Machine Learning Time Series Forecasting for Solar Data Prediction

Roger K Alexander

2023-09-14

Machine Learning Time Series Forecasting Project

This (R Markdown) document provides the final project report and documentation for the work done on the Data Science PH125 Course 9, 2nd Capstone Project (Choose Your Own Project). The work undertaken is the development of a Time Series Forecasting System using Machine Learning (ML) algorithms.

> Note: The project uses a number of R packages and includes .csv data files that must be dowloaded as specified in this Markdown document. Based on the included visualization and analysis code, compiling of the Markdown document on a standard multi-core PC/laptop will require many hours to run to completion. Time stamps in the code chunks give an indication of the overall execution time. In the code, as well as the companion R script, a scaling parameter ('scaler') is included that allows for extreme compression of the data so that the full analysis can be completed in minutes (for ex. by setting scaler = 12, which compresses the data to 2 aggregated measurements per day in addition to the monthly data average already applied). This however is done at the expense of limiting some of the granularity of the analysis and associated data visualizations. The **'scaler'** parameter can be found in the {r "Further data set size reduction"} code chunk.

Introduction

This project is an application of machine learning (ML) to time series data forecasting. The forecast variable is direct normal irradiance (DNI), a solar radiation measurement used for assessing the energy harvesting potential of solar photo-voltaic (PV) panels at a given geographic location. Employing ML techniques to time series data forecasting is a general problem with wide application across a range of different fields from ecommerce to industrial production. The specific focus of this project is to understand how well ML-based forecasting performs for a given forecast time horizon that extends significantly beyond the time period for which data is available for model training.

This study is motivated by the fact that training data used for solar prediction, while generally of very high quality, is limited by when the measured data is available for use. For practical deployed systems, even where extensive archival data exists, recent data is only available after a delayed collection and aggregation period that currently has a lag of up to three years.

Many studies have been undertaken on the use of ML for solar forecasting (see References [1], [2], [3]). The setup and execution of this project however applies related analysis techniques developed in the course of the HarvardX Data Science course series. It should also be noted that at least one recent study [5] has questioned whether ML algorithms can provide better performance forecasting than classical methods such as Autoregressive Integrated Moving Average (ARIMA). We acknowledge this potentially important caveat while still primarily focusing this project on the development and application of ML-based models. However, in keeping with techniques learned in the course series, we introduce a number of ensemble models as a means

of examining whether the concerns of ML forecasting relative to classical prediction models can at least be somewhat overcome through performance improvements obtained by ensemble modeling.

The data used for this project is obtained from the publicly available US National Renewable Energy Labs (NREL) National Solar Radiation Database (NSRDB) [8]. This database provides one of the premier international sources of serially complete measurement and modeling data on solar radiation. An R script program, included in this report, was implemented to access the database using the NSRDB-specified application programming interface (API). The compiled data set used in the analysis is included as .csv files that are part of the project submission.

Background & Goals

Solar Radiation Data

In solar analysis calculations there are three key measurement parameters: Direct Normal Irradiance (DNI), which represents the quantity of radiation received per unit area on a surface perpendicular to the sun, Global Horizontal Irradiance (GHI), which defines the total radiation absorbed on a horizontal surface on the Earth, and Diffuse Horizontal Irradiance (DHI) which is the terrestrial irradiance received by a flat surface, after scattering or diffusion by the atmosphere. Collectively, these three measurement parameters together with meteorological and environmental data such as Wind Speed, Surface Temperature, and Solar Zenith Angle will determine the amount of energy that can be harvested by a photo-voltaic (PV) solar panel. The amount of solar energy that can be harvested during periods of sunlight must be matched with energy storage capacity in an autonomously powered system.

The NREL NSRDB is an authoritative source for the collection, maintenance and access for the type of data that is used in solar-related renewable energy research and operational activities. The database can provide up to half-hourly measurement and modelling data for the defined solar radiation variables at 4 km-square geographic areas across the globe. Depending on location, the available data spans decades.

Since the key solar measurement parameters are independently measured and available, for the purpose of this project we examine just one of these, the DNI (Direct Normal Irradiance). However, the univariate ML-based forecasting approach and analysis undertaken for DNI can be readily applied to any of the other data parameters downloadable from the NSRDB site. To limit the size of the data sets and to allow a focus on the univariate prediction problem, the implemented NSRDB API script restricts the data download to the single DNI solar radiation attribute for this project.

The compiled analysis data comprises twenty-one years of hourly time-series data for the DNI variable measured at a single geographic location. Each year of hourly data comprises approximately 125 kB (kilobytes). The total data size for the project is therefore approximately 2.6 MB. The location used for the study is a point in New York (NSRDB Location ID 1244690). This point was selected solely as a familiar geographic reference location.

Data Forecasting Objective

The goal of the analysis will be to use multiple years of DNI measurement training data to make forecasts over time periods that extend beyond the period of the available training data. Given that there are seasonal and annual variations in the solar variables of interest, the period of forecasting performance assessment must at least be up to one year. With the current NSRDB data availability, forecasts are required for up to three years to accommodate the worst case lag in accessible solar radiation data. A multi-year forecast period that covers the diurnal, seasonal, and annual cycles will allow an assessment of how well the ML forecasting model captures the different cyclic variations.

To provide forecasts beyond the available model training data, the study will use a baseline assumption of a fixed number of years of prior training data. This fixed training data duration is determined by the total duration of the available data set, though also limited in this project by the available processing

capacity required for model development and analysis. With time series forecasting, unlike typical cases of bootstrapping where data sets can be randomly drawn across the available training data set, there is a requirement that training and forecasting be done using sequential and ideally full complete data sets. As such, the maximum twenty-one (21) years of hourly DNI measurement data was downloaded to provide sufficient data for the generation of time consecutive cross-validation (CV) training and testing data sets during the model development.

A useful ML-algorithm performance metric for the time-series forecasting assessment is the Mean Absolute Percentage Error (MAPE) where for a forecast of N time period values, the MAPE loss function is given by:

$$\frac{1}{N}\sum_{i=1}^{N}\frac{|(\hat{y}_i - y_i)|}{y_i} * 100\%$$

where $\hat{y}_i$ and y_i are the forecast and actual data values respectively.

MAPE provides a representative measure and as a percentage value is variable-independent. It offers an indicative representation of how actual and forecast values differ over the entire duration for which forecasts are made. Like the commonly used Root Mean Square Error (RMSE) loss function, MAPE provides a good measure of performance by capturing the mean of the absolute difference between actual and predicted/forecast values. Additionally, a percentage value does provide a more intuitive indication of performance even without any appreciation for the magnitudes of the actual variable quantities. As will be discussed in the report however, MAPE can become indeterminate where the measurement quantity can have zero values. In this project where this is indeed the case, we introduce a different but functionally equivalent percentage error performance measure.

While the interest in the analysis is the percentage error performance for the given forecast duration, the analysis will also examine the specific daily DNI data curves that are forecast to determine how well they capture and correspond to the known and expected diurnal solar radiation patterns.

Overview and Analysis Approach

This project makes central use of the caret package and its framework of machine learning models and capabilities for helping to find optimal models. The caret package integrates various activities such as data partitioning, cross-validation, and parameter tuning selection related to model development and allows easy incorporate of many of the major ML algorithms needed for the regression analysis. For the particular time series forecasting, the purpose-built "Conformal Time Series Forecasting Using State of Art Machine Learning Algorithms", caretForecast package, is used https://cran.r-project.org/web/packages/caretForecast/caretForecast.pdf.

As previously indicated, the study will also include the implementation of ensemble modeling that allows multiple ML algorithms to be applied collectively to potentially provide improvements over that of a single ML algorithm. While there is also a caretEnsemble package that provides the capabilities for automating the creation of ensemble models, the ensemble models developed in this study are custom-built as part of the undertaken project code development.

> Note: The caretForecast package used in the study applies RMSE as the internal default metric for algorithm regression performance assessment and optimization. The package is also able to provide the calculated Mean Absolute Percentage Error (MAPE) performance measure. However, as will be discussed further, MAPE or other percentage measures become indeterminate when zero-measurement values are encountered. This is the case with DNI solar measurements and so a different percentage measure, the Mean Absolute ** Daily** Percentage Error (MADPE) is introduced.

Initial Data Exploration

As with any data analysis undertaking, we begin with an exploration of the data that will be used in the study. We can examine both the specific data that will be evaluated as well as briefly take a look at the data that is more generally available via the NREL NSRDB database. The data set used for this project is included as separate csv files. Also included for background reference is a data file that provides an example of the more general data that can be obtained from the NSRDB site.

!!! IMPORTANT !!! Please download and copy the included project csv files to your current R working directory (given by the path obtained from the *getwd()* function call). Once the csv files are copied to the working directory the R script will be able to access them by referencing the relative local path address.

> Note: Since the undertaken analysis is based on univariate prediction, only a single attribute, the Direct Normal Irradiance (DNI) is included in the compiled working data set. This allows longer time spans of time series data to be provided for the analysis while still limiting the overall data size.

```
############ Initial Data Set Retrieval and Exploration ########################

# !!! IMPORTANT !!!

# Please copy the downloaded the .csv data set files ('source_file' and 'example_file')
# to your current R working directory given by the path obtained from the getwd()
# function call

# Once the csv files are copied to the working directory the R script will access them
# from there with the following code that uses the relative local path addresses for the
# file access

# Determine your current R working directory
getwd()
```

```
## [1] "C:/Users/roger/Documents/EDX Online Courses/HarvardX/R-Projects/ds_PH125_capstone_project"
```

```
# Set the path object to the current working directory
path_wd <- getwd()

# These file labels are the names of the csv data file provided for review
datafile_label <- "solar_data_LOCID_1244690_2000-2020.csv"
metafile_label <- "solar_data_LOCID_1244690_2000-2020_meta.csv"

# File name and path for retrieving and reading in source data file
local_source_filepath <- paste0(path_wd, "/", datafile_label)
local_source_metafilepath <- paste0(path_wd, "/", metafile_label)

# Reading in source file for analysis (plus metadata file)
source_data <- read.csv(local_source_filepath)
source_metadata <- read.csv(local_source_metafilepath)

# Object for solar measurements metadata from the input source file
solar_metadata <- source_metadata

# Displaying metadata contents from the input source file (including the parameter units
```

```r
# used for measurement and environmental variables available from NSRDB)
str(solar_metadata)
```

```
## 'data.frame':    1 obs. of  46 variables:
##  $ Source                   : chr "NSRDB"
##  $ Location.ID              : int 1244690
##  $ City                     : chr "-"
##  $ State                    : chr "-"
##  $ Country                  : chr "-"
##  $ Latitude                 : num 40.7
##  $ Longitude                : num -74
##  $ Time.Zone                : int -5
##  $ Elevation                : int 0
##  $ Local.Time.Zone          : int -5
##  $ Clearsky.DHI.Units       : chr "w/m2"
##  $ Clearsky.DNI.Units       : chr "w/m2"
##  $ Clearsky.GHI.Units       : chr "w/m2"
##  $ Dew.Point.Units          : chr "c"
##  $ DHI.Units                : chr "w/m2"
##  $ DNI.Units                : chr "w/m2"
##  $ GHI.Units                : chr "w/m2"
##  $ Solar.Zenith.Angle.Units: chr "Degree"
##  $ Temperature.Units        : chr "c"
##  $ Pressure.Units           : chr "mbar"
##  $ Relative.Humidity.Units : chr "%"
##  $ Precipitable.Water.Units: chr "cm"
##  $ Wind.Direction.Units     : chr "Degrees"
##  $ Wind.Speed.Units         : chr "m/s"
##  $ Cloud.Type..15           : chr "N/A"
##  $ Cloud.Type.0             : chr "Clear"
##  $ Cloud.Type.1             : chr "Probably Clear"
##  $ Cloud.Type.2             : chr "Fog"
##  $ Cloud.Type.3             : chr "Water"
##  $ Cloud.Type.4             : chr "Super-Cooled Water"
##  $ Cloud.Type.5             : chr "Mixed"
##  $ Cloud.Type.6             : chr "Opaque Ice"
##  $ Cloud.Type.7             : chr "Cirrus"
##  $ Cloud.Type.8             : chr "Overlapping"
##  $ Cloud.Type.9             : chr "Overshooting"
##  $ Cloud.Type.10            : chr "Unknown"
##  $ Cloud.Type.11            : chr "Dust"
##  $ Cloud.Type.12            : chr "Smoke"
##  $ Fill.Flag.0              : chr "N/A"
##  $ Fill.Flag.1              : chr "Missing Image"
##  $ Fill.Flag.2              : chr "Low Irradiance"
##  $ Fill.Flag.3              : chr "Exceeds Clearsky"
##  $ Fill.Flag.4              : chr "Missing CLoud Properties"
##  $ Fill.Flag.5              : chr "Rayleigh Violation"
##  $ Surface.Albedo.Units     : chr "N/A"
##  $ Version                  : chr "3.2.0"
```

```
# Note: The NREL NSRDB API that was used for the download of individual location solar
# data includes metadata in each (date-defined) csv data file file as well as the ability
# to specify the set of solar radiation measurement attribute information that is
# downloaded.

# In downloading and compiling the input data set for this project, the metadata is
# separated from the measurement data but maintained as part of the compiled data list
# containing the aggregated data files.

# Object for solar measurements solar data set from the input source data file
solar_datafile <- source_data

# Examination of the structure and contents of the input data set.  Note: Since the
# undertaken analysis is based on univariate prediction, only a single attribute, the
# Direct Normal Irradiance (DNI) is included in the compiled data set.
str(solar_datafile)
```

```
## 'data.frame':    183960 obs. of  6 variables:
##  $ Year  : int  2000 2000 2000 2000 2000 2000 2000 2000 2000 2000 ...
##  $ Month : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ Day   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ Hour  : int  0 1 2 3 4 5 6 7 8 9 ...
##  $ Minute: int  30 30 30 30 30 30 30 30 30 30 ...
##  $ DNI   : int  0 0 0 0 0 0 0 0 0 63 ...
```

```
# With hourly interval data and no leap year data (as explicitly specified in API
# database request), the expected number of data points for 21 years of data (maximum
# available for download at selected location) is:
24 * 365 * 21
```

```
## [1] 183960
```

```
# The oldest time period covered by the full data file is from January 1, 2000, as shown
# below at the head of the data frame.  Note: The hourly data is provided on the
# half-hour mark (30 minute)
head(solar_datafile)
```

```
##   Year Month Day Hour Minute DNI
## 1 2000     1   1    0     30   0
## 2 2000     1   1    1     30   0
## 3 2000     1   1    2     30   0
## 4 2000     1   1    3     30   0
## 5 2000     1   1    4     30   0
## 6 2000     1   1    5     30   0
```

```
# As can be confirmed below, the most recent date-time in the data file is for Year=2020,
# Month=12, Day=31, and Hour=23, as shown at the tail of the data frame.
tail(solar_datafile)
```

```
##        Year Month Day Hour Minute DNI
```

```
## 183955 2020    12  31    18     30    0
## 183956 2020    12  31    19     30    0
## 183957 2020    12  31    20     30    0
## 183958 2020    12  31    21     30    0
## 183959 2020    12  31    22     30    0
## 183960 2020    12  31    23     30    0
```

Data Set Time Series and Date Specification

For analysis purposes an R date-time object is derived from the measurement time information provided in the NSRDB data set file. This Date object is then used to convert the DNI measurements into a time-series (*ts*) object. The converted *ts* object is also a required input to the caretForecast package and for supporting a number of other R time-manipulating packages used in the analysis.

