Forecasting

SYS 6018 | Spring 2023

forecasting.pdf

Contents

1 Time Series Data

1.1 Chicago Arrest Data

The Chicago Police Department has published arrest data from 2014-2017.

```
library(tidyverse)
url = "https://home.chicagopolice.org/wp-content/uploads/2018/07/PublicReleaseArrestDataUPDATE.csv"
arrests = read_csv(url)
```

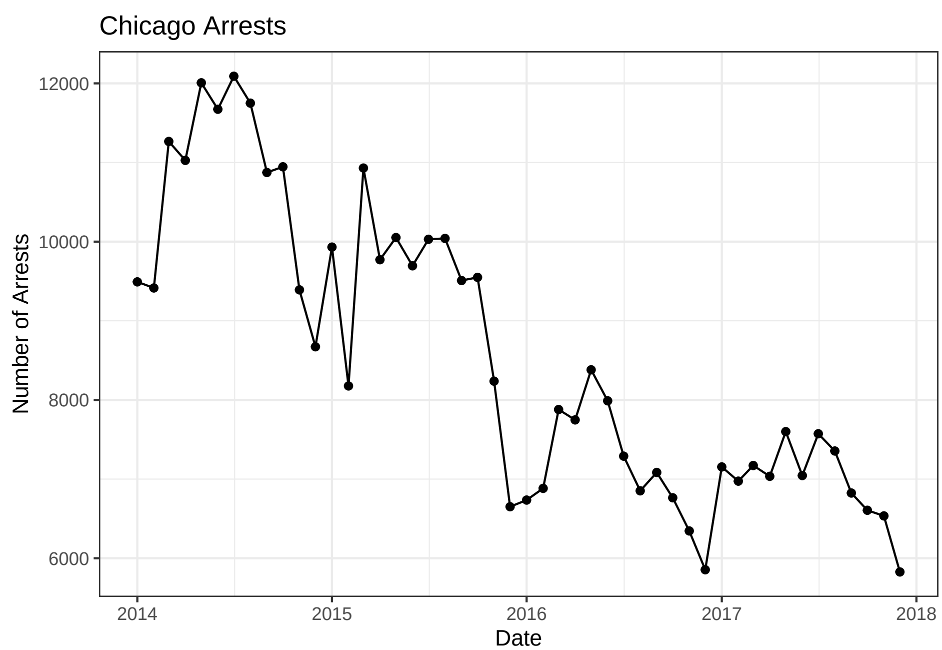
```
#> # A tibble: 6 x 10
#>   ARR_DISTRICT ARR_BEAT ARR_YEAR ARR_M~1 RACE_~2 FBI_C~3 STATUTE STAT_~4 CHARG~5
#>          <dbl>    <dbl>    <dbl>   <dbl> <chr>   <chr>   <chr>   <chr>   <chr>
#> 1           10     1033     2017       8 BLK     18      720 IL~ MFG/DE~ X
#> 2            9      923     2017       8 WWH     WRT     725 IL~ FUGITI~ Z
#> 3           10     1024     2017       8 BLK     WRT     725 IL~ ISSUAN~ Z
#> 4           11     1112     2017       8 BLK     18      720 IL~ MFG/DE~ X
#> 5           25     2524     2017       8 WHI     18      720 IL~ PCS - ~ 4
#> 6           11     1122     2017       9 BLK     7       720 IL~ CRIMIN~ A
#> # ... with 1 more variable: CHARGE_TYPE_CD <chr>, and abbreviated variable
#> #   names 1: ARR_MONTH, 2: RACE_CODE_CD, 3: FBI_CODE, 4: STAT_DESCR,
#> #   5: CHARGE_CLASS_CD
```

We're going to use the total number of arrest per month as the outcome variable of interest. This calculates the total number of arrests per year-month (`n`) and `date` (using first of month).

```
library(lubridate)  # good package for working with dates and times
arrest_counts = arrests %>%
  count(year = ARR_YEAR, month = ARR_MONTH) %>%   # aggregate number of arrests by month, year
  arrange(year, month) %>%
  mutate(
    date = lubridate::make_date(year, month, day = 1),
    index = 1:n()
  )

head(arrest_counts)
#> # A tibble: 6 x 5
#>    year month     n date       index
#>   <dbl> <dbl> <int> <date>     <int>
#> 1  2014     1  9492 2014-01-01     1
#> 2  2014     2  9415 2014-02-01     2
#> 3  2014     3 11266 2014-03-01     3
#> 4  2014     4 11027 2014-04-01     4
#> 5  2014     5 12007 2014-05-01     5
#> 6  2014     6 11673 2014-06-01     6
```

