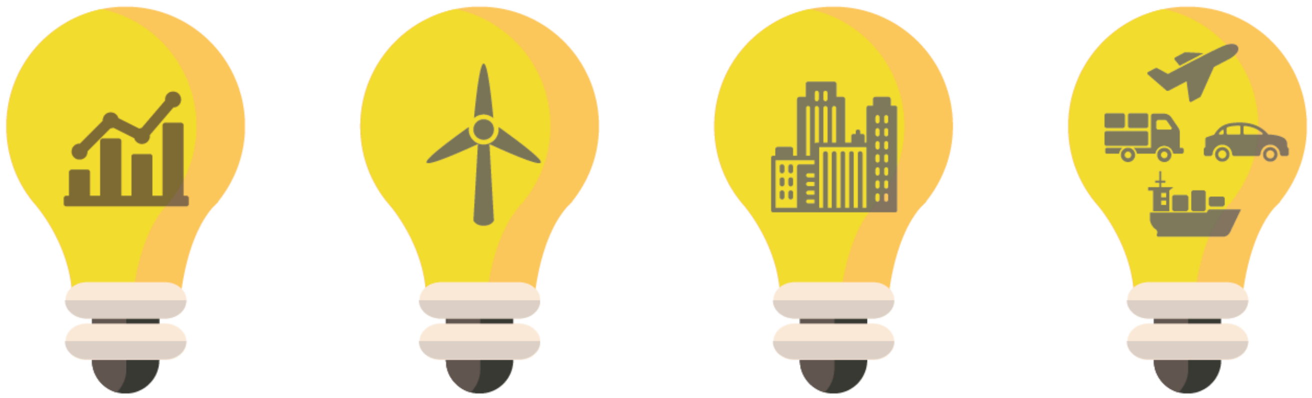
BENV0091 Lecture 5: Dates and Times

Patrick de Mars



Lecture Overview

- Intro to dates, times and the lubridate package
- Time series forecasting (Renewables Ninja)
- Upcoming competitions!

Challenges of Dates and Times

- Handling dates and times (AKA date-times) in R (and other languages) can be challenging
- There is no single format for dates and times:
  - 9am 25th Dec 2021
  - 2021-12-25 09:00
  - 9:00:00 25/12/2021
- Furthermore, local time zones (including daylight savings) mean that the time isn't the same everywhere in the world
- Coordinated Universal Time (**UTC**) is the standard time zone that is most widely used as a reference point, but often the local time is more informative
- Plus leap years (and even leap seconds) can pose problems!

Parsing Dates and Times

- Dates and times should be treated as their own data types: not as strings!
- There are 3 types of date/time object:
  - date
  - time
  - date-time (AKA POSIXct)
- When you read in data with a timestamp or date column, it will often not be automatically converted to date/time data
- The **lubridate** package has intelligent functions like `ymd()` or `dmy_hms()` for converting to date/time objects
- Identify the order of years, months, days, hours, minutes and seconds, then apply the relevant function on the right
- Task: convert the following dates/times to date or date-time variables:
  - March 3rd 2003
  - 16:00:00 14.11.2030
  - Tuesday 2nd August 1966 7:30pm

From the lubridate cheat sheet

2017-11-28T14:02:00	**ymd_hms(), ymd_hm(), ymd_h().** ymd_hms("2017-11-28T14:02:00")
2017-22-12 10:00:00	**ydm_hms(), ydm_hm(), ydm_h().** ydm_hms("2017-22-12 10:00:00")
11/28/2017 1:02:03	**mdy_hms(), mdy_hm(), mdy_h().** mdy_hms("11/28/2017 1:02:03")
1 Jan 2017 23:59:59	**dmy_hms(), dmy_hm(), dmy_h().** dmy_hms("1 Jan 2017 23:59:59")
20170131	**ymd(), ydm().** ymd(20170131)
July 4th, 2000	**mdy(), myd().** mdy("July 4th, 2000")
4th of July '99	**dmy(), dym().** dmy("4th of July '99")
2001: Q3	**yq()** Q for quarter. yq("2001: Q3")
07-2020	**my(), ym().** my("07-2020")
2:01	hms::**hms()** Also lubridate::**hms(), hm()** and **ms()**, which return periods.* hms::hms(sec = 0, min= 1, hours = 2, roll = FALSE)

Converting Time Zones

- When you create a date-time object, the time zone will be UTC unless you specify otherwise with the `tz` argument
  - E.g. ymd_hms(d, tz = …)
- The time zone codes (which should be passed as strings) and UTC offsets are taken from the **tz database** and generally have the form "region/place"
- The tz database takes into consideration daylight savings time when necessary
- Convert between timezones using `with_tz()`
- Task: create a date-time variable for 1st April 1980 5am with the UTC time zone, then convert to:
  - Europe/London (06:00 BST)
  - Asia/Tokyo (14:00 JST)
  - America/Los_Angeles (21:00 PST)

TZ Name	UTC Offset
Africa/Cairo	+02:00
Pacific/Honolulu	-10:00
Europe/Brussels	+01:00
America/Chicago	-06:00

Date-Time Arithmetic

- Date/time data can be added or subtracted
- Using `seconds()`, `months()`, `years()` etc. creates a **period**
- Adding/subtracting a period from a date-time returns a new date-time
- Subtracting two date-times returns a **difftime**
- Tasks:
  - Calculate the time in 123456789 seconds from now
  - Calculate your age in <u>seconds</u> by taking the difference between the time now and your DOB

There are some subtleties when adding or subtracting date/times: you must consider whether you want to consider discontinuities in the time line.

