# Let’s Get Started

library(tidyverse) library(timetk)

# Setup for the plotly charts (# FALSE returns ggplots) interactive <- TRUE

# Correlation Plots

plot\_acf\_diagnostics() returns the ACF and PACF of a target and optionally CCF’s of one or more lagged predictors in interactive plotly plots. We also scale to multiple time series using group\_by().

ACF = Autocorrelation between a target variable and lagged versions of itself. PACF = Partial Autocorrelation removes the dependence of lags on other lags highlighting key seasonalities.

CCF = Shows how lagged predictors can be used for prediction of a target variable.

### Lag Specification

Lags (.lags) can either be specified as:

A time-based phrase indicating a duraction (e.g. 2 months) A maximum lag (e.g. .lags = 28)

A sequence of lags (e.g. .lags = 7:28)

### Scales to Multiple Time Series with Groups

The plot\_acf\_diagnostics() works with grouped\_df’s, meaning you can group your time series by one or more categorical columns with dplyr::group\_by() and then apply plot\_acf\_diagnostics() to return group-wise lag diagnostics.

### Special Note on Groups

Unlike other plotting utilities, the .facet\_vars arguments is NOT included. Use dplyr::group\_by() for processing multiple time series groups.

### Calculating the White Noise Significance Bars

The formula for the significance bars is +2/sqrt(T) and -2/sqrt(T) where T is the length of the time series. For a white noise time series, 95% of the data points should fall within this range. Those that don’t may be significant autocorrelations.

**Grouped ACF Diagnostics**

m4\_hourly %>%

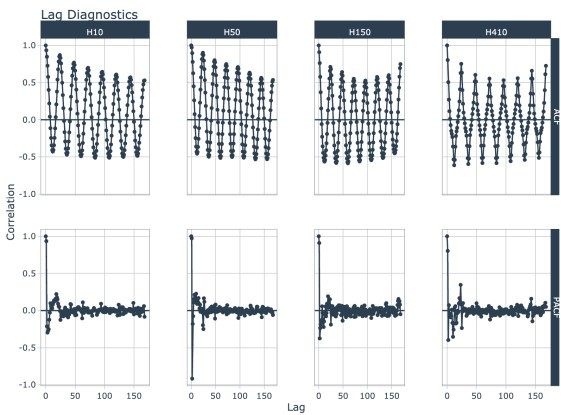
group\_by(id) %>% plot\_acf\_diagnostics(

date, value, # ACF & PACF

.lags = "7 days", # 7-Days of hourly lags

.interactive = interactive

)



## Grouped CCF Plots

walmart\_sales\_weekly %>%

select(id, Date, Weekly\_Sales, Temperature, Fuel\_Price) %>% group\_by(id) %>%

plot\_acf\_diagnostics(

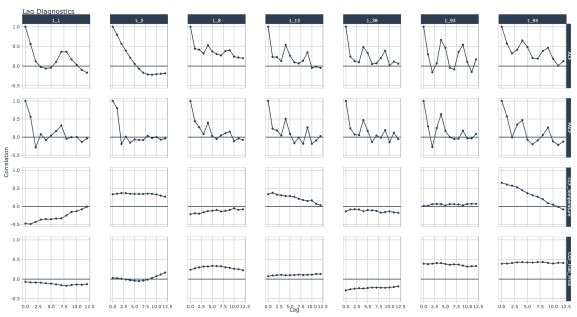
Date, Weekly\_Sales, # ACF & PACF

.ccf\_vars = c(Temperature, Fuel\_Price), # CCFs

.lags = "3 months", # 3 months of weekly lags

.interactive = interactive

)



# Seasonality

plot\_seasonal\_diagnostics() is an interactive and scalable function for visualizing time series seasonality.

### Automatic Feature Selection

Internal calculations are performed to detect a sub-range of features to include using the following logic:

The *minimum* feature is selected based on the median difference between consecutive timestamps

The *maximum* feature is selected based on having 2 full periods.

Example: Hourly timestamp data that lasts more than 2 weeks will have the following features: “hour”, “wday.lbl”, and “week”.

### Scalable with Grouped Data Frames

This function respects grouped data.frame and tibbles that were made with dplyr::group\_by().

For grouped data, the automatic feature selection returned is a collection of all features within the sub- groups. This means extra features are returned even though they may be meaningless for some of the groups.

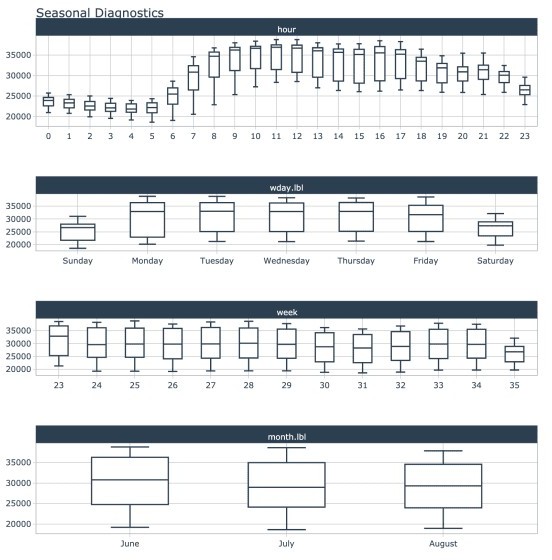
### Transformations

The .value parameter respects transformations (e.g. .value = log(sales))

**Seasonal Visualizations**

taylor\_30\_min %>%

plot\_seasonal\_diagnostics(date, value, .interactive = interactive)

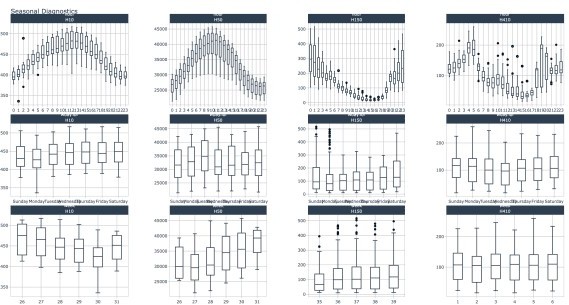


## Grouped Seasonal Visualizations

m4\_hourly %>%

group\_by(id) %>%

plot\_seasonal\_diagnostics(date, value, .interactive = interactive)



# STL Diagnostics

The plot\_stl\_diagnostics() function generates a Seasonal-Trend-Loess decomposition. The function is “tidy” in the sense that it works on data frames and is designed to work with dplyr groups.

### STL method

The STL method implements time series decomposition using the underlying stats::stl(). The decomposition separates the “season” and “trend” components from the “observed” values leaving the “remainder”.

### Frequency & Trend Selection

The user can control two parameters: .frequency and .trend.

The .frequency parameter adjusts the “season” component that is removed from the “observed” values.

The .trend parameter adjusts the trend window (t.window parameter from stl()) that is used.

The user may supply both .frequency and .trend as time-based durations (e.g. “6 weeks”) or numeric values (e.g. 180) or “auto”, which automatically selects the frequency and/or trend based on the scale of the time series.

m4\_hourly %>%

group\_by(id) %>% plot\_stl\_diagnostics(

date, value,

.frequency = "auto", .trend = "auto",

.feature\_set = c("observed", "season", "trend", "remainder"),

.interactive = interactive)

