
  labs(title = "Median log-PSD Spectrum (Weighted)",          # Plot title and axis labels
       x = "Frequency (Hz)",
       y = "Median log-PSD  (dB, m²/Hz)") +
  theme_minimal(base_size = 11) +                              # Minimal theme with base font size
```

```
  theme(
    legend.position = "right",                          # Legend on the right
    panel.grid.minor = element_blank()                  # Remove minor gridlines for clarity
  )

print(p_spec)                                           # Render the plot
```

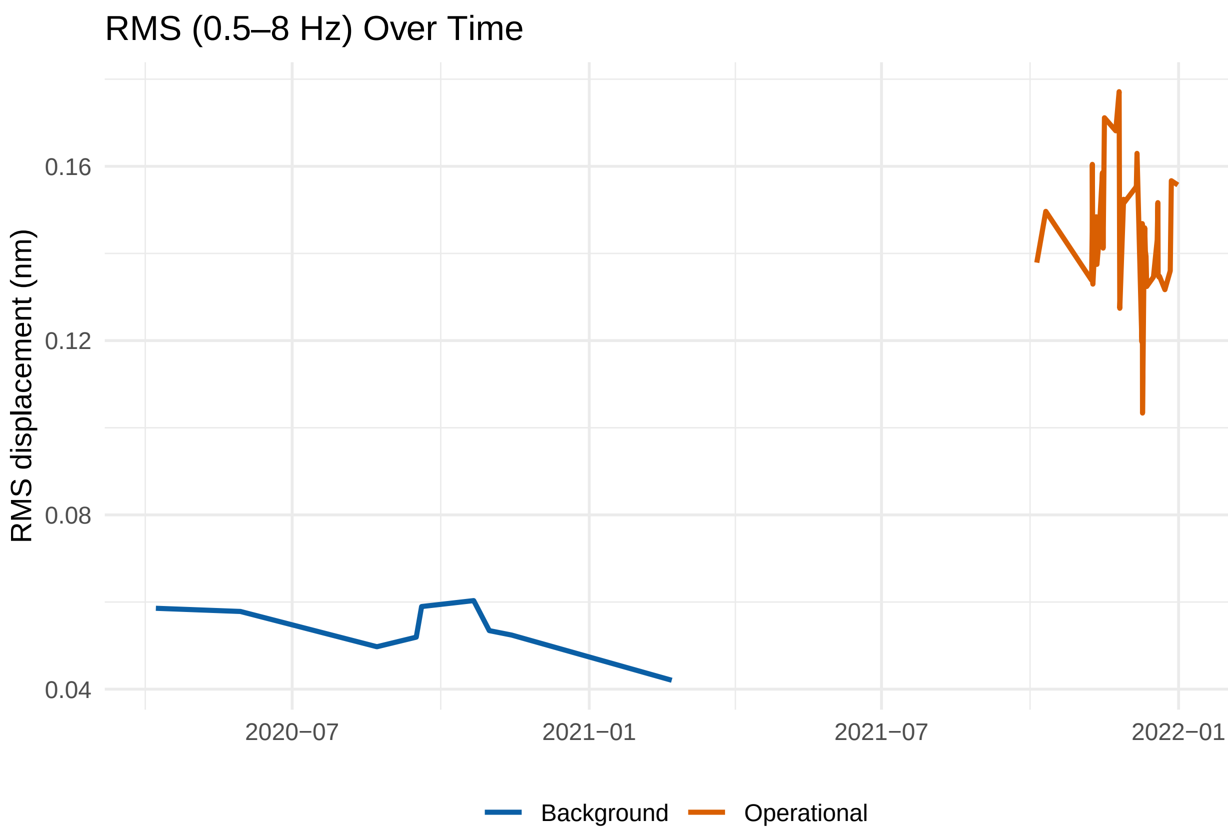



```
### Plot 10 - Autocorrelation of log-Energy

# Ensure time series is sorted chronologically by timestamp
e_ts <- energy %>% arrange(Datetime)

# Plot autocorrelation function (ACF) of log-energy values
# Useful to check for temporal dependence or seasonality in the 10-minute blocks
acf(e_ts$logE, na.action = na.pass, main = "ACF of log-Energy (10-min samples)")
```



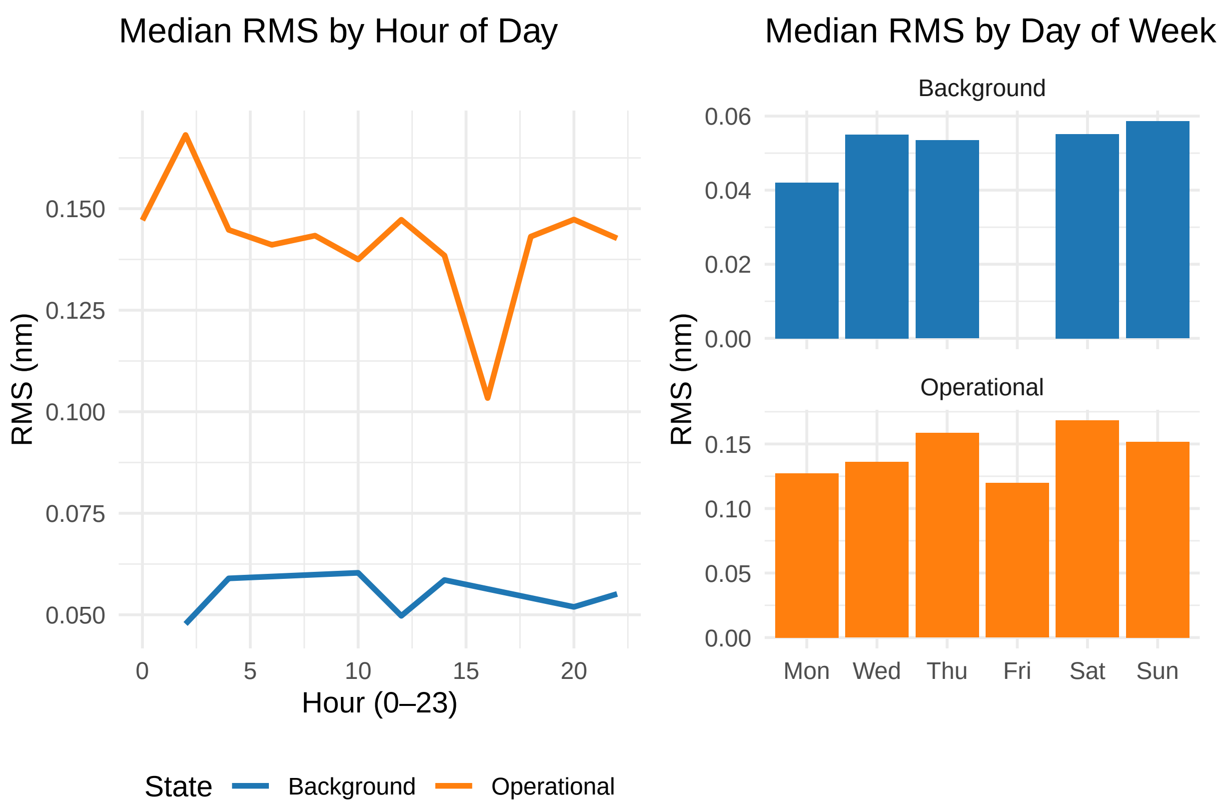
2. Temporal Patterns

```
### Plot 11 - Time Series of RMS Displacement (0.5-8 Hz Band)

# Time-series plot of RMS displacement (in nm) over time, coloured by turbine state
p_ts <- ggplot(energy, aes(Datetime, RMS_nm, colour = Type)) +
  geom_line(size = 0.9,         # Use thicker lines for better visibility
            alpha = 1) +        # Fully opaque lines (no transparency)
  scale_colour_manual(values = c("#0b5fa5",  # dark blue for Background
                                 "#d95f02"), # dark orange for Operational
                      name = "") +           # No legend title
  labs(title = "RMS (0.5-8 Hz) Over Time",   # Main plot title
       y = "RMS displacement (nm)",          # Y-axis label
       x = NULL) +                           # Omit X-axis label (Datetime self-explanatory)
  theme_minimal() +                          # Apply clean minimal theme
  theme(legend.position = "bottom")          # Move legend below plot

print(p_ts)                                  # Render the plot
```



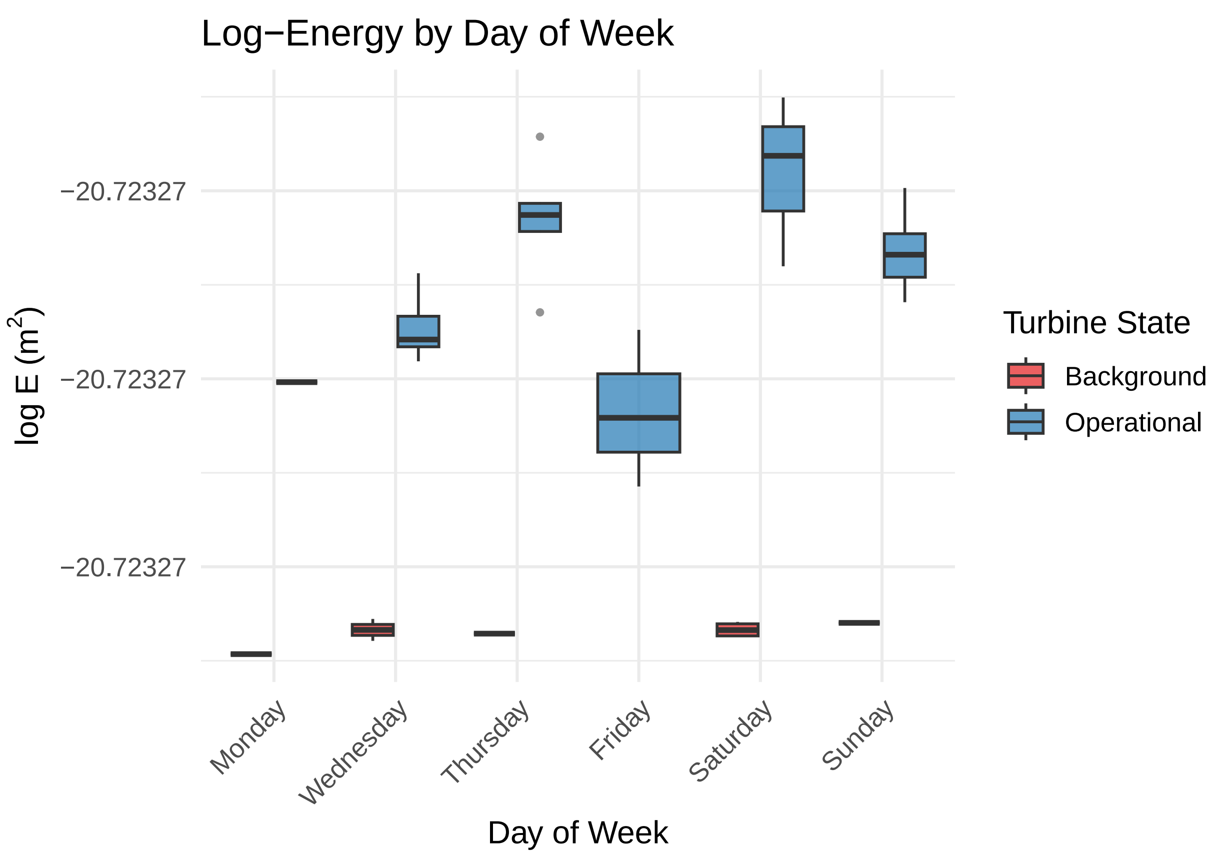

```
### Plot 12 - Median RMS by Hour and Day of Week

# --- Compute median RMS by hour --- #
hourly_rms <- energy %>%
  mutate(State = factor(Operational, labels = c("Background", "Operational"))) %>%
  group_by(Hour, State) %>%
  summarise(med_rms = median(RMS_nm, na.rm = TRUE), .groups = "drop")

# --- Compute median RMS by weekday --- #
weekday_rms <- energy %>%
  mutate(
    State   = factor(Operational, labels = c("Background", "Operational")),
    Weekday = factor(weekdays(as.Date(Date)),
                     levels = c("Monday", "Tuesday", "Wednesday", "Thursday",
                                "Friday", "Saturday", "Sunday"),
                     labels = c("Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"))
  ) %>%
  group_by(Weekday, State) %>%
  summarise(med_rms = median(RMS_nm, na.rm = TRUE), .groups = "drop")

# --- Plot A: Median RMS by Hour of Day --- #
p_hour <- ggplot(hourly_rms, aes(x = Hour, y = med_rms, colour = State)) +
  geom_line(linewidth = 1) +
  labs(title = "Median RMS by Hour of Day", x = "Hour (0-23)", y = "RMS (nm)") +
  scale_colour_manual(values = c("Background" = "#1f77b4", "Operational" = "#ff7f0e")) +
  theme_minimal() +
```

```r
  theme(legend.position = "bottom")

# --- Plot B: Median RMS by Day of Week, Faceted by State --- #
p_week <- ggplot(weekday_rms, aes(x = Weekday, y = med_rms, fill = State)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~State, ncol = 1, scales = "free_y") +
  labs(title = "Median RMS by Day of Week", x = NULL, y = "RMS (nm)") +
  scale_fill_manual(values = c("Background" = "#1f77b4", "Operational" = "#ff7f0e")) +
  theme_minimal()

# --- Combine side-by-side with patchwork --- #
p_hour + p_week + plot_layout(widths = c(1.2, 1))
```



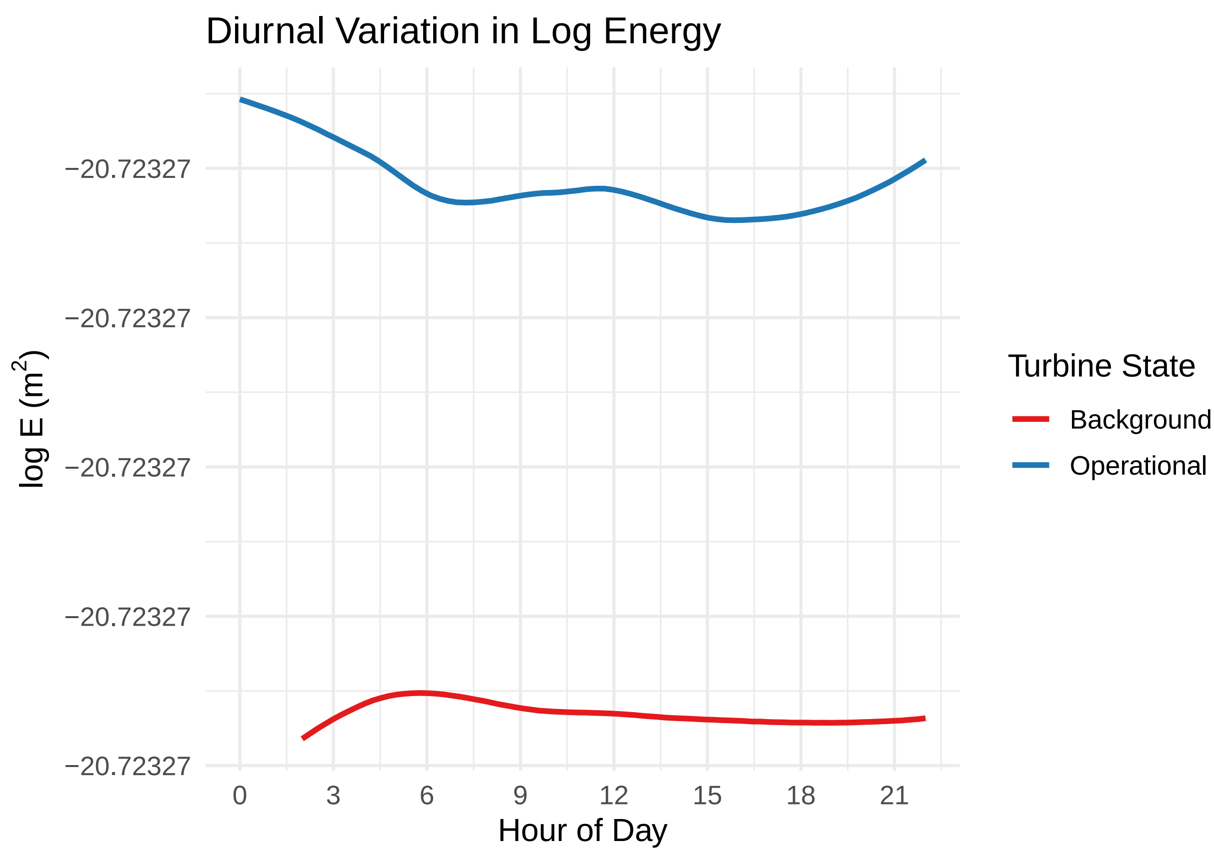

```r
# Add Weekday as a factor with correct order (optional)
energy$Weekday <- factor(weekdays(as.Date(energy$Date)),
                         levels = c("Monday", "Tuesday", "Wednesday", "Thursday",
                                    "Friday", "Saturday", "Sunday"))

### Plot 13 - Log-Energy variation across weekdays by turbine status
p_weekday <- ggplot(energy, aes(x = Weekday, y = logE, fill = factor(Operational))) +
  geom_boxplot(alpha = 0.7, outlier.size = 0.8, outlier.color = "grey40") +
  scale_fill_manual(values = c("0" = "#e41a1c", "1" = "#1f77b4"),
                    labels = c("Background", "Operational"),
                    name   = "Turbine State") +
  labs(title = "Log-Energy by Day of Week",
```

```r
       x = "Day of Week",
       y = expression(log~E~"(m"^2*")")) +
  theme_minimal(base_size = 12) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

# Print
print(p_weekday)
```

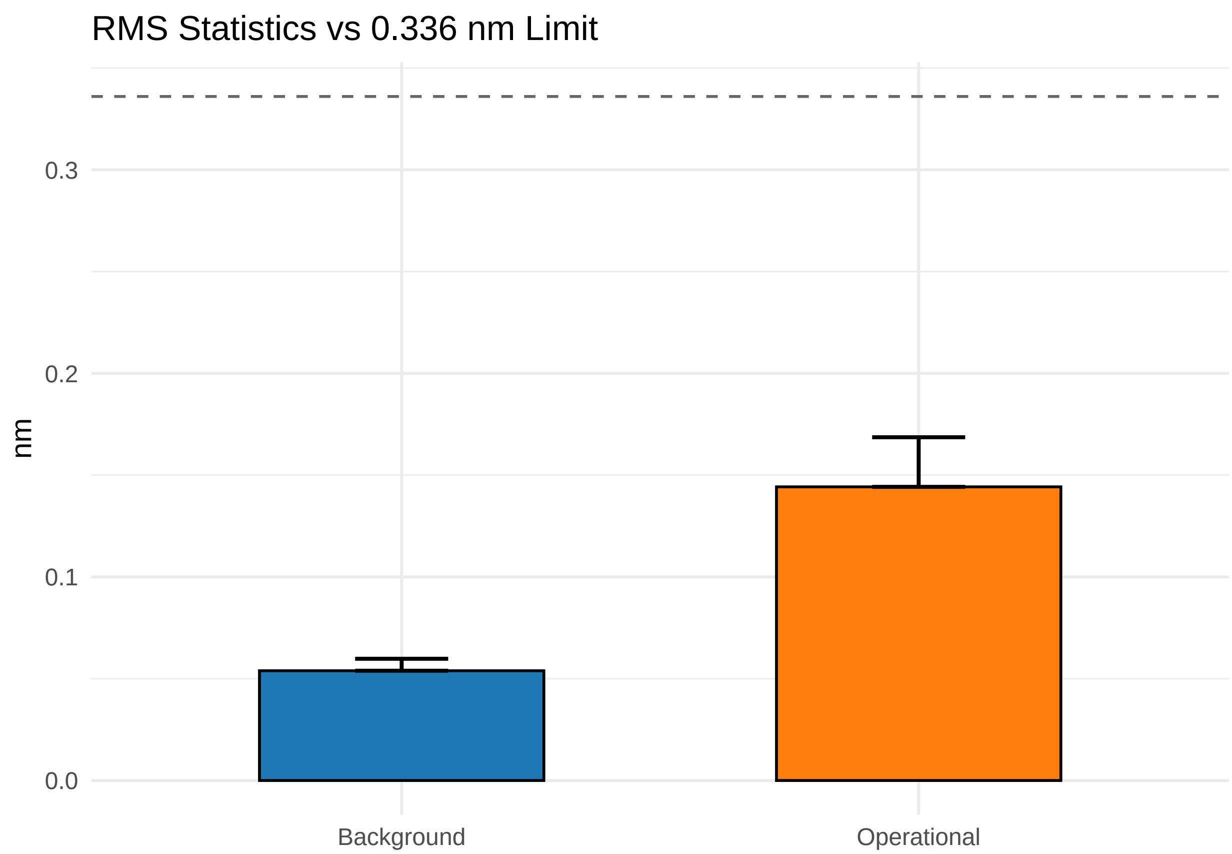



```r
### Plot 14 - Hourly smoothed variation in log-energy, faceted by turbine state
p_diurnal <- ggplot(energy, aes(x = Hour, y = logE, color = factor(Operational))) +
  geom_smooth(se = FALSE, linewidth = 1) +
  scale_color_manual(values = c("0" = "#e41a1c", "1" = "#1f77b4"),
                     labels = c("Background", "Operational"),
                     name   = "Turbine State") +
  scale_x_continuous(breaks = seq(0, 24, by = 3)) +
  labs(title = "Diurnal Variation in Log Energy",
       x = "Hour of Day",
       y = expression(log~E~"(m"^2*")")) +
  theme_minimal(base_size = 12)