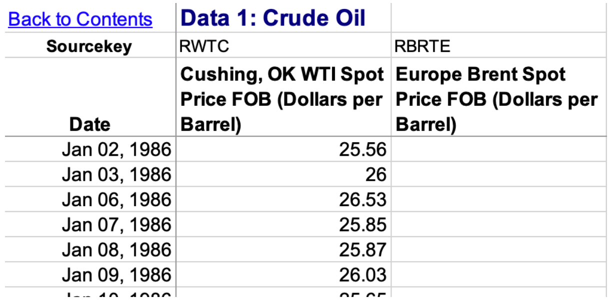
Suppose you want to know the time 3 hours after Saturday 30th October 2021 at 11pm (the night the clocks go back): 2am or 1am?

now() gives the current time as a date-time object

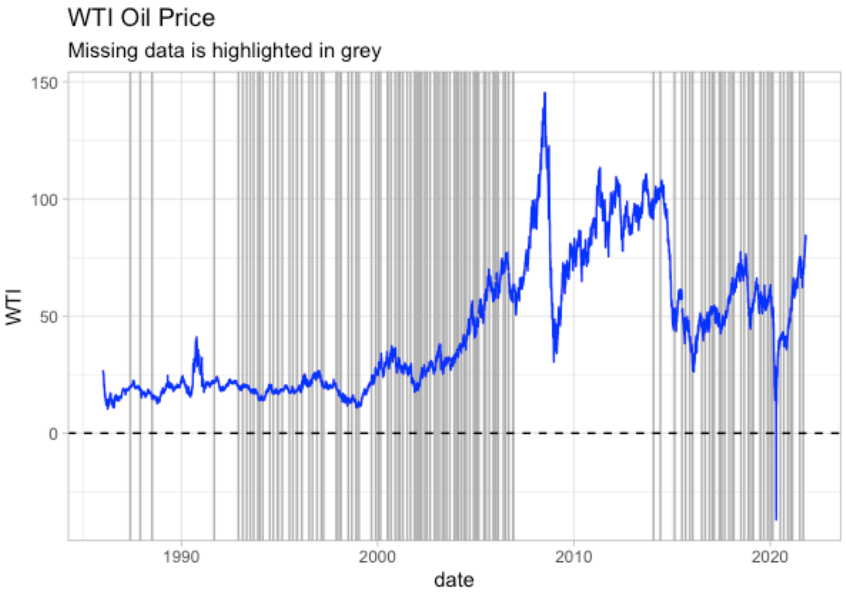
Filling NAs

```r
df <- read_excel(f, sheet = 'Data 1', skip = 2)
names(df) <- c('date', 'WTI', 'Brent') # rename columns
```

- Task: read the crude oil price data using `read_excel()` from the **readxl** package:
  - Skip the first two rows
  - Choose the sheet named “Data 1”
  - Rename the columns (see right)
- This daily crude oil price (WTI) data has a few missing values
- Task: count the number of NAs in the WTI column
- We can get rid of these values with `drop_na()` but we will lose valuable data!
- Alternatively, we can attempt to fill in (**impute**) the missing data points

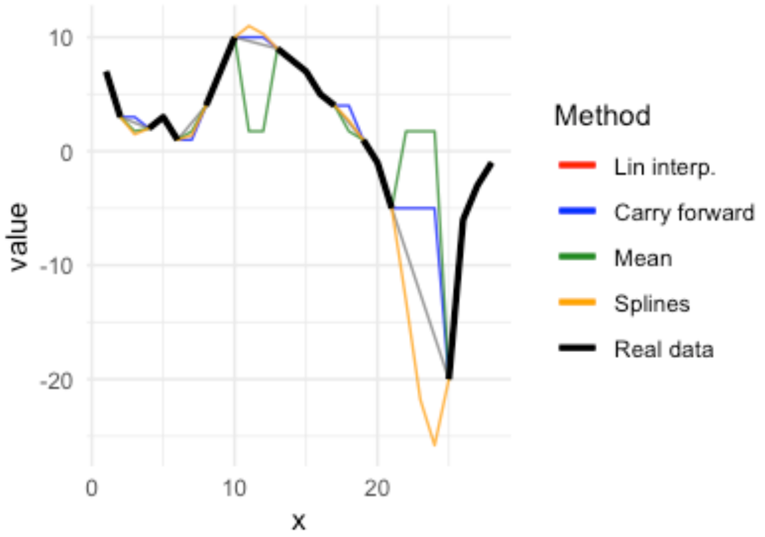


Back to Contents	Data 1: Crude Oil	
Sourcekey	RWTC	RBRTE
Date	Cushing, OK WTI Spot Price FOB (Dollars per Barrel)	Europe Brent Spot Price FOB (Dollars per Barrel)
Jan 02, 1986	25.56	
Jan 03, 1986	26	
Jan 06, 1986	26.53	
Jan 07, 1986	25.85	
Jan 08, 1986	25.87	
Jan 09, 1986	26.03	



Imputation

- Methods which impute missing values of column X using information from column X alone are **univariate**. Methods include:
  - Taking the mean, max or other statistic
  - Interpolation (linear, splines…)
  - Carry forward/backward
- **Multivariate imputation** uses other variables to **predict** the missing values
- Univariate methods are particularly appropriate for time series, because often adjacent values are similar to each other
- Task: use na.approx() from the **zoo** package to fill the NA values in the WTI oil price time series



zoo functions for univariate imputation:
- *na.approx()*
- *na.locf()*
- *na.aggregate()*
- *na.spline()*