Time series plot

```
arrest_counts %>%
  ggplot(aes(date, n)) +
  geom_point() + geom_line() +
  labs(title="Chicago Arrests", x = "Date", y = "Number of Arrests")
```



Your Turn #1

1. What patterns do you see?

1.2 Data Example 2

TODO / Traffic (hourly, seconds)

1.3 Data Example 3

TODO

1.4 Time Series Patterns

A time series is data that is recorded at sequentially at *regular intervals*[1] of time. We can write the data as $D = \{(t_i, y_i)\}$ where:

- $t_{i-1} < t_i < t_{i+1}$ is the time of the ith observation
- $\delta = t_i - t_{i-1}$ is the fixed interval between times
  - units: years, quarters, months, weeks, days, hours, mins, sec
- The outcome variable y_i can be anything (real valued, integer/count, categorical, graph, image)
- For regular (i.e., fixed interval) time series, it's common to simplify notation by moving time into subscript of y denote the time $\ldots, y_{t-1}, y_t, y_{t+1}, \ldots$

There are some common patterns in time series data that can be exploited to make good forecasts.

1. Trends: observations *close* in time are often similar
   - Observations close in time exhibit simple trends (e.g., linear, quadratic, logistic)
   - e.g., y_t close to βy_{t-1} (or $\beta + y_{t-1}$)
2. Seasonality: Observations at *similar* times are often similar.
   - Human regular routines often lead to seasonal patterns
   - E.g., compare this month's outcomes to 12 months ago
   - e.g., y_t close to y_{t-12} (yearly seasonality)
3. Special times
   - Holidays, Events
   - These are short periods of time in which human behavior changes from normal
4. Exogenous dependence/association
   - other time series influencing (or associating with) outcomes
   - e.g., number of arrests may be related to the number of officers
5. Cycles: regular fluctuations, but with non-fixed frequency
   - economic cycles (expansion, peak, contraction, and trough)
   - weather cycles (El Niño and La Niña)
6. Shocks: significance changes to the data generating system
   - The data generating system may experience shocks that temporarily or persistently change the process.
   - E.g., large weather events, change in political leaders
   - These can sometimes be captured in *trends* and *special times*, but there is a field of time series analysis called *intervention analysis* that attempts to properly handle such shocks in presence of temporally correlated errors.

1.5 Forecasting / Predicting

The goal of forecasting is predict the outcome for *future* times.

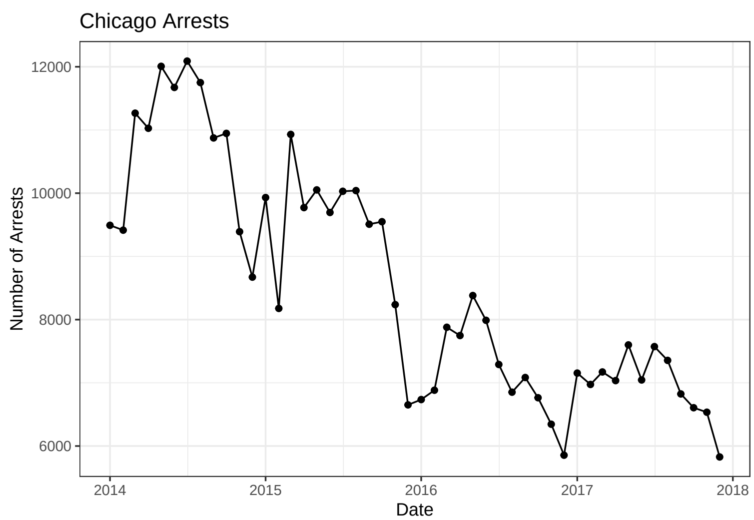
[1] Technically, you can have irregularly spaced time series (e.g., stock market closed weekends and holidays).

$$\hat{y}(t+h) = \hat{f}_h(y_{1:t}, X_{1:t})$$

- $h > 0$ is the forecast horizon
- $\hat{f}_h$ is the forecasting model (for horizon h)
- $y_{1:t} = y_1, \ldots, y_t$ are the past outcomes
- $X_{1:t} = X_1, \ldots, X_t$ are the other predictive features