# Print
print(p_diurnal)
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

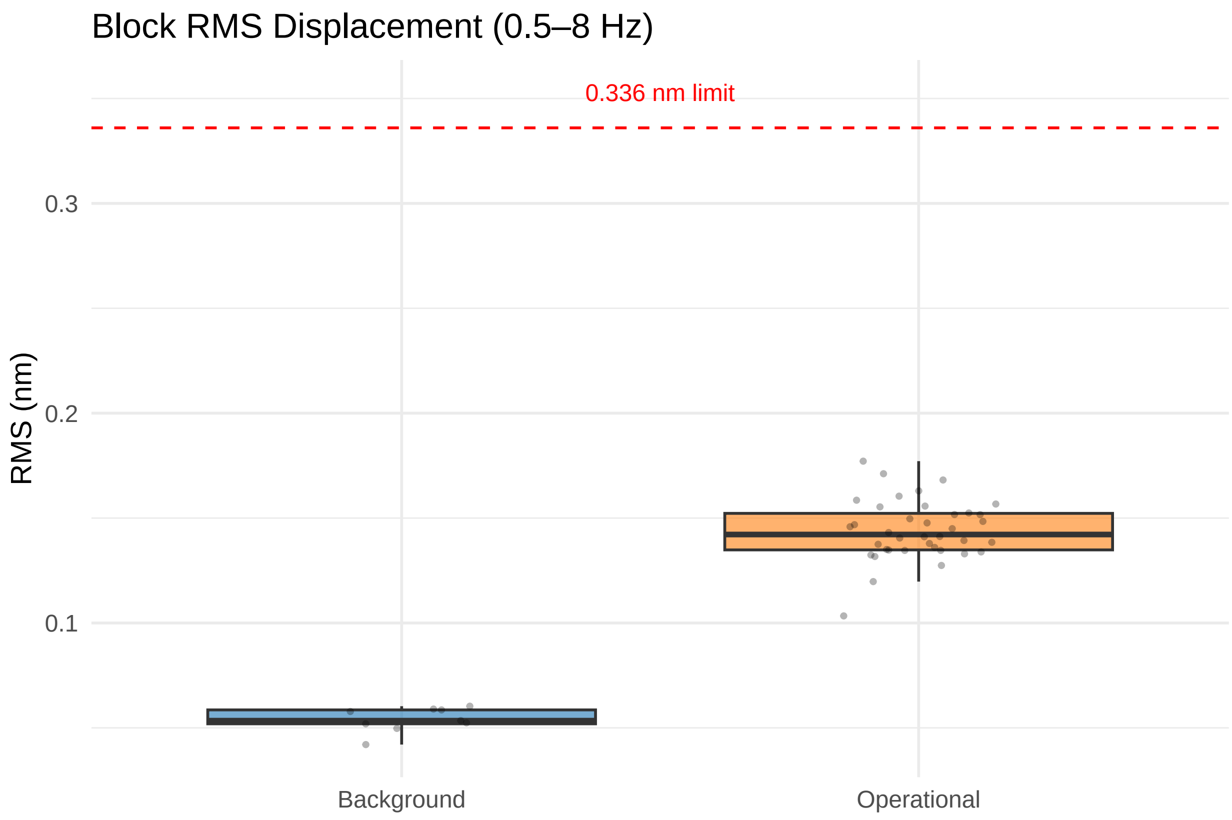



```
### plot 15 - RMS Displacement (0.5-8 Hz band) vs Regulatory Limit

# Define compliance limit and color palette for turbine states
limit_nm <- 0.336
cols      <- c("Background"="#1f77b4", "Operational"="#ff7f0e")

# Summarise RMS stats: mean and 95th percentile by state
stat_df <- energy %>%
  group_by(Type) %>%
  summarise(
    mean_nm = mean(RMS_nm),                    # mean RMS in nanometers
    p95_nm  = quantile(RMS_nm, .95)            # 95th percentile RMS
  ) |>
  mutate(Type = factor(Type, levels = c("Background","Operational")))  # set plotting order

# Create bar plot with error bars and threshold line
ggplot(stat_df, aes(Type, mean_nm, fill = Type)) +
  geom_col(width = .55, colour = "black", show.legend = FALSE) +        # bar: mean value
  geom_errorbar(aes(ymin = mean_nm, ymax = p95_nm),                     # error bar: up to 95th %ile
                width = .18, size = .7) +
  geom_hline(yintercept = limit_nm, linetype = "dashed", colour = "grey40") +  # compliance threshold
  scale_fill_manual(values = cols) +                                    # color mapping
  labs(title = "RMS Statistics vs 0.336 nm Limit",                      # plot title and labels
       y = "nm", x = NULL) +
  theme_minimal()                                                       # clean theme
```




```
### Plot 16 - RMS Displacement vs Turbine State (Boxplot + Threshold)

p_rms <- energy %>%
  mutate(State = factor(Operational, labels = c("Background", "Operational"))) %>%
  ggplot(aes(State, RMS_nm, fill = State)) +

  # Boxplot without outlier points for clarity
  geom_boxplot(outlier.shape = NA, alpha = 0.6) +

  # Overlay individual jittered points for distribution visibility
  geom_jitter(width = 0.15, alpha = 0.3, size = 0.6) +

  # Add horizontal reference line for compliance threshold
  geom_hline(yintercept = limit_nm, lty = 2, colour = "red") +

  # Annotate the threshold line
  annotate("text", x = 1.5, y = limit_nm * 1.05,
           label = "0.336 nm limit", color = "red", size = 3) +

  # Custom fill color for state categories
  scale_fill_manual(values = c("Background" = "#1f77b4",
                               "Operational" = "#ff7f0e")) +

  # Labels and theme
  labs(title = "Block RMS Displacement (0.5-8 Hz)",
       y = "RMS (nm)", x = "") +
```

```r
  theme_minimal() +
  theme(legend.position = "none")

# Display the plot
print(p_rms)
```




```r
### STATISTICAL MODELING APPROACH

#  Preprocessing for GAM modeling
# Standardize wind speed and wind direction for numerical stability
energy <- energy %>%
  mutate(
    ws_s   = scale(Wind.speed, center = TRUE, scale = TRUE)[,1],  # standardized wind speed
    wd_s   = scale(Wind.dir,    center = TRUE, scale = TRUE)[,1]  # standardized wind direction
  )

# Ensure date factor is available for random effect smooth
energy <- energy %>%
  mutate(DateF = factor(Date))  # convert Date to factor for s(DateF, bs = "re")

# Count unique values to dynamically select basis dimensions
n_ws    <- length(unique(energy$Wind.speed))   # number of unique wind speed values
n_dir   <- length(unique(energy$Wind.dir))     # number of unique wind direction values
n_hour  <- length(unique(energy$Hour))         # number of unique hourly values
```

```r
# Set maximum basis size (k) for splines, constrained by unique values
k_ws    <- min(10, n_ws)     # max 10 basis functions for wind speed
k_dir   <- min(12, n_dir)    # max 12 for wind direction
k_hour  <- min(24, n_hour)  # max 24 for hour-of-day (cyclic)
```

```r
### GAM1: Baseline model with turbine operational status only
# This model includes only a parametric effect for turbine state (ON/OFF)
# It tests the fundamental uplift in seismic energy due to operation
gam1 <- gam(
  logE ~ Operational,      # turbine ON/OFF effect (binary)
  data   = energy,
  method = "REML"          # use Restricted Maximum Likelihood for smoother estimation
)

# Summarize model output: coefficient estimates, fit statistics, etc.
summary(gam1)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## logE ~ Operational
##
## Parametric coefficients:
##               Estimate Std. Error    t value Pr(>|t|)
## (Intercept) -2.072e+01  1.246e-12 -1.663e+13   <2e-16 ***
## Operational  1.811e-11  1.386e-12  1.307e+01   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##
## R-sq.(adj) =  0.786   Deviance explained = 79.1%
## -REML = -1117.3  Scale est. = 1.3979e-23  n = 47
```

```r
# Diagnostic plots: QQ-plot, residuals vs fitted, leverage, etc.
appraise(gam1)
```


```r
### GAM2: Additive Linear Effect of Wind Speed
# This model extends GAM1 by including wind speed as a *linear* covariate,
# alongside the binary turbine operational status.
# Purpose: To assess whether raw wind speed (without smoothing) explains
# additional variation in log-energy (logE).
# Note: Wind speed is treated linearly here, not as a smooth function.
gam2 <- gam(
  logE ~ Operational + Wind.speed,
  data   = energy,
  method = "REML"      # Restricted Maximum Likelihood for smooth estimation
)

# Summary of model fit and coefficients
summary(gam2)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## logE ~ Operational + Wind.speed
##
## Parametric coefficients:
##               Estimate Std. Error    t value Pr(>|t|)
## (Intercept) -2.072e+01  2.151e-11 -9.636e+11   <2e-16 ***
```

```
## Operational  1.804e-11  1.302e-12  1.386e+01   <2e-16 ***
## Wind.speed   4.805e-12  1.811e-12  2.654e+00    0.011 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##
## R-sq.(adj) =  0.811   Deviance explained = 81.9%
## -REML = -1094.5  Scale est. = 1.2328e-23  n = 47
```

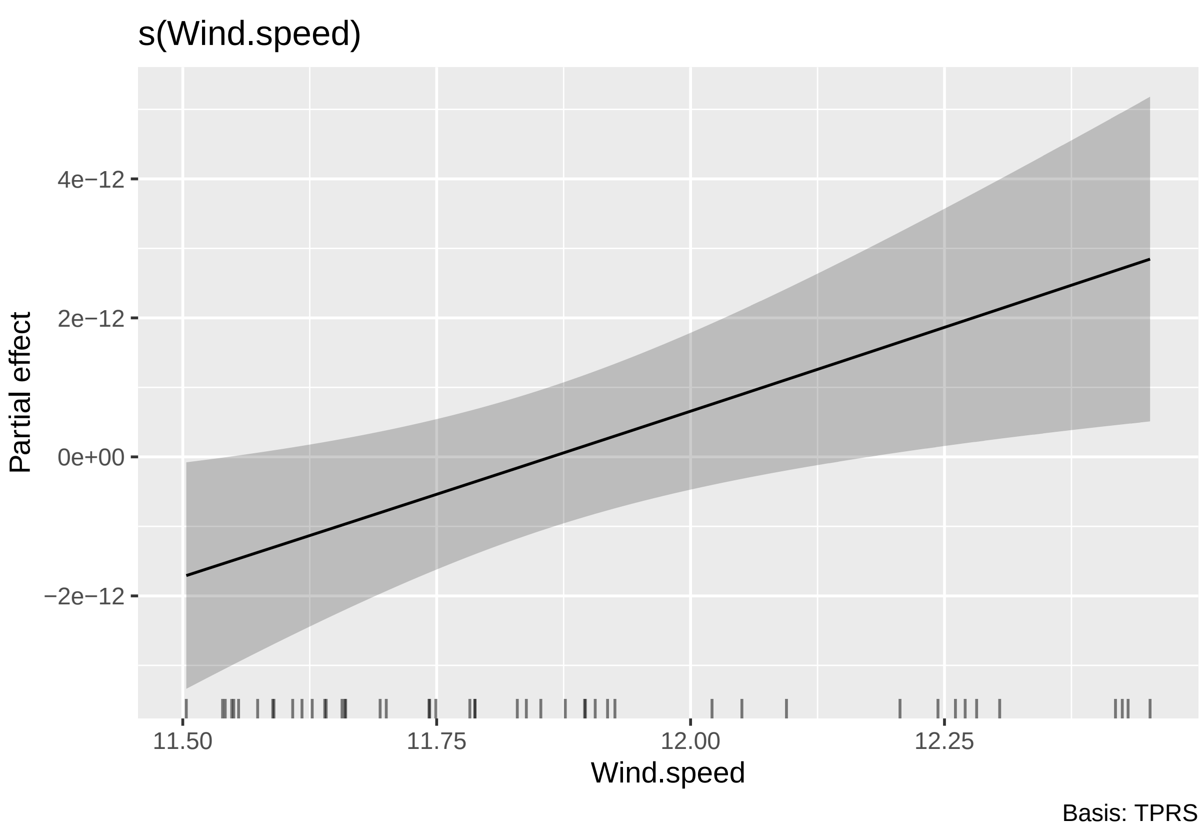
```r
# Diagnostic checks: residuals, QQ-plot, leverage, etc.
appraise(gam2)
```


```r
### GAM3: Add smooth term for wind speed
# Model log-energy as a function of turbine operation and a non-linear smooth for wind speed.
# This allows capturing non-linear background effects of wind speed on seismic energy.

gam3 <- gam(
  logE ~ Operational + s(Wind.speed, k = k_ws),  # add smooth spline for Wind.speed
  data   = energy,
  method = "REML"
)

# Summarize model: check parametric and smooth term significance
summary(gam3)
```

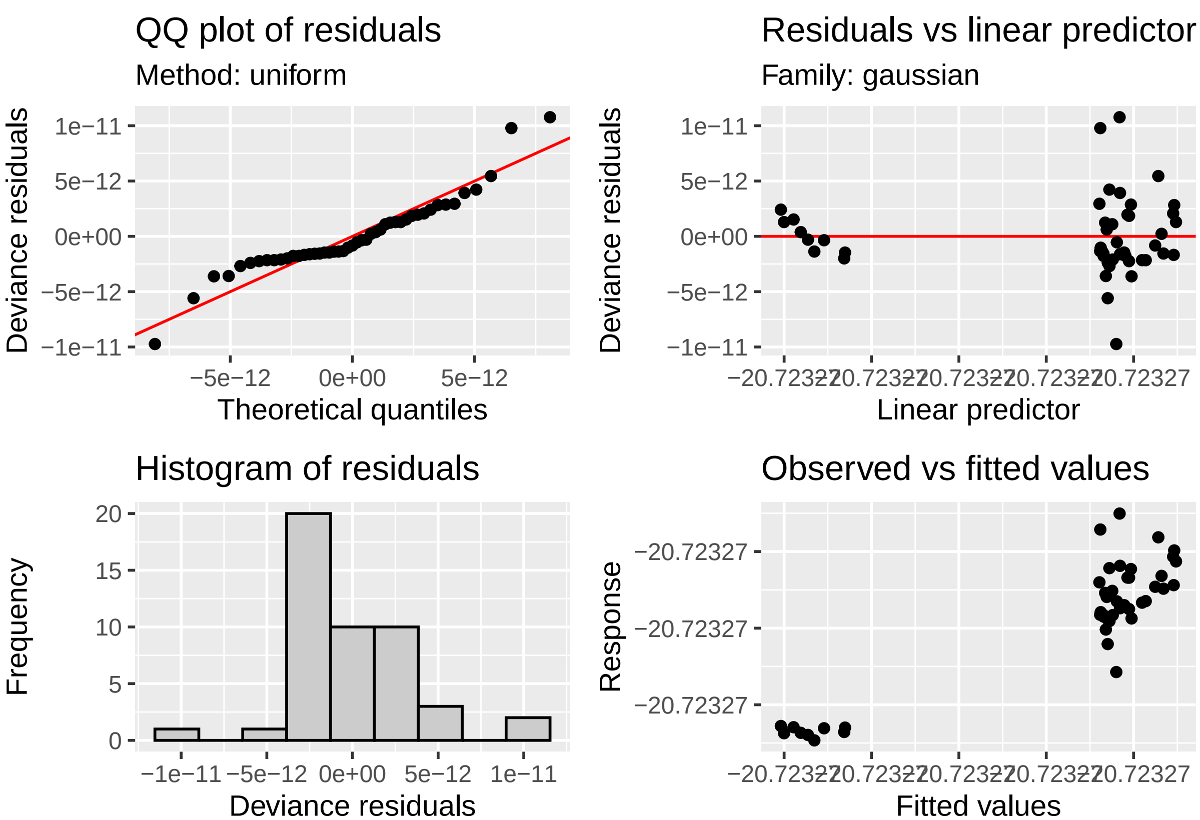
```
##
```

```
## Family: gaussian
## Link function: identity
##
## Formula:
## logE ~ Operational + s(Wind.speed, k = k_ws)
##
## Parametric coefficients:
##               Estimate Std. Error    t value Pr(>|t|)
## (Intercept) -2.072e+01  1.170e-12 -1.771e+13   <2e-16 ***
## Operational  1.805e-11  1.302e-12  1.387e+01   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                 edf Ref.df     F p-value
## s(Wind.speed) 1.013  1.026 6.901  0.0117 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.811   Deviance explained =   82%
## -REML = -1093.2  Scale est. = 1.2325e-23  n = 47
```

```
# Visualize smooth function of Wind.speed
draw(gam3, select = "s(Wind.speed)")
```



```r
# Residual diagnostics: QQ-plot, residual-vs-fitted, leverage
appraise(gam3)
```