R packages for multivariate imputation:
- *missForest*
- *mice*
- *Amelia*
- *mi*

Resampling Time Series

- Often, we want to change the resolution of a time series to match another data set
  - Electricity metered at 30 minute resolution, gas metered at 1 hour resolution
  - Daily oil prices, half-hourly electricity prices
- The methods for resampling depend on whether we want to increase or decrease the resolution:
  - Reduce resolution → aggregate
  - Increase resolution → impute or aggregate
- Task: upsample the oil price data to 30 minute resolution
- Task: impute the missing WTI data with a method of your choice

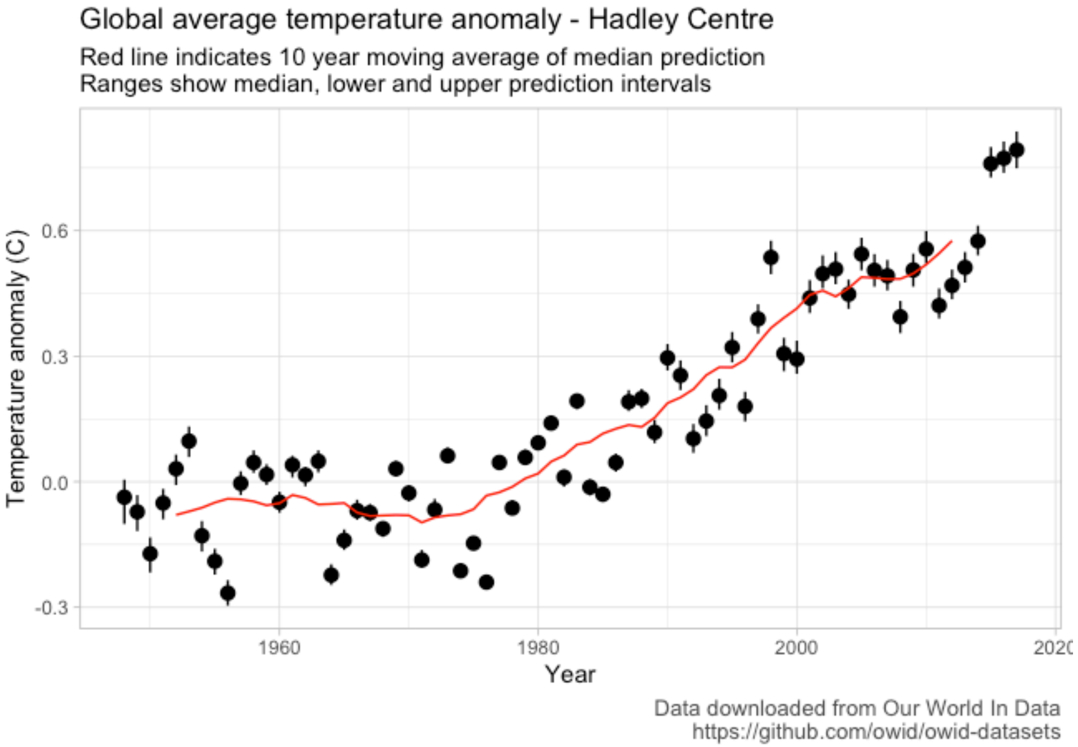
```r
# Create a data frame with the target date-time
# frequency (half-hour resolution)
hh_df <- tibble(date = seq(df_clean$date[1],
                           tail(df_clean$date, 1),
                           '30 mins'))

# Join the two dfs
df_clean_upsample <- left_join(hh_df, df_clean)
```

One upsampling method: join the data frame to a separate data frame with the desired date-time resolution

Moving Averages

- A useful and interpretable way to indicate a trend over time is a moving average
- Moving (or rolling) average typically calculates the **mean** over a rolling window
- The **`rollmean()`** function from **zoo** is a good option for calculating rolling averages (or you can write your own function!)