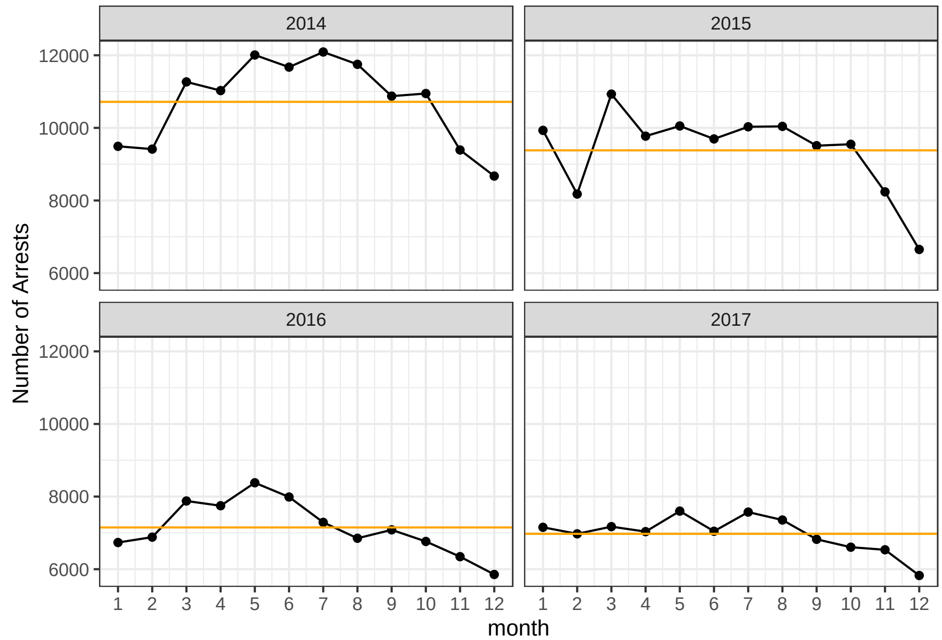
2 Chicago Arrest Analysis

2.1 Seasonality



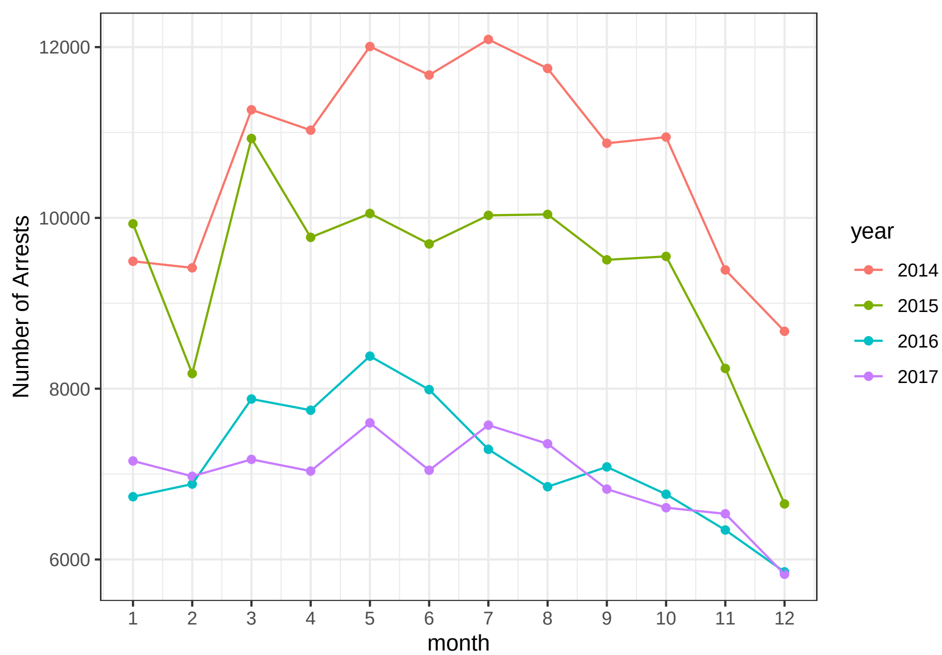
Notice the potential seasonality (e.g., decrease in Oct-Dec, increase in Mar). Let's look at the events for each year separately.

```
arrest_counts %>%
  # add yearly average
  group_by(year) %>% mutate(n_avg = mean(n)) %>% ungroup() %>%
  ggplot(aes(month, n)) +
  geom_point() + geom_line() +
  geom_hline(aes(yintercept = n_avg), color="orange") +
  scale_x_continuous(breaks = 1:12) +
  facet_wrap(~year) +
  labs(y = "Number of Arrests")
```



Or, view on single plot

```
arrest_counts %>% group_by(year) %>% mutate(n_avg = mean(n)) %>% ungroup() %>%
  mutate(year = factor(year)) %>% # make discrete for ggplot
  ggplot(aes(month, n, color=year)) +
  geom_point() + geom_line() +
  scale_x_continuous(breaks = 1:12) +
  labs(y = "Number of Arrests")
```



