Spring 2024 Forecasting Class Competition

Master Forecasters

2024-04-14

Contents

Name	Major Contribution
Cory Petersen	Model
Laila Saleh	Data Transforms and Visualizations
Daniel Moore	.Rmd organization

Executive Summary

This was our task: This is what we accomplished: This is why it's important:

1 Data Exploration

Our task is to predict the Plane of Array (POA) Irradiance $\left(\frac{W}{m^2}\right)$ for measurements made at the Rutgers University Energy Lab at Richard Weeks Hall in 10-minute increments for the next 12-hours. The POA has been measured by a pyranometer with the same orientation as the solar array. This measurement is critical for modeling the performance of a Photo-Voltaic (PV) system. Predicting future POA enables operators to plan for optimize Distributed Energy Resources (DER). The data is provided in 10-minute increments from June 1, 2023 to August 2, 2023 with the following measurements:

- DATE_TIME: Date/time information
- AIRTEMP: Air temperature (C)
- RH_AVG: Humidity (%)
- DEWPT: Dew point temperature (C)
- WS: Wind speed $\left(\frac{m}{s}\right)$
- GHI: Global Horizontal Irradiance $\left(\frac{W}{m^2}\right)$ measured from a horizontal pyranometer mounted on a sun tracker
- DNI: Direct Normal Irradiance $\left(\frac{W}{m^2}\right)$ measured from a horizontal pyranometer mounted on a sun tracker
- DIFF: Diffuse Irradiance $\left(\frac{W}{m^2}\right)$ measured from a horizontal pyranometer mounted on a sun tracker
- POA: Plane-of-Array Irradiance $\left(\frac{W}{m^2}\right)$ measured from a pyranometer that has the exact same tilting

1.1 Loading the Data

Load the data and concatenate, eliminating duplicates Processing DATE_TIME into a datetime object Make the data a tsibble object

DATE_TIME	AIRTEMP	RH_AVG	DEWPT	WS	GHI	DNI	DIFF	POA
2023-07-05 17:20:00	32.65	50.36	21.01	1.988	104.8	0	104.8	111.40
2023-07-05 17:30:00	31.48	61.72	23.28	2.144	96.0	0	96.0	103.90
2023-07-05 17:40:00	31.00	63.82	23.36	1.870	69.9	0	69.9	81.80
2023-07-05 17:50:00	30.70	64.63	23.29	1.978	63.5	0	63.5	71.17
2023-07-05 18:00:00	30.55	64.88	23.20	1.728	69.9	0	69.9	68.33

1.2 Loading External Data

Prediciting the POA is tantamont to predicting how sunny it is.

We have obtained a dataset which contains historical data for the period covered, in hourly data

We were not able to obtain historical day-ahead weather forecasts so we are using the historical data as a forecast

This is valid because hourly forecasts for the next 24 hours tend to be very accurate

Now we need to combine the data because the provided measurements are in 10-minute increments and the weather data is in hourly increments

We create a temporary column in the provided data which is just the datetime rounded down to the nearest hour

This gives a key that we can conduct a left join on

We can then remove the temporary column

It would be preferable to do linear interpolation on the weather data.

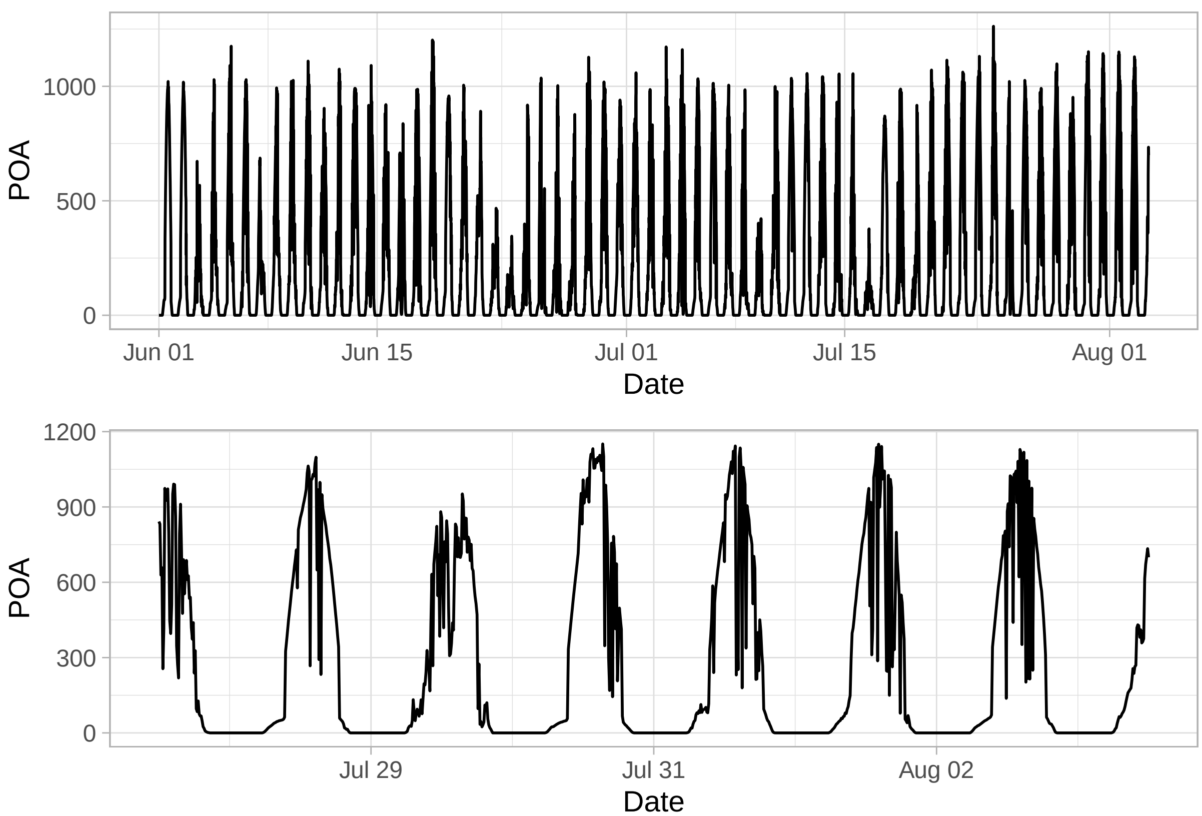
```
data <- data %>%
  mutate(datetime_rounded = floor_date(DATE_TIME, "hour"))

data <- left_join(
  data, rename(weather, DATE_TIME = datetime),
  by = c("datetime_rounded" = "DATE_TIME")
)

data <- select(data, -datetime_rounded)
```