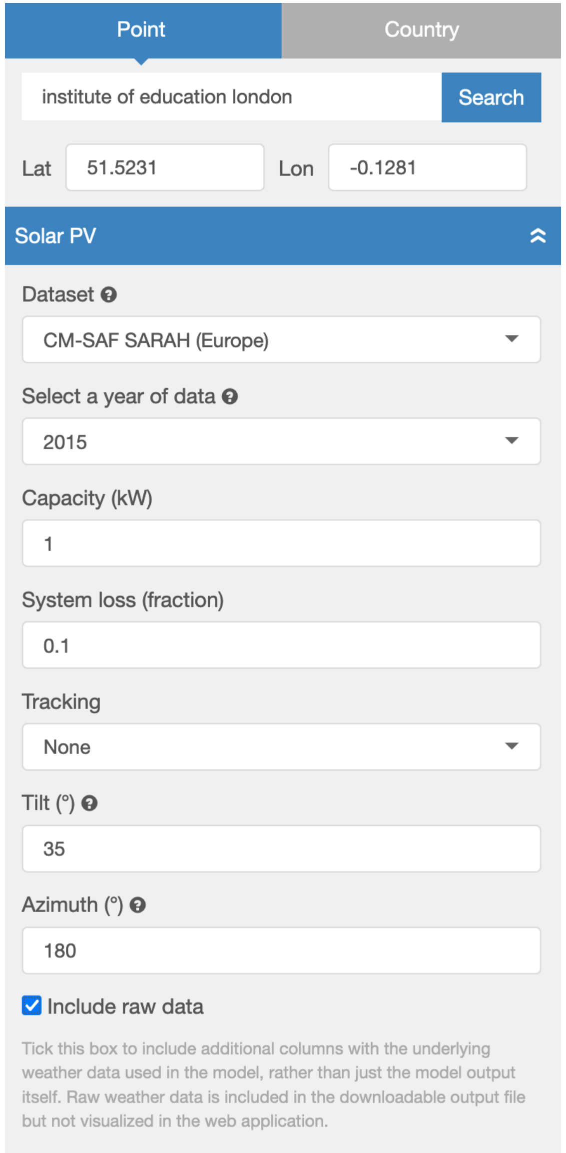


Code available from the Github repo

Introduction to Forecasting

Data: Renewables Ninja

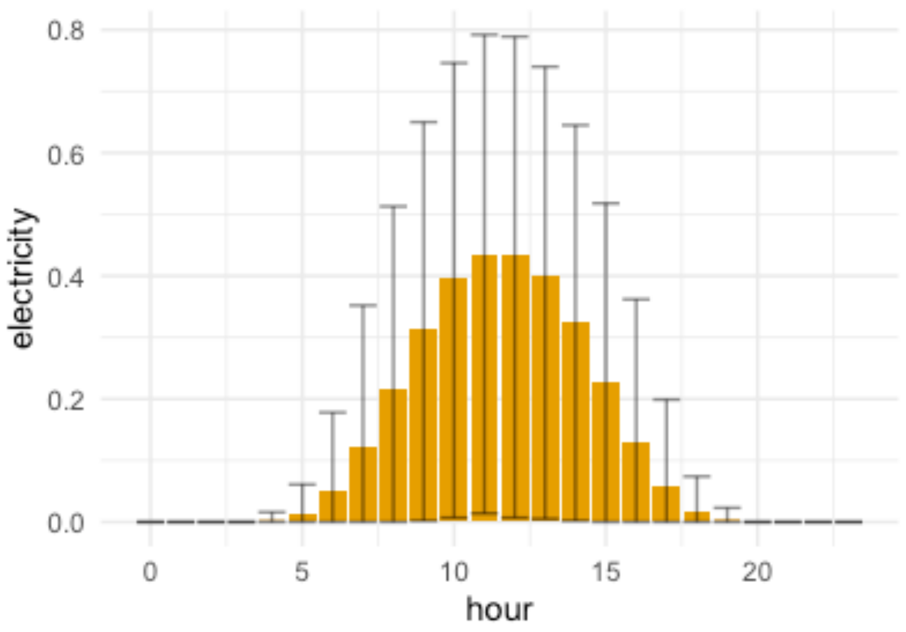
- Renewables Ninja runs simulations of hypothetical solar PV or wind turbines for a specified location, based on weather data
- We will use hourly simulated solar PV production for a 1 kW solar panel on the Institute of Education building (51.5231, -0.1281)
- Task: read the data into an object called `ninja`
- Note there are two time columns:
  - time (UTC)
  - local_time (considers local time zones, including daylight savings)



Retrieving Date-Time Components

- Once you have a correctly formatted a date-time, you can easily retrieve components such as week, hour, year etc.
- Task: use mutate() to add hour, week and date columns to `ninja`
- Task: produce the plot on the right hand side!
  - geom_col() shows the **mean** production by hour
  - geom_errorbar() shows the **range**

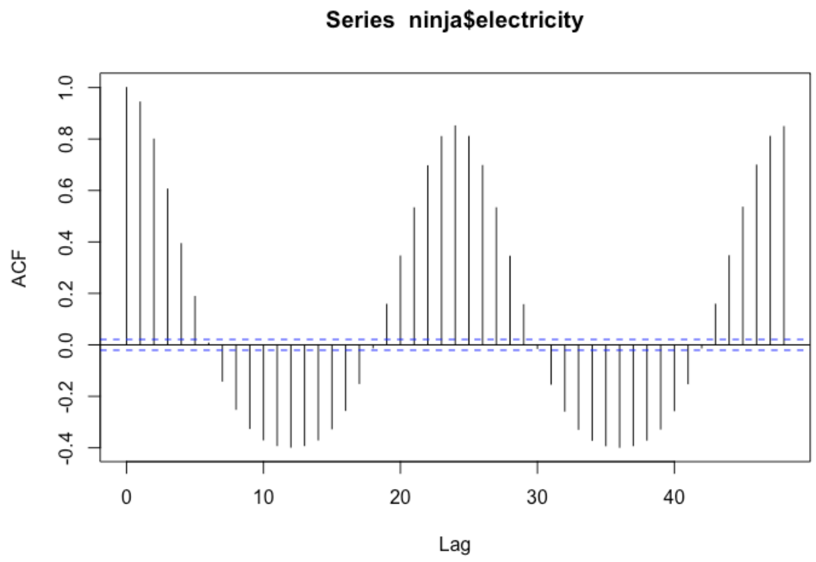
Use week() to retrieve week number from a date-time, year() to retrieve year etc.