Your Turn #2

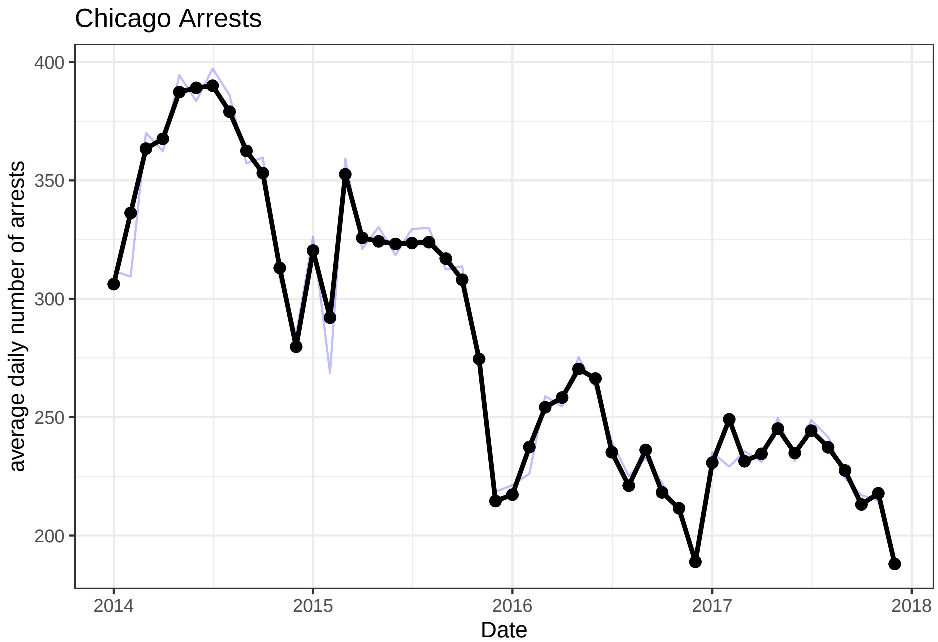
Why are the number of arrests lower in February compared to January and March?

2.1.1 Adjustments / Standardization

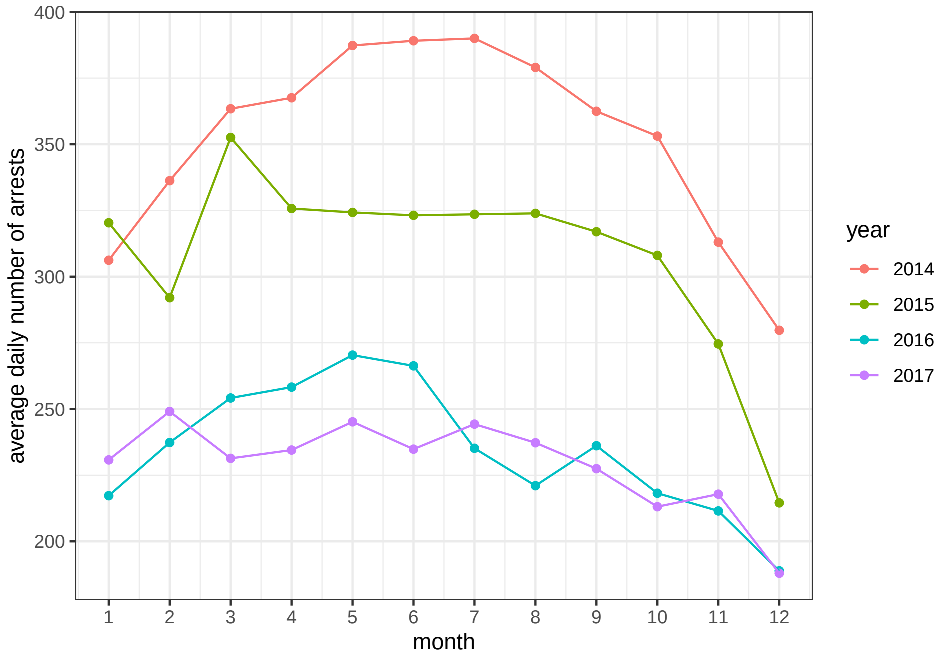
Let's standardize for days per month.

```
library(lubridate)
arrest_rate = arrest_counts %>%
  mutate(
    days_in_month = lubridate::days_in_month(date),
    daily_avg = n / days_in_month
  )
```

```
arrest_rate %>%
  ggplot(aes(date, daily_avg)) +
  geom_line(data = . %>% mutate(daily_avg = n/(365.25/12)),
            linewidth = .5,
            alpha = .25, color="blue") +
  geom_point(size = 2.25) + geom_line(linewidth=1.10) +
  labs(title="Chicago Arrests", x = "Date", y = "average daily number of arrests")
```




```
arrest_rate %>%
  mutate(year = factor(year)) %>%
  ggplot(aes(month, daily_avg, color = year)) +
  geom_point() + geom_line() +
  scale_x_continuous(breaks = 1:12) +
  labs(y = "average daily number of arrests")
```