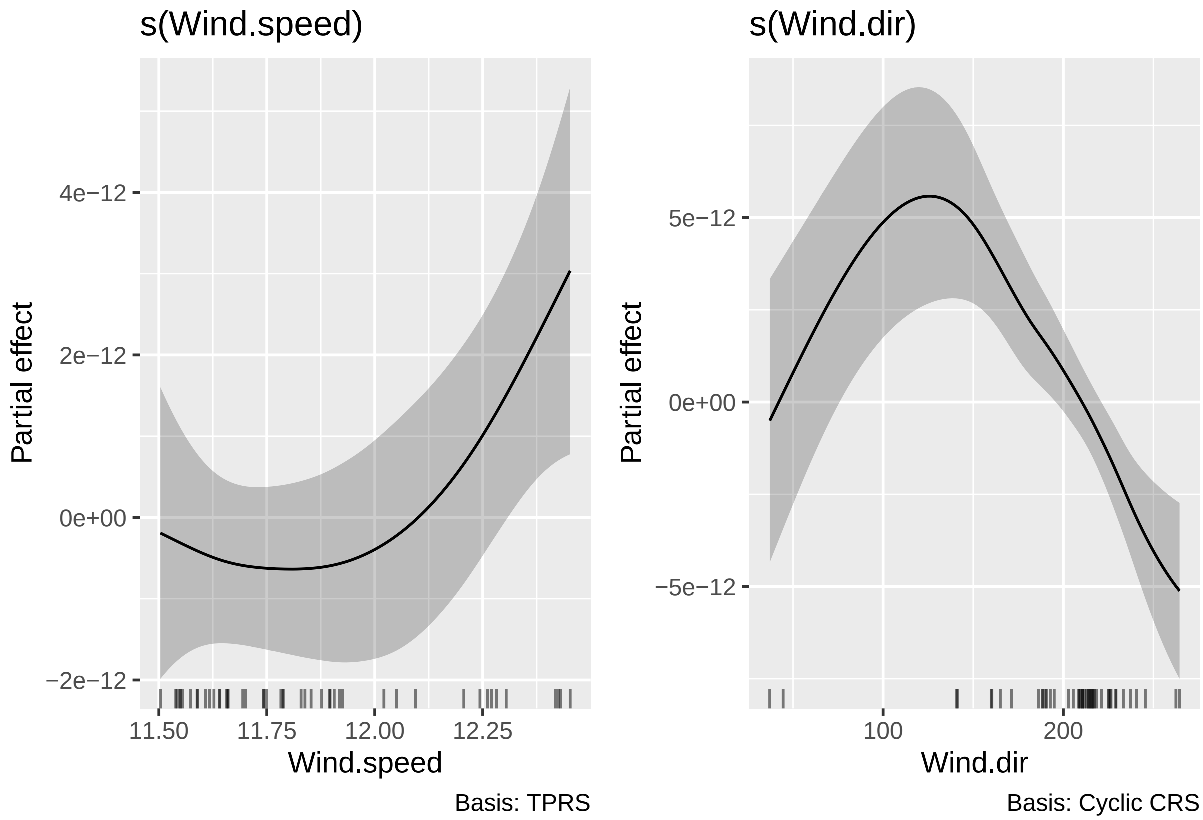

```r
### GAM4: Add cyclic wind direction term
# This model includes:
# - A baseline operational shift (ON/OFF effect),
# - A smooth effect for wind speed,
# - A cyclic smooth for wind direction (0-360° wrapped),
# - No temporal or date-based variation yet.

gam4 <- gam(
  logE ~ Operational
         + s(Wind.speed, k = k_ws)             # smooth for wind speed
         + s(Wind.dir, bs = "cc", k = k_dir),  # cyclic smooth for wind direction
  data   = energy,
  method = "REML",
  knots  = list(Wind.dir = c(0, 360))          # enforce cyclicity from 0 to 360 degrees
)
```

```
## Warning in newton(lsp = lsp, X = G$X, y = G$y, Eb = G$Eb, UrS = G$UrS, L = G$L,
## : Fitting terminated with step failure - check results carefully
```

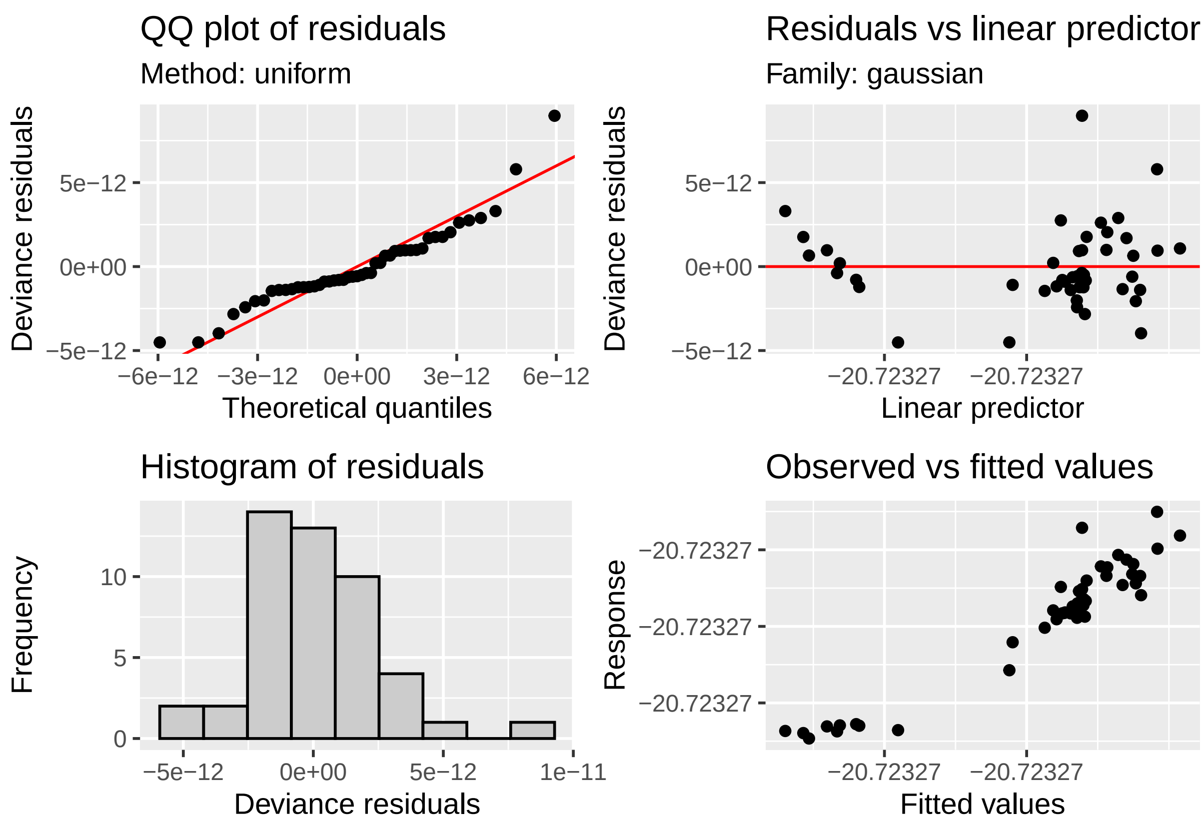
```r
# Display model summary including significance of each term
summary(gam4)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## logE ~ Operational + s(Wind.speed, k = k_ws) + s(Wind.dir, bs = "cc",
##     k = k_dir)
##
## Parametric coefficients:
##               Estimate Std. Error    t value Pr(>|t|)
## (Intercept) -2.072e+01  9.275e-13 -2.234e+13   <2e-16 ***
## Operational  1.773e-11  1.048e-12  1.691e+01   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                 edf Ref.df     F  p-value
## s(Wind.speed) 2.072  2.556 2.859   0.0449 *
## s(Wind.dir)   2.935 10.000 3.589 3.08e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.898   Deviance explained = 91.1%
## -REML = -1102.3  Scale est. = 6.6675e-24  n = 47
```

```r
# Plot smooth terms for wind speed and wind direction
draw(gam4, select = c("s(Wind.speed)", "s(Wind.dir)"))
```




```
# Run diagnostic checks: residual plots, QQ, histogram, etc.
appraise(gam4)
```

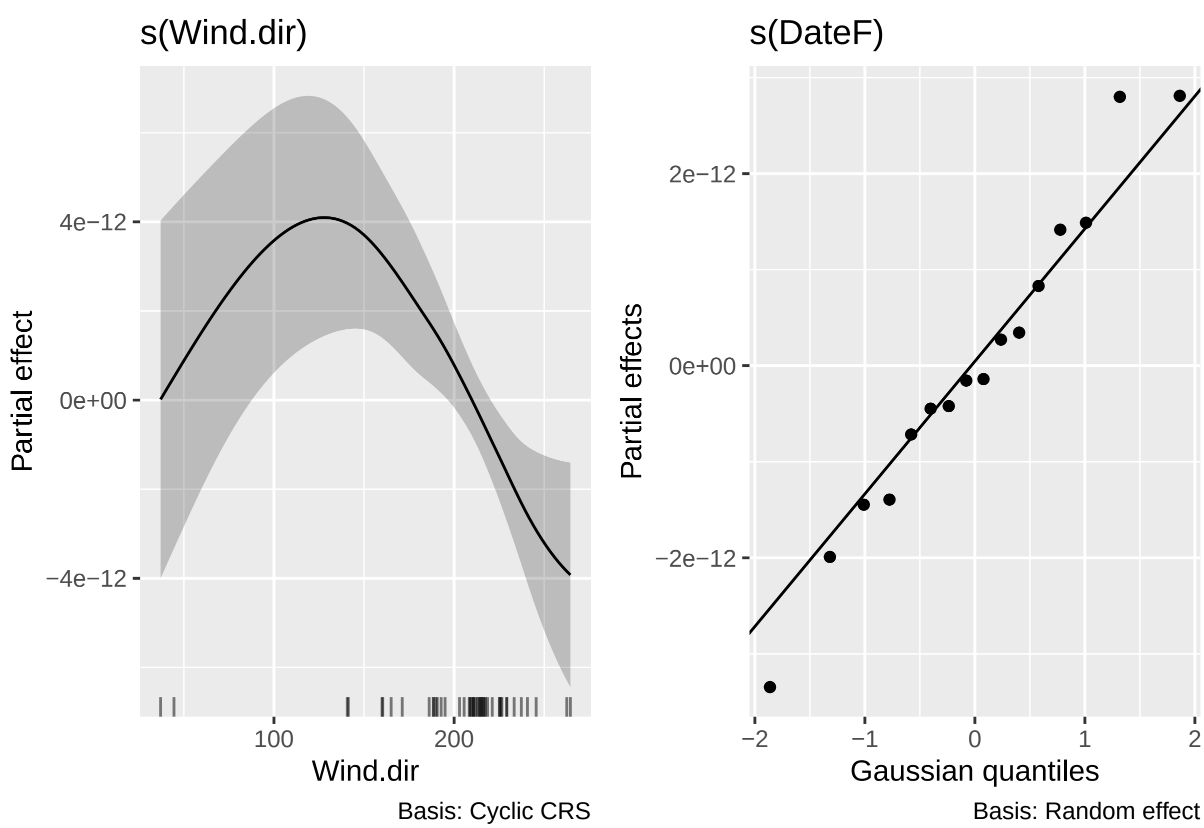



```r
# GAM5: Add cyclic wind-direction and daily random effect
# This model includes:
# - Operational status as a parametric term
# - A smooth spline for wind speed (linear basis)
# - A cyclic cubic spline ("cc") for wind direction
# - A random intercept smooth for date (captures unobserved day-level heterogeneity)

gam5 <- gam(
  logE ~ Operational                          # binary effect of turbine operation
       + s(Wind.speed, k = k_ws)              # spline for wind speed
       + s(Wind.dir, bs = "cc", k = k_dir)   # cyclic spline for wind direction (0-360°)
       + s(DateF, bs = "re"),                 # random effect for each day
  data   = energy,
  method = "REML",                            # penalized likelihood estimation
  knots  = list(Wind.dir = c(0,360))          # specify cyclic bounds for Wind.dir
)

# Output summary of model fit: coefficients, EDFs, p-values
summary(gam5)
```

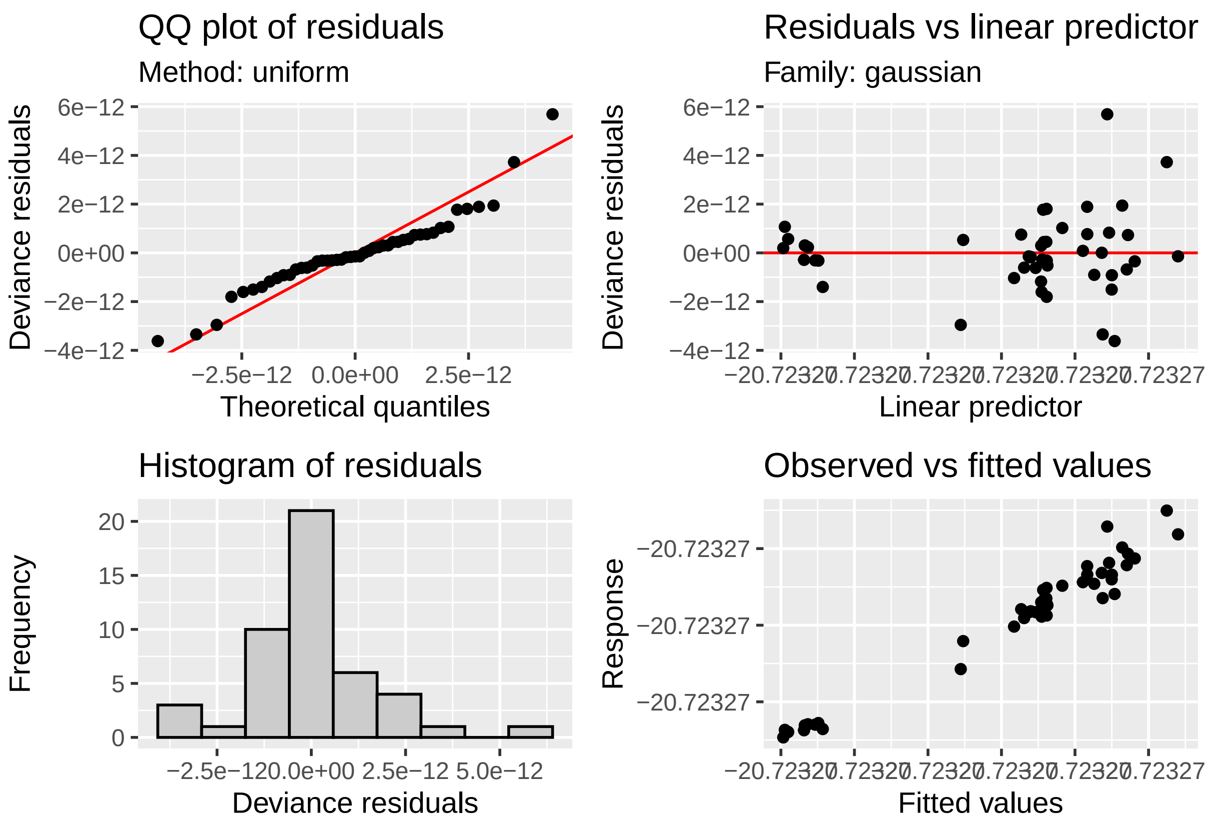
```
##
## Family: gaussian
## Link function: identity
##
## Formula:
```

```
## logE ~ Operational + s(Wind.speed, k = k_ws) + s(Wind.dir, bs = "cc",
##     k = k_dir) + s(DateF, bs = "re")
##
## Parametric coefficients:
##               Estimate Std. Error    t value Pr(>|t|)
## (Intercept) -2.072e+01  1.092e-12 -1.897e+13  < 2e-16 ***
## Operational  1.814e-11  1.437e-12  1.262e+01 5.11e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                 edf Ref.df      F  p-value
## s(Wind.speed) 2.111  2.563  1.456 0.172764
## s(Wind.dir)   2.377 10.000 16.118 0.000302 ***
## s(DateF)      8.287 14.000  2.653 0.000168 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.945   Deviance explained = 96.2%
## -REML = -1108.5  Scale est. = 3.5736e-24  n = 47
```

```
# Visualize selected smooth terms: Wind direction and Date-level effects
draw(gam5, select = c("s(Wind.dir)", "s(DateF)"))
```



```r
# Check residuals, QQ plot, leverage, fitted vs residuals
appraise(gam5)
```



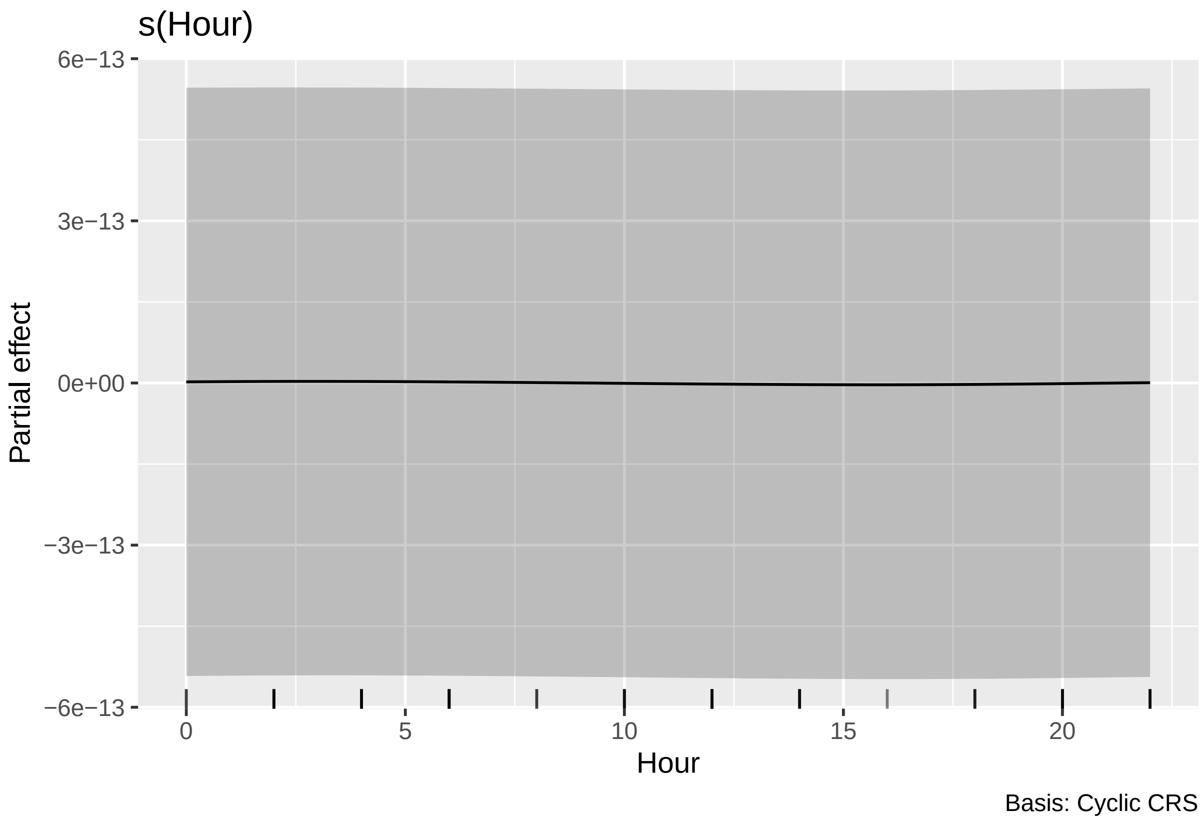

```r
# GAM6: Add diurnal cycle to previous model
# This model includes:
#   - A parametric term for turbine operational status (Operational)
#   - Smooth terms for wind speed and wind direction (cyclic)
#   - A random effect smooth for day-level variability (s(DateF, bs = "re"))
#   - A cyclic smooth for hour-of-day effects (s(Hour, bs = "cc"))
# The cyclic bases (bs = "cc") enforce continuity at endpoints (e.g., 0 = 24)

gam6 <- gam(
  logE ~ Operational                           # binary ON/OFF shift
         + s(Wind.speed, k = k_ws)            # smooth effect of wind speed
         + s(Wind.dir,     bs = "cc", k = k_dir)  # cyclic wind direction smooth
         + s(DateF,        bs = "re")         # random effect for daily fluctuations
         + s(Hour,         bs = "cc", k = k_hour),# diurnal cyclic smooth
  data   = energy,
  method = "REML",                            # restricted maximum likelihood
  knots  = list(Wind.dir = c(0, 360),         # enforce periodicity in wind direction
                Hour     = c(0, 24))          # and hour-of-day
)
```

```
## Warning in newton(lsp = lsp, X = G$X, y = G$y, Eb = G$Eb, UrS = G$UrS, L = G$L,
## : Fitting terminated with step failure - check results carefully
```