Autocorrelation

Use acf(vector, lag.max = N) to plot the autocorrelation function for vector with a maximum lag of N

- Autocorrelation measures the correlation between a series and a delayed version of itself
- It is useful for determining patterns and **periodicity** of time series, which can be useful for **feature generation**
- Task: produce the plot on the right with the \`acf()\` function

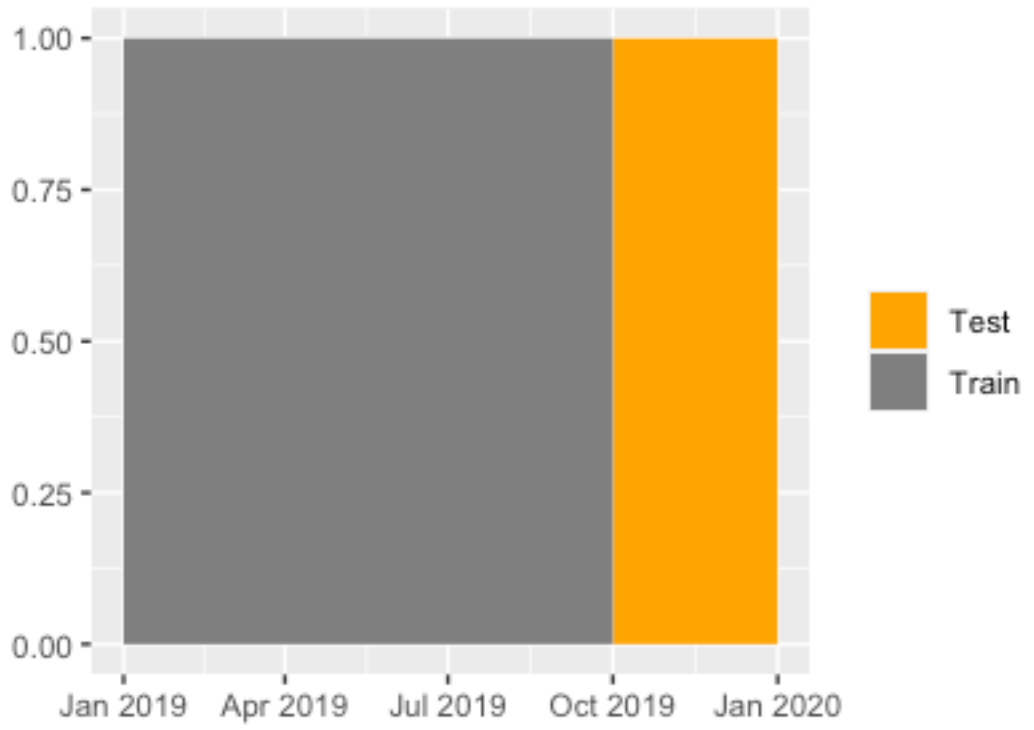


Solar PV Forecasting

- Our task will be to forecast the (simulated) solar PV output 24 hours ahead
- In this lecture we will adopt a machine learning approach, using supervised learning methods that are equally applicable in non-time series contexts
- However, there is a rich literature of traditional time series forecasting methods, including:
  - Exponential smoothing
  - Autoregressive and moving average models, e.g. ARMA, ARIMA

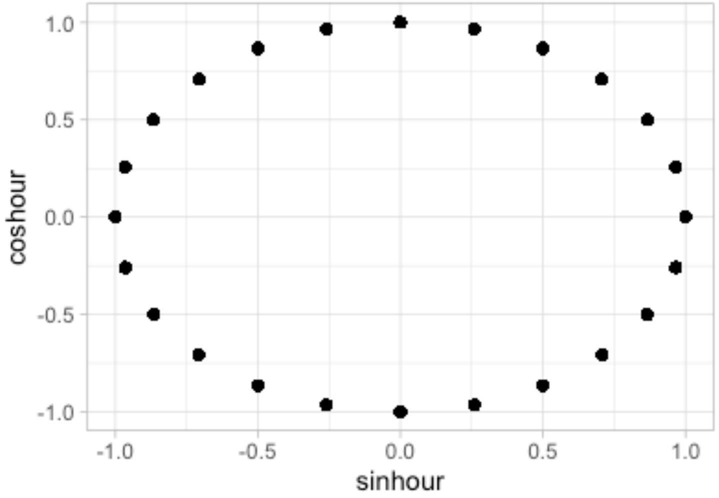
Data Pre-Processing and Train/Test Splitting

- It is important to get the data pre-processing and train/test split steps right for forecasting tasks as there are <u>serious</u> risks of **data leakage**:
    1. Including measurements from the target time as features in the model
    2. Including adjacent observations in test and train data
- To avoid (1), <u>observations from the target period shouldn't be used to train the model!</u>
- To avoid (2), the test set is often set to be the last segment
- Task: split the data such that all pre-October data is for training; the remainder is for testing



Cyclical Date/Time Features

- Dates and times are cyclical: hour 24 is adjacent to hour 1, December is adjacent to January
- If we encode dates/times as numeric variables, then we cannot immediately capture the cyclical nature
- A categorical encoding could also work, but this loses the continuous nature of dates and times and increases the dimensionality of the data significantly
- One option to create a numeric (continuous) encoding that captures the cyclical nature of dates/times is to use the **sine and cosine** functions
- Task: use mutate() to create new variables for hour and weeks:
    - sin_week, cos_week
    - sin_hour, cos_hour



For variable with period T:
cos_var = 2π cos(var / T)
sin_var = 2π sin(var / T)

Lagged Predictors

In general: if we are forecasting a variable y_t with a forecast horizon of H timesteps, we can use any observations $y_{t-k \,|\, k \geq H}$

- **Lagged explanatory variables** are among the most important inputs for forecasting models
- We can include observations of the target variable and other explanatory variables from any time before the forecast horizon, <u>but not after</u>
- Depending on the context, you may want to include 24hr, 1wk, 1yr lagged variables: you can use autocorrelation to inform your decision
- 1 year might be useful for solar PV, but we don't have enough data ☹