1.3 Data Visualizations

1.3.1 POA Time Series

First, we need to just examine the target variable

```
POQ_vs_TIME <- data %>% autoplot(POA) +
  xlab("Date")

POA_vs_LAST_7D <- data %>%
  filter(DATE_TIME >= max(DATE_TIME) - as.difftime(7, units = "days")) %>%
  autoplot(POA) +
  xlab("Date")

(POQ_vs_TIME + theme_light()) / (POA_vs_LAST_7D + theme_light())
```



1.3.2 POA vs Time and Cloud Cover

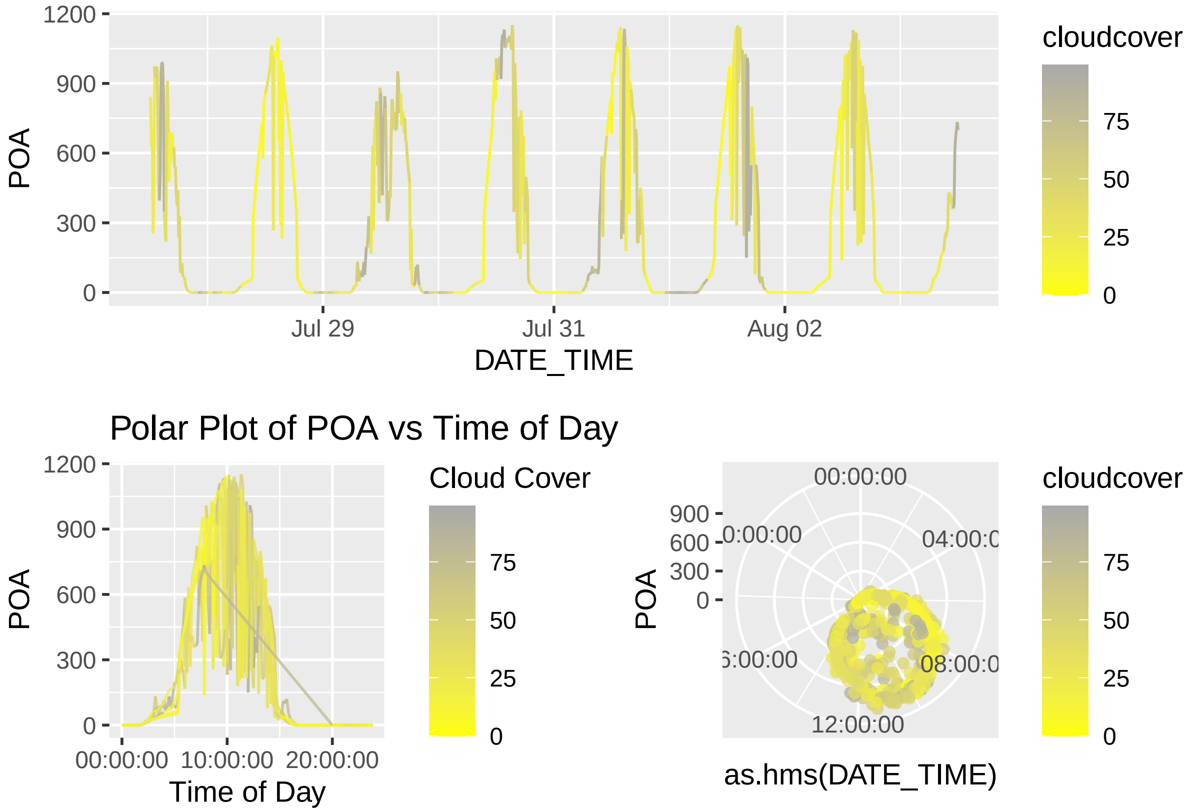
```r
POA_vs_LAST_7D_CC <- data %>%
  filter(DATE_TIME >= max(DATE_TIME) - as.difftime(7, units = "days")) %>%
  ggplot(aes(x = DATE_TIME, y = POA, color = cloudcover)) +
  geom_line() +
  scale_colour_gradient(low = "yellow", high = "darkgrey")

polar_cc <- data %>%
  filter(DATE_TIME >= max(DATE_TIME) - as.difftime(7, units = "days")) %>%
  ggplot(
    aes(
        x = as.hms(DATE_TIME),
        y = POA,
        group = yday(DATE_TIME),
        color = cloudcover)
  ) +
  geom_point(alpha = 0.75) +  # Scatter plot with 75% transparency
  scale_colour_gradient(low = "yellow", high = "darkgrey") +
  coord_polar()  # Converts the plot to polar coordinates
labs(
  title = "Polar Plot of POA vs Time of Day",
  x = "Time of Day",
  y = "POA",
  colour = "Cloud Cover"
)
```

```
## $x
## [1] "Time of Day"
##
## $y
## [1] "POA"
##
## $colour
## [1] "Cloud Cover"
##
## $title
## [1] "Polar Plot of POA vs Time of Day"
##
## attr(,"class")
## [1] "labels"
```

```r
line_cc <- data %>%
  filter(DATE_TIME >= max(DATE_TIME) - as.difftime(7, units = "days")) %>%
  ggplot(
    aes(
        x = as.hms(DATE_TIME),
        y = POA,
        group = yday(DATE_TIME),
        color = cloudcover)
  ) +
  geom_line(alpha = 0.75) +  # Scatter plot with 75% transparency
  scale_colour_gradient(low = "yellow", high = "darkgrey") +
  labs(title = "Polar Plot of POA vs Time of Day",
      x = "Time of Day",
```