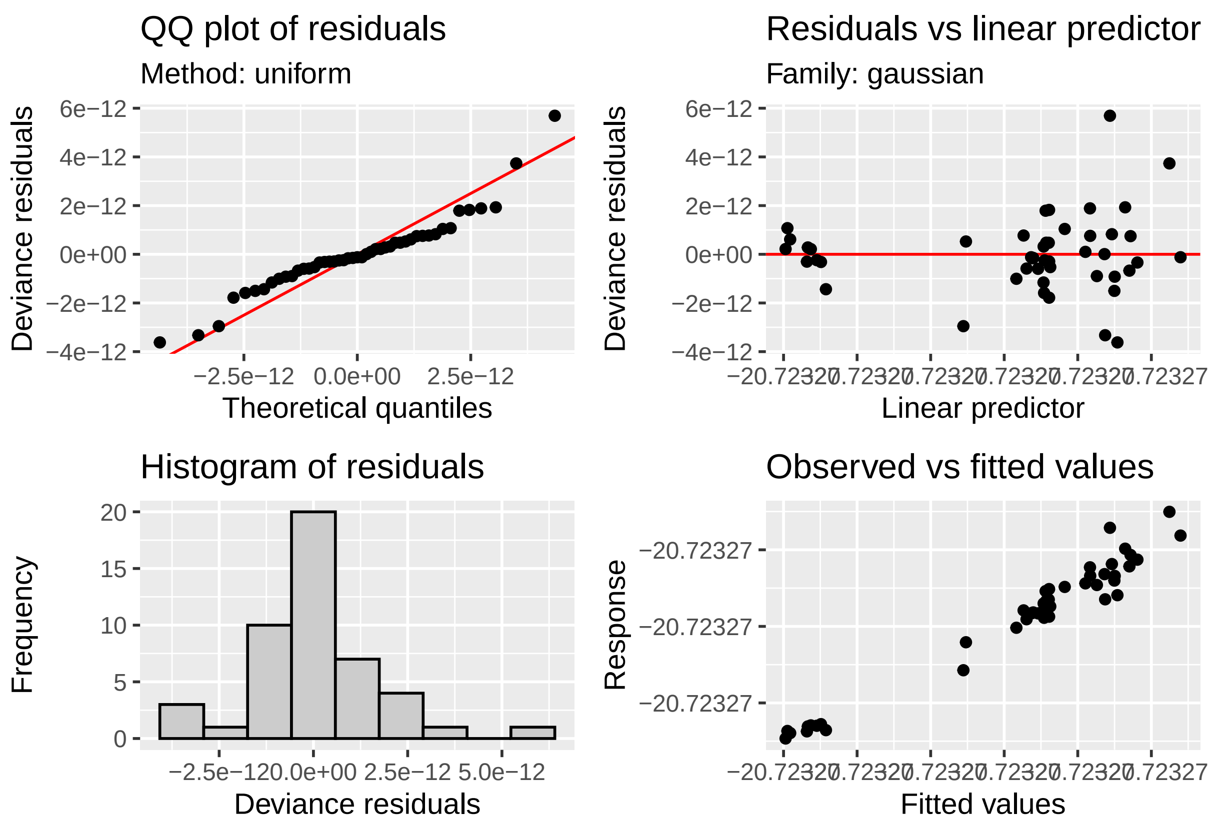
```r
# Model diagnostics and visualization
summary(gam6)                  # inspect coefficients and smooth significance
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## logE ~ Operational + s(Wind.speed, k = k_ws) + s(Wind.dir, bs = "cc",
##     k = k_dir) + s(DateF, bs = "re") + s(Hour, bs = "cc", k = k_hour)
##
## Parametric coefficients:
##               Estimate Std. Error    t value Pr(>|t|)
## (Intercept) -2.072e+01  1.088e-12 -1.904e+13  < 2e-16 ***
## Operational  1.814e-11  1.432e-12  1.267e+01 4.61e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                   edf Ref.df     F  p-value
## s(Wind.speed) 2.12880  2.584  1.44 0.175461
## s(Wind.dir)   2.38188 10.000 16.07 0.000310 ***
## s(DateF)      8.25120 14.000  2.65 0.000159 ***
## s(Hour)       0.01142 10.000  0.00 0.423718
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.945   Deviance explained = 96.2%
## -REML = -1108.6  Scale est. = 3.5711e-24  n = 47
```

```r
draw(gam6, select = "s(Hour)") # visualize the diurnal effect
```




```r
appraise(gam6)                    # residual diagnostics (QQ, hist, etc.)
```




```r
# ### Model Selection via AIC
# Compare Akaike Information Criterion (AIC) values across candidate models:
#   - Lower AIC indicates a better trade-off between model fit and complexity
#   - Used here to evaluate which GAM best explains variation in logE
#     while penalising excessive smoothness or overfitting

AIC(gam1, gam2, gam3, gam4, gam5, gam6)
```

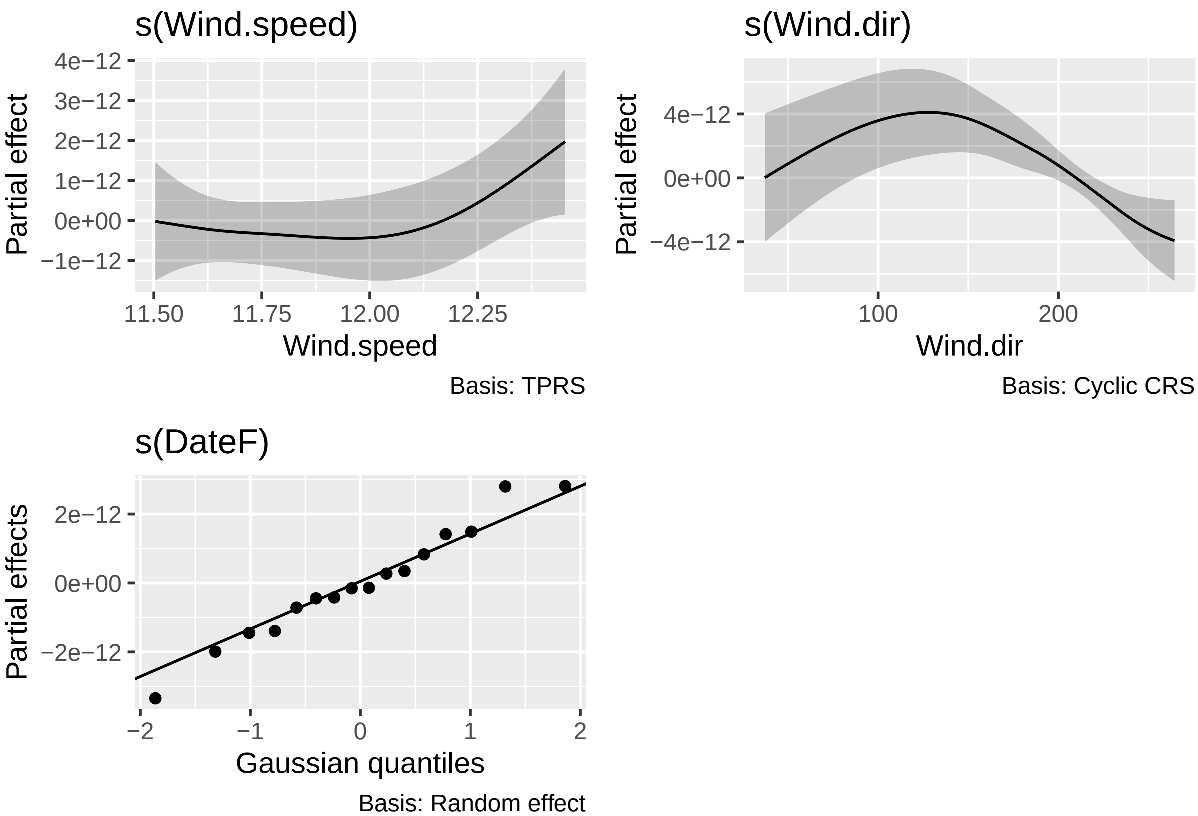
```
##              df       AIC
## gam1   3.000000 -2336.015
## gam2   4.000000 -2340.988
## gam3   4.025906 -2340.953
## gam4   9.197043 -2363.962
## gam5  18.337927 -2385.194
## gam6  18.416458 -2385.036
```

```r
# GAM5: Incorporating Temporal Random Effects
# Model formula:
#   logE ~ Operational + s(Wind.speed) + s(Wind.dir) + s(DateF, bs = "re")
#
# This model extends previous specifications by:
#   - Including a random effect (`s(DateF, bs = "re")`) to account for
#     unobserved daily variation in seismic energy.
#   - Capturing both environmental drivers (wind speed & direction)
#     and operational state (turbine ON/OFF).
```

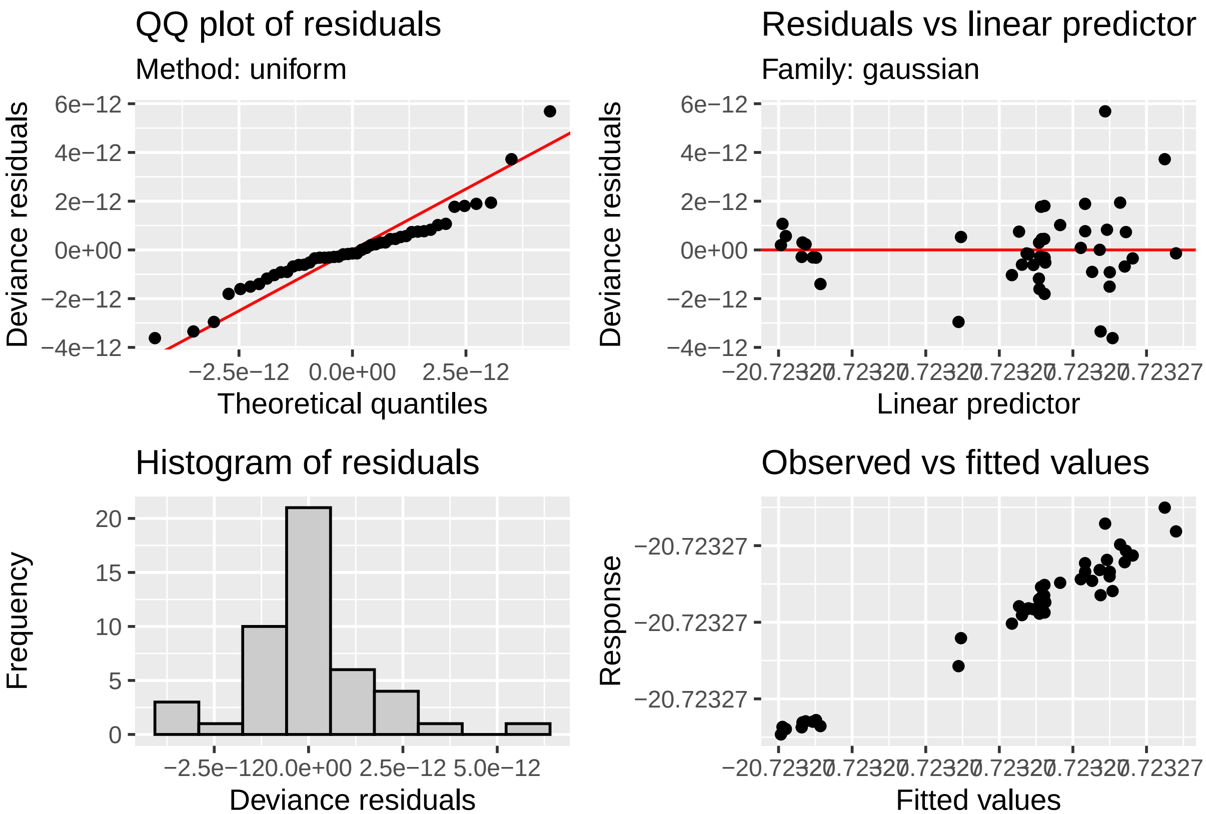
```r
#   - Allowing `Wind.dir` to vary smoothly and cyclically.
#
# Summary provides parametric and smooth term significance.
summary(gam5)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## logE ~ Operational + s(Wind.speed, k = k_ws) + s(Wind.dir, bs = "cc",
##     k = k_dir) + s(DateF, bs = "re")
##
## Parametric coefficients:
##               Estimate Std. Error    t value Pr(>|t|)
## (Intercept) -2.072e+01  1.092e-12 -1.897e+13  < 2e-16 ***
## Operational  1.814e-11  1.437e-12  1.262e+01 5.11e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                 edf Ref.df      F  p-value
## s(Wind.speed) 2.111  2.563  1.456 0.172764
## s(Wind.dir)   2.377 10.000 16.118 0.000302 ***
## s(DateF)      8.287 14.000  2.653 0.000168 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.945   Deviance explained = 96.2%
## -REML = -1108.5  Scale est. = 3.5736e-24  n = 47
```

```r
# --- Visual Diagnostics: Smooth Terms -------------------------------------
# Plot the estimated smooth functions for wind speed, direction, and
# random daily variation to assess their functional forms and confidence intervals.
draw(gam5, rug = FALSE)
```




```r
# Residual Diagnostics
# Assess model fit and assumption validity:
#    - Residual-vs-fitted plots
#    - Normal Q-Q plot
#    - Histogram of residuals
#    - Leverage and influential point check
appraise(gam5)
```




```r
# GAM 7: Add cyclic smooth for Hour to previous model
gam7 <- gam(
  logE ~
    Operational +                                    # binary ON/OFF turbine effect
    s(Wind.speed, k = k_ws) +                        # smooth wind speed effect
    s(Wind.dir,     bs = "cc", k = k_dir) +          # cyclic spline for wind direction (0-360°)
    s(DateF,        bs = "re") +                     # random effect for each date
    s(Hour,         bs = "cc", k = k_hour),          # cyclic spline for hour-of-day (0-24)
  data   = energy,
  method = "REML",                                   # restricted maximum likelihood
  knots  = list(                                     # ensure proper knot placement for cyclic terms
    Wind.dir = c(0, 360),
    Hour     = c(min(energy$Hour), max(energy$Hour))
  )
)
```

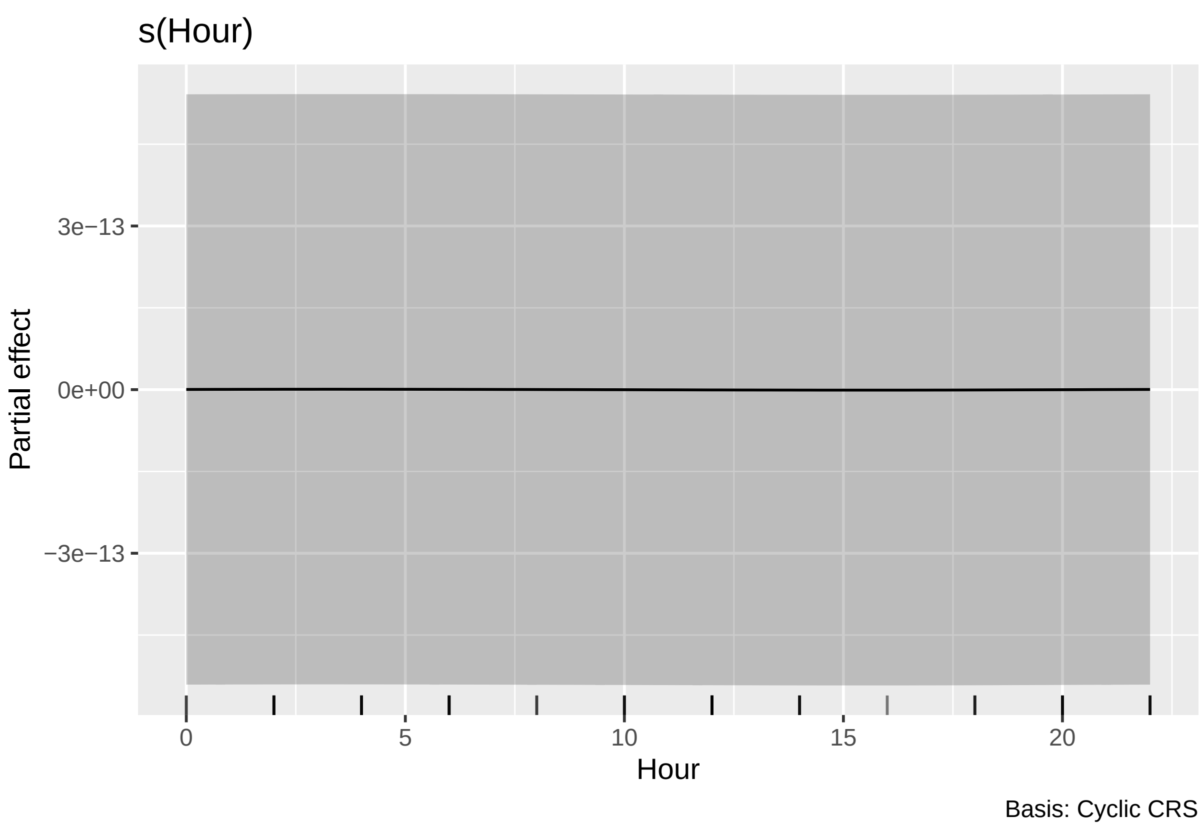
```
## Warning in newton(lsp = lsp, X = G$X, y = G$y, Eb = G$Eb, UrS = G$UrS, L = G$L,
## : Fitting terminated with step failure - check results carefully
```

```r
#  Model summary and diagnostics
summary(gam7)                     # check coefficients, EDFs, and p-values
```

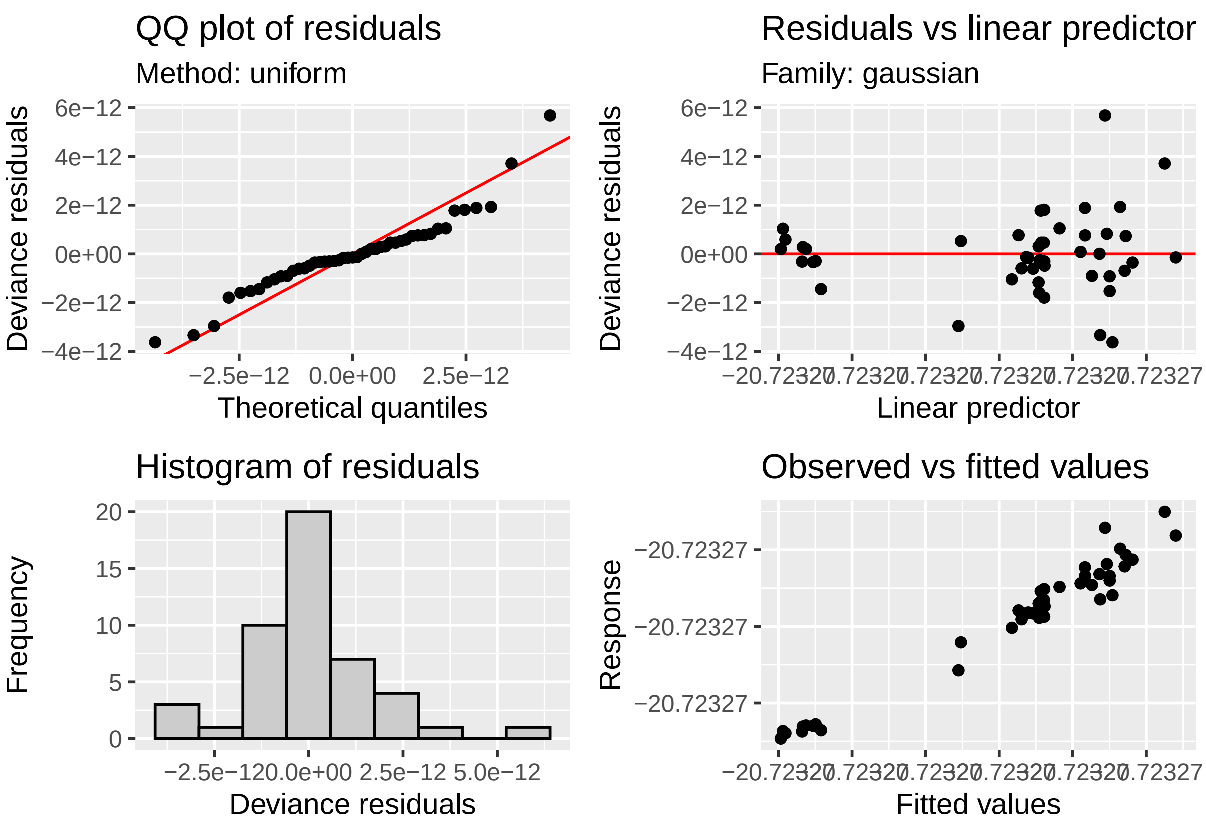
```
##
## Family: gaussian
## Link function: identity
```

```
##
## Formula:
## logE ~ Operational + s(Wind.speed, k = k_ws) + s(Wind.dir, bs = "cc",
##     k = k_dir) + s(DateF, bs = "re") + s(Hour, bs = "cc", k = k_hour)
##
## Parametric coefficients:
##               Estimate Std. Error    t value Pr(>|t|)
## (Intercept) -2.072e+01  1.089e-12 -1.902e+13  < 2e-16 ***
## Operational  1.812e-11  1.433e-12  1.264e+01 4.92e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                    edf Ref.df      F  p-value
## s(Wind.speed) 2.126212  2.581  1.448 0.173593
## s(Wind.dir)   2.383102 10.000 16.164 0.000297 ***
## s(DateF)      8.261474 14.000  2.660 0.000158 ***
## s(Hour)       0.003136 10.000  0.000 0.480145
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.945   Deviance explained = 96.2%
## -REML = -1108.5  Scale est. = 3.5705e-24  n = 47
```

```
draw(gam7, select = "s(Hour)")        # visualize diurnal variation
```



```r
appraise(gam7)                          # residuals, QQ plot, leverage
```




```r
#  Compare AIC across GAM 5-7
AIC(gam5, gam6, gam7)                   # lower AIC indicates better fit
```

```
##             df       AIC
## gam5 18.33793 -2385.194
## gam6 18.41646 -2385.036
## gam7 18.34924 -2385.130
```

```r
# GAM with Interaction: Wind Speed × Operational Status + Temporal Effects
gam_int <- gam(
  logE ~
    s(Wind.speed,                     k = k_ws) +               # baseline wind-speed effect
    s(Wind.speed, by = Operational, k = k_ws) +                # additional effect when turbines ON
    s(Wind.dir,     bs = "cc",        k = k_dir) +             # cyclic spline for wind direction
    s(Hour,         bs = "cc",        k = k_hour) +            # diurnal cyclic spline
    s(DateF,        bs = "re") +                               # daily random effects
    Operational,                                                # parametric shift (ON vs OFF)
  data   = energy,
  method = "REML",
  knots  = list(
    Wind.dir = c(0, 360),                                     # enforce cyclicity in wind direction
    Hour     = c(min(energy$Hour), max(energy$Hour))         # enforce cyclicity for time-of-day
```