Use lag(vector, N) to create a new vector that is lagged by N steps, with the initial values padded with NA

Lagged values with N=1

	values	lagged
	<int>	<int>
1	1	NA
2	2	1
3	3	2
4	4	3
5	5	4

Pre-Processing

- Task: create a function for pre-processing your data
  - Add 24 hour lagged variables for:
    - Electricity
    - Direct irradiance
    - Diffuse irradiance
    - Temperature
  - Add the sine and cosine transforms of week and hour variables
- Task: pre-process your training and testing data **separately** with the cleaning function
- Task: drop the NA values from your training data

```r
prep_df <- function(df){
  # ...
  # Your code here
  # ...
}

train_prep <- prep_df(train)
test_prep <- prep_df(test)
```

Fitting Models

- This week we will introduce a new model: the **random forest**
- Random forests fit many different decision trees using **different features** (sampled randomly) and **with different data**
- The forest of trees are used to calculate an average prediction
- Task: fit the following models to the training data
  - Linear Regression
  - Decision Tree (minsplit = 10)
  - Random Forest (ntree = 200)

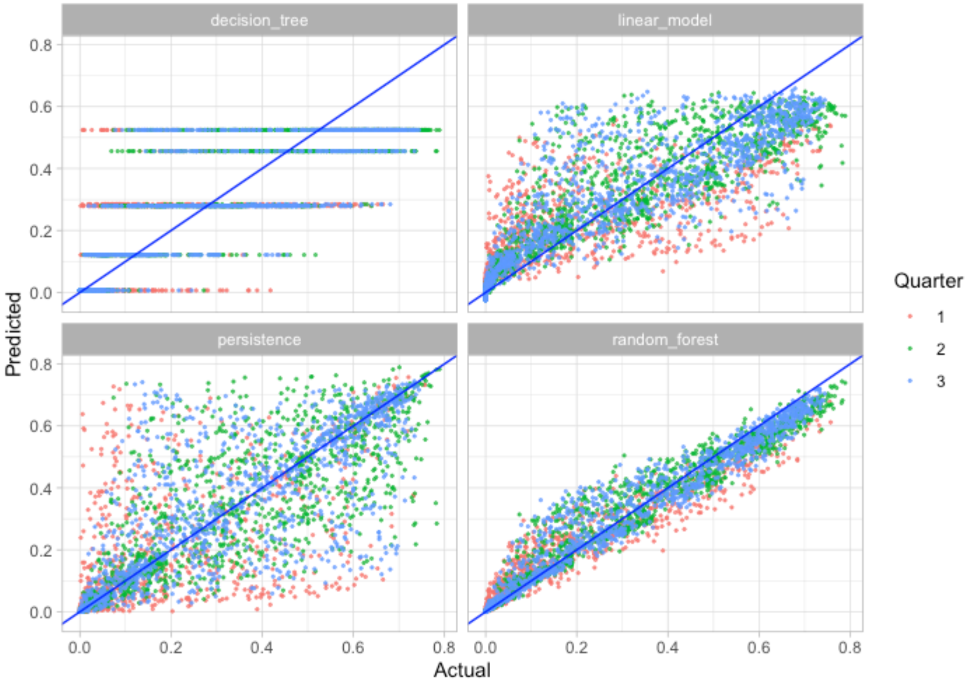
Use randomForest(formula, data, arguments…) to fit a random forest model

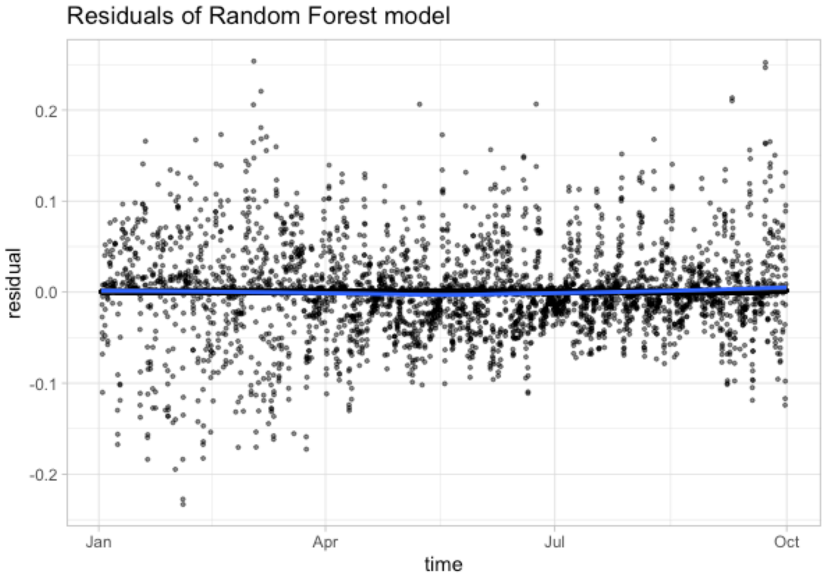
Evaluating the Models

- Task: calculate the predictions based on the training data, storing them in a data frame with the following columns:
  - Timestamp
  - Actual value
  - Linear model prediction
  - Decision tree prediction
  - Random forest prediction
- The simplest forecast assumes that the forecast is the same as the last observation: called a **naïve forecast** or **persistence forecast**
- Task: add a naïve (persistence) forecast to your predictions data frame
- Task: calculate the RMSE of each of the models

Plotting Residuals

- Now we will investigate the residuals for each model
- Task: plot the predicted vs. actual values for each model
- Task: colour each point by quarter
- Task: plot the residuals of the random forest model with respect to time