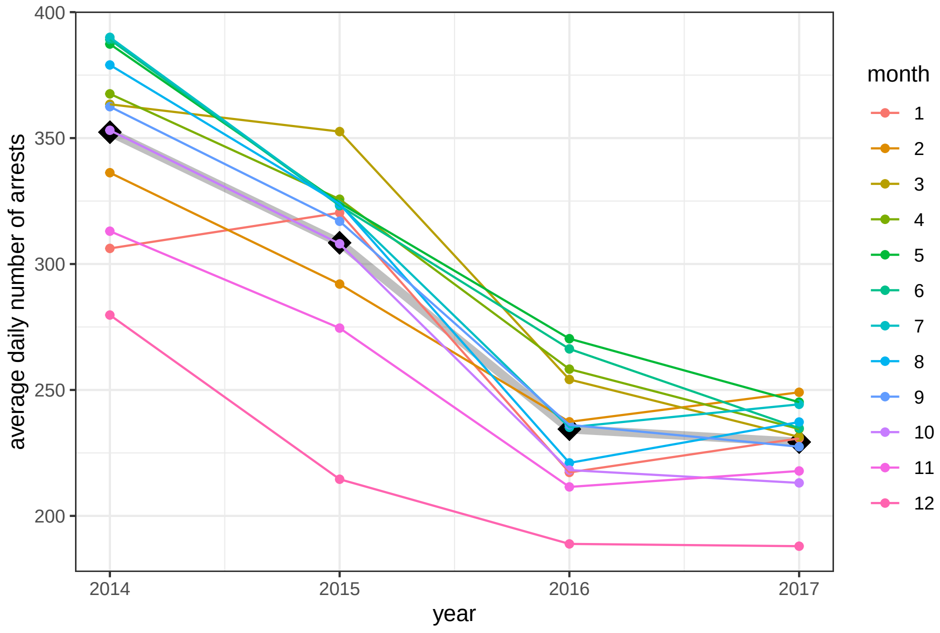


Much smoother path across months and also now a fairer comparison.

2.2 Trends

The number of arrests (and arrest rates) appear to be reducing over time. Here we examine each month separately as a function of year:

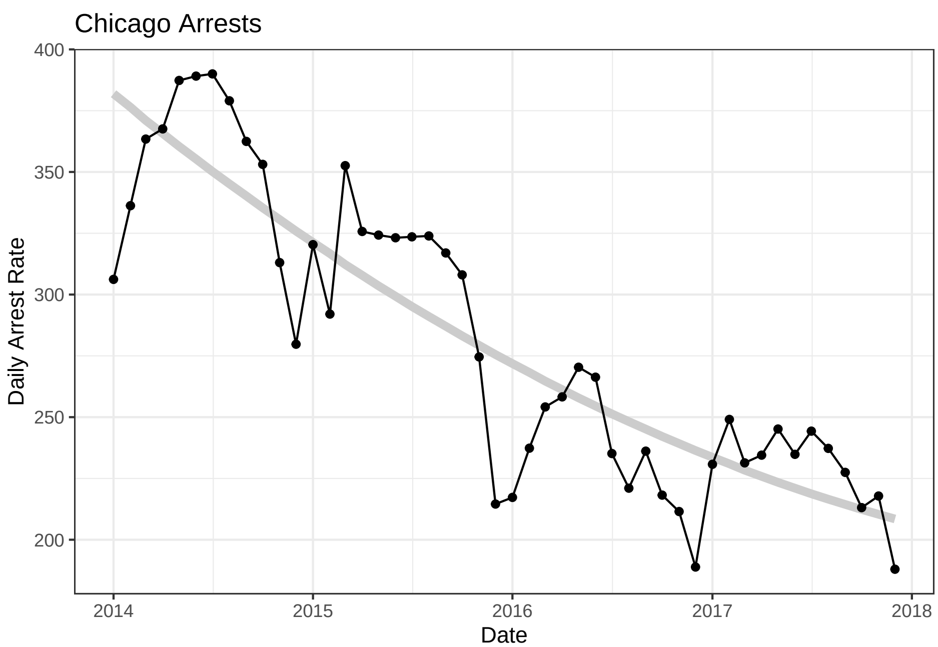
```
arrest_rate %>%
  mutate(
    days_in_year = ifelse(year %% 4 == 0, 366, 365)
  ) %>%
  group_by(year) %>% mutate(yearly_avg = sum(n) / days_in_year) %>% ungroup() %>%
  ggplot(aes(year, daily_avg, color = factor(month))) +
  geom_point(aes(y = yearly_avg), color="black", shape=18, size = 5) +
  geom_line(aes(y = yearly_avg), color="black", alpha = .25, linewidth = 2) +
  geom_point() + geom_line() +
  labs(y = "average daily number of arrests", color="month",
       caption = "black diamonds represent the yearly average")
```