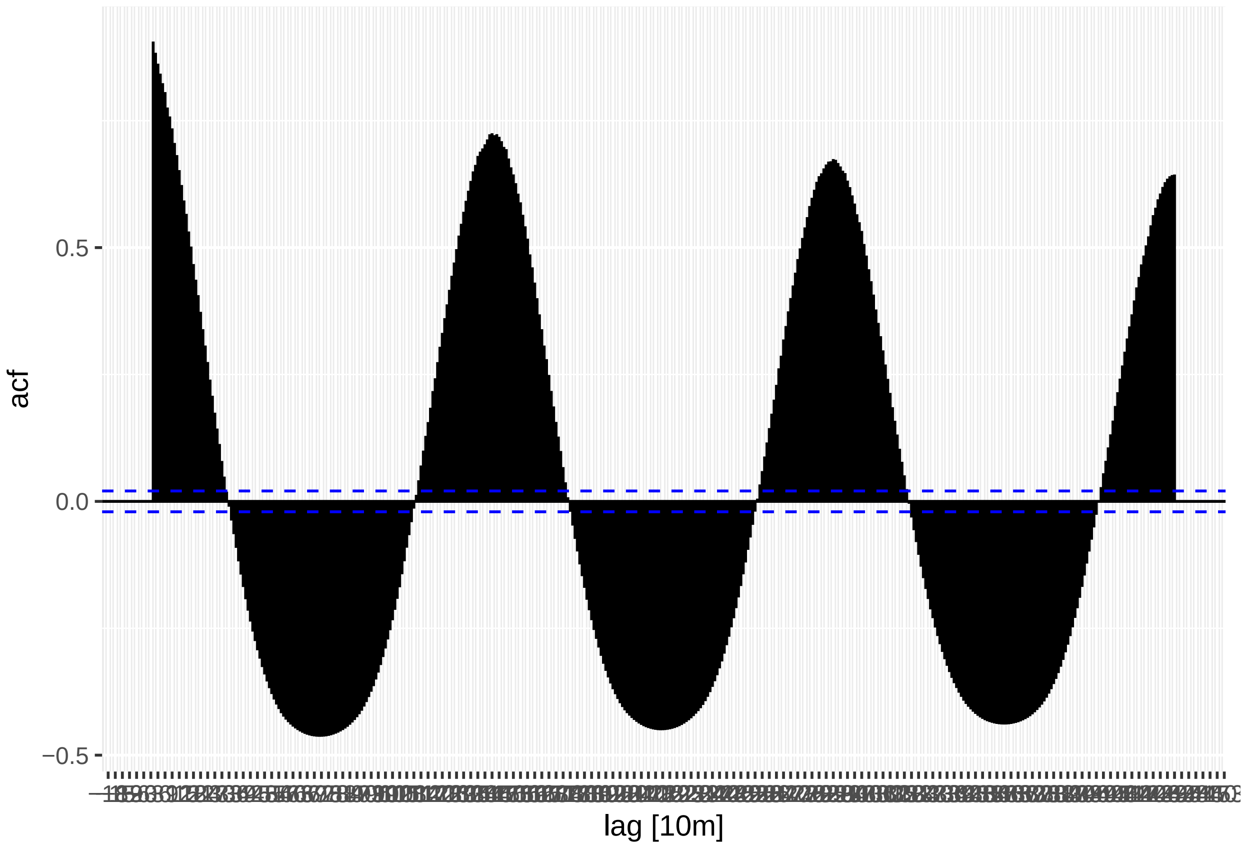
```r
       y = "POA",
       colour = "Cloud Cover")

POA_vs_LAST_7D_CC / (line_cc + polar_cc)
```

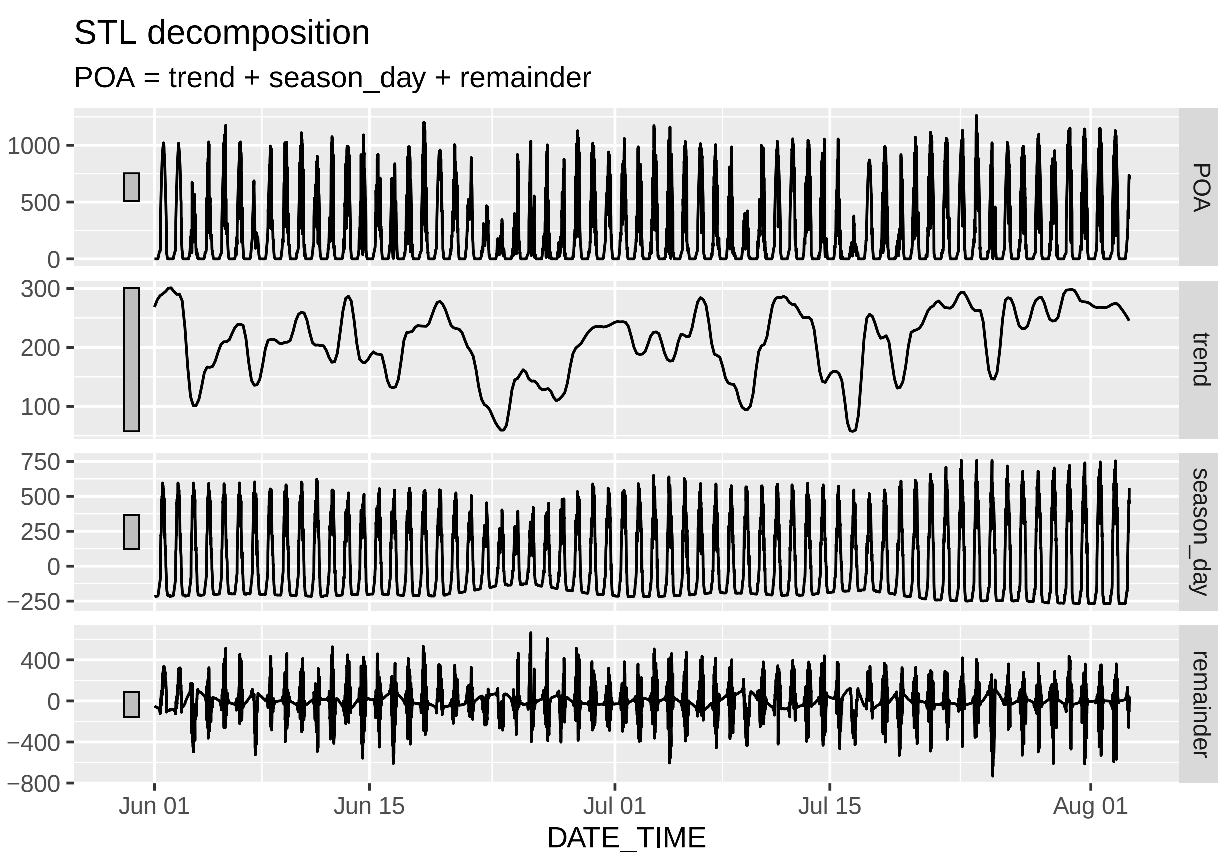
```
## Warning: `as.hms()` was deprecated in hms 0.5.0.
## i Please use `as_hms()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```