> R offeres a wide array of packages for working with date and time series data a number of which are used throughout the analysis.

```
############ Data Set Date-Time Specification and Date Reference Specification ####

# The Date variable can be used as the reference for variable time slicing of the data
# set for analysis and related time period referencing including data sets partitioning.

solar_dataset <- solar_datafile %>%
    mutate(Date = ymd_hm(paste(Year, Month, Day, Hour, Minute, sep = " "), tz = "EST"))
    ↪  %>%
    as_tibble()

# Note: The explicitly time zone setting to 'EST' (America/New_York) based on the NSRDB
# API download request to provide local time rather than UTC time whereas the ymd_hm()
# defaults to a 'UTC' time zone for date-time conversions.

# Solar data set with date-time object reference variable added to data frame
solar_dataset
```

```
## # A tibble: 183,960 x 7
##     Year Month   Day  Hour Minute   DNI Date
##    <int> <int> <int> <int>  <int> <int> <dttm>
##  1  2000     1     1     0     30     0 2000-01-01 00:30:00
##  2  2000     1     1     1     30     0 2000-01-01 01:30:00
##  3  2000     1     1     2     30     0 2000-01-01 02:30:00
##  4  2000     1     1     3     30     0 2000-01-01 03:30:00
##  5  2000     1     1     4     30     0 2000-01-01 04:30:00
##  6  2000     1     1     5     30     0 2000-01-01 05:30:00
##  7  2000     1     1     6     30     0 2000-01-01 06:30:00
##  8  2000     1     1     7     30     0 2000-01-01 07:30:00
##  9  2000     1     1     8     30     0 2000-01-01 08:30:00
## 10  2000     1     1     9     30    63 2000-01-01 09:30:00
## # i 183,950 more rows
```

```
tail(solar_dataset)
```

```
## # A tibble: 6 x 7
##    Year Month   Day  Hour Minute   DNI Date
##   <int> <int> <int> <int>  <int> <int> <dttm>
## 1  2020    12    31    18     30     0 2020-12-31 18:30:00
## 2  2020    12    31    19     30     0 2020-12-31 19:30:00
## 3  2020    12    31    20     30     0 2020-12-31 20:30:00
## 4  2020    12    31    21     30     0 2020-12-31 21:30:00
## 5  2020    12    31    22     30     0 2020-12-31 22:30:00
## 6  2020    12    31    23     30     0 2020-12-31 23:30:00
```

```
# Ex. using Date variable to check on the solar data set time period

# Earliest measurement data point
min(solar_dataset$Date)
```

```
## [1] "2000-01-01 00:30:00 EST"
```

```
# Most recent measurement data point
max(solar_dataset$Date)
```

```
## [1] "2020-12-31 23:30:00 EST"
```

```
############ Converting DNI data to a time series object ########################

# From the solar radiation data set we create a time series object for the (hourly) DNI
# variable
dni_ts <- solar_dataset %>%
    mutate(dni_ts = ts(DNI, start = c(Year[1], 1), frequency = 8760)) %>%
    pull(dni_ts)

# The frequency argument specifies the maximum lag or cycle in the data given in data
# points
str(dni_ts)
```

```
##  Time-Series [1:183960] from 2000 to 2021: 0 0 0 0 0 0 0 0 0 63 ...
```

```
# This time series object will be used as input to the caretForecast analysis package
# that will support the undertaken analyses.
```

Preliminary Data Review

Data completeness and consistency is an important prerequisite to any analysis and even more critical with time series data to be used for forecasting. Before moving to any possible missing value imputation and/or variable pre-processing, we can observe the descriptive statistics of the data in the original solar data file, particularly the DNI variable.

The skimr package provides the capability to readily show key descriptive stats for the data frame variables. Interesting descriptive statistics can generally be gleaned from a data frame of numeric data. However, in this case there is just primarily a single variable of interest, DNI. Nonetheless, the skim() function can still be used to verify completeness of the DNI and transformed Date variables and can display related numeric stats including DNI value extremes.

```
# Note use of skim_without_charts() to overcome encoding issue for Rmarkdown pdf output
dat_skim <- skim_without_charts(solar_dataset)

# Skimr data output
dat_skim
```

Table 1: Data summary

Name	solar_dataset
Number of rows	183960
Number of columns	7
Column type frequency:	
numeric	6
POSIXct	1
Group variables	None

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100
Year	0	1	2010.00	6.06	2000	2005.00	2010.0	2015.00	2020
Month	0	1	6.53	3.45	1	4.00	7.0	10.00	12
Day	0	1	15.72	8.80	1	8.00	16.0	23.00	31
Hour	0	1	11.50	6.92	0	5.75	11.5	17.25	23
Minute	0	1	30.00	0.00	30	30.00	30.0	30.00	30
DNI	0	1	177.18	290.59	0	0.00	0.0	292.00	1024

Variable type: POSIXct

skim_variable	n_missing	complete_rate	min	max	median	n_unique
Date	0	1	2000-01-01 00:30:00	2020-12-31 23:30:00	2010-07-02 12:00:00	183960

As seen, there are 183960 unique Date points which is as expected for 21 years of hourly data and also confirms that there has been no overlaps or other errors introduced in the processing and merging of the per-year data sets that were individually downloaded from the NREL NSRDB servers.

Data Pre-processing Prior to Partitioning

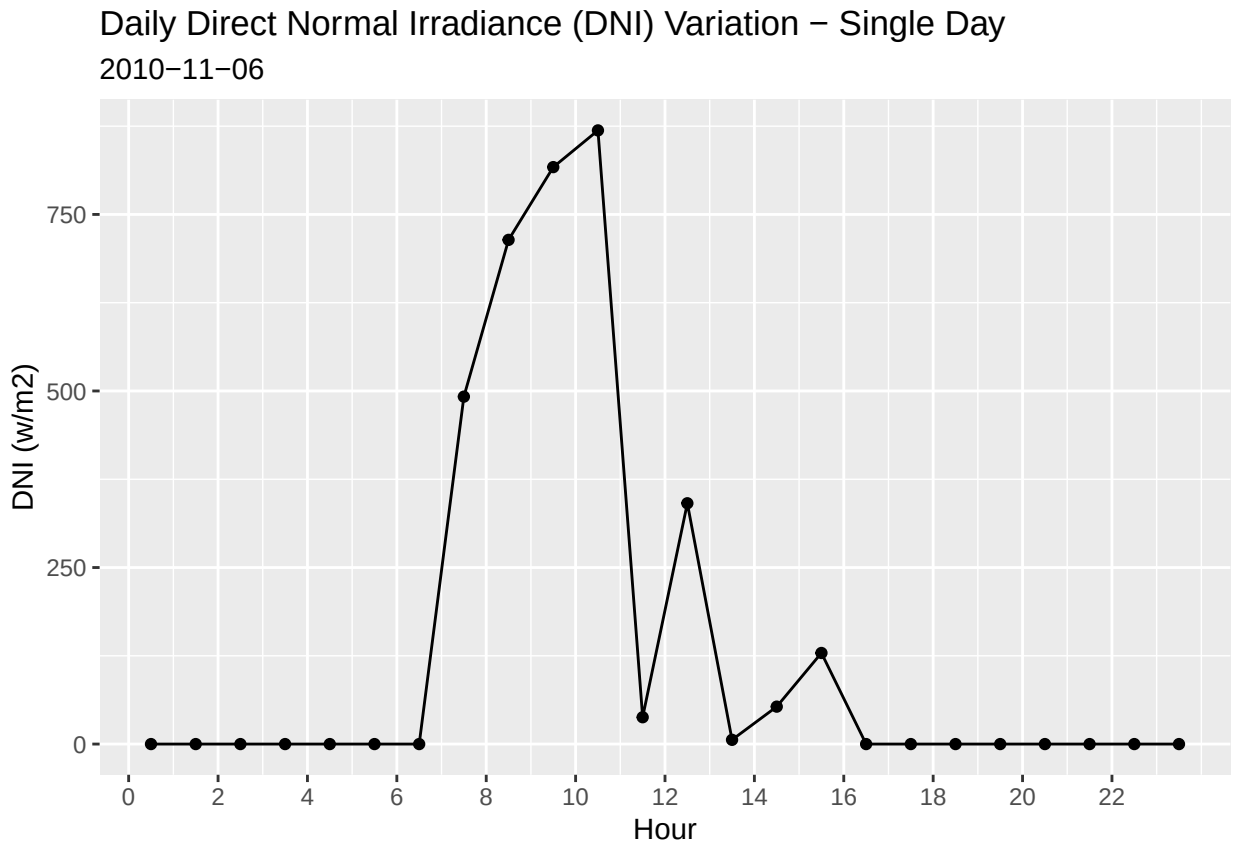
Since there does not appear to be missing data in the full data set we can proceed with data partitioning. Had there been missing data (not expected given the known quality of the NREL NSRDB source) we could have chosen to impute it by filling in with some appropriate values (see [4]).

Different techniques might apply based on the data type. In the case of the continuous DNI solar data, for example, we know that values are positive during daylight and 0 during nigh times which could be used in refining any needed data imputing. The caret package also offers a preProcess() function that could also have been applied, were it needed. With the NSRDB data no such requirements are needed.

Data Content Exploration

Having verified the general data completeness and consistency it is valuable to look more closely into the content of the data itself to understand the intrinsic data patterns and trends that exist. We can explore the data by observing the daily and monthly patterns and variations. Given the large quantity of data we will use random snapshots of time to obtained more visually digestible data subsets.

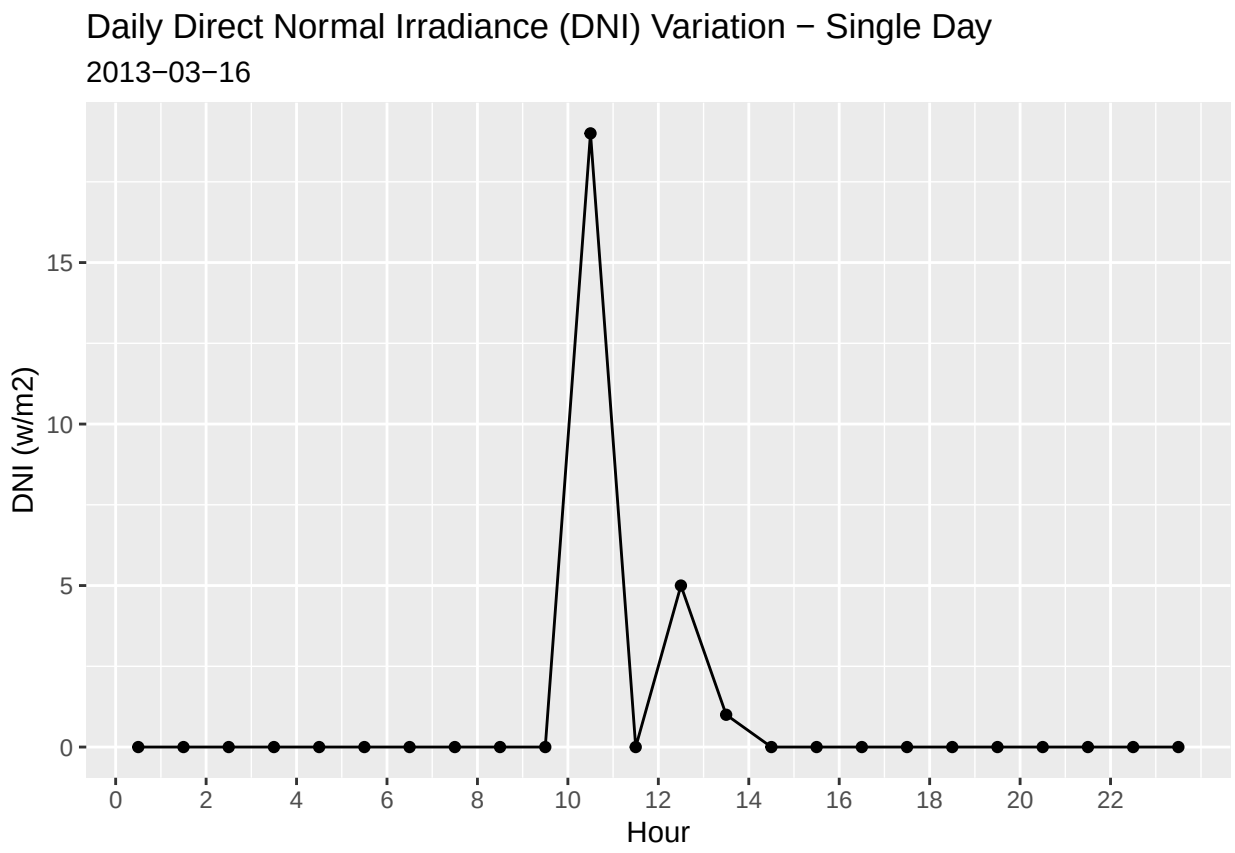
```
# We can begin by taking a randomly selected day from the data set Randomly selected day
s_day <- sample(solar_dataset$Date, 1)

solar_dataset %>%
    filter(Year == year(s_day) & Month == month(s_day) & Day == day(s_day)) %>%
    mutate(Hour = hour(Date) + minute(Date)/60) %>%
    ggplot(aes(x = Hour, y = DNI)) + geom_point() + geom_line() +
    ↪  scale_x_continuous(breaks = seq(0,
    23, 2)) + ggtitle("Daily Direct Normal Irradiance (DNI) Variation - Single Day",
    ↪  as.Date(s_day)) +
    labs(x = "Hour", y = "DNI (w/m2)")
```



The data shown is as would be expected where solar radiation is positive only during daylight hours but can vary as a function of local weather conditions and cloud cover. The duration of the periods where DNI solar radiation is positive is also function of the time of the year for the given geographic location (which in this project is New York on the US East coast).