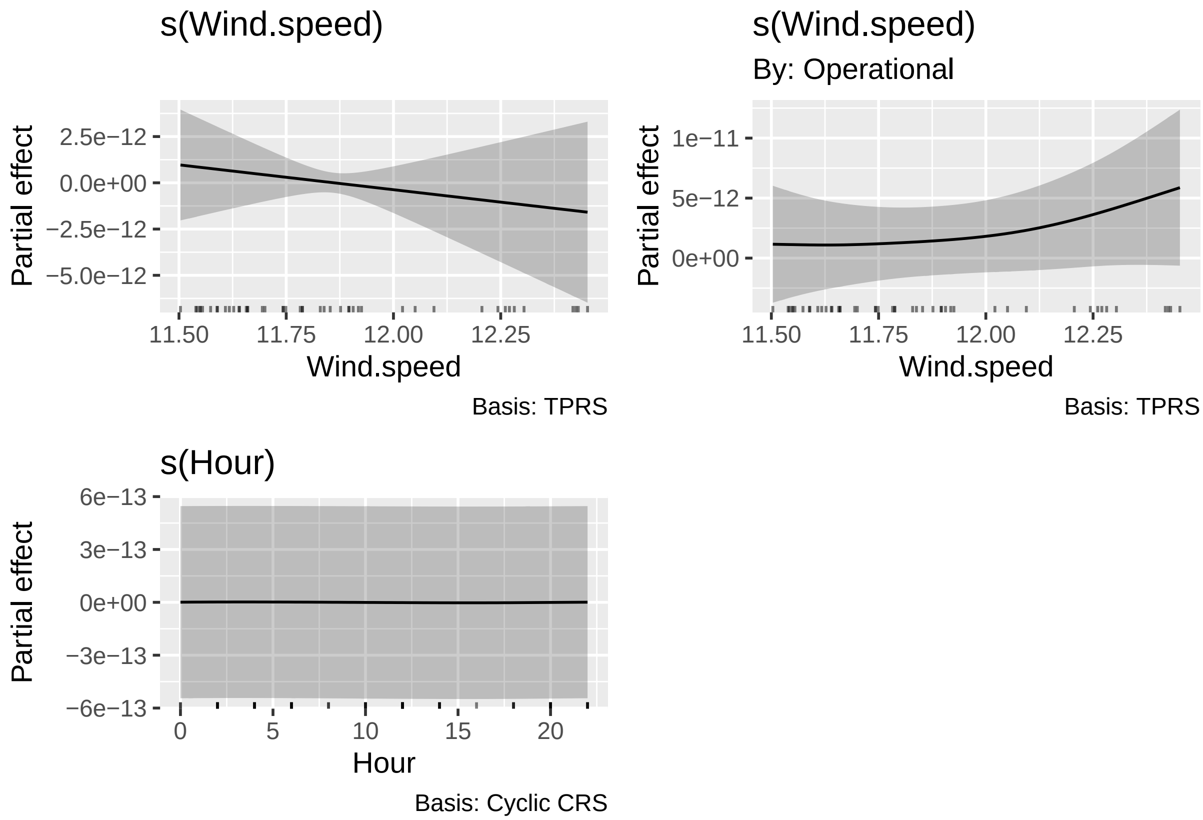
```r
  )
)
```

```
## Warning in newton(lsp = lsp, X = G$X, y = G$y, Eb = G$Eb, UrS = G$UrS, L = G$L,
## : Fitting terminated with step failure - check results carefully
```

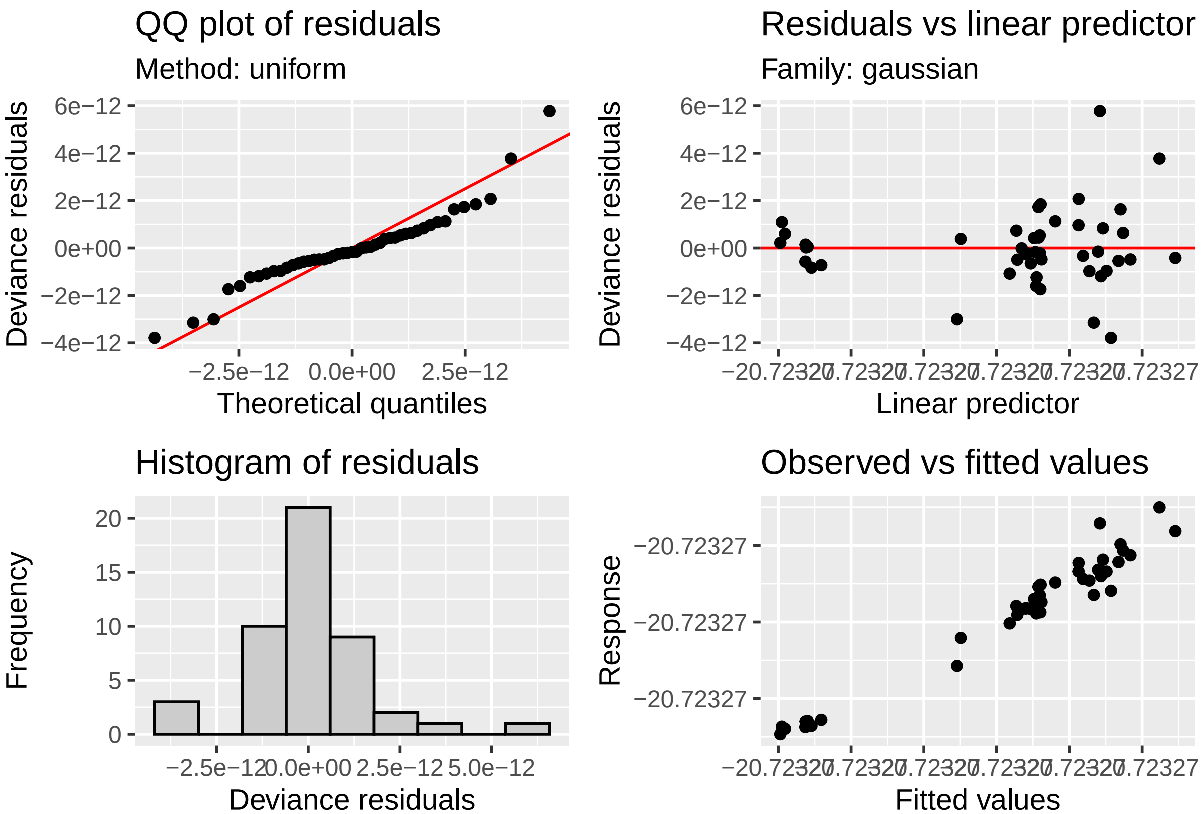
```r
# Model diagnostics and visualization
summary(gam_int)                                                # model summary
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## logE ~ s(Wind.speed, k = k_ws) + s(Wind.speed, by = Operational,
##     k = k_ws) + s(Wind.dir, bs = "cc", k = k_dir) + s(Hour, bs = "cc",
##     k = k_hour) + s(DateF, bs = "re") + Operational
##
## Parametric coefficients:
##               Estimate Std. Error    t value Pr(>|t|)
## (Intercept) -2.072e+01  1.045e-12 -1.984e+13  < 2e-16 ***
## Operational  1.641e-11  2.041e-12  8.041e+00 3.41e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                                edf Ref.df      F  p-value
## s(Wind.speed)              1.00064  1.001  0.409 0.527021
## s(Wind.speed):Operational 2.11167  2.582  1.036 0.346482
## s(Wind.dir)               2.42964 10.000 14.524 0.586888
## s(Hour)                   0.01014 10.000  0.000 0.950207
## s(DateF)                  7.31407 14.000  2.094 0.000664 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Rank: 56/57
## R-sq.(adj) =  0.945   Deviance explained = 96.1%
## -REML = -1056.8  Scale est. = 3.5958e-24  n = 47
```

```r
draw(gam_int, select = c("s(Wind.speed)",
                         "s(Wind.speed):Operational",
                         "s(Hour)"))                            # plot key smooth terms
```




```
appraise(gam_int)                                          # residual diagnostics
```



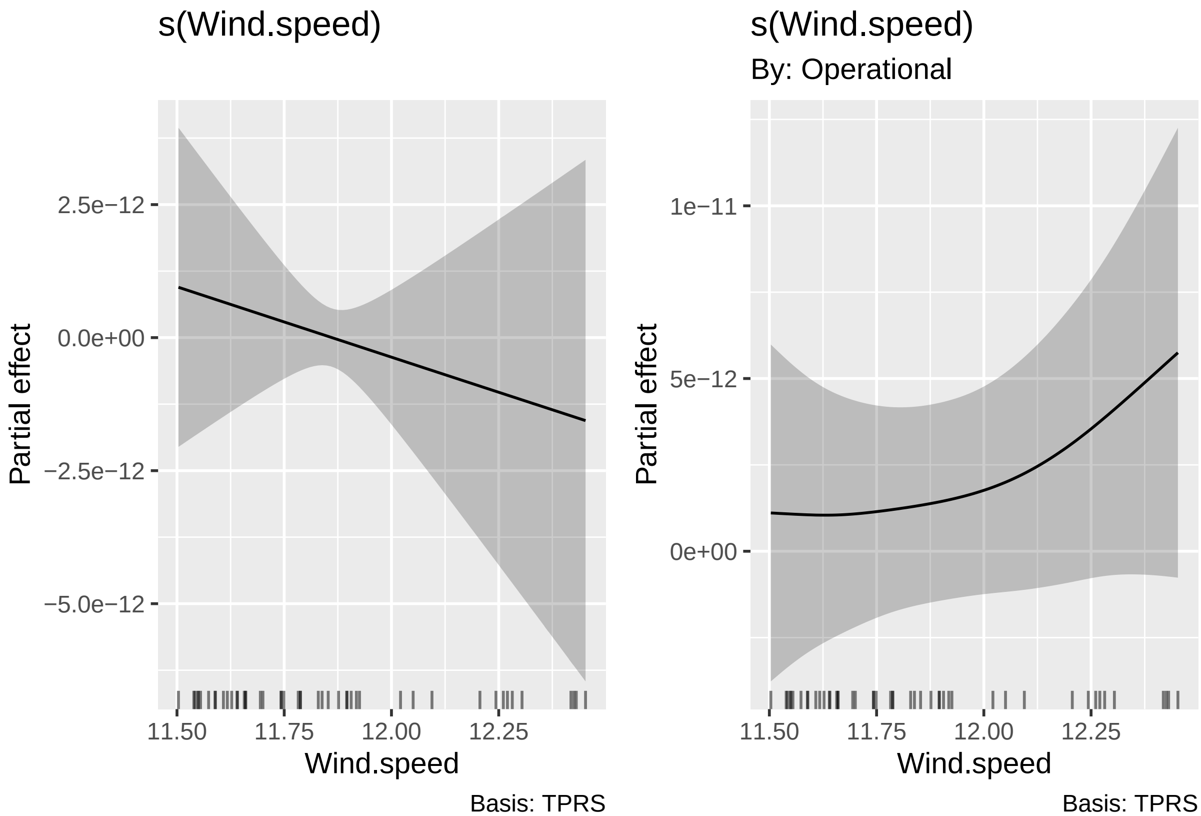

```r
AIC(gam5, gam6, gam_int)                                   # model comparison using AIC
```

```
##               df       AIC
## gam5    18.33793 -2385.194
## gam6    18.41646 -2385.036
## gam_int 19.22162 -2383.170
```

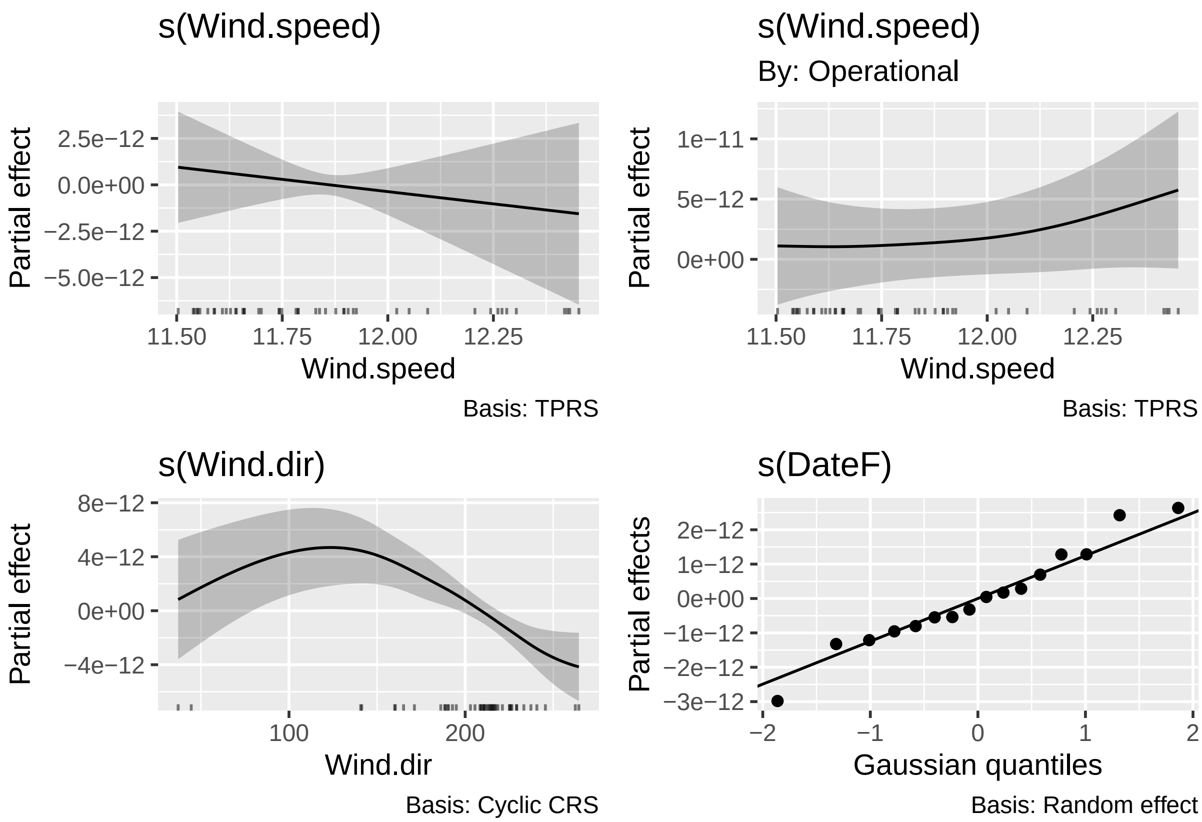
```r
### Final Generalized Additive Model (GAM)
# Best model based on AIC, residual diagnostics, and smooth term significance

gam_combo <- gam(
  logE ~
    Operational +                               # baseline ON/OFF shift due to turbines
    s(Wind.speed, k = k_ws) +                   # smooth background wind-speed effect
    s(Wind.speed, by = Operational, k = k_ws) + # interaction: wind-speed effect under turbine ON
    s(Wind.dir,    bs = "cc", k = k_dir) +      # cyclic spline for wind direction
    s(DateF,       bs = "re"),                  # random intercept per day
  data   = energy,
  method = "REML",
  knots  = list(Wind.dir = c(0, 360))           # ensure cyclicity in wind direction
)

# Model summary (parametric + smooth terms)
summary(gam_combo)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## logE ~ Operational + s(Wind.speed, k = k_ws) + s(Wind.speed,
##     by = Operational, k = k_ws) + s(Wind.dir, bs = "cc", k = k_dir) +
##     s(DateF, bs = "re")
##
## Parametric coefficients:
##               Estimate Std. Error    t value Pr(>|t|)
## (Intercept) -2.072e+01  1.046e-12 -1.981e+13  < 2e-16 ***
## Operational  1.650e-11  2.043e-12  8.077e+00 3.09e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                             edf Ref.df      F  p-value
## s(Wind.speed)             1.004  1.006  0.391 0.536308
## s(Wind.speed):Operational 2.110  2.579  1.054 0.278081
## s(Wind.dir)               2.432 10.000 12.677 0.000104 ***
## s(DateF)                  7.320 14.000  2.012 0.000659 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Rank: 46/47
## R-sq.(adj) =  0.945   Deviance explained = 96.2%
## -REML = -1056.8  Scale est. = 3.5994e-24  n = 47
```