```r
data %>% ACF(POA, lag_max = 3 * 24 * 6, season = "day") |> autoplot()
```

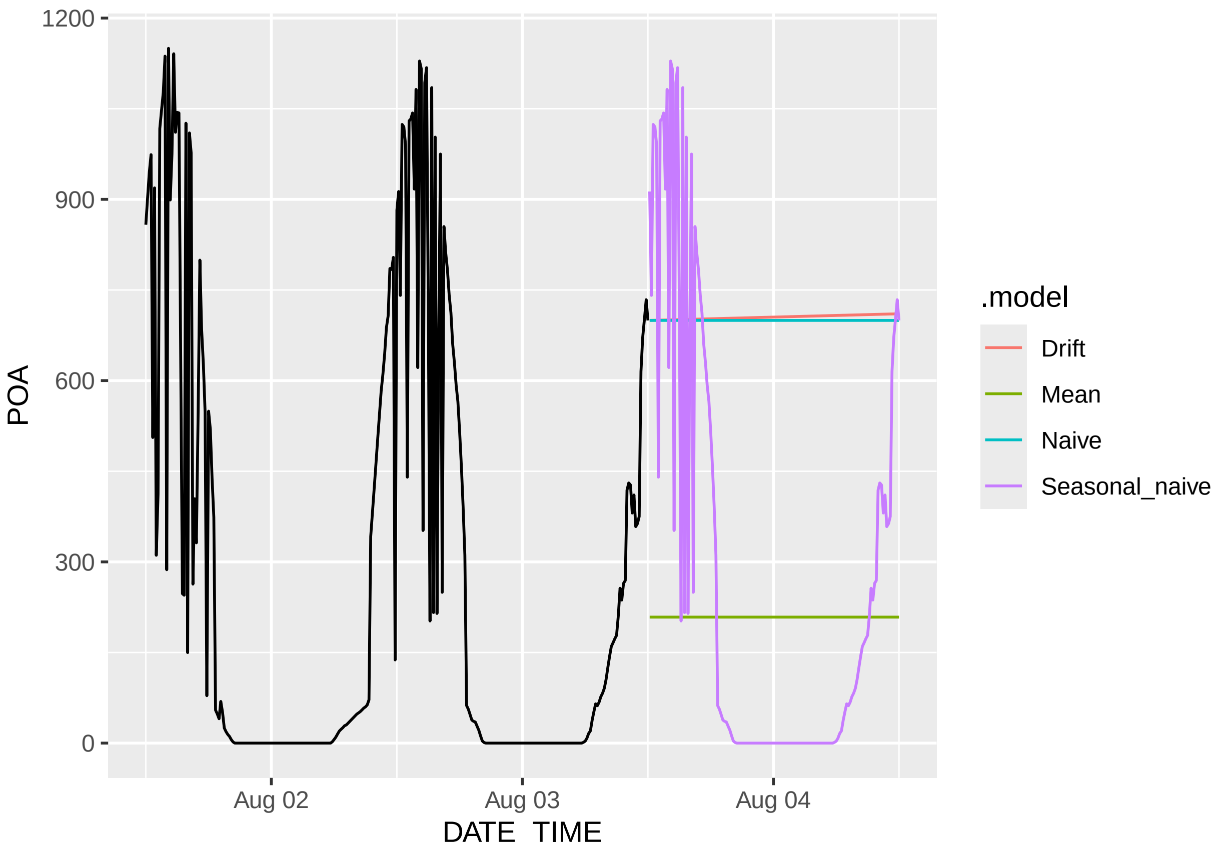
```
## Warning: The `...` argument of `PACF()` is deprecated as of feasts 0.2.2.
## i ACF variables should be passed to the `y` argument. If multiple variables are
##   to be used, specify them using `vars(...)`.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.