We can also confirm the data consistency with additional, similarly randomly drawn days from the overall data set.

```r
# Randomly selected day
s_day <- sample(solar_dataset$Date, 1)

solar_dataset %>%
    filter(Year == year(s_day) & Month == month(s_day) & Day == day(s_day)) %>%
    mutate(Hour = hour(Date) + minute(Date)/60) %>%
    ggplot(aes(x = Hour, y = DNI)) + geom_point() + geom_line() +
    ↪  scale_x_continuous(breaks = seq(0,
    23, 2)) + ggtitle("Daily Direct Normal Irradiance (DNI) Variation - Single Day",
    ↪  as.Date(s_day)) +
    labs(x = "Hour", y = "DNI (w/m2)")
```



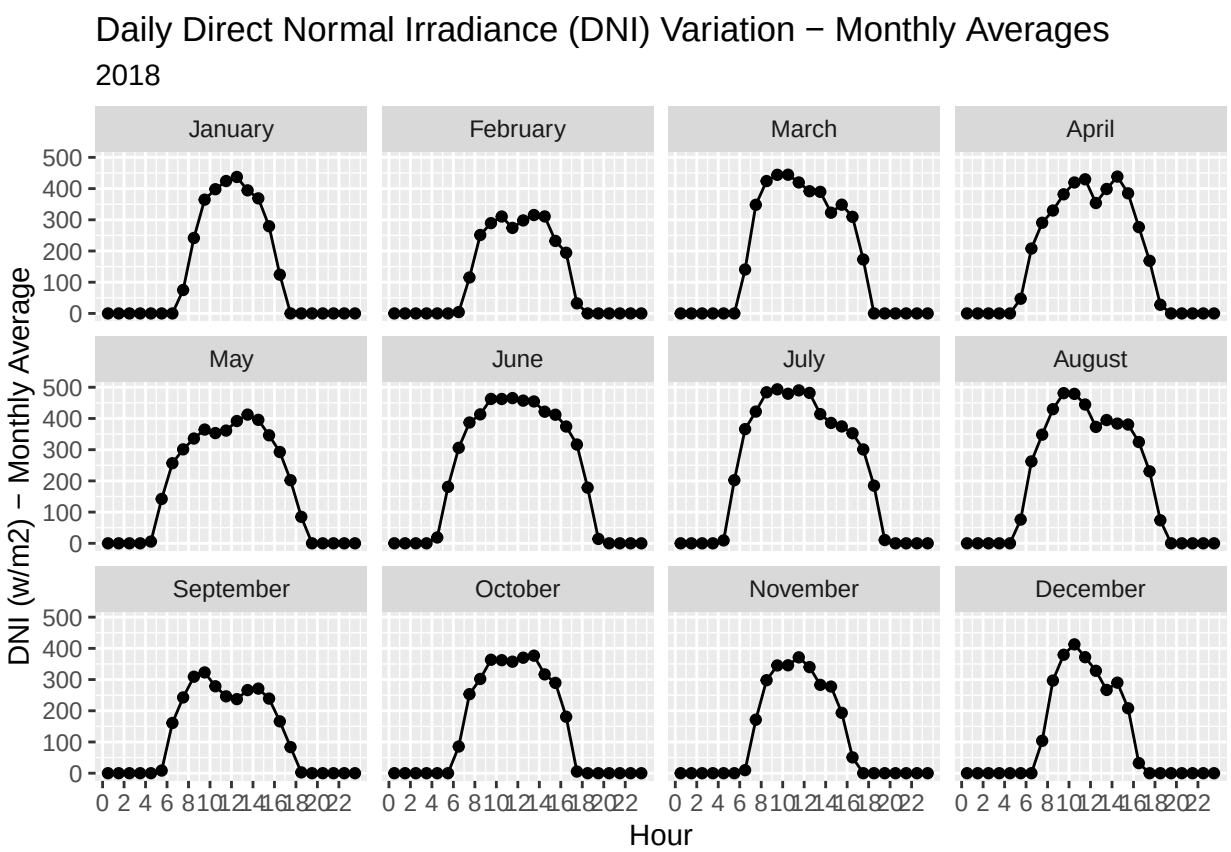

```r
# The number of hours of positive sunlight also depends on the time of the year for the
# given location - New York in this data set.
```

We can also examine the daily DNI pattern aggregation over the course of a month for a randomly selected year in the data set. The data is more readily visualized by calculating and displaying monthly averages of the daily DNI radiation patterns. Even though there is some fidelity that is loss at the level of solar activity of a specific day, this monthly aggregation of the data does provide a useful way to compress the available measurement data.

```r
############ Plotting Daily DNI Variation - Monthly Averages

# Randomly selected year
```

```
s_year <- year(sample(solar_dataset$Date, 1))

solar_dataset %>%
    filter(Year == s_year) %>%
    mutate(Hour = hour(Date) + minute(Date)/60, Calendar_Month =
    ↪  factor(month.name[month(Date)],
        levels = month.name)) %>%
    group_by(Month, Calendar_Month, Hour) %>%
    summarize(DNI_Hourly_Avg = mean(DNI)) %>%
    ggplot(aes(x = Hour, y = DNI_Hourly_Avg, group = desc(Calendar_Month))) +
    ↪  geom_point() +
    geom_line() + scale_x_continuous(breaks = seq(0, 23, 2)) +
    ↪  facet_wrap(~Calendar_Month) +
    ggtitle("Daily Direct Normal Irradiance (DNI) Variation - Monthly Averages", s_year)
    ↪  +
    labs(x = "Hour", y = "DNI (w/m2) - Monthly Average")
```

```
## `summarise()` has grouped output by 'Month', 'Calendar_Month'. You can override using the
## `.groups` argument.
```



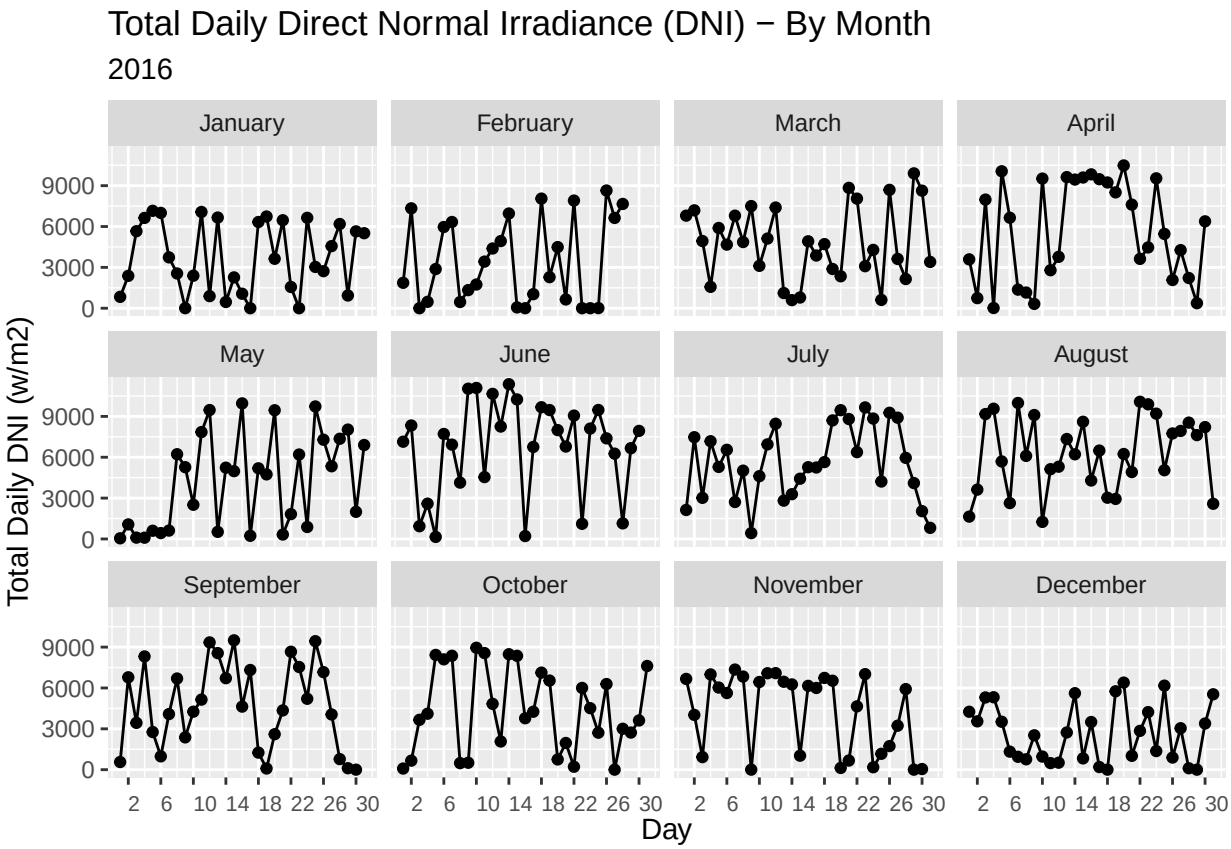
Difference in both the peaks and the daily duration of positive (non-zero) DNI values can be observed over the course of the year. This monthly-averaged, daily DNI variation over a year is what will be of further interest for reference comparison with the ML-algorithm generated forecasts.

The total daily DNI across each month is another interesting measure given its impact on the amount of

solar energy harvesting that can be achieved for deployed PV panels in the field. As seen, the accumulated daily totals are highest in the summer months and lowest in winter.

```
############ Plotting Daily DNI Totals

# Randomly selected year
s_year <- year(sample(solar_dataset$Date, 1))

solar_dataset %>%
    filter(Year == s_year) %>%
    mutate(Hour = hour(Date) + minute(Date)/60, Calendar_Month =
    ↪  factor(month.name[month(Date)],
        levels = month.name)) %>%
    group_by(Month, Calendar_Month, Day) %>%
    summarize(DNI_Daily_Total = sum(DNI)) %>%
    ggplot(aes(x = Day, y = DNI_Daily_Total, group = desc(Calendar_Month))) +
    ↪  geom_point() +
    geom_line() + scale_x_continuous(breaks = seq(2, 30, 4)) +
    ↪  facet_wrap(~Calendar_Month) +
    ggtitle("Total Daily Direct Normal Irradiance (DNI) - By Month", s_year) + labs(x =
    ↪  "Day",
    y = "Total Daily DNI (w/m2)") + theme(axis.text.x = element_text(angle = 0, vjust =
    ↪  0,
    hjust = 0, size = 8))
```

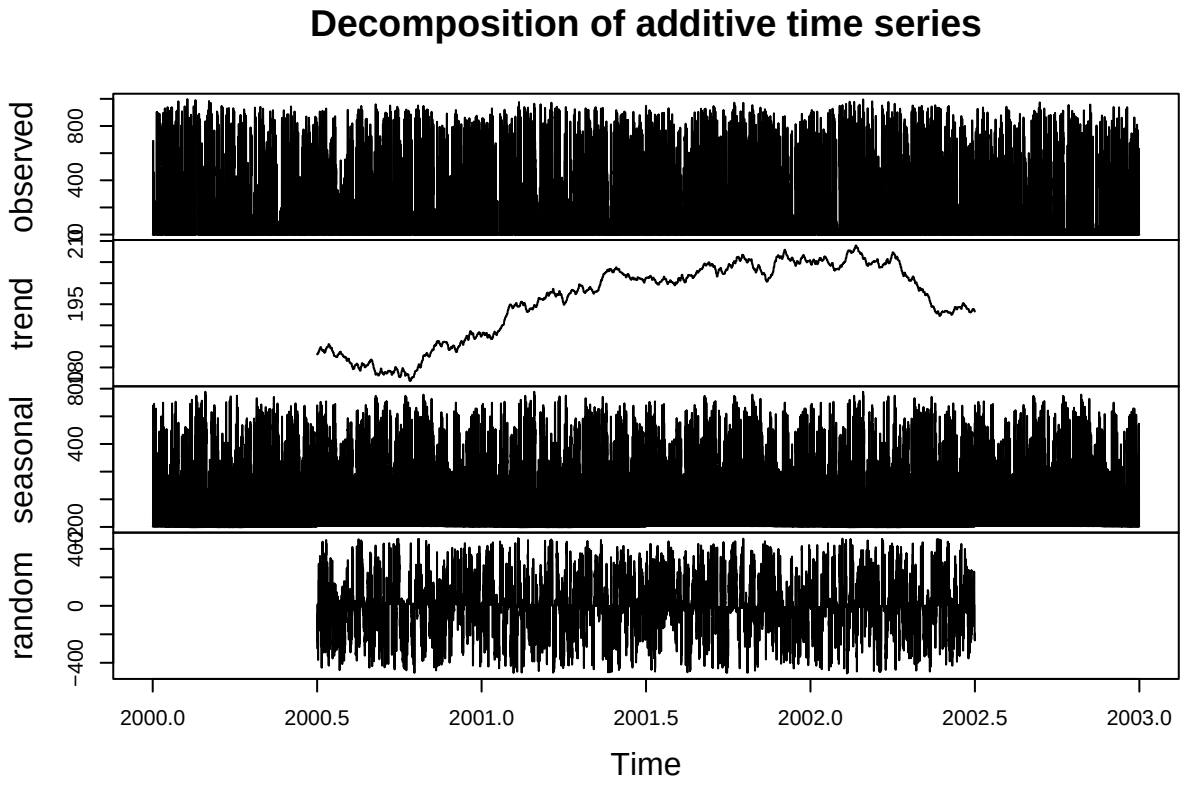
```
## `summarise()` has grouped output by 'Month', 'Calendar_Month'. You can override using the
## `.groups` argument.
```



These data plots again confirm the validity and consistency of the data set as no anomalies or unexpected data is observed. Daily DNI totals are higher in the months leading up to the June Summer Solstice with longer periods of daylight for those months (for the Northern hemisphere). The daily totals are correspondingly lowest in the Winter months (November - January).

Another useful measure of the data completeness and consistency can be seen in the generic decomposition of the data set. With time series data the decompose() function can be used to extract different cyclic trends in the data. Again, given the large quantity of data available, a subset of three years is taken.

```
############ Decomposing Trends and Cycles in the DNI Time Series Data

# Sub-setting first three years of data set dni time series data
dni_ts_3yr <- window(dni_ts, start = c(year(solar_dataset$Date[1]), 1), end =
↪  c(year(solar_dataset$Date[1]) +
    2, 8760))

# Apply decomposition function and plotting decomposed data components
dni_dcomp <- decompose(dni_ts_3yr)

plot(dni_dcomp)
```



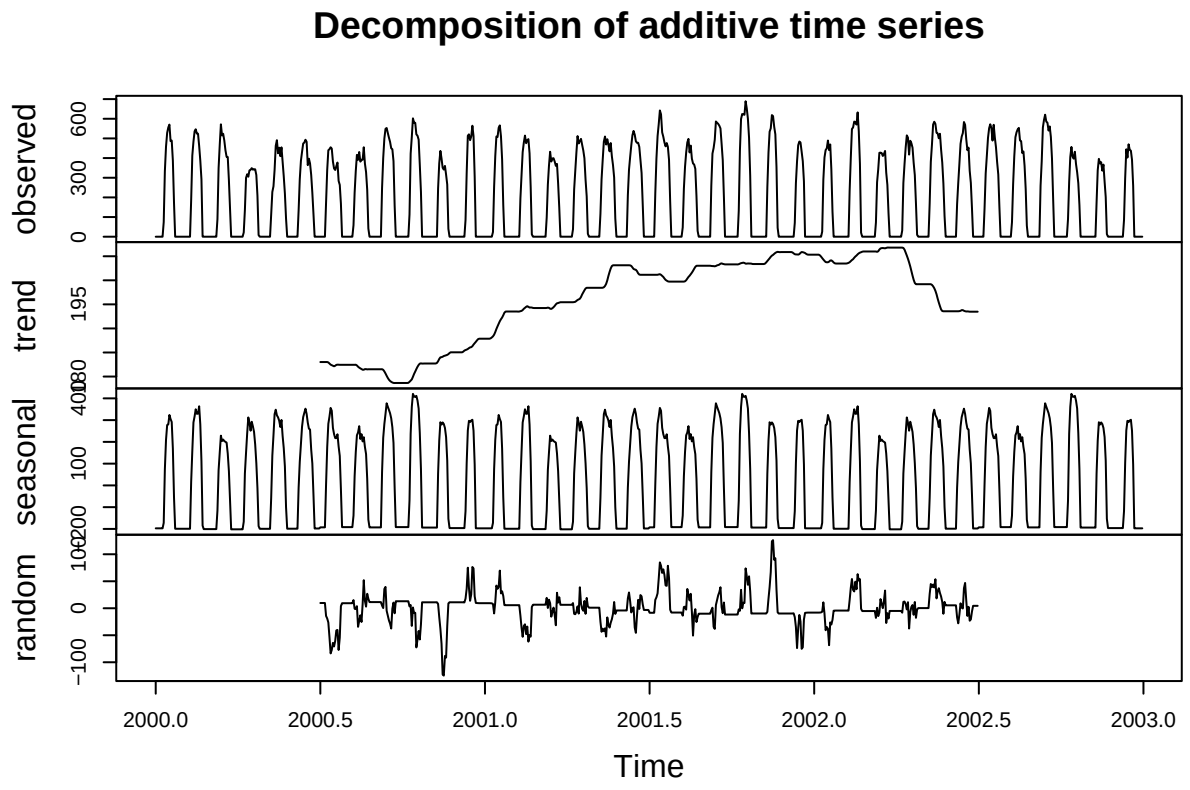
Other than the trend plot it is not possible to discern very much from this decomposition of the granular hourly data. To provide a bit more perspective we can instead look at the hourly DNI values each day on a monthly-averaged basis.

```
############ Decomposing Trends and Cycles in the DNI Hourly Values - Monthly Averaged
############ Time Series Data

# Defining 3-year period from first year (2000) in the data set
year_range <- seq(year(solar_dataset$Date[1]), year(solar_dataset$Date[1]) + 2)

# Filtering and calculating monthly average DNI values
dni_avgs_3yrs <- solar_dataset %>%
    filter(Year %in% year_range) %>%
    mutate(Year = year(Date), Hour = hour(Date) + minute(Date)/60, Calendar_Month =
    ↪  factor(month.name[month(Date)],
        levels = month.name)) %>%
    group_by(Year, Month, Calendar_Month, Hour) %>%
    summarize(DNI_Hourly_Avg = mean(DNI))
```

```
## `summarise()` has grouped output by 'Year', 'Month', 'Calendar_Month'. You can override
## using the `.groups` argument.
```

```
# Creating DNI hourly averaged time series with (24*12)= 288 samples per year Note the
# need to coerse group_tibble to tibble to ensure that time series object is obtained.
dni_avgs_ts <- as_tibble(dni_avgs_3yrs) %>%
    mutate(dni_avgs_ts = ts(DNI_Hourly_Avg, start = c(Year[1], 1), frequency = 288)) %>%
```

```
    pull(dni_avgs_ts)

# class(dni_avgs_ts) str(dni_avgs_ts)

# Apply decomposition function and plotting decomposed data components
dni_avgs_dcomp <- decompose(dni_avgs_ts)

plot(dni_avgs_dcomp)
```



The seasonal patterns and trends in the DNI data are more readily seen in the monthly average daily data. This offers a clearer visualization reference for use with algorithm forecast data comparisons. A similar visualization review of the ML model forecasts will also be later undertaken for comparison between actual measurement data and that generated by the developed prediction models.