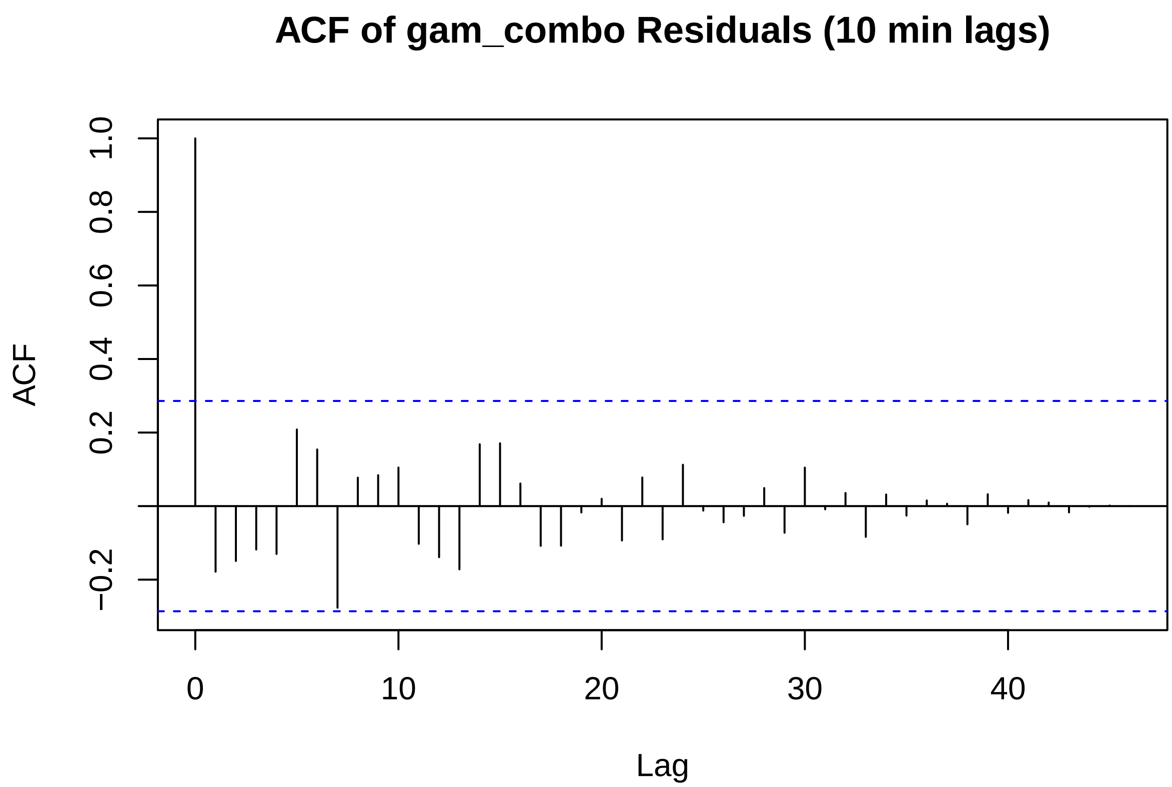
```r
# Visualize partial effects of wind speed (overall and by operational state)
draw(gam_combo, select = c("s(Wind.speed)", "s(Wind.speed):Operational"))
```



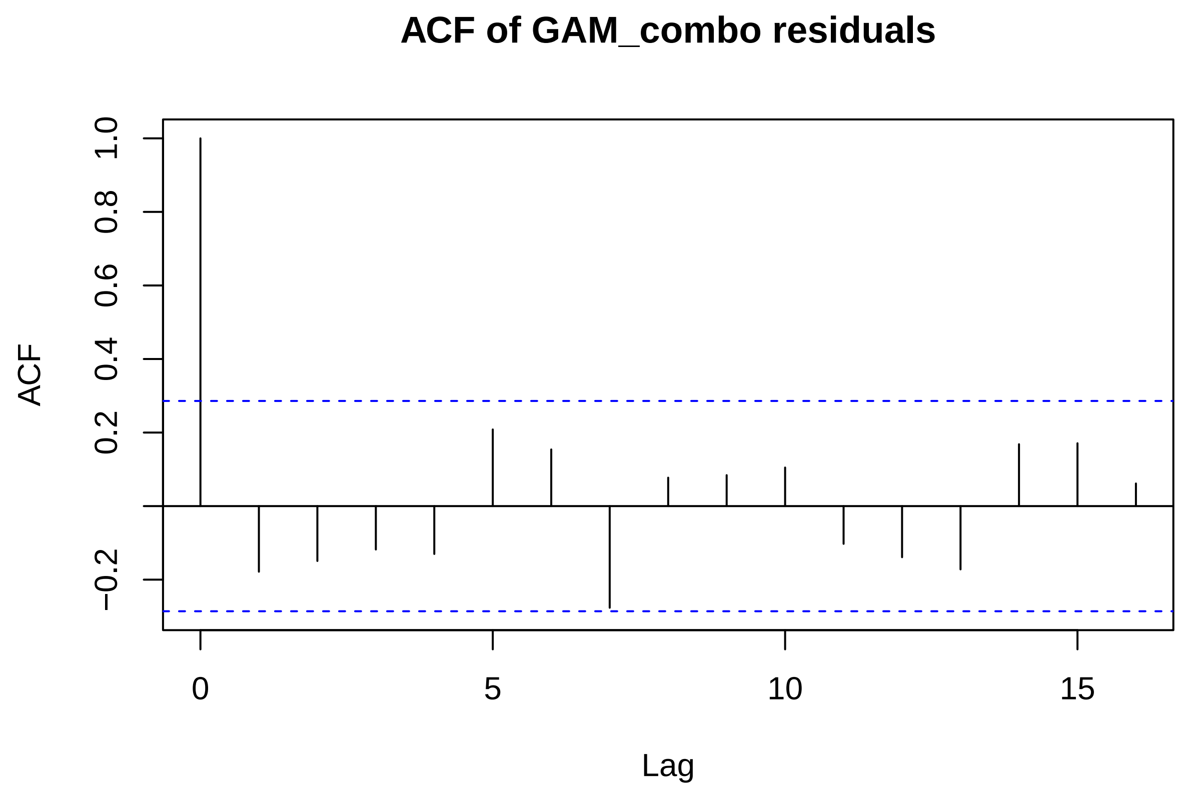

```r
# Visualize all smooths for interpretability
draw(gam_combo, select = c("s(Wind.speed)", "s(Wind.speed):Operational",
                           "s(Wind.dir)", "s(DateF)"))
```



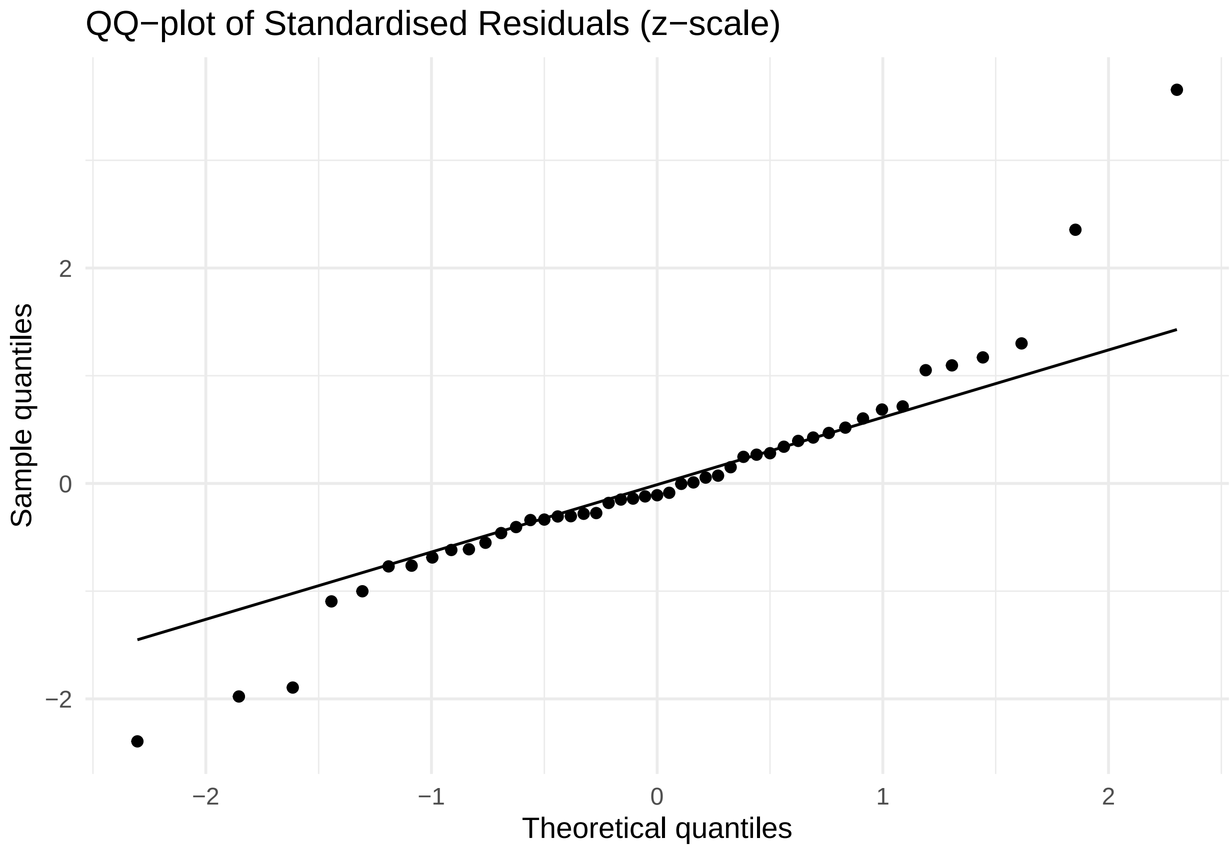

```r
### Residual Diagnostics

# Extract Pearson residuals for autocorrelation check
resid_combo <- resid(gam_combo, type = "pearson")

# Autocorrelation function (ACF) plots at 10-min lag intervals
acf(resid_combo, lag.max = 48, main = "ACF of gam_combo Residuals (10 min lags)")
```




```r
acf(resid(gam_combo), main = "ACF of GAM_combo residuals")
```




```r
# Standardized residuals (deviance scale) for QQ plot
z_resid <- scale(resid(gam_combo, type = "deviance"))[,1]
qq_df <- data.frame(sample = z_resid)

# QQ-plot to assess normality of residuals
ggplot(qq_df, aes(sample = sample)) +
  stat_qq() + stat_qq_line() +
  labs(title = "QQ-plot of Standardised Residuals (z-scale)",
       x = "Theoretical quantiles", y = "Sample quantiles") +
  theme_minimal()
```




```
### AIC Comparison Across Competing GAMs
# Compare Akaike Information Criterion for model selection
AIC(gam1, gam2, gam3, gam4, gam5, gam6, gam_int, gam_combo)
```

```
##                  df       AIC
## gam1       3.000000 -2336.015
## gam2       4.000000 -2340.988
## gam3       4.025906 -2340.953
## gam4       9.197043 -2363.962
## gam5      18.337927 -2385.194
## gam6      18.416458 -2385.036
## gam_int   19.221615 -2383.170
## gam_combo 17.349441 -2387.019
```

```
### Estimated Effect of Turbine Operation on Seismic Energy

# Extract the GAM coefficient for turbine operation (log-scale effect)
beta_op <- coef(gam_combo)["Operational"]                          # point estimate
se_op   <- sqrt(vcov(gam_combo)["Operational", "Operational"]) # standard error


# Compute 95% confidence interval on the log-scale
ci_log  <- beta_op + c(-1.96, 1.96) * se_op

# Convert to the multiplicative energy scale
```

```r
mult    <- exp(beta_op)      # multiplier: E_op / E_bg
ci_mul  <- exp(ci_log)       # 95% CI on multiplicative scale


# Report effect of turbine operation

cat(
  sprintf("Multiplier (turbines ON / OFF) = %.12f\n",  mult),
  sprintf("95%% confidence interval        = [%.12f, %.12f]\n",
          ci_mul[1], ci_mul[2])
)
```

```
## Multiplier (turbines ON / OFF) = 1.000000000017
##  95% confidence interval        = [1.000000000012, 1.000000000021]
```

```r
### Energy Uplift Calculation: Background vs Operational

# Compute mean integrated seismic energy in each turbine state
E_bg  <- mean(energy$E[energy$Operational == 0], na.rm = TRUE)   # mean energy during background (turbi
E_op  <- mean(energy$E[energy$Operational == 1], na.rm = TRUE)   # mean energy during operational (turb
E_all <- mean(energy$E,                          na.rm = TRUE)      # mean energy across all samples

# Calculate absolute and relative uplift due to turbine operation
deltaE <- E_op - E_bg               # absolute difference in mean energy
mult   <- E_op / E_bg               # multiplicative uplift factor (ratio)


# Display formatted results

cat(
  sprintf("Mean background energy (E_bg)      = %.3e  m²\n", E_bg),
  sprintf("Mean operational energy (E_op)     = %.3e  m²\n", E_op),
  sprintf("Overall mean energy  (E_all)       = %.3e  m²\n\n", E_all),
  sprintf("Absolute uplift  ΔE  (E_op - E_bg) = %.3e  m²\n", deltaE),
  sprintf("Multiplicative uplift (E_op / E_bg)= %.12f ×\n", mult)
)
```

```
## Mean background energy (E_bg)      = 2.937e-21  m²
##  Mean operational energy (E_op)     = 2.100e-20  m²
##  Overall mean energy  (E_all)       = 1.754e-20  m²
##
##  Absolute uplift  ΔE  (E_op - E_bg) = 1.807e-20  m²
##  Multiplicative uplift (E_op / E_bg)= 7.150170276535 ×
```

```r
### Estimate Turbine-ON Effect in Decibels

# Extract coefficient and standard error for 'Operational' term from GAM
beta_op <- coef(gam_combo)["Operational"]                              # log-scale coefficient
se_op   <- sqrt(vcov(gam_combo)["Operational", "Operational"])        # standard error

# Convert log-scale estimate to decibels (dB)
ln10     <- log(10)                                                   # natural log of 10 for conversion
```

```r
delta_dB <- 10 * beta_op / ln10                                    # estimated dB uplift
se_dB    <- 10 * se_op  / ln10                                     # standard error in dB
ci_dB    <- delta_dB + c(-1.96, 1.96) * se_dB                      # 95% confidence interval

# Nicely formatted output
cat(
  "Turbine-ON increment (0.5-8 Hz band)\n",
  sprintf("Δ = %.3e dB",  delta_dB), "\n",
  sprintf("95%% CI = [%.3e, %.3e] dB", ci_dB[1], ci_dB[2]), "\n"
)
```

```
## Turbine-ON increment (0.5-8 Hz band)
##  Δ = 7.167e-11 dB
##  95% CI = [5.428e-11, 8.906e-11] dB
```

```r
# 1. Convert Energy (E, in m²) to RMS Displacement (nm)
#    Formula: RMS = √E       [E = integrated PSD over 0.5-8 Hz]
#    Conversion: 1 metre = 1e9 nanometres

energy <- energy %>%
  mutate(
    RMS_m  = sqrt(E),        # RMS displacement in metres
    RMS_nm = RMS_m * 1e9     # Convert to nanometres
  )


# 2. Summarise RMS Displacement by Turbine Operational State
#    Statistics: mean, 95th percentile, and maximum

summary_nm <- energy %>%
  group_by(Type, Operational) %>%  # "Background" (0) and "Operational" (1)
  summarise(
    mean_nm = mean(RMS_nm, na.rm = TRUE),               # average RMS
    p95_nm  = quantile(RMS_nm, 0.95, na.rm = TRUE),    # 95th percentile
    max_nm  = max(RMS_nm, na.rm = TRUE),                # maximum value
    .groups = "drop"
  )

print(summary_nm)
```

```
## # A tibble: 2 x 5
##   Type        Operational mean_nm p95_nm max_nm
##   <chr>             <dbl>   <dbl>  <dbl>  <dbl>
## 1 Background            0  0.0539 0.0598 0.0603
## 2 Operational           1  0.144  0.169  0.177
```

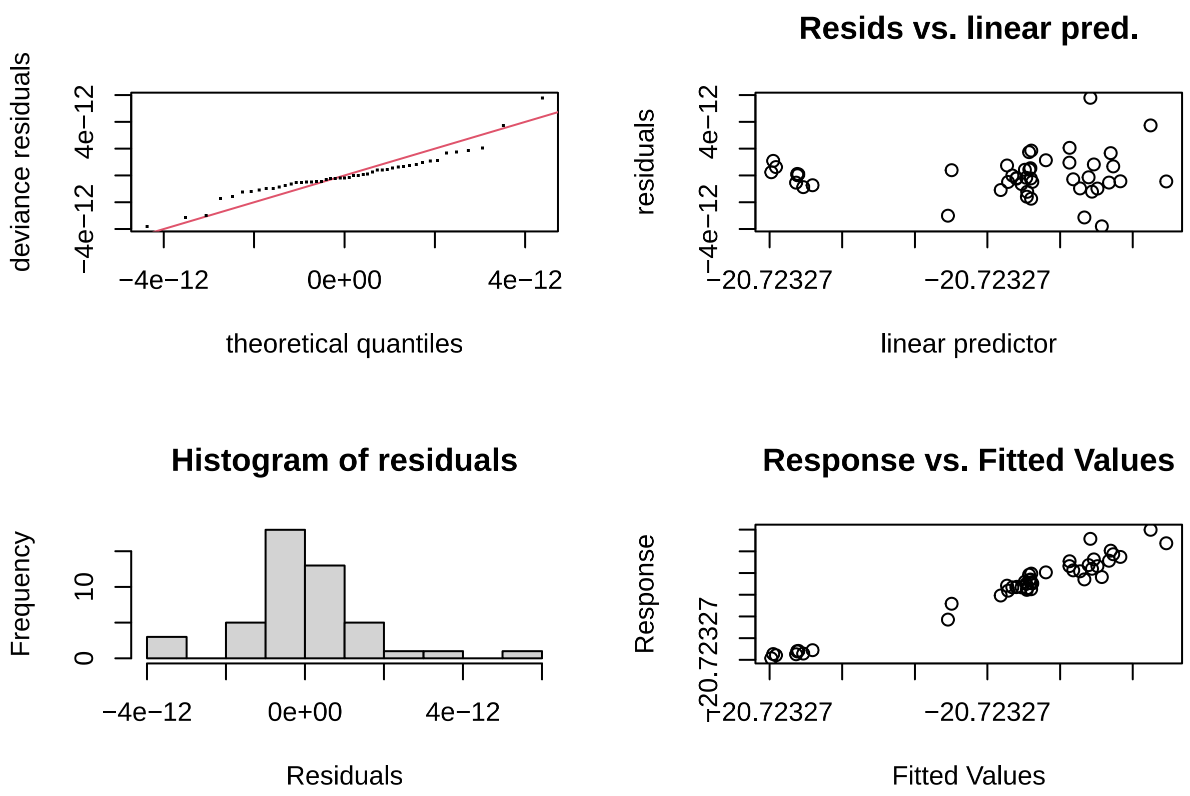
```r
# 3. Compare Results to the Eskdalemuir Defensible Threshold
#    Limit: 0.336 nm (maximum RMS displacement allowed)

limit_nm <- 0.336  # specified defensible RMS threshold

# Calculate percentage of the limit
```

```r
summary_nm %>%
  mutate(
    mean_pct = mean_nm / limit_nm * 100,  # % of threshold (mean)
    p95_pct  = p95_nm  / limit_nm * 100,  # % of threshold (95th percentile)
    max_pct  = max_nm  / limit_nm * 100   # % of threshold (max)
  ) %>%
  select(Type, starts_with("mean_"), starts_with("p95_"), starts_with("max_")) %>%
  knitr::kable(digits = 2,
    col.names = c("State", "Mean RMS (nm)", "% of Limit",
                  "95th-%ile RMS (nm)", "% of Limit",
                  "Max RMS (nm)", "% of Limit"))
```

State	Mean RMS (nm)	% of Limit	95th-%ile RMS (nm)	% of Limit	Max RMS (nm)	% of Limit
Background	0.05	16.05	0.06	17.79	0.06	17.96
Operational	0.14	42.93	0.17	50.18	0.18	52.72