## Warning: ACF currently only supports one column, `POA` will be used.
```


acf
0.5
0.0
−0.5
lag [10m]

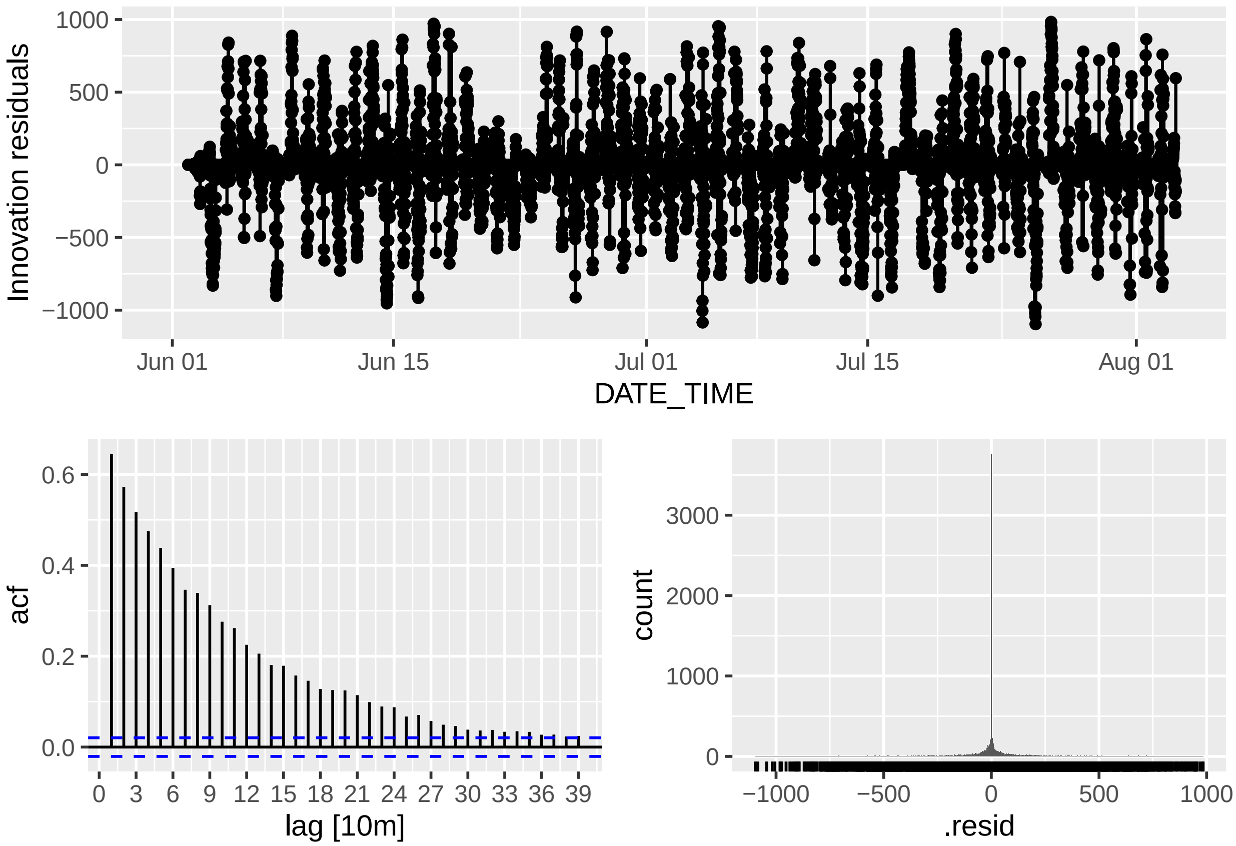

STL decomposition
POA = trend + season_day + remainder
POA
trend
season_day
remainder
DATE_TIME
Jun 01
Jun 15
Jul 01
Jul 15
Aug 01

2 Benchmark Models

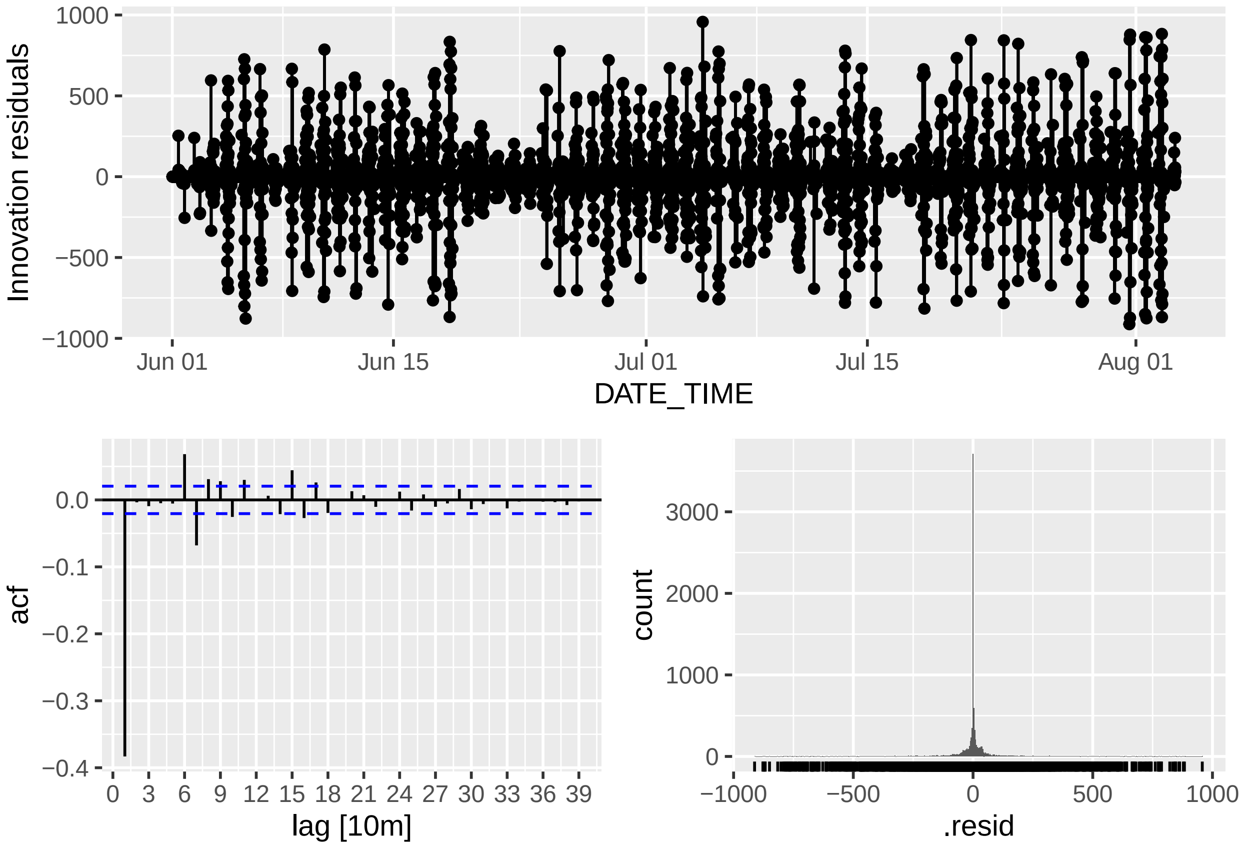
```
benchmarks <- data %>% model(
  Seasonal_naive = SNAIVE(POA ~ lag("1 day")),
  Naive = NAIVE(POA),
  Drift = RW(POA ~ drift()),
  Mean = MEAN(POA)
)