Data Partitioning Data into Training and Testing Data Sets

With the prepared data set, the next step is to split it into training and testing data sets. For building the forecasting model the full input data set will be partitioned into initial training and testing data sets with the partitioned test set reserved exclusively for evaluation testing after the ML algorithm model has been selected, trained, and tuned.

With time series data and the univariate parameter forecasting that is being addressed, there is a need to preserve time continuity when creating data sets. Rather than the typical random selection for partitioning of the data set, some form of time-slicing is instead required. The approach using the caret package is discussed below. With time series data, time slicing approach is required for creating CV data sets that are made up of sequential training data of fixed duration immediately followed by sequential testing data also

of fixed duration. This generates the continuous time-series data needed for forecasting model development and evaluation.

It will also be desirable to apply Cross-validation (CV) in the model development and optimization by generating and utilizing training and testing data sets that are drawn entirely from within the initial partitioned training data set. As described below, the caretForecast package allows for automation of the CV implementation within the model evaluation.

```
############ Partitioning Data into Training and Testing Data Sets #############

# With the solar data DNI measurement already specified as a time series object, the
# window() function can be used to partition the data set.

# With hourly data, each year of data corresponds to 24*365 = 8760 hourly data samples
# (recall, no leap year data). Hence a 2-year validation testing set will comprise 2*8760
# = 17520 hourly DNI data points. The training data set will comprise 19*8760 = 166440
# hourly DNI data points. The training and testing data thus including the 21 years of
# data downloaded from the NREL NSRDB site.

# The training and evaluation testing data sets are given by the following:

test_set <- window(dni_ts, start = c(2020, 1))  # validation testing data set
length(test_set)
```

```
## [1] 8760
```

```
train_set <- window(dni_ts, start = c(2000, 1), end = c(2019, 8760))  # training data set
length(train_set)
```

```
## [1] 175200
```

```
# Additionally given that we know the validation test_set will be 26280 hourly DNI data
# samples, the caretForecast package can also be used to perform the time series data
# split. The validation test_set is automatically selected as the final consecutive data
# samples given by the specified test_size argument.  The training train_set is the
# remainder of the data set samples prior to the start of the test_set.

# Using caretForect data splits
datlists <- caretForecast::split_ts(dni_ts, test_size = 8760)

test_set <- datlists$test
length(test_set)
```

```
## [1] 8760
```

```
train_set <- datlists$train
length(train_set)
```

```
## [1] 175200
```

```r
# Both the test and train data sets retain their time series properties including time
# series frequency
head(test_set)
```

```
## Time Series:
## Start = c(2020, 1)
## End = c(2020, 6)
## Frequency = 8760
## [1] 0 0 0 0 0 0
```

```r
# Note: End is just the 6th sample of the year since head() only provides the top 6
# values of the data set

tail(train_set)
```

```
## Time Series:
## Start = c(2019, 8755)
## End = c(2019, 8760)
## Frequency = 8760
## [1] 0 0 0 0 0 0
```

```r
# Note: Start is the 8755th sample of the year since tail() only provides the bottom 6
# values of the data set

# Further verifying the data set split by combining the two time series
data_sets_ts <- ts(c(train_set, test_set), start = start(train_set), frequency =
↪  frequency(train_set))

# Comparing with the original ts data set
identical(data_sets_ts, dni_ts)
```

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

Reducing Analysis Data Set Size

Platform processing limits were encountered in working with increasing data sample sizes and the caret-Forecast package. With lengthened training and forecast data, the execution times extended to multiple hours and eventually resulting in error warnings being given, including, > "protection stack overflow" and "model fit failed for Training. . . result would exceed 2^31-1 bytes", clearly indicating memory exhaustion on the current laptop PC.

To continue the analysis with the limited processing capability of a standard laptop computer the solar radiation time series data is compressed to monthly daily averages. With this approach, rather than having (365x24) 8760 hourly DNI samples per year, there will just be (12*24) 288 hourly DNI samples per year with just 24 per-day, monthly-averaged hourly samples per-month. This reduces the data size by a factor of approximately 30 (the average number of days per month) while still maintaining the per-month character of the data measurements.

```r
# Calculating monthly averaged daily DNI values
dni_avgs <- solar_dataset %>%
    mutate(Year = year(Date), Hour = hour(Date) + minute(Date)/60, Calendar_Month =
    ↪  factor(month.name[month(Date)],
        levels = month.name)) %>%
    group_by(Year, Month, Calendar_Month, Hour) %>%
    summarize(DNI_Hourly_Avg = mean(DNI))
```

```
## `summarise()` has grouped output by 'Year', 'Month', 'Calendar_Month'. You can override
## using the `.groups` argument.
```

```r
# Creating DNI hourly averaged time series with (24*12)= 288 samples per year Note the
# need to coerce group_tibble to tibble to ensure that time series object is obtained.
dni_avgs_ts <- as_tibble(dni_avgs) %>%
    mutate(dni_avgs_ts = ts(DNI_Hourly_Avg, start = c(Year[1], 1), frequency = 288)) %>%
    pull(dni_avgs_ts)

class(dni_avgs_ts)
```

```
## [1] "ts"
```

```r
str(dni_avgs_ts)
```

```
##  Time-Series [1:6048] from 2000 to 2021: 0 0 0 0 0 ...
```

```r
############ Compressed Data Sets ################################################

test_set <- window(dni_avgs_ts, start = c(2019, 1))  # validation testing data set
length(test_set)
```

```
## [1] 576
```

```r
train_set <- window(dni_avgs_ts, start = c(2000, 1), end = c(2018, 288))  # training data
↪  set
length(train_set)
```

```
## [1] 5472
```

```r
# Training data set for input to forecasting model
training_data <- train_set

# Testing (holdout evaluation) data set for use in forecasting with generated algorithm
# models
testing_data <- test_set
```

Aggregating Time Series Data to Further Reduce Processing Time

While working with the monthly-averaged daily DNI data did allow for significant reduction in the time samples to be processed, the time taken for model execution was still significant. Further data reduction was therefore implemented to reduce the number of time series steps being processed by reducing the number of time interval measurements per day. This achieved a reduction in the number of measurement data points per day by combining consecutive hourly measurements. For example, to create time series of bi-hourly or tri-hourly aggregated DNI measurements.

The **'scaler'** parameter is introduced to allow more automated control of the compressing of the data used in the analysis. The scaler variable will control how many points in the daily time series measurements are aggregated from the original baseline of 24 hourly points per day (with scaler=1) to just one total DNI measurement per day (with scaler=24).

```
############ Further Compressed Data Set ########################################

scaler <- 2  # number of 1-hour periods per day that are aggregated (as sub-multiple of
↪  24)

# dni_comp_ts is the new multiple-hour aggregated time series (where per day hourly
# aggregation is added to the prior monthly averaged data)
dni_comp_ts <- aggregate(dni_avgs_ts, 288/scaler, sum)

# With the further data compression a 2-year test data set is defined
test_set <- window(dni_comp_ts, start = c(2019, 1))  # validation testing data set
length(test_set)
```

```
## [1] 288
```

```
# With the further data compression, the training set duration is further extended to use
# more of the available data set
train_set <- window(dni_comp_ts, start = c(2006, 1), end = c(2018, 288/scaler))  #
↪  training data set
length(train_set)
```

```
## [1] 1872
```

Model Evaluation and Analysis

As indicated in the introduction, the automated setup and cross-validation tuning capabilities of the caret-Forecast package is used as the primary tool for the analysis.

Even with the use of the automated caret package capabilities a random sampling seed is still set to provide repeatability during model code verification and review. Even as caret involves different randomization in its execution, the random seed initialization allows the same results to be obtained when the model setup and data conditions are repeated. That consistency is important for analysis comparisons and debugging as new elements are added or changes made in the selection and refinement of the algorithm models.

The analysis initially selects a single ML algorithm and uses the caretForecast package to generate an optimal model through cross-validation and tuning on the training data set. The derived model is then used to generate forecast values for the period specified. Performance evaluation against the provided validation test data set is automatically undertaken and provided as part of the result output. The caretForecast output includes a number of different error function measures, including RMSE and MAPE.

While the caretForecast package uses RMSE (and MAPE) as its default internal performance optimization metric we will extract that measure as well as independently calculate a Mean Absolute Daily Percentage Error (MADPE) that will provide a reference for the analysis.

The sections of the report below will show the setup and results for the DNI data forecasting using a single ML algorithm. In this case the caretForecast default "cubist" model is selected. This single model analysis will be followed by the introduction of code to evaluate multiple different selected ML algorithms and to compare the performance of the single ML algorithms with forecasts from ensemble models.

Forecasting Performance Metric

The reference algorithm performance metric that was intended to be used in the univariate time-series forecasting assessment is the Mean Absolute Percentage Error (MAPE). This percentage error provides an appropriate measure as it is metric-independent while offering an easily interpretible and indicative representation of how actual and forecast values vary over the duration for which forecasts are made. Similar to the Root Mean Square Error (RMSE) loss function, MAPE captures the mean of the absolute difference between actual and predicted/forecast values. However, being a percentage, it provides an intuitive indication of performance even without needing an appreciation for the magnitude of the involved measurement quantities.

A drawback of MAPE however is that like other metrics involving a percentage, it becomes infinite or indeterminate when the measurement values are zero (since it results in a divide by zero). Based on the known patterns of solar radiation being zero during non-daylight and extreme overcast conditions, the MAPE measure would be indeterminate. However, since we know that the desired forecasts should indeed mirror the daily patterns, a percentage measure defined on a daily basis would be appropriate to capture the percentage error representation without a zero value concern.

Additionally, the goal of solar radiation prediction is to ensure that there is adequate storage of harvested solar energy from the daily cumulative available radiation so that the system is able to operate during periods of low and no available sunlight. As such, being able to reference the percentage irradiation error in the forecast over the course of a day does provide a very useful summary metric for the amount of energy storage that is needed in a system.

For the project analysis therefore a percentage error measurement for assessing performance, called the Mean Absolute **Daily** Percentage Error (MADPE), is introduced. By using the daily total solar radiation measurement as the percentage base (denominator), this metric overcomes the issue of division by zero in calculating the percentage error.

For forecasts provided over a period of M days where there are N time measurements per day, the MADPE loss function is given by:

$$\frac{1}{M}\sum_{j=1}^{M}\frac{\sum_{i=1}^{N}|(\hat{y}_i - y_i)|}{\sum_{i=1}^{N} y_i} * 100\%$$

where $\hat{y}_i$ and y_i are the forecast and actual data values for each time period measurement respectively.

> Note: Because the MADPE measure is based on a percentage error per day, it localizes percentage error assessment over a day as opposed to over each measurement. Thus, unlike MAPE or the Root Mean Square Error (RMSE) where each (absolute) measurement error is equally weighted, with MADPE there is a wider percentage averaging window.

Since the caret package uses RMSE as the default metric for evaluating algorithm regression performance and for internal parameter optimization it is still therefore important to capture and reference the RMSE in the undertaken analysis.

```r
# Calculating MADPE - Mean Absolute Daily Percentage Error used as additional forecasting
# performance assessment reference

# This function takes as input the actual and the forecasts measurements and the number
# of measurement points per day given by the daily_points parameter (as specified for the
# defined time series data). In the case of hourly data, there are 24 daily measurement
# point (groups) per day.

############ Forecasting Performance - Daily Percentage Error Metric ###########

MADPE <- function(true_measurements, forecast_measurements, daily_points) {
    dat_df <- data.frame(actual = true_measurements, forecast = forecast_measurements)
    madpe <- dat_df %>%
        mutate(Abs_Error = abs(actual - forecast), DayNumber = (row_number() -
        ↪  1)%/%daily_points) %>%
        group_by(DayNumber) %>%
        summarize(adpe = 100 * sum(Abs_Error)/sum(actual)) %>%
        pull(adpe) %>%
        mean()
}

# For each day, the absolute value of the forecast error for each data point (period of
# the day) is calculated and the sum of these absolute errors used to derive the total
# absolute daily forecast error. The daily percentage error is the total daily forecast
# error divided by the total daily DNI measurement. The mean absolute daily percentage
# error for the forecast duration is the average of the absolute daily percentage errors.
```

Using caretForecast (Conformal Time Series Forecasting Using State of Art Machine Learning Algorithms)

The following sections show the application of the caretForecast time series forecasting package. The "cubist" model is the default model for the package.

Single Model Time Series Forcasting and Evaluation using CaretForecast

Given the range of parameters that must be tuned for each model, the execution time can be quite significant when CV is applied. To gauge approximate execution times for a given model, some initial exploratory testing was conducted with CV disabled. Since resampling is also automated as part of the caret cross validation, the duration of the total training data set also affects execution time since with time series data needing to be continuous, the CV samples are shifted as a function of the max lag time (which in this work is specified as 1 year - 288 hourly monthly-averaged samples for scaler = 1). This time shifting of resamples limits how many time-shifted resamples can be drawn from the available training data set.