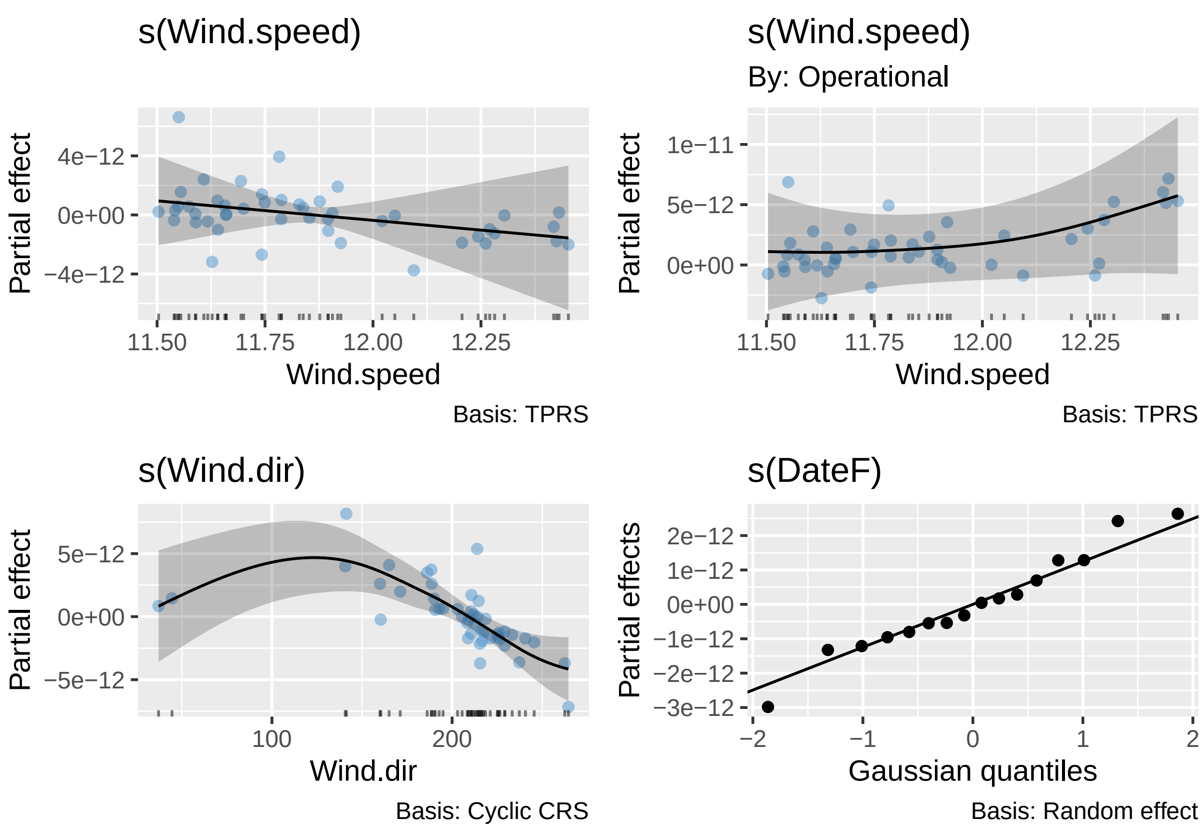
MODELLING & RESULTS PLOTS

```r
### Plot 1 - GAM Diagnostic Checks: Basis Dimension & Residual Inspection

# 1. Check adequacy of basis dimension (k-index):
#    - A k-index < 1.2 indicates that the chosen basis dimension is sufficient.
#    - A value > 1.2 may suggest underfitting or the need for higher k in s() terms.
gam.check(gam_combo)
```

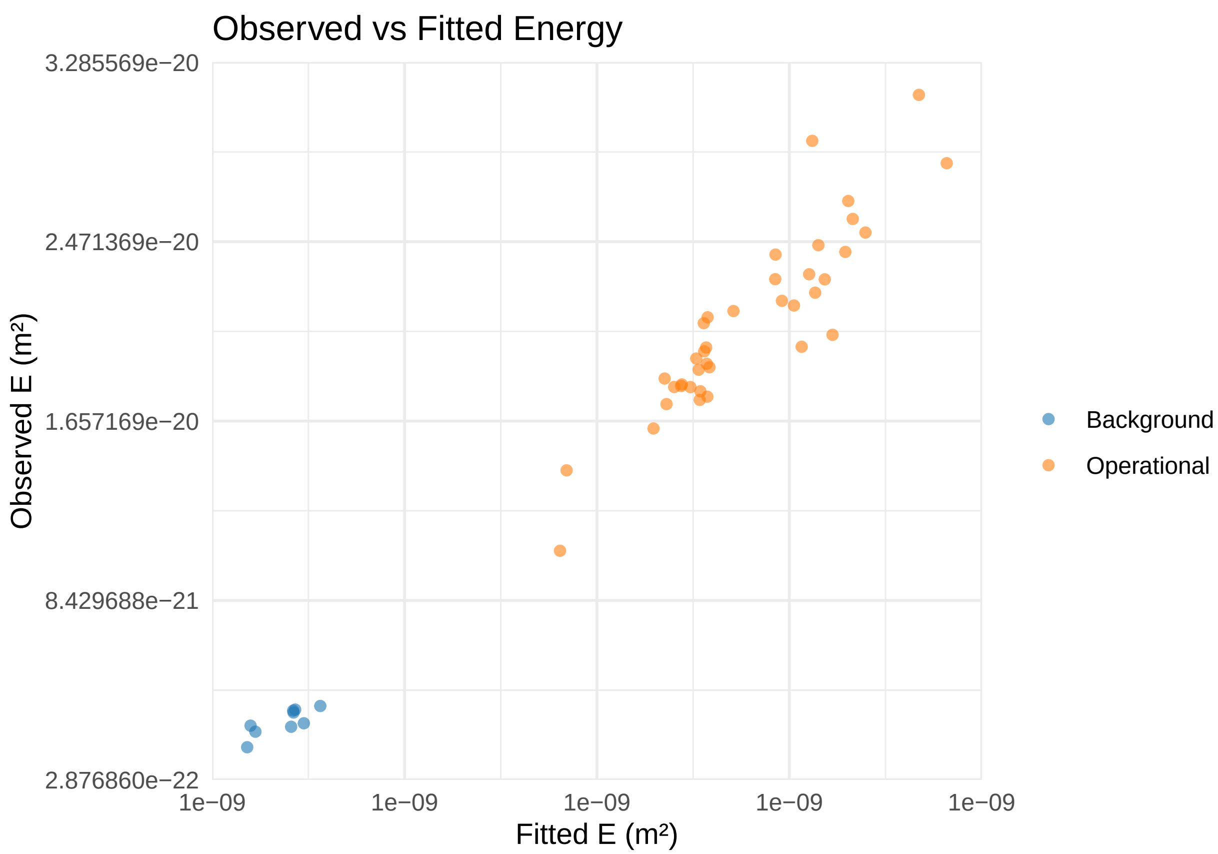



```
##
## Method: REML   Optimizer: outer newton
## full convergence after 8 iterations.
## Gradient range [-0.00166361,0.003669081]
## (score -1056.797 & scale 3.599446e-24).
## Hessian positive definite, eigenvalue range [0.001239878,21.78138].
## Model rank =  46 / 47
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##                             k'   edf k-index p-value
## s(Wind.speed)             9.00  1.00    1.03    0.48
## s(Wind.speed):Operational 10.00  2.11    1.03    0.46
## s(Wind.dir)               10.00  2.43    0.96    0.39
## s(DateF)                  16.00  7.32      NA      NA
```

```r
# 2. Visual diagnostic: Residuals plotted over each smooth term
#    - Uses 'gratia' package to draw smooth terms with residual clouds.
#    - Helps visually assess model fit and detect systematic bias.
#    - Grey points represent residuals; patterns could indicate poor fit.
draw(gam_combo, residuals = TRUE)
```



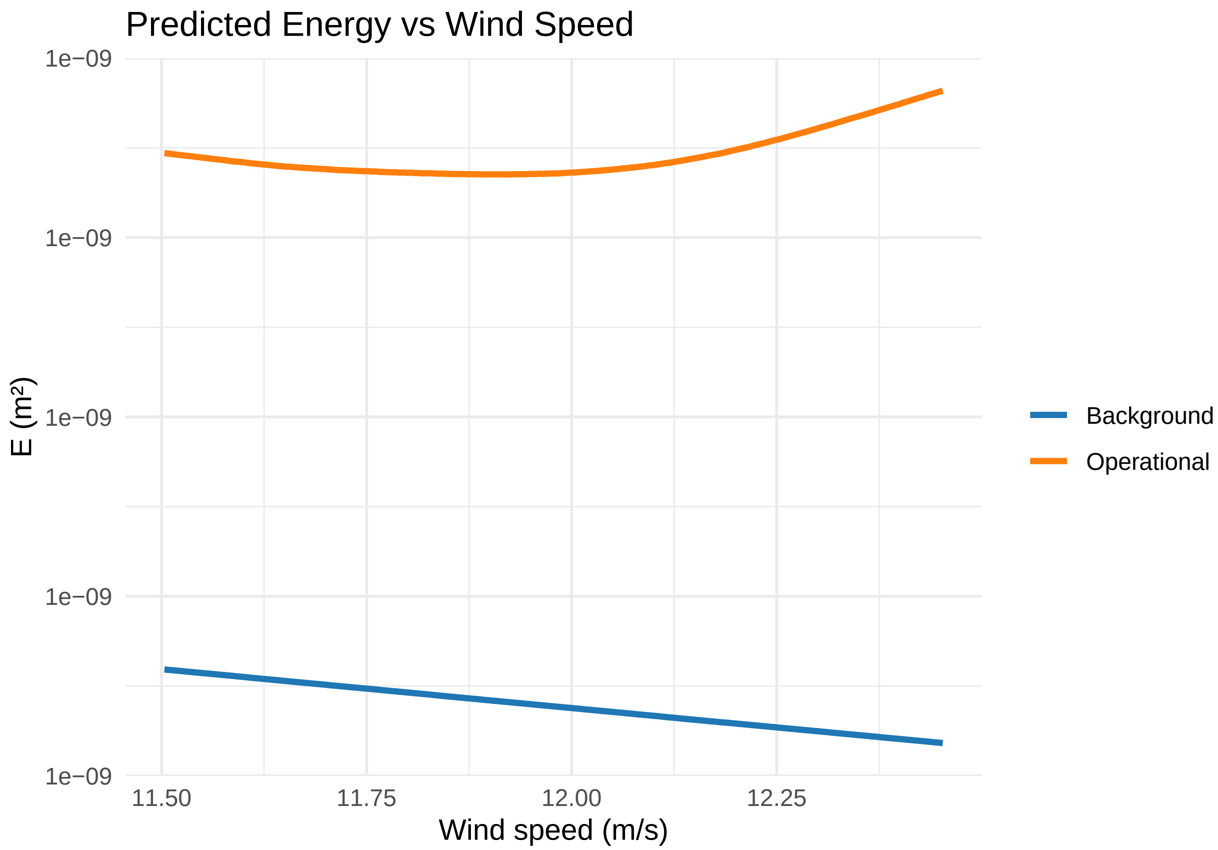

```
### Plot 2 - Observed vs Fitted Energy Plot

cols <- c("Background" = "#1f77b4", "Operational" = "#ff7f0e")   # Define colour palette for turbine st
limit_nm <- 0.336                                                 # Compliance RMS displacement threshol

obs_fit <- energy %>%
  mutate(Fitted = exp(fitted(gam_combo)))      # Back-transform log-scale model predictions to original

ggplot(obs_fit, aes(Fitted, E, colour = Type)) +  # Scatterplot: Fitted vs Observed Energy, coloured by
  geom_point(alpha = .6) +                        # Semi-transparent points for visual clarity
  geom_abline(slope = 1, intercept = 0, linetype = "dashed") +  # Identity line to indicate perfect pre
  scale_colour_manual(values = cols, name = "") +              # Apply custom colours with no legend t
  labs(title = "Observed vs Fitted Energy",                    # Axis titles and plot title
       x = "Fitted E (m²)", y = "Observed E (m²)") +
  theme_minimal()                                              # Clean, minimal visual style
```



Predictor Effect Visualizations

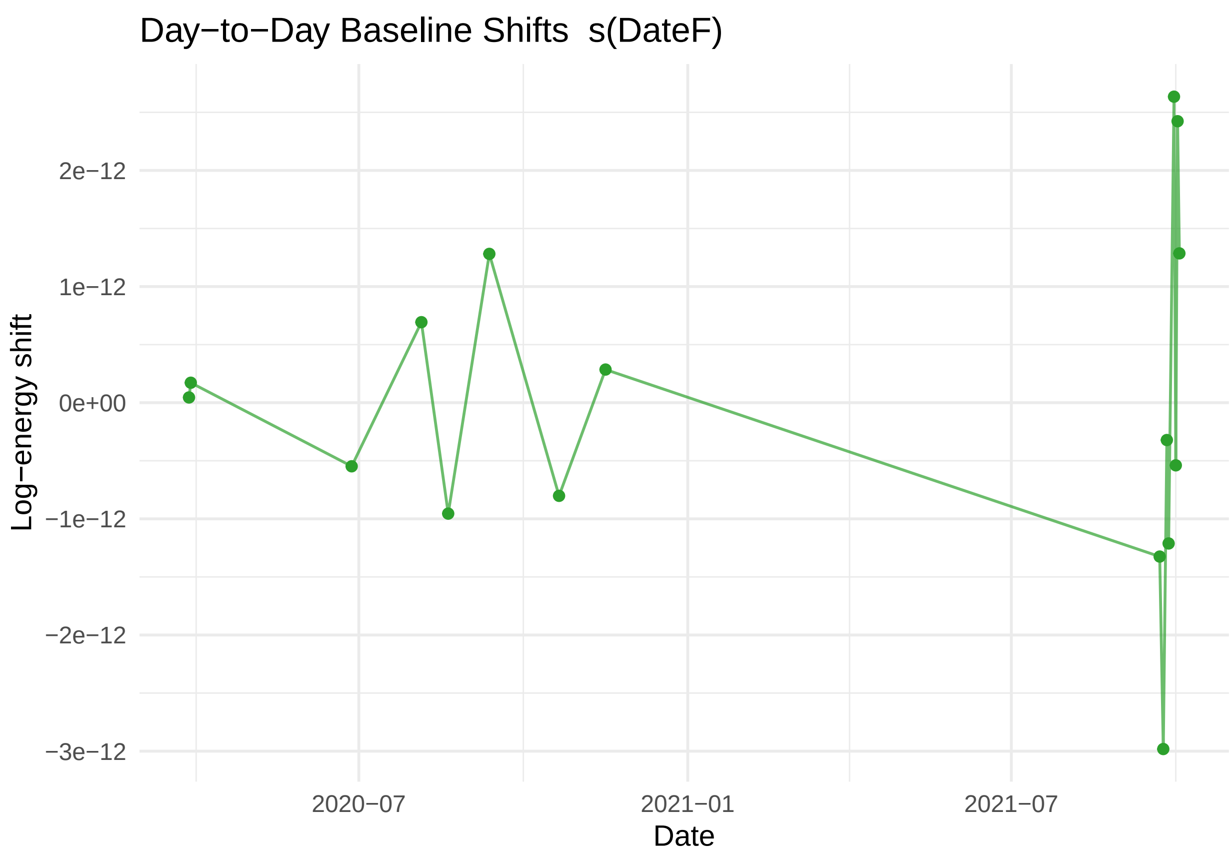
```
### Plot 3 - Predicted Energy vs Wind Speed (ON vs OFF)

# Grid of speeds in observed range
grid_ws <- data.frame(
  Wind.speed  = seq(min(energy$Wind.speed), max(energy$Wind.speed), length = 200),  # sequence of wind
  Wind.dir    = mean(energy$Wind.dir),           # hold wind direction at mean
  DateF       = energy$DateF[1]                  # fixed date factor for prediction
)

# Add Operational status = 0 (Background) and = 1 (Operational) to grid
grid_ws_bg <- cbind(grid_ws, Operational = 0)
grid_ws_op <- cbind(grid_ws, Operational = 1)

# Predict energy for both states and combine results
p_ws <- bind_rows(
  grid_ws_bg  |>
    mutate(Type = "Background",
           fit  = exp(predict(gam_combo, newdata = grid_ws_bg))),  # back-transform log prediction
  grid_ws_op  |>
    mutate(Type = "Operational",
           fit  = exp(predict(gam_combo, newdata = grid_ws_op)))   # back-transform log prediction
) |>
  ggplot(aes(Wind.speed, fit, colour = Type)) +  # plot fitted energy vs wind speed
    geom_line(size = 1.1) +                      # add smooth lines
```

```
    scale_colour_manual(values = cols, name = "") +  # custom colour mapping
    labs(title = "Predicted Energy vs Wind Speed",
         x = "Wind speed (m/s)", y = "E (m²)") +
    theme_minimal()                                  # clean theme

# Display the plot
print(p_ws)
```



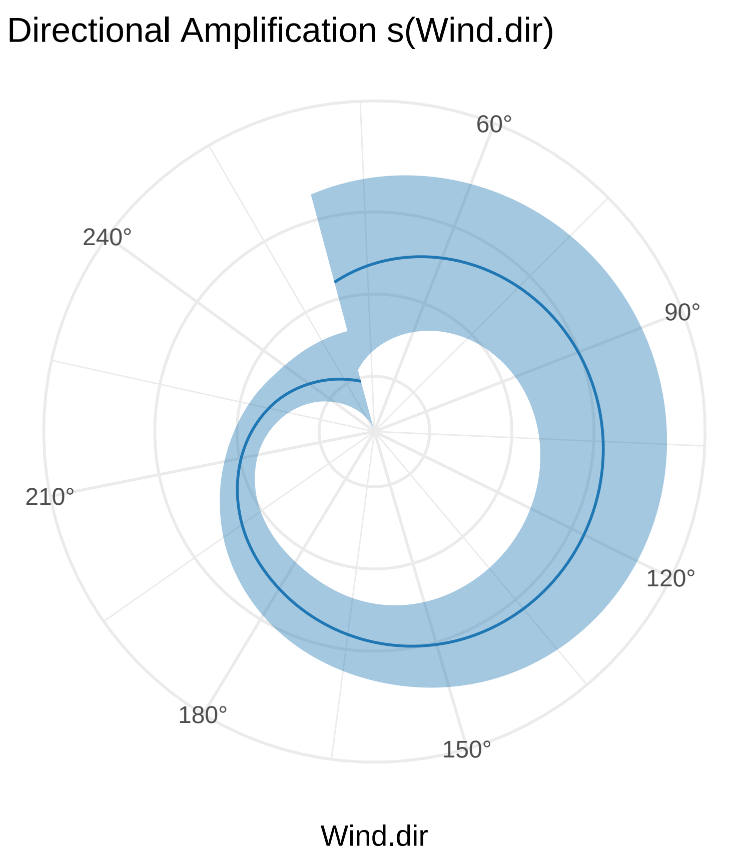

```
# Plot 4 - Daily Random Intercepts - Smooth Term s(DateF)

# 1. Extract daily intercepts from s(DateF)
#    - Each date gets its own partial effect (random smooth)
#    - These reflect baseline energy shifts per day
re_df <- tibble(
  Date      = as.Date(levels(energy$DateF)),    # Convert factor to Date
  Intercept = coef(gam_combo)[grep("^s\\(DateF\\)", names(coef(gam_combo)))]   # Extract coefficients fo
)

# 2. Plot: Daily shifts in log-energy baseline
#    - Useful for visualizing temporal heterogeneity
p_day <- ggplot(re_df, aes(Date, Intercept)) +
  geom_line(color = "#2ca02c", alpha = .7) +     # green line
  geom_point(color = "#2ca02c") +                # green dots
  labs(
    title = "Day-to-Day Baseline Shifts  s(DateF)",
```

```
    y      = "Log-energy shift"
  ) +
  theme_minimal()

# 3. Display plot
print(p_day)
```