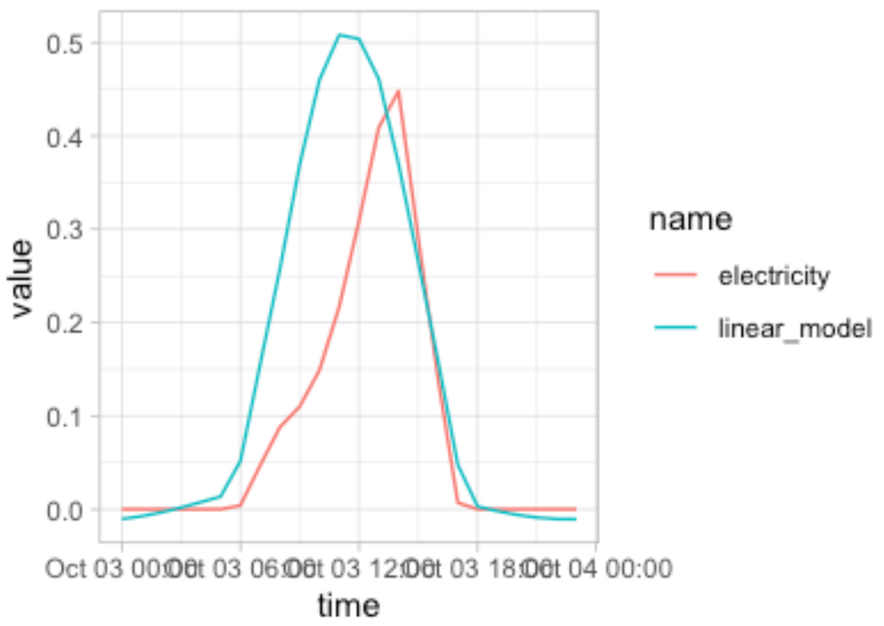




Model Evaluation on Test Data

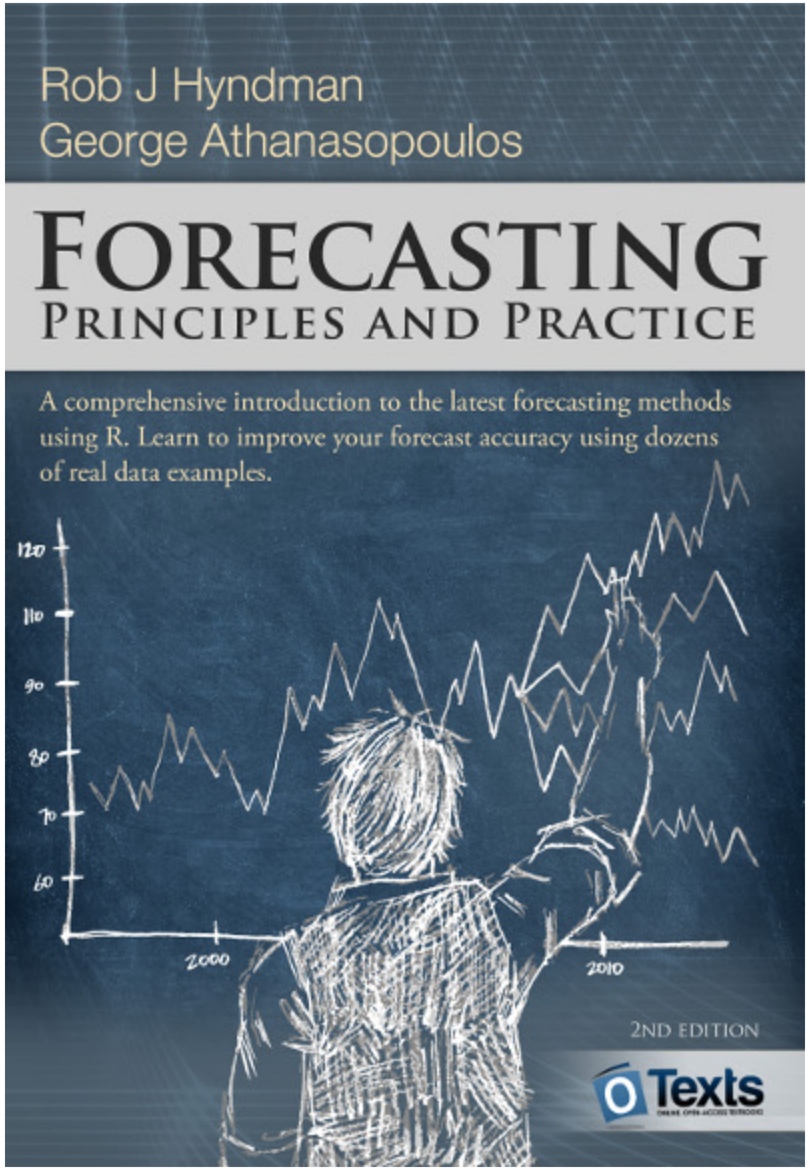
- We will now employ our models to predict the solar PV output on the out-of-sample data (October onwards)
- Task: make your predictions on the (pre-processed) test data
- Task: calculate the RMSE of each model on the test data
- Bonus task: write a function that plots the predicted and actual time series for a specified model and random day (see right)
- Which model performed best? What are the limitations of the random forest model?

```
plot_random_day <- function(model_name){
  # ...
  # Your code here
  # ...
}
```