> **Prior to running the code, check the 'scaler' parameter value that is set in accordance with expectations for code completion times!!**
>
> The presented analysis reduces both the data training period and the period of the available training data used by the caretForecast package to reduce the time taken for the analysis and report generation (which is still setup to run for several hours!). Reducing the training data size

was determined to be one way to limit the extent of the CV and tuning optimizations performed by the package.

```
############ caretForecast Single Model Forecasting Setup ######################

set.seed(1, sample.kind = "Rounding")

# Even with the use of the automated caret package capabilities a random sampling seed is
# still set to provide repeatability during model code debugging verification and review

training_data <- train_set
testing_data <- test_set

# This print code provides visual feedback and time logging for the on-going analysis
# code execution
print(paste0(" Initiating Model Evaluation: Train_set data size : ",
↪  length(training_data),
    " data points"))
```

```
## [1] " Initiating Model Evaluation: Train_set data size : 1872 data points"
```

```
start_time <- Sys.time()
print(paste0("Model Evaluation Start time: ", Sys.time()))
```

```
## [1] "Model Evaluation Start time: 2023-09-14 12:58:26.052922"
```

```
# Using caretForecast (Conformal Time Series Forecasting Using State of Art Machine
# Learning Algorithms)

daily_points <- 24/scaler
# 'scaler' parameter determines the number of hourly measurements that are combined

lag <- 288/scaler
# 288 corresponds to a 1-year period of 12 months x 24 monthly-averaged hourly points per
# day profile

window <- 1728/scaler
# 1728 (288*6) corresponds to specified 6-year model training period (per resample) for
# 2-year forecasts

horizon <- 576/scaler  # 576 corresponds to a 2-year (monthly-averaged data) test period
↪  for CV samples

# Testing execution time without CV
fit_cub <- ARml(training_data, max_lag = lag, initial_window = window, caret_method =
↪  "cubist",
    cv = TRUE, cv_horizon = horizon, fixed_window = TRUE, metric = "RMSE", verbose =
    ↪  FALSE)
```

Note: The max_lag argument is the maximum period over which data repetition cycles occur. Since the solar data has annual cycles, that corresponds to a max_lag of 24x365 = 8760 hourly data points (using the specified NSRDB download API to explicitly remove leap year data).

The fixed_window=TRUE uses fixed size training-testing data sets for cross-validation (cv=TRUE)

The initial_window argument is the number of consecutive values in each training set sample. Since the goal of the analysis is to train on fixed 8-year training data, that corresponds to 6x24x365 = 52560 hourly values (before monthly averaging compression).

The cv_horizon argument is the number of consecutive values in the cv test set sample. Since the analysis goal is to perform 2-year forecasts, that corresponds to 2x24x365 = 17520 hourly values (before monthly averaging compression).

The metric argument specifies what summary metric will be used to select the optimal model. By default, possible values are "RMSE" and "Rsquared" for regression.

Note: For configured cross-validation with fixed window=TRUE, the following relationship must hold: length(training_data) - length(initial_window), MUST be an integer of (max_lag)!!

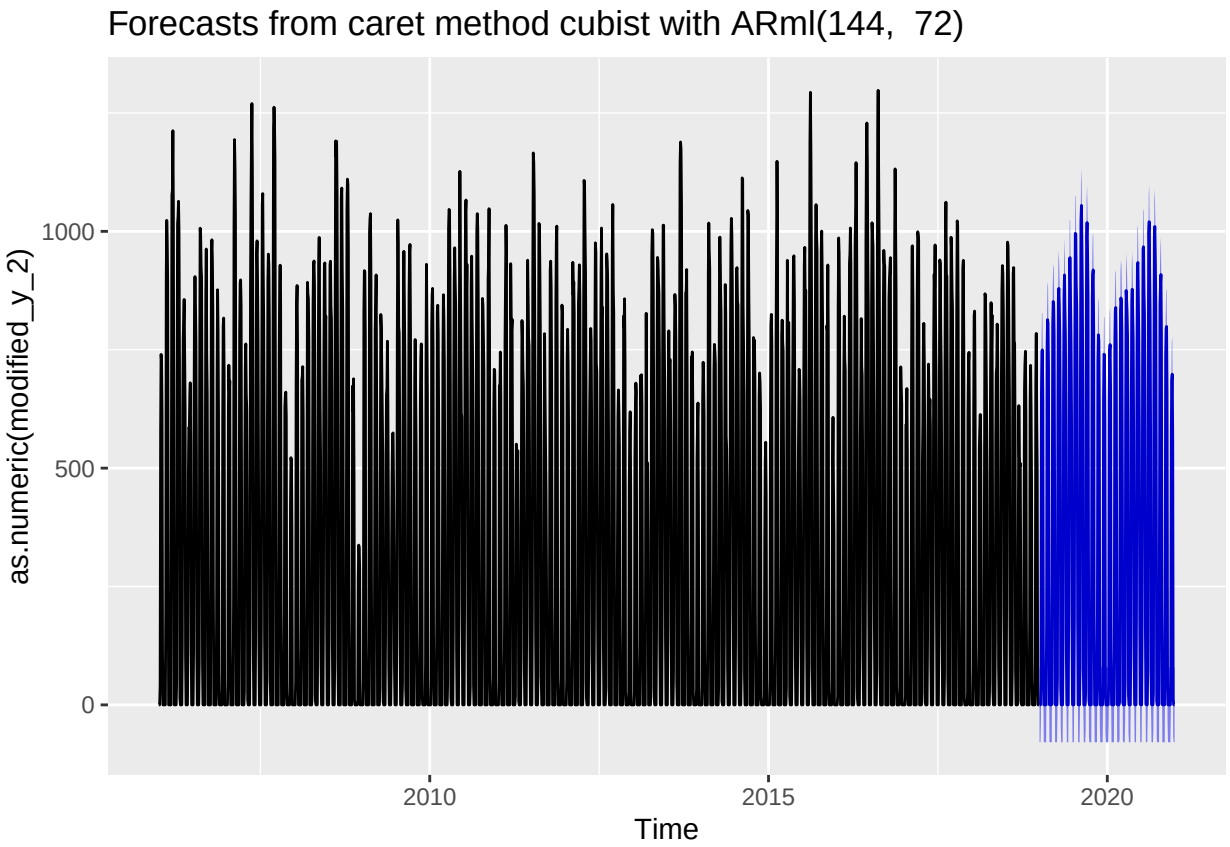
The single model analysis applies the "cubist" method, a rule-based prediction regression model that is an extension of Quinlan's M5 model tree https://cran.r-project.org/web/packages/Cubist/Cubist.pdf.

With the above given setup and the respective training and testing data, the forecast values, fc and cv-tuned accuracy measures are as given below. The forecast values, *fc_cub*, are the median values and the level argument allows upper and lower confidence interval values to be provided as part of the data output.

```
############ caretForecast Single Model Forecast Results ######################

# Using the single model generated data to derive DNI bi-hourly forecast
forecast(fit_cub, h = length(testing_data), level = 95) -> fc

# Forecast performance for caretForecast CV-optimized model
accuracy(fc, testing_data) %>%
    kable()
```

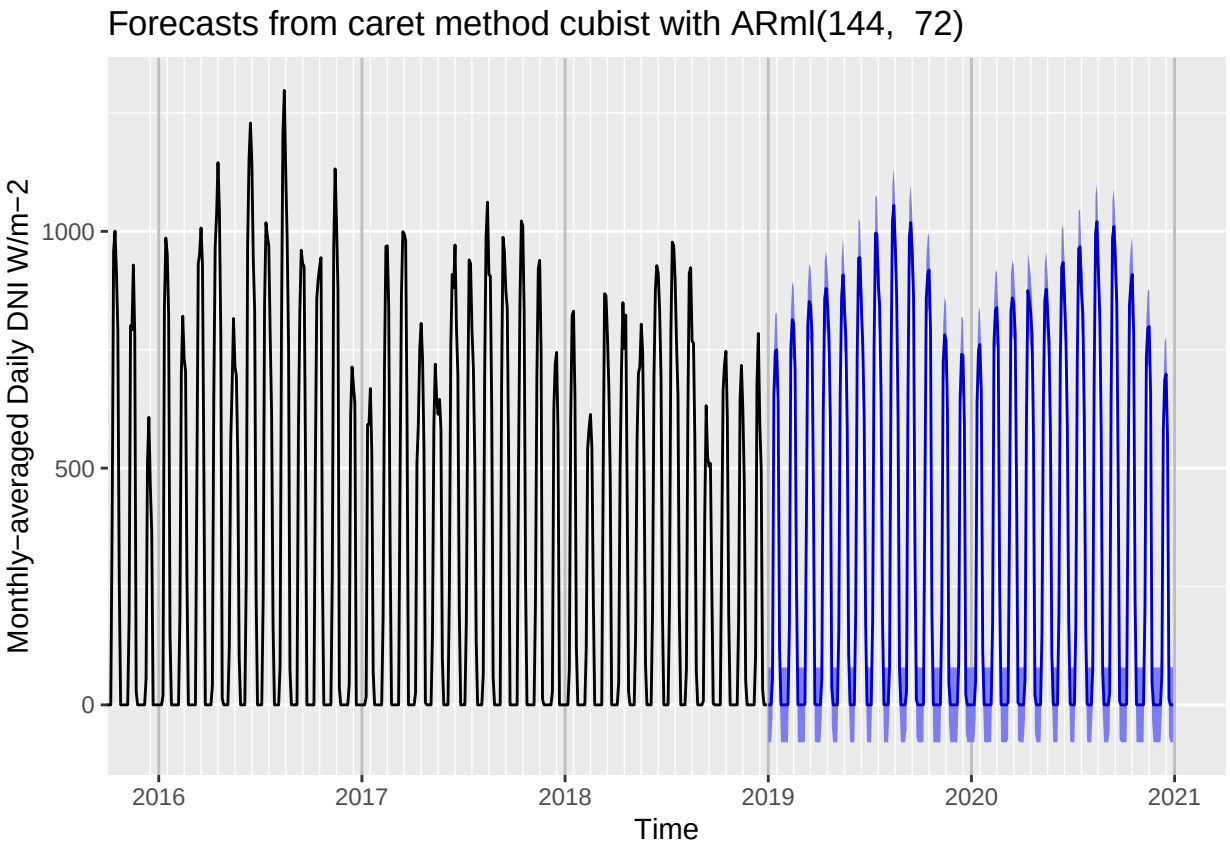
	ME	RMSE	MAE	MPE	MAPE	MASE	ACF1	Theil's U
Training set	-0.3048608	34.15554	19.13236	-Inf	Inf	0.2705440	0.0219332	NA
Test set	5.4330285	79.20831	46.73731	-Inf	Inf	0.6608957	0.6666525	0

```
# Note: MPE and MAPE measures are -Inf and Inf respectively (for cases where scaler <=
# 6). As discussed, this is the result of zero values and hence the occurrence of zero
# value denominators in the measurement percentage calculation.

# Output plot of training and associated forecast data. Note the automated generation of
# confidence intervals on the forecast plots based on the 95% levels specified.
ggplot2::autoplot(fc)
```



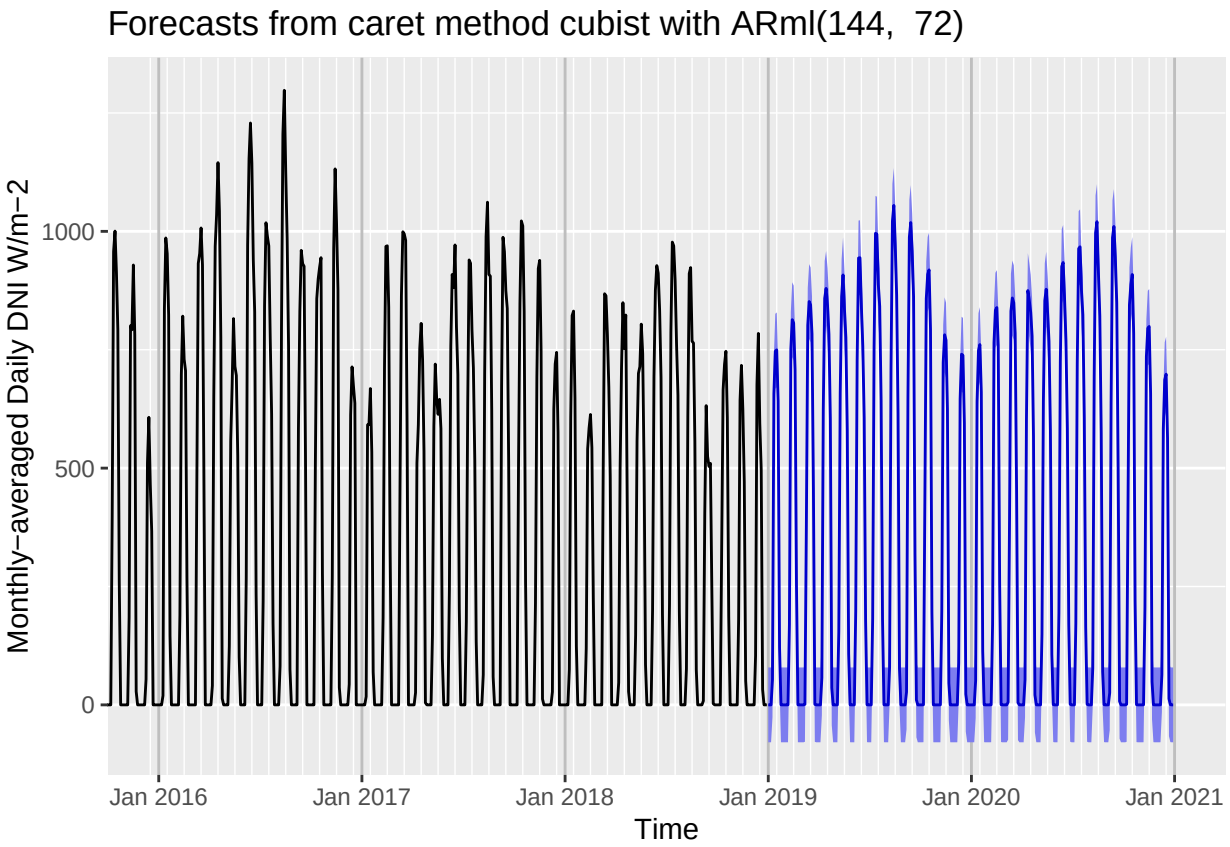

```
ggplot2::autoplot(fc) + labs(y = "Monthly-averaged Daily DNI W/m-2") +
↪  coord_cartesian(xlim = c(2016,
    2021)) + scale_x_continuous(breaks = c(seq(2016, 2021)), minor_breaks = c(seq(2016,
    ↪  2021,
    by = 1/12) - 1/24)) + theme(panel.grid.major.x = element_line(color = "grey",
    ↪  linewidth = 0.5))
```

```
## Scale for x is already present.
## Adding another scale for x, which will replace the existing scale.
```




```
# Note: Decimal time can be converted to year month using as.yearmon(time(fc$x))

ggplot2::autoplot(fc) + labs(y = "Monthly-averaged Daily DNI W/m-2") +
↪  coord_cartesian(xlim = c(2016,
    2021)) + scale_x_continuous(breaks = c(seq(2016, 2021)), minor_breaks = c(seq(2016,
    ↪  2021,
    by = 1/12) - 1/24), labels = as.yearmon(seq(2016, 2021))) + theme(panel.grid.major.x
    ↪  = element_line(color = "grey",
    linewidth = 0.5))
```

```
## Scale for x is already present.
## Adding another scale for x, which will replace the existing scale.
```




```
# The Mean Absolute Daily Percentage Error MADPE measure can be calculated based on the
# defined MADPE function
madpe_accuracy <- data.frame(MADPE = MADPE(testing_data, fc$mean, daily_points))
rownames(madpe_accuracy) <- c("Test set")

madpe_accuracy %>%
    kable()
```

	MADPE
Test set	13.41322

```
# Note that with the twice hourly DNI values (averaged per month), the MADPE is
# calculated over 24/'scaler' bi-hourly points per day

# The result is therefore a DNI daily average percentage error of
print(paste0(round(madpe_accuracy[1, 1], 2), " %"))
```

```
## [1] "13.41 %"
```

The output forecast appears to show good correspondence with the patterns of the training data on which the forecasts were based.

Analysis of Ensemble Models for DNI Forecasting

With limited time to delve deeply into understanding and finding the best ML algorithms for time series data, a number of the popular regression models mentioned in the literature were selected for the forecasting evaluation and to determine whether an ensemble model could be created that could provide even better performance. The models eventually selected and applied were chosen based on their availability within caret's built-in library and their ability to execute relatively quickly when tested on a small data set.