black diamonds represent the yearly average

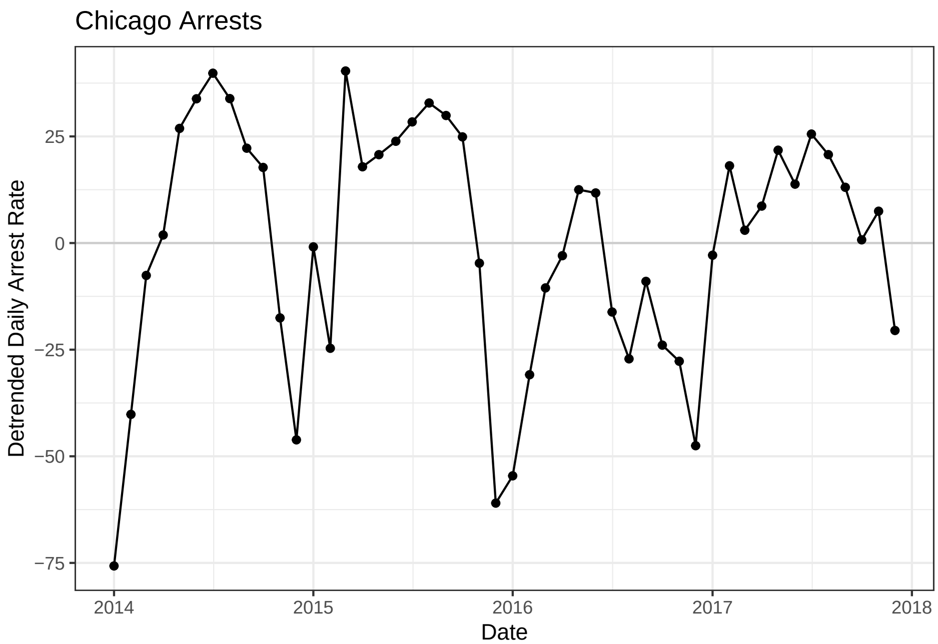
```
#: fit a quadratic trend
trend_quad = lm(daily_avg ~ poly(index, 2), data = arrest_rate)

#: add trend to data
arrest_rate_trend = arrest_rate %>%
  mutate(trend = predict(trend_quad, .))
```

```
#: add trend line to time series
arrest_rate_trend %>%
  ggplot(aes(date, daily_avg)) +
  geom_line(aes(y = trend), color = "grey80", linewidth=2) +
  geom_point() + geom_line() +
  labs(title="Chicago Arrests", x = "Date", y = "Daily Arrest Rate")
```