benchmark_forecasts <- benchmarks %>% forecast(h = "1 days")

benchmark_forecasts %>%
  autoplot(level = NULL) +
  autolayer(data %>% filter(
    DATE_TIME >= max(DATE_TIME) - as.difftime(2, units = "days")), POA
  )
```




```
gg_tsresiduals(benchmarks["Seasonal_naive"])
```




```
gg_tsresiduals(benchmarks["Naive"])
```



Assume the residuals are white noise - Use ljung-box to determine whether the residuals are indistringuishable from white noise - If lb_pvalue > 0.05, then

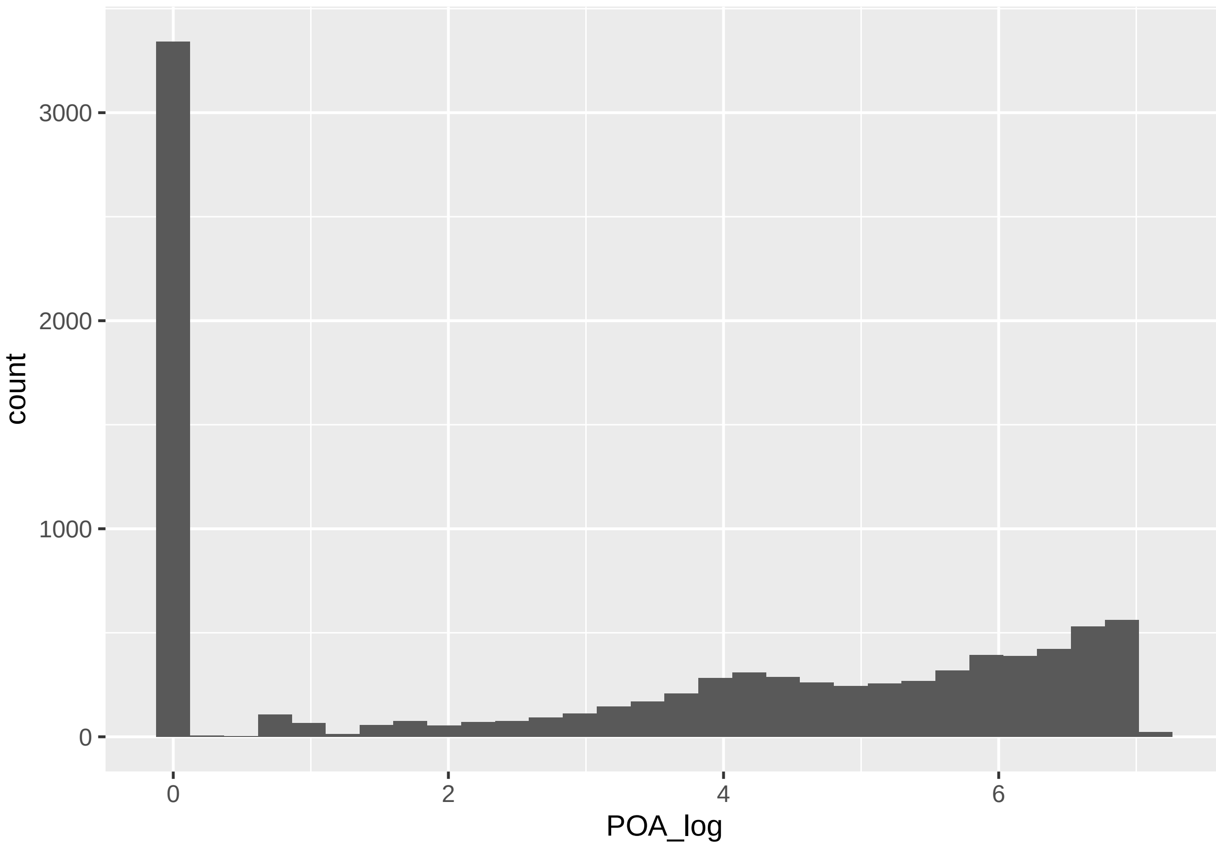
```
augment(benchmarks) %>% features(.resid, ljung_box, lag = 2 * 6 * 24)
```

```
## # A tibble: 4 x 3
##   .model         lb_stat lb_pvalue
##   <chr>            <dbl>     <dbl>
## 1 Drift            1929.         0
## 2 Mean           493151.         0
## 3 Naive            1929.         0
## 4 Seasonal_naive  30037.         0
```

3 Data Transforms