The number of different caret ML algorithms did also allow some exploration and experimentation of different options when searching for an algorithm to be used for combining the output of the individual model to create an ensemble regression model.

```
set.seed(1, sample.kind = "Rounding")

training_data <- train_set
testing_data <- test_set

models <- c("cubist", "glm", "ranger", "svmLinear2", "ridge")

fcts <- lapply(models, function(model) {

    # This print code provides visual feedback and time logging for the on-going analysis
    # code execution
    print(paste0("Initiating Model Evaluation: Train_set data size : ",
    ↪  length(training_data),
        " data points"))
    start_time <- Sys.time()
    print(paste0("Model Evaluation Start time: ", Sys.time()))

    print(model)  # visual feedback while running

    # Since the 'cubist' model has been previously run, the ifelse() statement allows the
    # prior 'fit_cub' model data to be used when that model is called

    ifelse(model == "cubist", fit <- fit_cub, fit <- ARml(training_data, max_lag = lag,
    ↪  initial_window = window,
        caret_method = model, cv = TRUE, cv_horizon = horizon, fixed_window = TRUE,
        ↪  verbose = FALSE,
        trace = FALSE))

    forecast(fit, h = length(testing_data)) -> fc

    # Extracting forecast values
    dni_fct <- fc$mean

    # Extracting RMSE from the different output accuracy performance measures
    rmse_fct <- accuracy(fc, testing_data)["Test set", "RMSE"]

    # Calculating the project defined MADPE (mean absolute daily percentage error)
    madpe_fct <- MADPE(testing_data, fc$mean, daily_points)

    # Recording and displaying execution run time on-screen for visual confirmation and
    # processing time benchmarking when setting up different models and evaluation.
    stop_time <- Sys.time()
```

```
    exec_time <- difftime(stop_time, start_time, units = "mins")
    print(paste0("Model Evaluation executiion time: ", round(exec_time, 1), " minutes"))
    cat("  \n")

    return(list(dni_fct, rmse_fct, madpe_fct))
})
```

```
## [1] "Initiating Model Evaluation: Train_set data size : 1872 data points"
## [1] "Model Evaluation Start time: 2023-09-14 14:14:59.403654"
## [1] "cubist"
## [1] "Model Evaluation executiion time: 0.2 minutes"
##
## [1] "Initiating Model Evaluation: Train_set data size : 1872 data points"
## [1] "Model Evaluation Start time: 2023-09-14 14:15:11.175821"
## [1] "glm"
## [1] "Model Evaluation executiion time: 0.8 minutes"
##
## [1] "Initiating Model Evaluation: Train_set data size : 1872 data points"
## [1] "Model Evaluation Start time: 2023-09-14 14:16:01.902965"
## [1] "ranger"
## [1] "Model Evaluation executiion time: 36.3 minutes"
##
## [1] "Initiating Model Evaluation: Train_set data size : 1872 data points"
## [1] "Model Evaluation Start time: 2023-09-14 14:52:17.490637"
## [1] "svmLinear2"
## [1] "Model Evaluation executiion time: 15.4 minutes"
##
## [1] "Initiating Model Evaluation: Train_set data size : 1872 data points"
## [1] "Model Evaluation Start time: 2023-09-14 15:07:39.875398"
## [1] "ridge"
## [1] "Model Evaluation executiion time: 24.3 minutes"
##
```

```
# Set names for forecasts output by model used
names(fcts) <- models

# Verify forecasts output results
str(fcts)
```

```
## List of 5
##  $ cubist    :List of 3
##   ..$ : Time-Series [1:288] from 2019 to 2021: 0.00 0.00 5.48e-03 5.30e+01 6.57e+02 ...
##   ..$ : num 79.2
##   ..$ : num 13.4
##  $ glm       :List of 3
##   ..$ : Time-Series [1:288] from 2019 to 2021: -1.66 7.61 7.11 19.41 648.22 ...
##   .. ..- attr(*, "names")= chr [1:288] "1" "1" "1" "1" ...
##   ..$ : num 81.4
##   ..$ : num 14.3
##  $ ranger    :List of 3
##   ..$ : Time-Series [1:288] from 2019 to 2021: 0 0 0 55.6 658.7 ...
##   ..$ : num 81.8
```

```
##   ..$ : num 13.2
##  $ svmLinear2:List of 3
##   ..$ : Time-Series [1:288] from 2019 to 2021: 9.54 31.15 37.12 37.25 651.25 ...
##   .. ..- attr(*, "names")= chr [1:288] "1" "1" "1" "1" ...
##   ..$ : num 84.8
##   ..$ : num 16.2
##  $ ridge     :List of 3
##   ..$ : Time-Series [1:288] from 2019 to 2021: -1.15 7.86 6.83 18.64 645.91 ...
##   .. ..- attr(*, "names")= chr [1:288] "349" "349" "349" "349" ...
##   ..$ : num 81.4
##   ..$ : num 14.3
```

```
# fcts[1]

# Extracting the lists of forecast dni values
dni_fcts <- sapply(fcts, function(n) {
    n[[1]]
})

# Extracting the lists of RMSE performance metric for each forecast by model used
rmse_fcts <- sapply(fcts, function(n) {
    n[[2]]
})

# Extracting the lists of MADPE performance metric for each forecast by model used
madpe_fcts <- sapply(fcts, function(n) {
    n[[3]]
})

# Creating data frames of dni forecasts by model and MADPE by model
dni_fcts_df <- data.frame(dni_fcts)
head(dni_fcts_df)
```

```
##         cubist        glm    ranger svmLinear2      ridge
## 1 0.000000e+00  -1.660881   0.00000   9.543477  -1.149808
## 2 0.000000e+00   7.608140   0.00000  31.152271   7.861836
## 3 5.481667e-03   7.111795   0.00000  37.117399   6.826194
## 4 5.304837e+01  19.407786  55.61906  37.247111  18.636689
## 5 6.571526e+02 648.215033 658.72607 651.252881 645.913382
## 6 7.468364e+02 721.381908 719.05496 735.901387 718.927474
```

```
madpe_fcts_df <- data.frame(MADPE = madpe_fcts)
madpe_fcts_df
```

```
##               MADPE
## cubist     13.41322
## glm        14.31552
## ranger     13.16627
## svmLinear2 16.24874
## ridge      14.32536
```

```
############ Output Results of applied single ML Models ########################

results_fcts_df <- data.frame(model = rownames(madpe_fcts_df), RMSE = rmse_fcts, MADPE =
↪  madpe_fcts)
rownames(results_fcts_df) <- seq(nrow(madpe_fcts_df))

# MADPE-ordered results on ML models
results_fcts_df %>%
    arrange(MADPE) %>%
    kable()
```

model	RMSE	MADPE
ranger	81.78639	13.16627
cubist	79.20831	13.41322
glm	81.42761	14.31552
ridge	81.36671	14.32536
svmLinear2	84.77879	16.24874

The results show that the MADPE performance metric values tracks generally with that of the RMSE but do not align exactly in terms of performance ordering. As previously discussed this is a function of the fact that MADPE is being calculated on a per-day basis while RMSE weighs all errors uniformly.

```
############ Removal of Lower Performing Forecast Models ########################

# Implement a MADPE performance cutoff to keep the better performing forecasting
# algorithms
filtered_madpe_fcts_df <- results_fcts_df %>%
    filter(MADPE < 20)  # This 20% MADPE value was based arbitrarily chosen to remove
    ↪  appreciably higher MADPEs

# Results for models with MADPE better (lower) than 20% daily average percentage error
# (Table is only output if there have been models removed)

if (nrow(filtered_madpe_fcts_df) < nrow(results_fcts_df)) {
    filtered_madpe_fcts_df %>%
        arrange(MADPE) %>%
        kable()
}

# Identifying any removed models
unfiltered_models <- intersect(results_fcts_df$model, filtered_madpe_fcts_df$model)

# Filtering the dni forecasts data frame based on common (un-removed) models
filtered_dni_fcts_df <- dni_fcts_df[c(unfiltered_models)]
```

Creating Ensemble Models

Two types of ensemble models are created. The first is a combined average of the previous ML (sub) model results where an ML model is used to combine the individual regression results. Many choices exist for the combining ML model used. Testing of smaller sample iterations used to find select a model that provides a good ensemble performance.

Note: the search for a "best" model has not been exhaustive and instead simply compares the execution time and MADPE performance for a number of regression model candidates.

The second approach to creating an ensemble model is through ensemble averaging. In this ensemble model, the mean, median, and a weighted (MADPE) average is used to create the average ensemble models.

Creating ML-Ensemble Model based on Regression Combining of sub-model Forecasts

A combined ensemble model is created by regression combining the sub-model forecast results to produce a fitted forecast. The MADPE measure for this combined ensemble forecast DNI is assessed and later also compared together other sub-models and ensemble models.

```
# The time series time variable is explicitly added to the data frame of sub-model
# forecasts and the data long-pivoted.  Since all model forecasts are based on the same
# time series range, time is from the first model in the forecast list 'fcts'.

dni_ml_ens_df <- dni_fcts_df %>%
    mutate(time = as.numeric(time(fcts[[1]][[1]])))

# Pivoting longer to create combined data frame of sub-model values versus time
ens_data <- dni_ml_ens_df %>%
    pivot_longer(-time, names_to = "model", values_to = "dni_ml_ens")

# Using 'random forest' to combine the sub-model forecasts into ML-ensemble forecast
# Note: This has not been a researched choice but simply based on results obtained from
# small sample data testing
ens_fit <- ens_data %>%
    train(dni_ml_ens ~ time, data = ., method = "rf")

# Creating ML-ensemble forecast data vector
dni_ensCombine <- as.vector(predict(ens_fit, newdata = dni_ml_ens_df))

# Verifying output result of ML-ensemble forecast head(dni_ensCombine)
str(dni_ensCombine)
```

```
##  num [1:288] 1.42 9.26 10.77 36.71 645.01 ...
```

Creating Average Ensemble Models

For the average ensemble, three different models are derived using the forecast results of the individual sub-models which are combined to create simple average, a median, and a MADPE-weighted average. For the MADPE-weighted average the forecast MADPE is normalized relative to the best performing MADPE forecast (i.e, the one with the lowest MAPE). This weighted average is only expected to produce a better ensemble model forecast only if there is some absence of correlation between the sub-model results that are exploited by the ensemble combining.

```
# Weighted Average Ensemble

# An inverse MADPE weighting metric is created where the weight for a forecast is an
# inverse function of its MAPE performance value. The weighting factor is relative to the
# best (lowest) performing model such that models with lower MAPE (lower forecast %
```

```
# error) are assigned a higher weighting factor.

# Calculating MADPE-based weighting factors Note: weights are based on the MADPE-filtered
# models (but ordered alphabetically)
ens_fctWeights_df <- filtered_madpe_fcts_df %>%
    mutate(inv_madpe_wght = ifelse(MADPE == 0, 1, min(MADPE)/MADPE), wghts =
    ↪  inv_madpe_wght/sum(inv_madpe_wght)) %>%
    arrange(model) %>%
    select(model, MADPE, weights = wghts)

# > Note: The weighting formulation assigns an inverse weighting of 1 if a model has a 0
# MADPE (zero error, perfect forecast). This results in a 0 weighting for any other
# non-zero MAPE models (and results in an equal weighting if there are multiple models
# with perfect forecast).

# Verifying weight and sum of weights Ensemble MADPE weights (note the ordering is not
# changed)
ens_fctWeights_df %>%
    arrange(MADPE) %>%
    kable()
```

model	MADPE	weights
ranger	13.16627	0.2159505
cubist	13.41322	0.2119746
glm	14.31552	0.1986140
ridge	14.32536	0.1984775
svmLinear2	16.24874	0.1749835

```
# Ensemble Weights vector
ens_fctWeights <- ens_fctWeights_df$weights
length(ens_fctWeights)
```

```
## [1] 5
```

```
# Sum of Ensemble Weights (equal to unity)
sum(ens_fctWeights)
```

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

```
# Average Ensemble Forecasts (Ensemble Average, Ensemble Median, and Ensemble Weighted
# Mean)
dni_ensMean <- apply(filtered_dni_fcts_df, 1, mean)
dni_ensMedian <- apply(filtered_dni_fcts_df, 1, median)

# For weighted ensemble calculation it is essential that the order of the model weights
# are the same as the filtered_dni_fcts_df data frame columns. Alphabetic ordering is
# used to match the imposed ordering used for the ensemble weights data frame
# ens_fctWeights_df
filtered_dni_fcts_df <- filtered_dni_fcts_df[, order(names(filtered_dni_fcts_df))]
```

```r
dni_ensWeightMean <- apply(filtered_dni_fcts_df, 1, weighted.mean, ens_fctWeights)

# Verifying Simple Mean Ensemble Average model
str(dni_ensMean)
```

```
##  num [1:288] 1.35 9.32 10.21 36.79 652.25 ...
```

```r
# head(dni_ensMean)

# Obtaining DNI measurements from testing data ts object for use in calculating MADPE for
# ensemble models
dni_actual <- as.vector(testing_data)
# dni_actual

# Creating list of the All Ensemble DNI Forecasts (including Average Ensemble and
# Combined ML Regression Ensemble)
dni_ens_fcts <- list(ensMean = dni_ensMean, ensMedian = dni_ensMedian, ensWeightMean =
↪  dni_ensWeightMean,
    ensCombine_rf = dni_ensCombine)
# dni_ens_fcts

# Ensemble Forecasts data frame
dni_ens_fcts_df <- as.data.frame(dni_ens_fcts)
head(dni_ens_fcts_df)
```

```
##      ensMean  ensMedian ensWeightMean ensCombine_rf
## 1   1.346558   0.000000      1.111866      1.424573
## 2   9.324449   7.608140      8.522614      9.264222
## 3  10.212174   6.826194      9.263442     10.768192
## 4  36.791803  37.247111     37.327118     36.710686
## 5 652.252003 651.252881    652.454191    645.012365
## 6 728.420419 721.381908    728.328619    727.671040
```