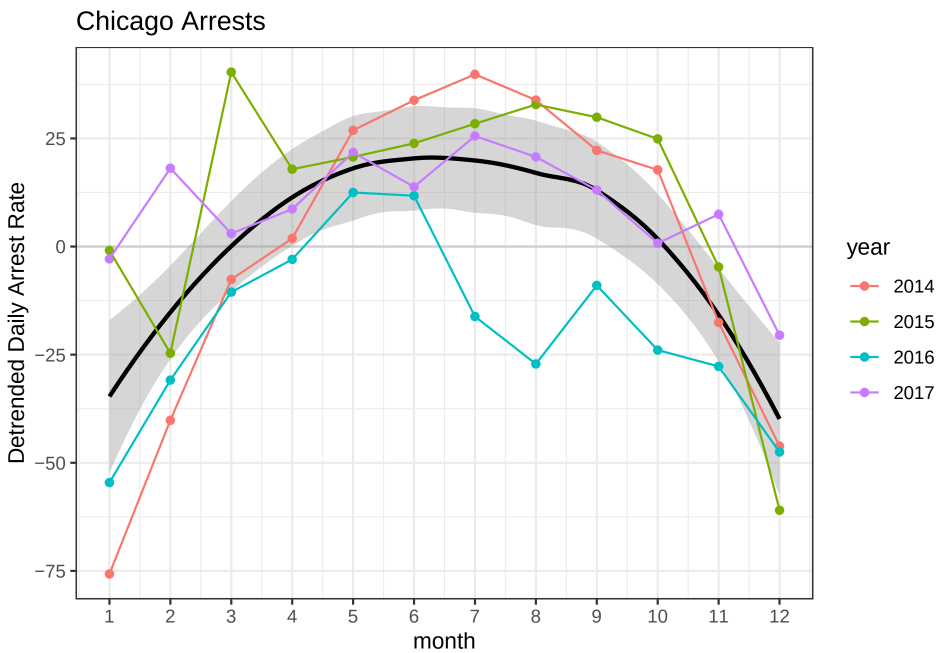
The *detrended* series is the outcome minus the trend

```
arrest_rate_trend %>%
  mutate(detrended = daily_avg - trend) %>%
  ggplot(aes(date, detrended)) +
  geom_hline(yintercept = 0, color="grey80") +
  geom_point() + geom_line() +
  labs(title="Chicago Arrests", x = "Date", y = "Detrended Daily Arrest Rate")
```



Use the detrended residuals to identify seasonality:

```
arrest_rate_trend %>%
  mutate(detrended = daily_avg - trend, year = factor(year)) %>%
  ggplot(aes(month, detrended)) +
  geom_hline(yintercept = 0, color="grey80") +
  geom_smooth(color="black") +
  geom_point(aes(color=year)) + geom_line(aes(color=year)) +
  scale_x_continuous(breaks = 1:12) +
  labs(title = "Chicago Arrests", y = "Detrended Daily Arrest Rate")
```