```
data <- data %>%
  mutate(POA_log = log(POA + 1), # Add 1 to POA to avoid log(0)
         POA_log = ifelse(is.infinite(POA_log) | is.nan(POA_log), NA, POA_log))
data <- data %>%
  mutate(POA_diffed_log = c(NA, diff(POA_log)))

data %>% ggplot(aes(x = POA_log)) +
  geom_histogram()
```

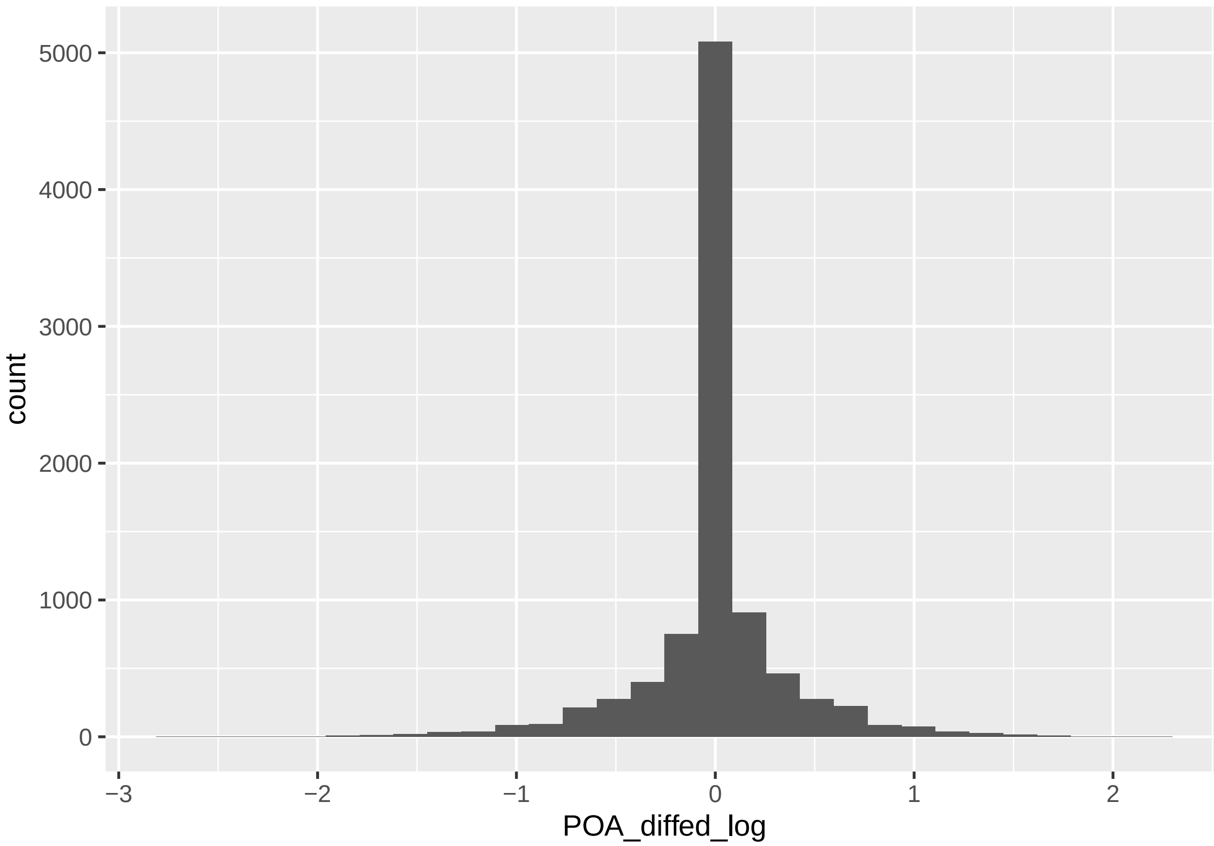
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```r
data %>% ggplot(aes(x = POA_diffed_log)) +
  geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

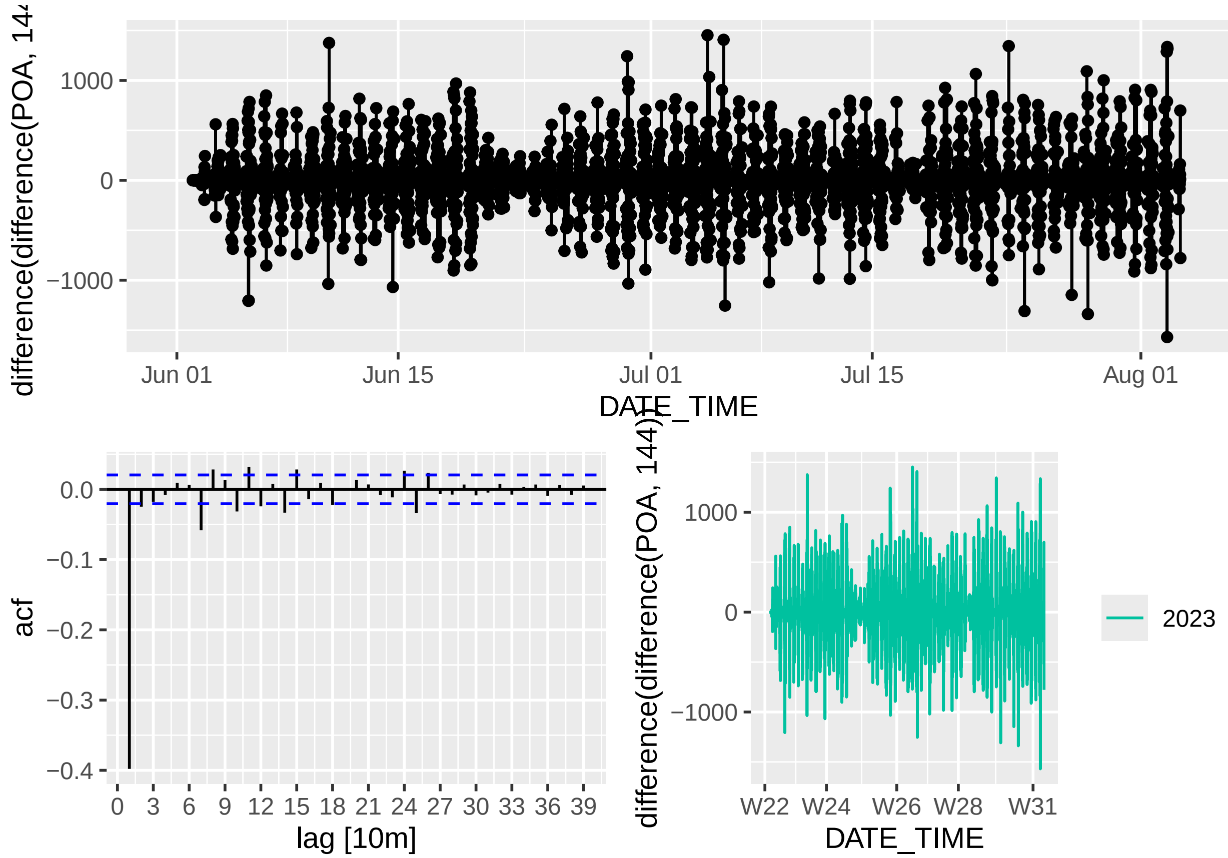
```
## Warning: Removed 1 row containing non-finite outside the scale range
## (`stat_bin()`).
```



4 Model Fitting

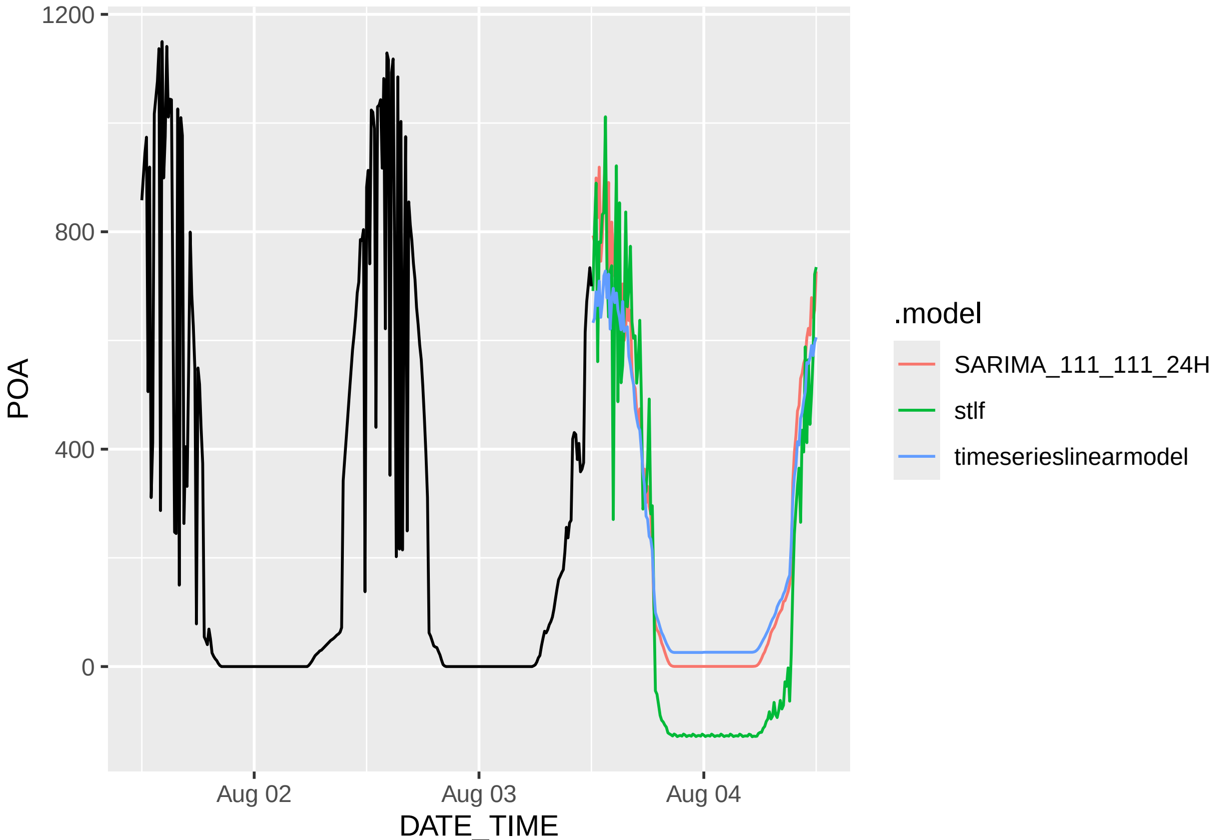
```
data %>% gg_tsdisplay(
  difference(POA, 144) |> difference()
)
```

```
## Warning: Removed 145 rows containing missing values or values outside the scale range
## (`geom_line()`).

## Warning: Removed 145 rows containing missing values or values outside the scale range
## (`geom_point()`).

## Warning: Removed 145 rows containing missing values or values outside the scale range
## (`geom_line()`).
```




```
fit <- data %>% model(
  stlf = decomposition_model(
    STL(POA ~ trend(), robust = TRUE),
    NAIVE(season_adjust)
  ),
  timeserieslinearmodel = TSLM(POA ~ trend() + season("1 day")),
  SARIMA_111_111_24H = ARIMA(
    log(POA + 1) ~ pdq(1, 1, 1) + PDQ(1, 1, 1, "1 day")
  ),
)
```

```
fit_forecasts <- fit %>% forecast(h = "1 days")

fit_forecasts <- fit_forecasts %>%
  mutate(.mean = pmax(.mean, 0))

fit_forecasts %>%
  autoplot(level = NULL) +
  autolayer(
    data %>% filter(
      DATE_TIME >= max(DATE_TIME) - as.difftime(2, units = "days")
    ), POA
  )
```



5 Model Evaluation

Here we talk about how we will evaluate the models

6 Final Predicitions and Conclusions

Here we talk about our final predictions and conclusions