Performance Results of single ML and Ensemble models

To compare the performance of the different ensemble models, their mean absolute daily percentage error (MADPE) and RMSE values are calculated. These results are then combined with those of the single ML models to allow for an overall performance evaluation review.

```r
############ Combined models and RMSE-MADPE performance results ################

# Combining all dni forecasts (individual ML model DNI forecasts and Ensemble model DNI
# forecasts) into single data frame
dni_all_fcts_df <- cbind(dni_fcts_df, dni_ens_fcts_df)
head(dni_all_fcts_df)
```

```
##         cubist        glm    ranger svmLinear2      ridge    ensMean  ensMedian
## 1 0.000000e+00  -1.660881   0.00000   9.543477  -1.149808   1.346558   0.000000
## 2 0.000000e+00   7.608140   0.00000  31.152271   7.861836   9.324449   7.608140
```

```
## 3 5.481667e-03   7.111795   0.00000  37.117399   6.826194  10.212174   6.826194
## 4 5.304837e+01  19.407786  55.61906  37.247111  18.636689  36.791803  37.247111
## 5 6.571526e+02 648.215033 658.72607 651.252881 645.913382 652.252003 651.252881
## 6 7.468364e+02 721.381908 719.05496 735.901387 718.927474 728.420419 721.381908
##   ensWeightMean ensCombine_rf
## 1      1.111866      1.424573
## 2      8.522614      9.264222
## 3      9.263442     10.768192
## 4     37.327118     36.710686
## 5    652.454191    645.012365
## 6    728.328619    727.671040
```

```r
# MADPE function is applied to calculate the MADPE for Ensemble DNI Forecasts

# Calculating MADPEs for Ensemble Model DNI Forecasts - output list
madpes_ens_lst <- lapply(dni_ens_fcts, FUN = MADPE, true_measurements = dni_actual,
↪  daily_points = daily_points)
# madpes_ens_lst

# RMSE function is applied to calculate the RMSE values for Ensemble DNI Forecasts
# Standard R RMSE <- function(pred, obs), a function of the predictions and observations
# (actuals)

# Calculating RMSEs for Ensemble Model DNI Forecasts - output list
rmses_ens_lst <- lapply(dni_ens_fcts, FUN = RMSE, obs = dni_actual)
# rmses_ens_lst

# Ensemble Model Forecast MAPEs and RMSEs assessment results data frame
rmses_ens_fcts <- t(as.data.frame(rmses_ens_lst))
# rmses_ens_fcts

madpes_ens_fcts <- t(as.data.frame(madpes_ens_lst))
# madpes_ens_fcts

# Combining and ordering model results ML-algorithm models and Ensemble Forecast results
# data frame

results_ens_fcts <- data.frame(rmses_ens_fcts, madpes_ens_fcts)
colnames(results_ens_fcts) <- c("RMSE", "MADPE")
# results_ens_fcts

results_ens_fcts_df <- results_ens_fcts %>%
    mutate(model = rownames(.)) %>%
    as_tibble() %>%
    select(model = model, RMSE, MADPE)
# results_ens_fcts_df

# Combined Single ML-Models and Ensemble Models RMSEs and MADPEs
all_results_df <- rbind(results_fcts_df, results_ens_fcts_df)
# all_results_df

# Combined and MADPE-Ordered results
madpe_all_results_df <- all_results_df %>%
    arrange(MADPE) %>%
```

```
    select(model = model, MADPE, RMSE)

madpe_all_results_df %>%
    kable()
```

model	MADPE	RMSE
ranger	13.16627	81.78639
cubist	13.41322	79.20831
ensWeightMean	13.45215	78.61856
ensMean	13.52754	78.79729
ensCombine_rf	13.54290	78.91172
ensMedian	13.57796	79.69502
glm	14.31552	81.42761
ridge	14.32536	81.36671
svmLinear2	16.24874	84.77879

On the basis of the defined Mean Average Daily Percentage Error (MADPE) the ranger model proves to be the superior forecast model even against random-forest ensemble combining of the forecasts of other models, which however provides similar performance! Other than the ranger model, the ensemble model provides improved performance relative to other non-ensemble models.

The table shows that the average ensemble median and MADPE-weighted mean is also one of the higher-performing models. The best performing models for forecasting are not any of the ensemble models but instead the single cubist and ranger models. It is interesting that simple average ensemble models do also provide improved forecasting performance relative the single ML models.

> Note: A key caveat is that the caretForecast package is optimizing and tuning on the basis of RMSE performance so it is possible that the single ML models are not as optimized for the MADPE metric as the MADPE-ensemble. Certainly it is possible to control the tuning of the caretForecasting to be MADPE aligned as part of a future study effort.

```
# Combined and RMSE-Ordered results
rmse_all_results_df <- all_results_df %>%
    arrange(RMSE) %>%
    select(model = model, RMSE, MADPE)

rmse_all_results_df %>%
    kable()
```

model	RMSE	MADPE
ensWeightMean	78.61856	13.45215
ensMean	78.79729	13.52754
ensCombine_rf	78.91172	13.54290
cubist	79.20831	13.41322
ensMedian	79.69502	13.57796
ridge	81.36671	14.32536
glm	81.42761	14.31552
ranger	81.78639	13.16627
svmLinear2	84.77879	16.24874

On the basis of the standard RMSE performance measure, where all measurement errors are equally average, the MADPE-weighted ensemble mean provides the best forecast performance. This is followed by caret ML cubist model. Other than the cubist model, the different ensemble models all delivered better forecast performance than the single ML algorithms. This reinforces the value of applying ensemble models.

Using Best MADPE-Performing Forecast for Revisiting Exploratory Review

As had been done with the earlier initial DNI data exploration, the same is applied for the forecasts made with the best performing ML algorithm.

```
# Reviewing Best Forecasting Model

# Best forecasting model based on MADPE results
best_model <- madpe_all_results_df$model[1]

############ Identified 'Best' model ##############
best_model
```

```
## [1] "ranger"
```

```
# DNI forecast for 'best' model
dni_best_fct <- dni_all_fcts_df[, best_model]
head(dni_best_fct)
```

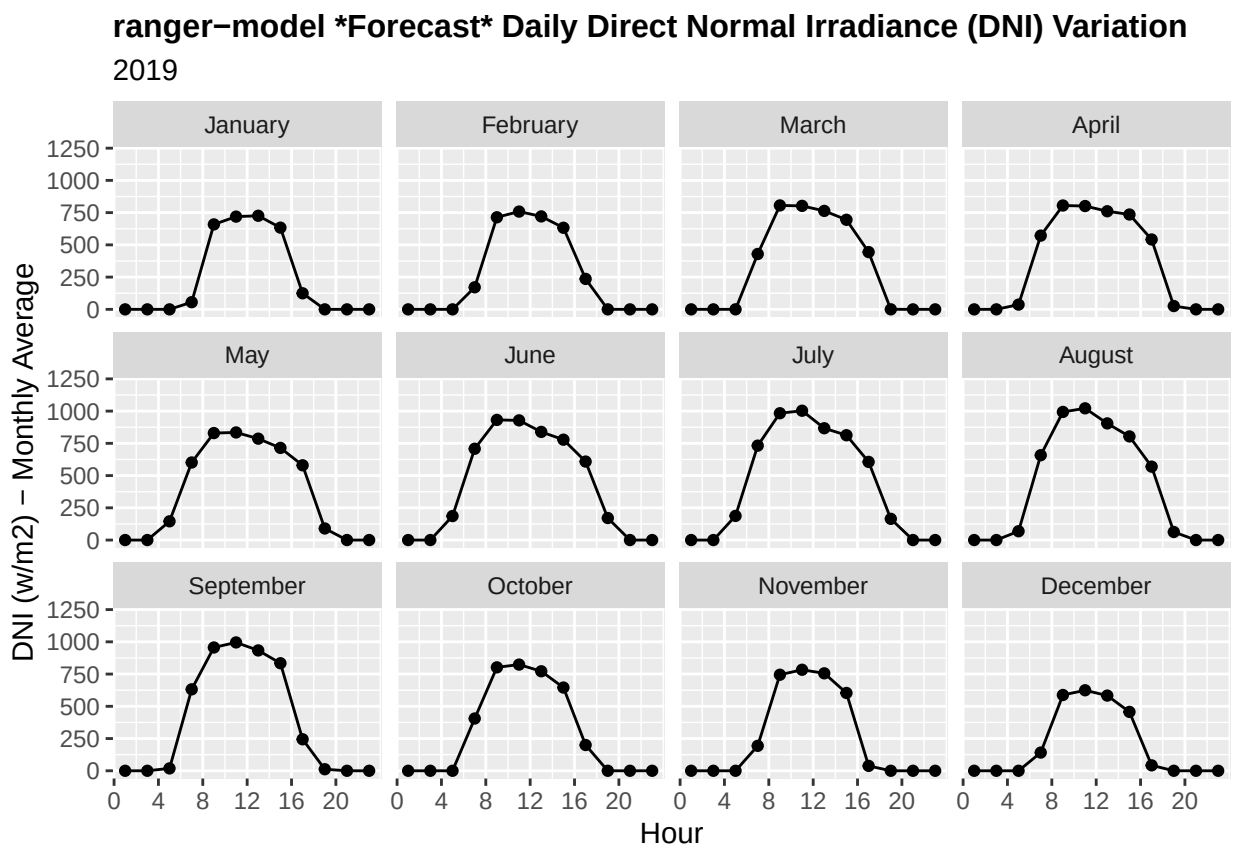
```
## [1]   0.00000   0.00000   0.00000  55.61906 658.72607 719.05496
```

```
# Convert dni_best_fct to a time series
dni_best_fct_ts <- ts(dni_best_fct, start = start(test_set), frequency =
↪  frequency(test_set))

# Adding the Best Forecast DNI Values to Test_set which includes the associated Date and
# other date-related information variables

# Take solar data set and subset by the Date???
dni_best_fct_vct <- as.vector(dni_best_fct_ts)
dni_best_fct_tsv <- (time(dni_best_fct_ts))

DNI_Hourly_Avg <- as.vector(dni_best_fct_ts)
DNI_Year <- as.vector(format(yearmon(time(dni_best_fct_ts)), "%Y"))
DNI_Month <- as.vector(month.name[as.numeric(format(yearmon(time(dni_best_fct_ts)),
↪  "%m"))])
DNI_Hour <- as.vector(scaler * (row_number((time(dni_best_fct_ts))) - 1)%%(24/scaler) +
↪  (scaler/2))

dni_best_fct_df <- data.frame(DNI_Year, DNI_Month, DNI_Hour, DNI_Hourly_Avg)
head(dni_best_fct_df)
```

```
##   DNI_Year DNI_Month DNI_Hour DNI_Hourly_Avg
## 1     2019   January        1        0.00000
## 2     2019   January        3        0.00000
```

```
## 3     2019    January         5        0.00000
## 4     2019    January         7       55.61906
## 5     2019    January         9      658.72607
## 6     2019    January        11      719.05496
```

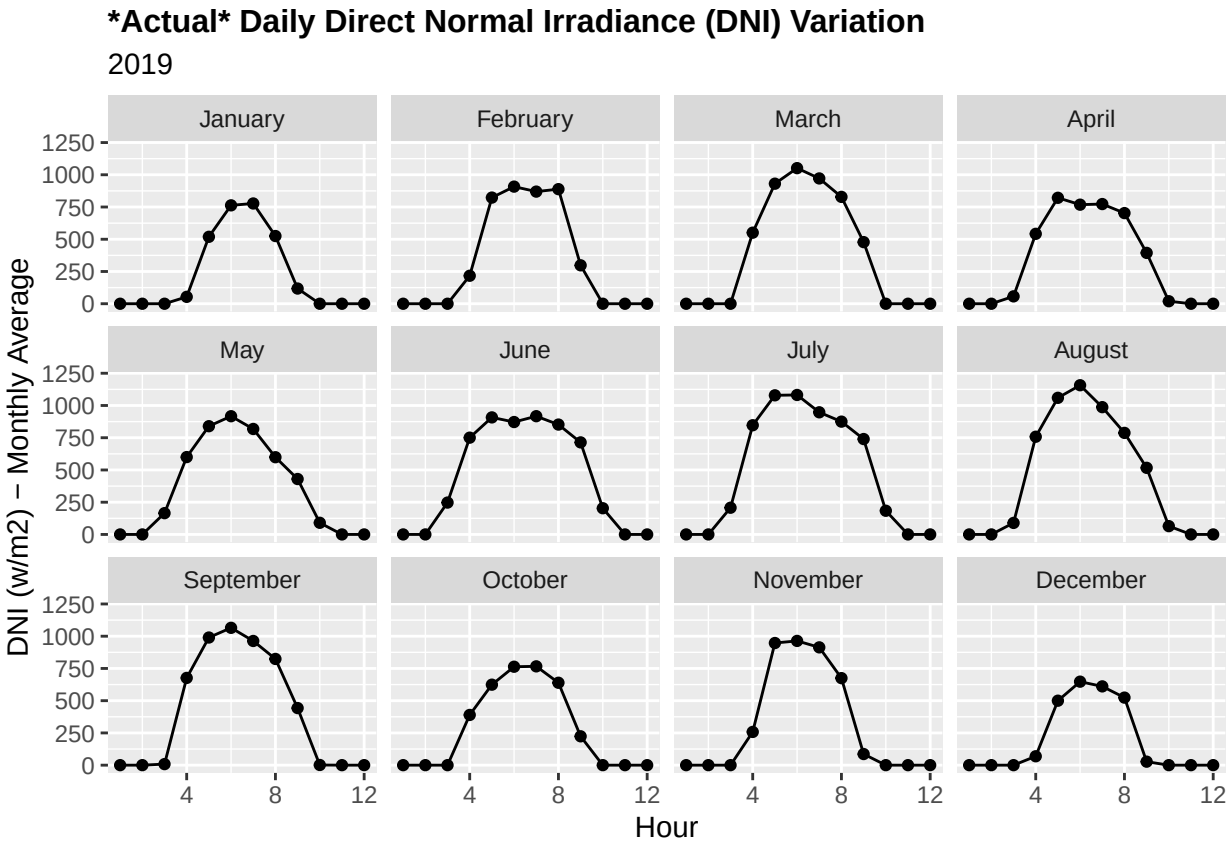
```
# First year of the forecast years
fs_year1 <- dni_best_fct_df$DNI_Year[1]

dni_best_fct_df %>%
    filter(DNI_Year == fs_year1) %>%
    mutate(Hour = DNI_Hour, Calendar_Month = factor(DNI_Month, levels = month.name)) %>%
    group_by(DNI_Month, Calendar_Month, Hour) %>%
    ggplot(aes(x = Hour, y = DNI_Hourly_Avg, group = desc(Calendar_Month))) +
    ↪  geom_point() +
    geom_line() + scale_x_continuous(breaks = seq(0, 23, 2 * scaler)) +
    ↪  facet_wrap(~Calendar_Month) +
    ggtitle(paste0(best_model, "-model *Forecast* Daily Direct Normal Irradiance (DNI)
    ↪  Variation"),
        fs_year1) + theme(plot.title = element_text(size = 12, face = "bold")) +
↪   coord_cartesian(ylim = c(0,
    1200)) + labs(x = "Hour", y = "DNI (w/m2) - Monthly Average")
```




```
# The forecast data plots can be compared with the plots of the actual data for the same
# year (fs_year1) as shown:
```

```r
# To represent the original data in the same aggregated hourly increments as that used in
# the analysis, some manipulation of the Hour variable is required.

solar_dataset %>%
    filter(Year == fs_year1) %>%
    mutate(Hour = hour(Date) + minute(Date)/60, Calendar_Month =
    ↪  factor(month.name[month(Date)],
        levels = month.name)) %>%
    group_by(Calendar_Month, Hour) %>%
    summarize(DNI_Hourly_Avg = mean(DNI)) %>%
    mutate(Hour = (Hour%/%scaler) + scaler/2) %>%
    group_by(Calendar_Month, Hour) %>%
    summarize(DNI_xHourly_Avg = sum(DNI_Hourly_Avg)) %>%
    ggplot(aes(x = Hour, y = DNI_xHourly_Avg, group = desc(Calendar_Month))) +
    ↪  geom_point() +
    geom_line() + scale_x_continuous(breaks = seq(0, 23, 2 * scaler)) +
    ↪  facet_wrap(~Calendar_Month) +
    ggtitle("*Actual* Daily Direct Normal Irradiance (DNI) Variation", fs_year1) +
    ↪  theme(plot.title = element_text(size = 12,
    face = "bold")) + coord_cartesian(ylim = c(0, 1200)) + labs(x = "Hour", y = "DNI
    ↪  (w/m2) - Monthly Average")
```