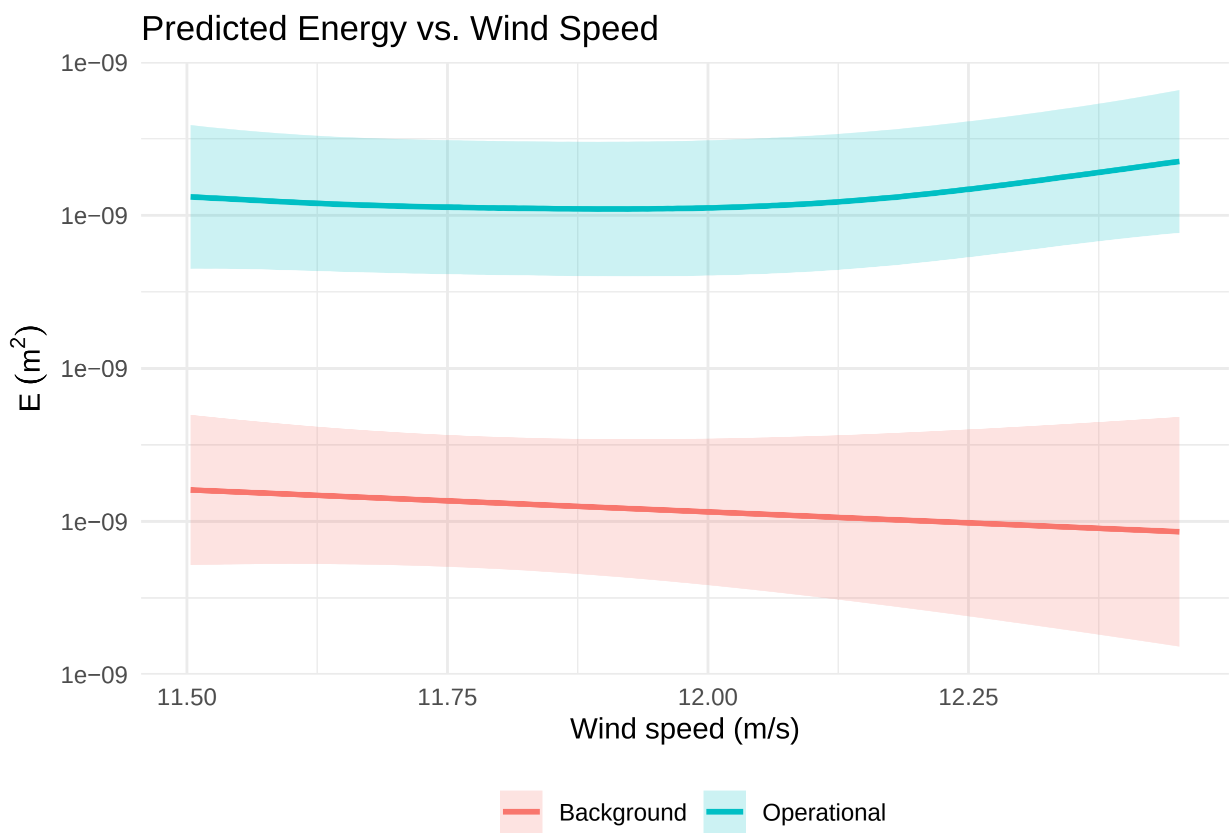

```
### Plot 5 - Directional Amplification via s(Wind.dir) Term

# 1. Extract smooth estimates for s(Wind.dir)
#    - The term shows how energy is amplified as a function of wind direction
#    - Convert log-scale effect to multiplicative factor on energy
dir_eff <- smooth_estimates(gam_combo, select = "s(Wind.dir)", n = 200) %>%
  transmute(
    Wind.dir = Wind.dir,
    mult     = exp(.estimate),                # effect multiplier
    lo       = exp(.estimate - 1.96 * .se),   # 95% CI lower bound
    hi       = exp(.estimate + 1.96 * .se)    # 95% CI upper bound
  )

# 2. Plot: Polar (rose) chart of directional effect
#    - Ribbon shows confidence interval
#    - Line represents estimated multiplier
#    - Helps visualize directionality of seismic energy amplification
p_rose <- ggplot(dir_eff, aes(Wind.dir, mult)) +
```

```r
  geom_ribbon(aes(ymin = lo, ymax = hi), fill = "#1f77b4", alpha = .4) +
  geom_line(color = "#1f77b4") +
  coord_polar(start = -pi/12) +  # rotate start angle
  scale_x_continuous(
    breaks = seq(0, 330, 30),
    labels = paste0(seq(0, 330, 30), "°")
  ) +
  labs(title = "Directional Amplification s(Wind.dir)",
       y = "Multiplier on Energy") +
  theme_minimal() +
  theme(
    axis.title.y = element_blank(),
    axis.text.y  = element_blank()
  )

# 3. Display plot
print(p_rose)
```



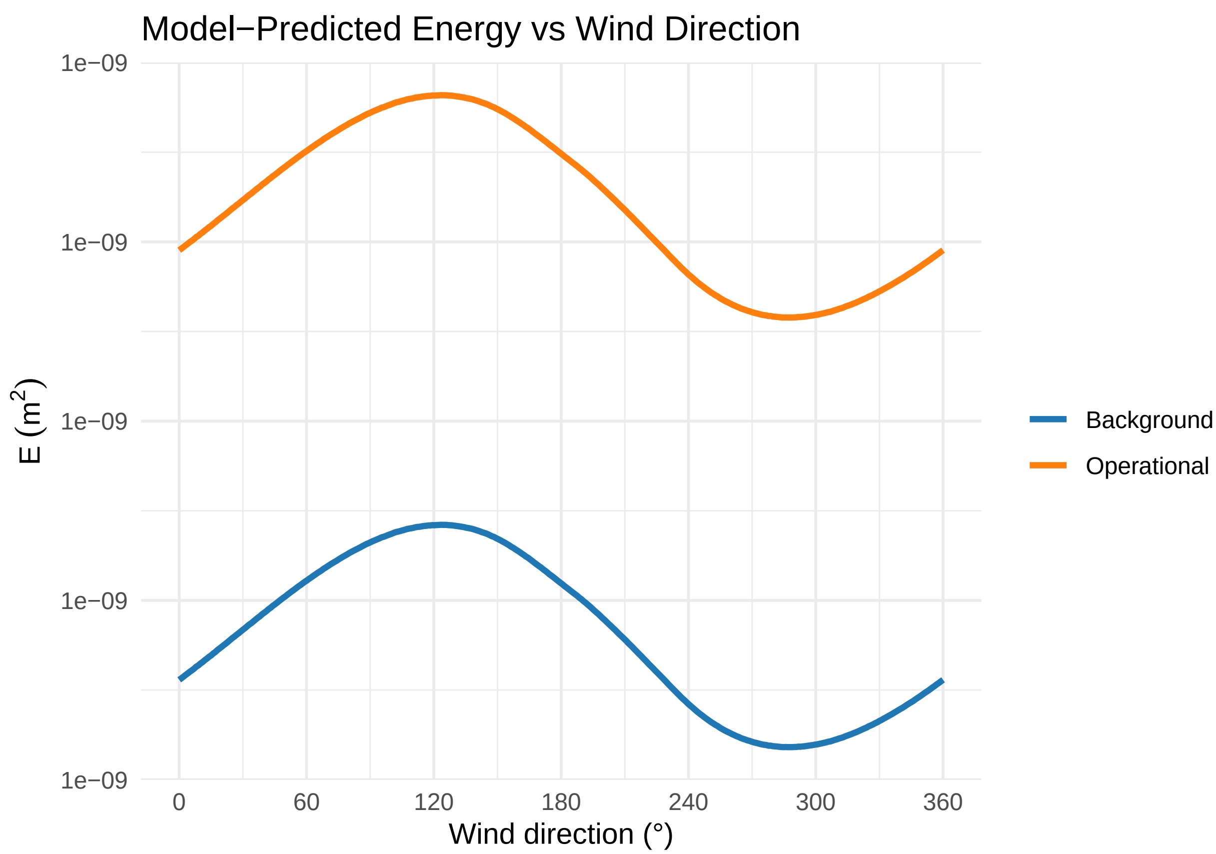

```r
### Plot 6 - Wind Speed Interaction Curves - GAM-predicted Energy (E)

# 1. Define grid of wind speeds
ws_seq <- seq(min(energy$Wind.speed),
              max(energy$Wind.speed),
              length.out = 200)
```

```r
# 2. Create prediction grid
#    - Vary wind speed across observed range
#    - Fix wind direction and date
#    - Include both turbine states: Operational and Background
grid_ws <- expand_grid(
  Wind.speed  = ws_seq,
  ws_s        = (ws_seq - mean(energy$Wind.speed)) / sd(energy$Wind.speed),  # scaled wind speed
  Wind.dir    = mean(energy$Wind.dir),   # fixed average direction
  DateF       = first(energy$DateF),     # fixed date (could use median too)
  Operational = c(0, 1)                  # both states
)

# 3. Predict from GAM model (log-scale)
prd_ws <- predict(gam_combo, newdata = grid_ws, se.fit = TRUE)

# 4. Back-transform predictions and structure output
plot_ws <- grid_ws %>%
  mutate(
    fit   = exp(prd_ws$fit),                              # mean predicted energy
    lo    = exp(prd_ws$fit - 1.96 * prd_ws$se.fit),       # lower CI
    hi    = exp(prd_ws$fit + 1.96 * prd_ws$se.fit),       # upper CI
    State = factor(Operational, labels = c("Background", "Operational"))
  )

# 5. Plot: Energy vs Wind Speed by turbine state
p_ws <- ggplot(plot_ws, aes(Wind.speed, fit, colour = State, fill = State)) +
  geom_ribbon(aes(ymin = lo, ymax = hi), alpha = 0.20, colour = NA) +
  geom_line(size = 1) +
  labs(
    title = "Predicted Energy vs. Wind Speed",
    x     = "Wind speed (m/s)",
    y     = expression(E~(m^2)),
    colour = "", fill = ""
  ) +
  theme_minimal() +
  theme(legend.position = "bottom")

print(p_ws)
```




```r
### Plot 7 - Model-Predicted Energy vs Wind Direction

# Create prediction grid for wind direction

# Define a grid of wind direction values (0° to 360°)
wd_grid <- data.frame(
  Wind.dir   = seq(0, 360, length = 360),          # 1° resolution
  Wind.speed = mean(energy$Wind.speed),            # hold wind speed constant
  DateF      = energy$DateF[1]                     # representative date
)

# Add turbine operational states to grid
grid_bg <- cbind(wd_grid, Operational = 0)         # Background scenario
grid_op <- cbind(wd_grid, Operational = 1)         # Operational scenario

# Predict model output across wind directions

# Combine predictions under both states
pred_dir <- bind_rows(
  grid_bg |> mutate(Type = "Background",
                    fit  = exp(predict(gam_combo, newdata = grid_bg))),   # back-transform log(E)
  grid_op |> mutate(Type = "Operational",
                    fit  = exp(predict(gam_combo, newdata = grid_op)))
)

# Plot: Model-Predicted Energy vs Wind Direction
```

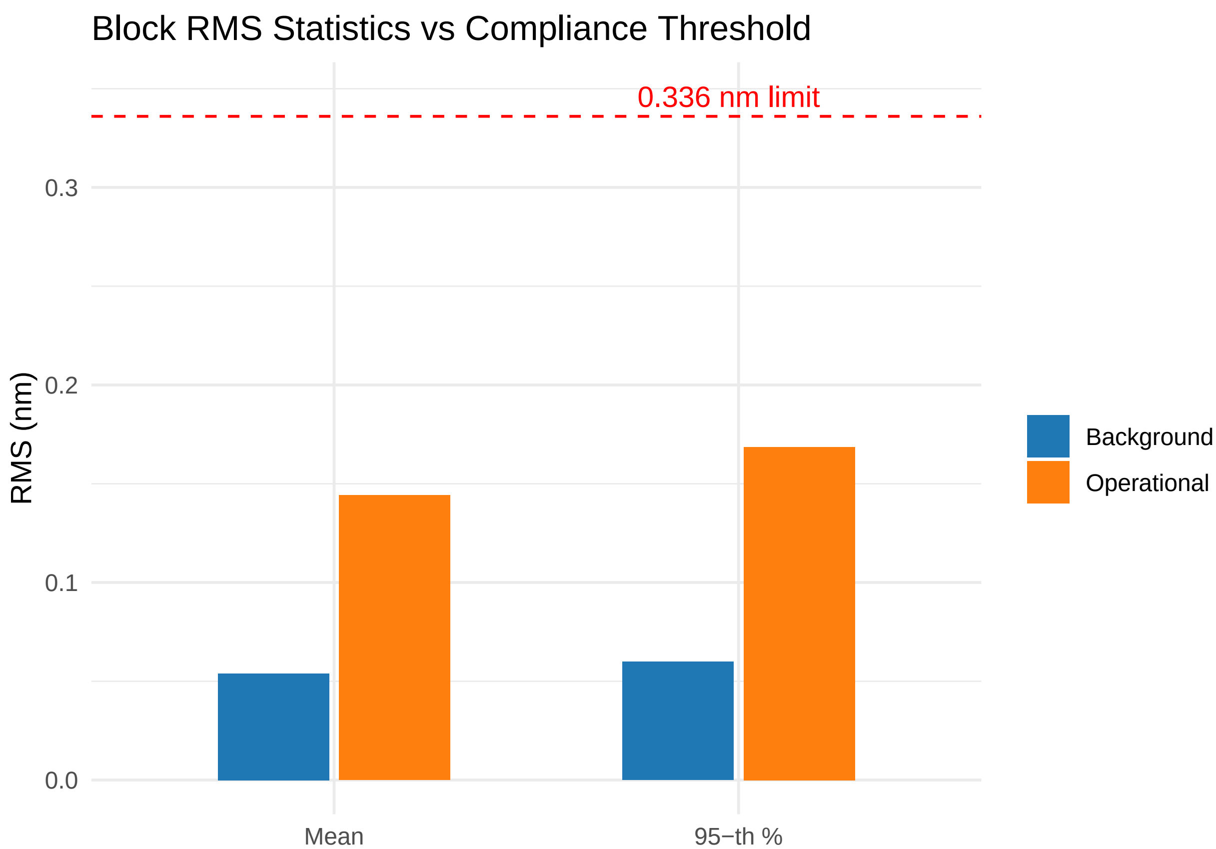
```r
p_wd <- ggplot(pred_dir, aes(Wind.dir, fit, colour = Type)) +
  geom_line(size = 1.1) +
  scale_colour_manual(values = cols, name = "") +
  scale_x_continuous(breaks = seq(0, 360, by = 60)) +
  labs(title = "Model-Predicted Energy vs Wind Direction",
       x = "Wind direction (°)",
       y = expression(E~(m^2))) +
  theme_minimal()

# Print the plot
print(p_wd)
```



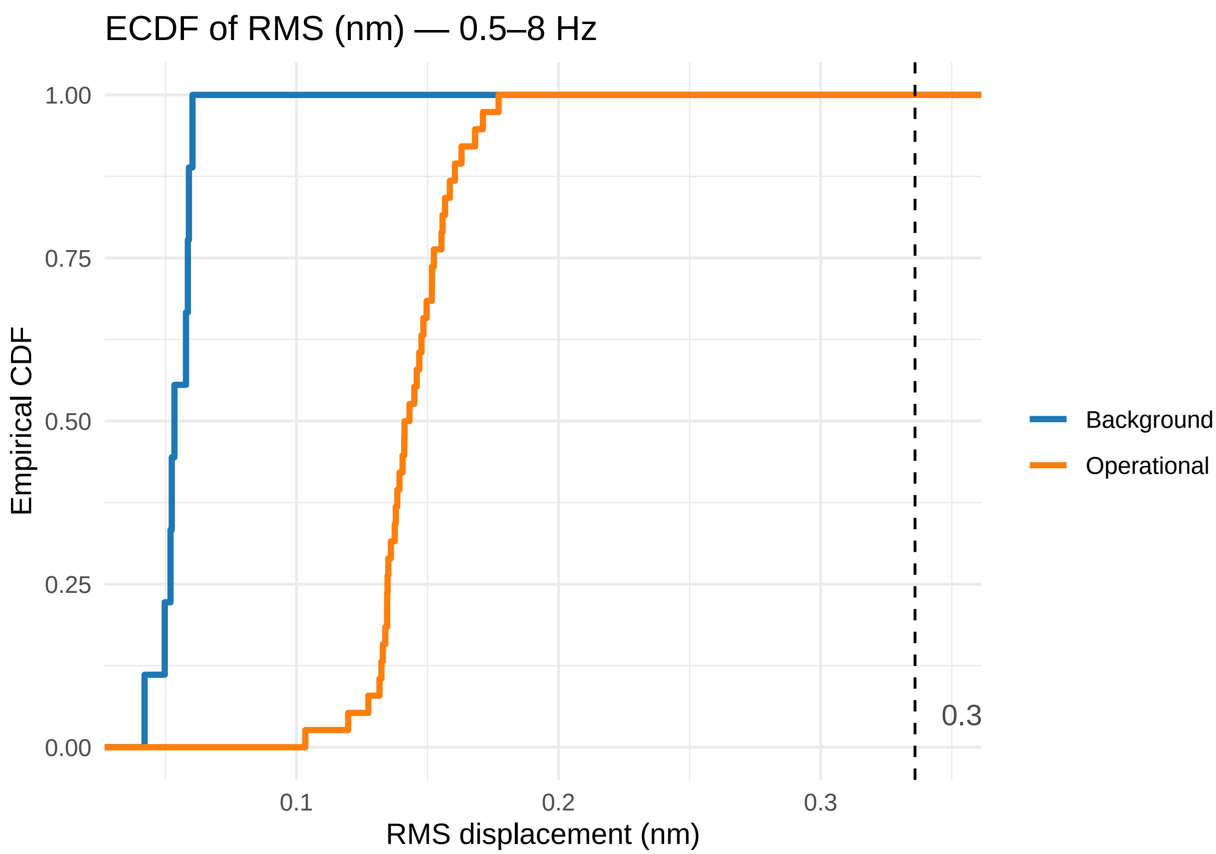
SUMMARY AND COMPLIANCE

```r
### Plot 8 - Block RMS Statistics vs Compliance Threshold
# Summary of mean and 95th percentile RMS displacement by turbine state#

# 1. Summarize RMS statistics (mean and 95th percentile) by turbine state
summary_nm <- energy %>%
  group_by(State = factor(Operational,
                          labels = c("Background", "Operational"))) %>%
  summarise(
    mean_nm = mean(RMS_nm),
    p95_nm  = quantile(RMS_nm, 0.95),
    .groups = "drop"
```

```r
  ) %>%
  pivot_longer(cols = c(mean_nm, p95_nm),
               names_to = "Metric", values_to = "value")

# 2. Create bar plot comparing statistics against 0.336 nm threshold
p_bar <- ggplot(summary_nm,
                aes(x = Metric, y = value, fill = State)) +

  # Bar plot with dodged positioning for comparability
  geom_col(position = position_dodge(width = 0.6), width = 0.55) +

  # Add compliance limit line (e.g., Eskdalemuir threshold)
  geom_hline(yintercept = limit_nm, linetype = 2, colour = "red") +

  # Annotate threshold label
  annotate("text", x = 1.75, y = limit_nm * 1.03,
           label = "0.336 nm limit", colour = "red", hjust = 0) +

  # Clean up axis labels and colors
  scale_x_discrete(labels = c("Mean", "95-th %")) +
  scale_fill_manual(values = c("Background" = "#1f77b4",
                               "Operational" = "#ff7f0e"),
                    name = "") +

  # Titles and theme
  labs(title = "Block RMS Statistics vs Compliance Threshold",
       x = NULL, y = "RMS (nm)") +
  theme_minimal()

# 3. Print the plot
print(p_bar)
```




```r
### Plot 9 - ECDF of RMS Displacement (nm)

# Empirical Cumulative Distribution Function (ECDF) of RMS Displacement
energy |> ggplot(aes(RMS_nm, colour = Type)) +

  # Plot ECDF curves for each turbine state
  stat_ecdf(size = 1.1) +

  # Add compliance threshold line at 0.336 nm
  geom_vline(xintercept = limit_nm, linetype = "dashed") +

  # Manually set line colours for each turbine state
  scale_colour_manual(values = cols, name = "") +

  # Add title and axis labels
  labs(title = "ECDF of RMS (nm) - 0.5-8 Hz",
       x = "RMS displacement (nm)", y = "Empirical CDF") +

  # Annotate the compliance limit line with a label
  annotate("text", x = limit_nm*1.03, y = .05,
           label = "0.336 nm limit", hjust = 0, colour = "grey30") +

  # Apply minimal theme
  theme_minimal()
```



ECDF of RMS (nm) — 0.5–8 Hz
1.00
0.75
0.50
0.25
0.00
Empirical CDF
0.3
0.1
0.2
0.3
RMS displacement (nm)
Background
Operational