Further Reading

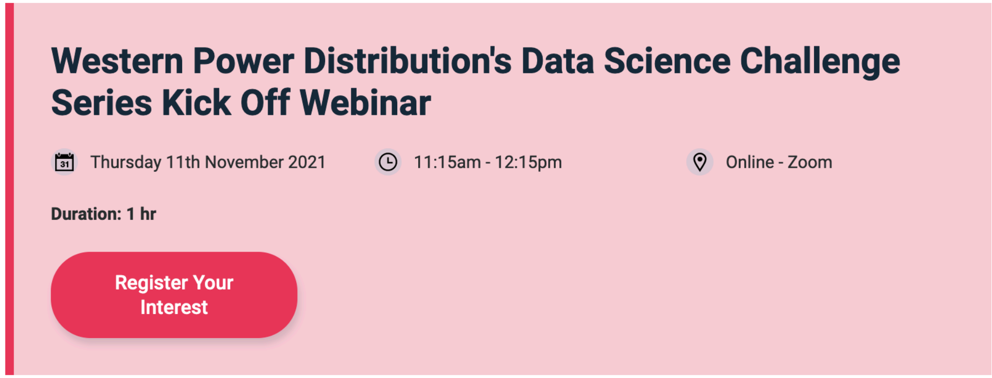
- *Forecasting: Principles and Practice by Rob J Hyndman and George Athanasopoulos*
- Excellent (free) book with R examples and more focus on traditional time series methods



Upcoming Competitions and Challenges

Western Power Distribution: Upcoming Challenges

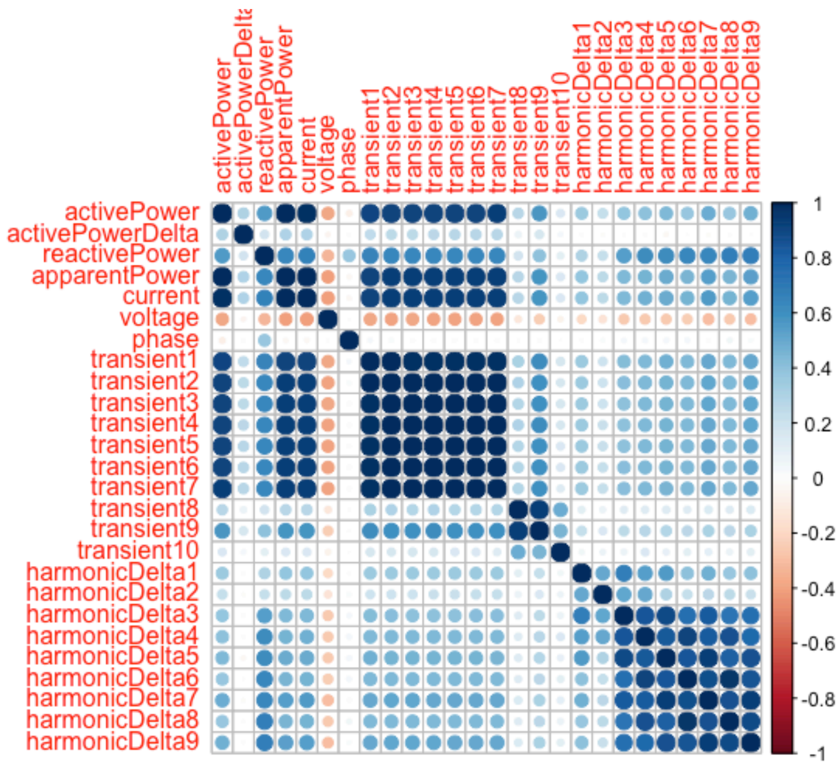
- The Energy Systems Catapult (ESC) and Western Power Distribution (WPD) are launching three short (3-week) data science challenges
- First challenge investigates whether low resolution demand data (e.g. 30-min) can be used to estimate short duration, high resolution events (such as 1-min demand spikes)
- Kick-off event: 11/11/2021
- Excellent opportunity to practice your data science skills; meet like-minded people working in the sector; study a real-world problem



Kaggle Competition

- Today we are launching a Kaggle competition looking at non-intrusive load monitoring (NILM)
- Your task is to predict **appliances** being used based on recorded samples of voltage and current
- 75% of the data is available in a training data set, 25% is held back in a test set
- You will be marked on **classification accuracy**
- **£100 for winner; £50 for 2nd and 3rd places!**
- Deadline is Wednesday 17th November @ 11:59pm

Correlation of training features



Class labels (28 in total)

	appliances	n
	<chr>	<int>
1	+fridge+washing_machine+washer_dryer	31183
2	+fridge	28575
3	+fridge+tumble_dryer+washer_dryer	15475
4	+fridge+washer_dryer	3328
5	+fridge+microwave	1969
6	+fridge+washing_machine+washer_dryer+microwave	1172
7	+fridge+shower	593

Minimum Viable Submission

- Make a submission is as simple as uploading a CSV file with 2 columns:
  - id (corresponding to the ids in the test set)
  - appliances (your prediction)
- We have made available a minimum viable submission: fitting a model, making predictions and saving the submission to a .CSV file
- **If nothing else, try submitting an improved decision tree with some hyper-parameter tuning!**
- More details are available on the Kaggle webpage ☺ Good luck!

Submission format

```
        id appliances
     <dbl> <fct>
1 87320507 +fridge
2 87319373 +fridge
3 87318280 +fridge
4 87317132 +fridge
5 87316743 +fridge+washing_machine+washer_dryer
6 87315652 +fridge+washing_machine+washer_dryer
```

https://www.kaggle.com/c/esda-nilm-2021