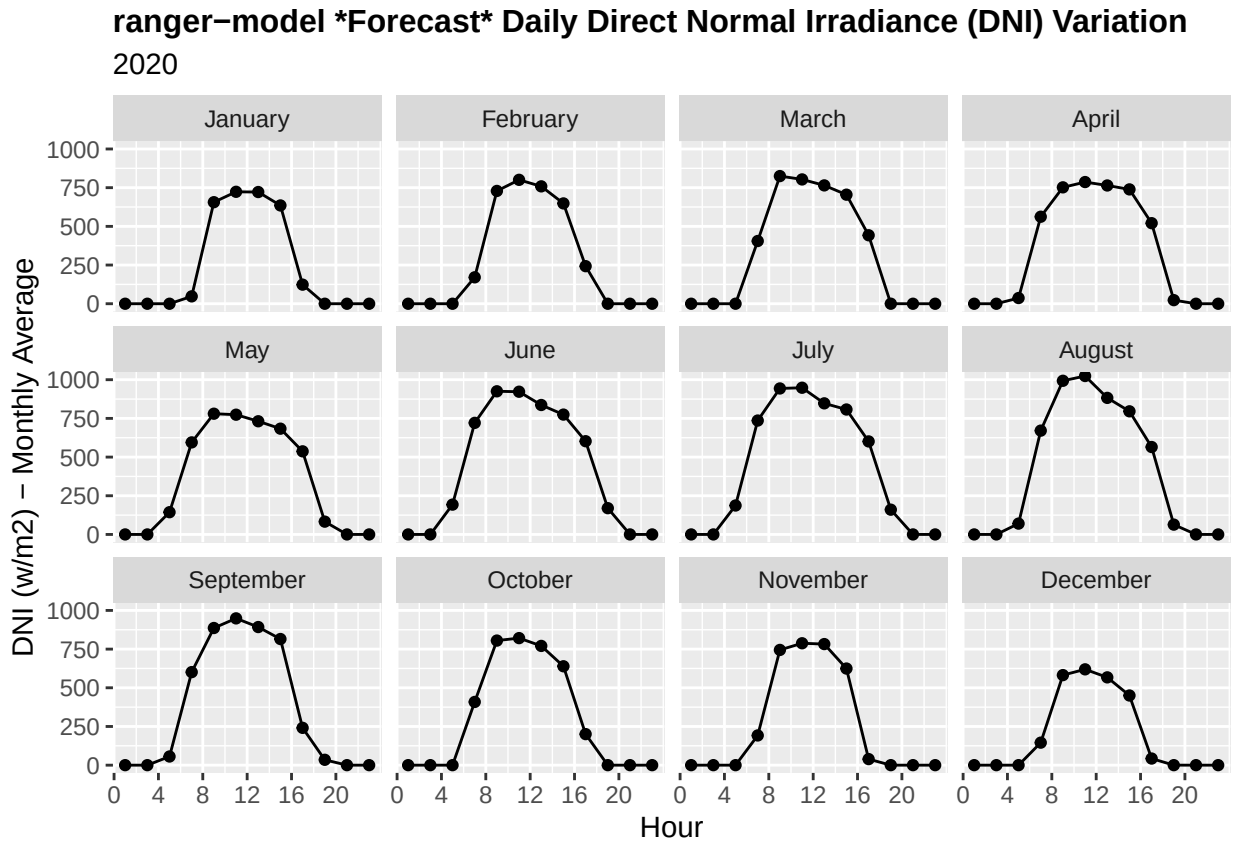
```
## `summarise()` has grouped output by 'Calendar_Month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'Calendar_Month'. You can override using the
## `.groups` argument.
```



For 2019, some differences in both the peaks and the daily duration of positive DNI values can be observed between the ranger model forecast and the actual data measurements over the course of the year though the general correspondence is good. Forecast peak (monthly-averaged) Daily DNI forecasts for December, for example, are approximately 625 w/m2 corresponding to the actual measurements in a similar range.

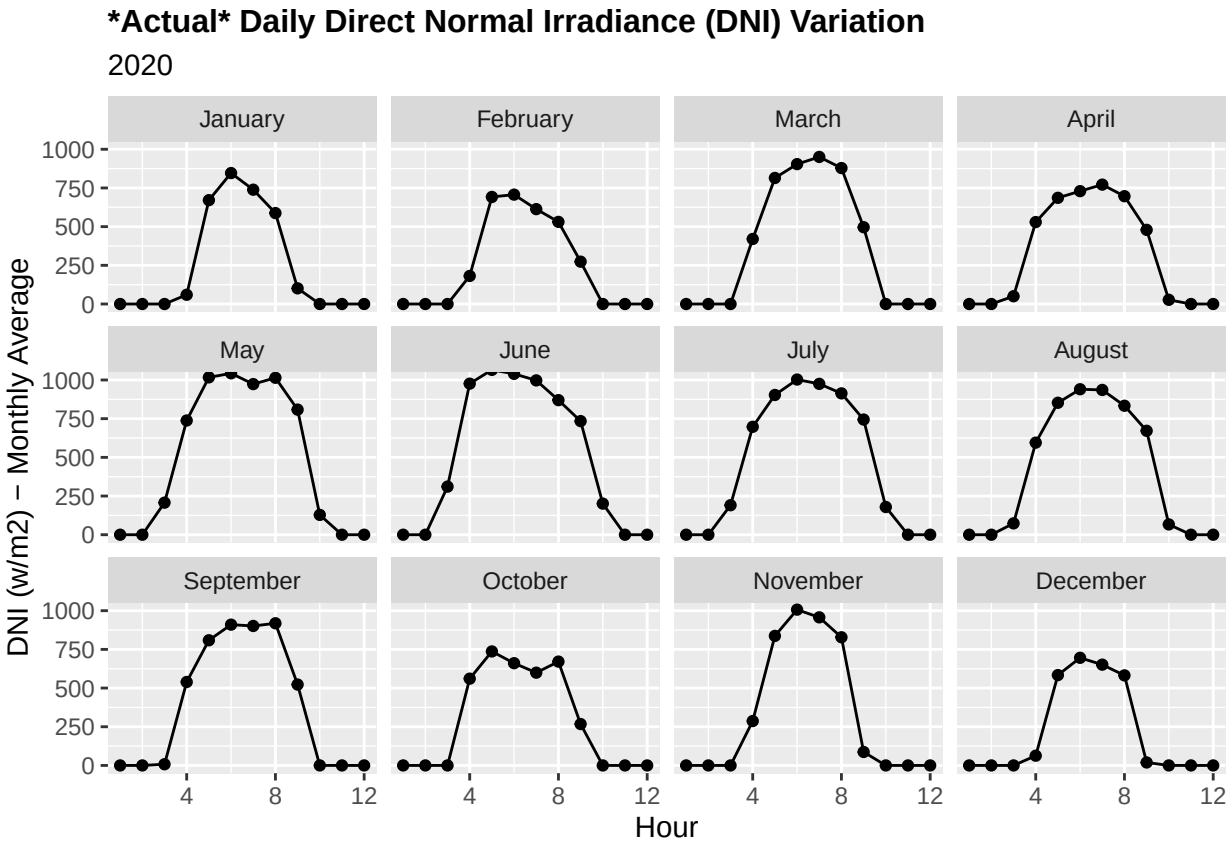
Overall the monthly-averaged daily DNI variation over the course of the year is a close correspondence with what would be expected given the mean absolute daily percentage error (MADPE) values obtained. Therefore, beyond the singular average daily percentage error measure for the DNI values, the time series forecast provide a reasonably accurate representation of the actual data.

```
# Second year of the forecast years
fs_year2 <- as.numeric(fs_year1) + 1

dni_best_fct_df %>%
    filter(DNI_Year == fs_year2) %>%
    mutate(Hour = DNI_Hour, Calendar_Month = factor(DNI_Month, levels = month.name)) %>%
    group_by(DNI_Month, Calendar_Month, Hour) %>%
    ggplot(aes(x = Hour, y = DNI_Hourly_Avg, group = desc(Calendar_Month))) +
    ↪  geom_point() +
    geom_line() + scale_x_continuous(breaks = seq(0, 23, 2 * scaler)) +
    ↪  facet_wrap(~Calendar_Month) +
    ggtitle(paste0(best_model, "-model *Forecast* Daily Direct Normal Irradiance (DNI)
    ↪  Variation"),
        fs_year2) + theme(plot.title = element_text(size = 12, face = "bold")) +
↪   coord_cartesian(ylim = c(0,
    1000)) + labs(x = "Hour", y = "DNI (w/m2) - Monthly Average")
```




```
# The forecast data plots can be compared with the plots of the actual data for the same
# year (fs_year2) as shown:

# To represent the original data in the same aggregated hourly increments as that used in
# the analysis, some manipulation of the Hour variable is required.

solar_dataset %>%
    filter(Year == fs_year2) %>%
    mutate(Hour = hour(Date) + minute(Date)/60, Calendar_Month =
    ↪  factor(month.name[month(Date)],
        levels = month.name)) %>%
    group_by(Calendar_Month, Hour) %>%
    summarize(DNI_Hourly_Avg = mean(DNI)) %>%
    mutate(Hour = (Hour%/%scaler) + scaler/2) %>%
    group_by(Calendar_Month, Hour) %>%
    summarize(DNI_xHourly_Avg = sum(DNI_Hourly_Avg)) %>%
    ggplot(aes(x = Hour, y = DNI_xHourly_Avg, group = desc(Calendar_Month))) +
    ↪  geom_point() +
    geom_line() + scale_x_continuous(breaks = seq(0, 23, 2 * scaler)) +
    ↪  facet_wrap(~Calendar_Month) +
    ggtitle("*Actual* Daily Direct Normal Irradiance (DNI) Variation", fs_year2) +
    ↪  theme(plot.title = element_text(size = 12,
    face = "bold")) + coord_cartesian(ylim = c(0, 1000)) + labs(x = "Hour", y = "DNI
    ↪  (w/m2) - Monthly Average")
```

```
## `summarise()` has grouped output by 'Calendar_Month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'Calendar_Month'. You can override using the
## `.groups` argument.
```



Consistent with the derived RMSE and MADPE, which covered the entire forecast duration, the comparison of the DNI data forecasts for year 2 (2020) of the forecast period with that of the actual measurements again show good correspondence. There is also not an appreciably falloff in the forecast at two years into the future.

Conclusions

This study has involved the development and application of machine learning algorithm to the forecasting of time series data. While the effort has been specific to the particulars of a solar radiation variable, the investigation, setup and analysis are generally applicable. The work has provided an opportunity to develop an understanding of working with time series data and with the various data analysis tools and packages available. The use of the caretForecast package allowed for the automation of a number of important data analysis steps.

The study effort included the development of an R script to access and download data from the National Renewable Energy Lab (NREL) National Solar Radiation Database (NSRDB). That developed application programming interface (API) was used to obtain the source data for the project.

As part of the undertaken analysis, a mean absolute **daily** percentage error (MADPE) measure was introduced to address the desire to have an easily referenced percentage performance measure that reflected the known daily patterns of zero and non-zero solar radiation. This measure is also particularly relevant to the

background project needs where an autonomous solar-powered system relies on being able to harvest and store solar energy across daily daylight and nighttime variations. The analysis results showed that with a 2-year forecast horizon and using aggregated hourly data it is possible to capture daily patters of the solar radiation data and achieve a forecast performance of better than 20% mean absolute daily percentage error.

On the basis of standard RMSE, the created ensemble models in general produced better prediction forecasts than the different individual ML models applied in the analysis. On the basis on the project-introduced MADPE measure, "cubist" and "ranger" were the best performing forecasting models even against the created ensemble models. Overall however, other than the svmLinear2, all models produced similar performance on both RMSE and MADPE measures. The study suggests that the default "cubist" model is indeed a good default for the caretForecast package as the best-performing single ML model by RMSE, which was the internal regression evaluation and optimization metric used for the package. This is a somewhat expected result given that the "cubist" model is an ensemble learning algorithm that combines decision trees and linear models. On the basis of the project's MADPE measure, the overall 'best' performing model was "ranger", itself an implementation of random forests decision trees ensemble. While a rigorous review was not undertaken, among the different algorithms applied in the study it was observed that the 'best' model, on the basis of MADPE, did change between "cubist" and "ranger" when different compressed data sets were used.

In this study, the computing requirements for data analysis and model evaluation proved to be a significant factor in what could be performed. Indeed, while machine learning provides extremely powerful data analysis capabilities, it does also require increased platform processing and memory resources to be able to handle larger and more granular data. To address the limitations of operating on a personal computer, different data aggregation and compression approaches and some scope limitation steps were taken in the project. However, those measures were in themselves a useful exercise and likely to always be an important relevant tradeoff consideration in any data analysis undertaking.

Future Work

A future study item would be to use the current analysis setup to investigate the impact of different lengths of training period on the algorithm's forecasting performance. With the repetitive cyclical nature of the solar measurements, it would be interesting to explore whether and to what extent there might be diminishing returns from the use of longer periods of training data or whether there may be longer-term environmental trends that are occurring that might be captured in the solar radiation data.

Studies such as [6] have also shown that, as expected, longer training data periods results in more accurate forecasts results. It would be interesting to determine the degree to which that similarly applies with physical and environmental phenomena such as solar radiation. In the caretForecast package, using a non-fixed window size will allow the package to automate the testing of the training window size and provide an optimal size for the particular ML model. Of course, any such analysis will require much more significant computing platform processing capability. Not shown in this report but observed in the testing was the fact that increasing the training period did improve forecast performance. It would be beneficial to explore this more rigorously.

In addition to the use of different training data periods, another future work item would be to evaluate performance for longer solar forecasting time horizons. The current study was limited to a 2-year forecast and with source data measurements compressed to bi-hourly, monthly averaged vales. It would be useful and interesting to understand how the specific MADPE performance declined as the forecast horizon is increased and the extent to which such a performance degradation can be compensated for by increased training periods. Being able to quantify longer range forecasts and to operate with more granular data (including up to the 30 minutes available from the NRSRDB) would be valuable for operational systems where analysis must be performed before a autonomous solar-powered system must be fielded. Given the need to cover the operating life of a system that could extend beyond a decade, understanding the margin of forecast error would be beneficial to solar energy storage sizing.

References

1. Demir, V., Citakoglu, H. Forecasting of solar radiation using different machine learning approaches. Neural Computing & Applications 35, 887–906 (2023). https://doi.org/10.1007/s00521-022-07841-x

2. Liexing Huang, Junfeng Kang, Mengxue Wan, Lei Fang, Chunyan Zhang, Zhaoliang Zeng, Solar Radiation Prediction Using Different Machine Learning Algorithms and Implications for Extreme Climate Events, Frontiers Earth Science, 30 April 2021 Sec. Environmental Informatics and Remote Sensing Volume 9 - 2021 | https://doi.org/10.3389/feart.2021.596860

3. Cyril Voyant, Gilles Notton, Soteris Kalogirou, Marie-Laure Nivet, Christophe Paoli, Fabrice Motte, Alexis Fouilloy, Machine Learning methods for solar radiation forecasting: a review, Renewable Energy Volume 105, May 2017, Pages 569-582, https://doi.org/10.1016/j.renene.2016.12.095

4. Steffen Moritz, Thomas Bartz-Beielstein, imputeTS: Time Series Missing Value Imputation in R https://cran.r-project.org/web/packages/imputeTS/vignettes/imputeTS-Time-Series-Missing-Value-Imputation-in-R.pdf

5. Data Splitting for Time Series https://topepo.github.io/caret/data-splitting.html#time

6. Spyros Makridakis, Evangelos Spiliotis, Vassilios Assimakopoulos, "Statistical and Machine Learning forecasting methods: Concerns and ways forward", PLOS, March 27, 2018, https://doi.org/10.1371/journal.pone.0194889

7. Federico Divina, Miguel García Torres, Francisco A. Goméz Vela and José Luis Vázquez Noguera, "A Comparative Study of Time Series Forecasting Methods for Short Term Electric Energy Consumption Prediction in Smart Buildings", Energies 2019, 12(10), 1934; https://doi.org/10.3390/en12101934

8. Data Source: US National Renewable Energy Labs (NREL) National Solar Radiation Database (NSRDB) https://nsrdb.nrel.gov/data-sets/how-to-access-data