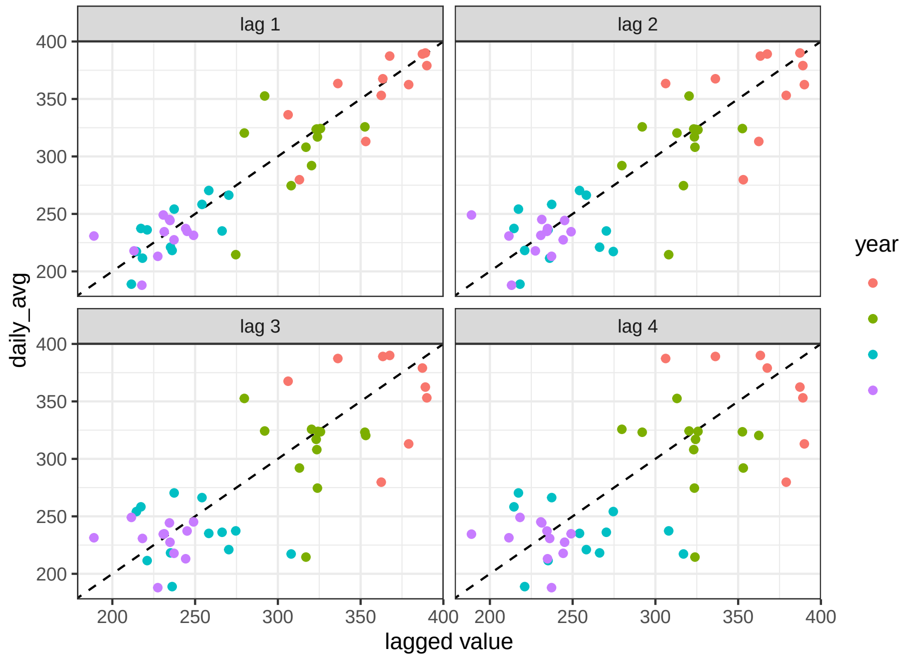


2.3 Autocorrelation

How similar are the arrest rates for nearby months? We can plot the lagged arrest rates. The lagged values are $\text{lag}_k(t) = y(t) - y(t-k)$.

```
lags = arrest_rate %>%
  mutate(
    year = factor(year)
  ) %>%
  arrange(date) %>%
  mutate(
    "lag 1" = lag(daily_avg),
    "lag 2" = lag(daily_avg, 2),
    "lag 3" = lag(daily_avg, 3),
    "lag 4" = lag(daily_avg, 4),
  )
head(lags)
#> # A tibble: 6 x 11
#>   year  month     n date       index days_in_m~1 daily~2 `lag 1` `lag 2` `lag 3`
#>   <fct> <dbl> <int> <date>     <int>       <int>   <dbl>   <dbl>   <dbl>   <dbl>
#> 1 2014      1  9492 2014-01-01     1          31    306.     NA      NA      NA 
#> 2 2014      2  9415 2014-02-01     2          28    336.    306.     NA      NA 
#> 3 2014      3 11266 2014-03-01     3          31    363.    336.    306.     NA 
#> 4 2014      4 11027 2014-04-01     4          30    368.    363.    336.    306.
#> 5 2014      5 12007 2014-05-01     5          31    387.    368.    363.    336.
#> 6 2014      6 11673 2014-06-01     6          30    389.    387.    368.    363.
#> # ... with 1 more variable: `lag 4` <dbl>, and abbreviated variable names
#> #   1: days_in_month, 2: daily_avg
```

```
lags %>%
  pivot_longer(cols = starts_with("lag"), names_to="lag") %>% filter(!is.na(value)) %>%
  ggplot(aes(value, daily_avg, color=year)) +
  geom_abline(slope = 1, intercept = 0, linetype = "dashed") +
  geom_point() +
  facet_wrap(~lag) +
  labs(x = "lagged value")
```



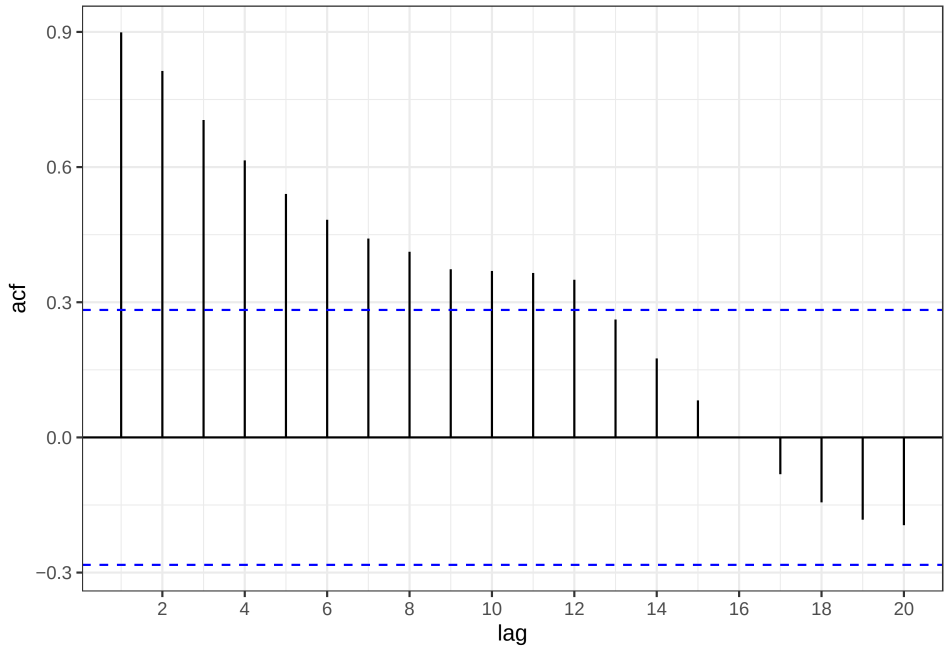
Notice that the lag 1 values have the highest correlation. We can explicitly calculate the (auto) correlation

```
lags %>%
  pivot_longer(cols = starts_with("lag"), names_to="lag") %>%
  filter(!is.na(value)) %>%
  group_by(lag) %>%
    summarize(r = cor(daily_avg, value))
#> # A tibble: 4 x 2
#>   lag       r
#>   <chr> <dbl>
#> 1 lag 1 0.926
#> 2 lag 2 0.860
#> 3 lag 3 0.779
#> 4 lag 4 0.713
```

The *autocorrelation* function (ACF) gives the correlation between y_t and a range of lagged values. To get the ACF, we'll need to convert our data into a proper time series object. I'll use the `tsibble` representation from the `tsibble` package (part of `fpp3`).

```
library(fpp3)
arrest_ts = arrest_rate %>%
  mutate(
    time = make_yearmonth(year, month) # use (month, year) for temporal index
  ) %>%
  as_tsibble(index = time)
```

```
arrest_ts %>%
  ACF(daily_avg, lag_max = 20) %>%
  autoplot() +
  scale_x_continuous(breaks = seq(0, 24, by=2)) +
  labs(x = "lag")
```



2.4 Time Series Decomposition

Consider the additive decomposition model:

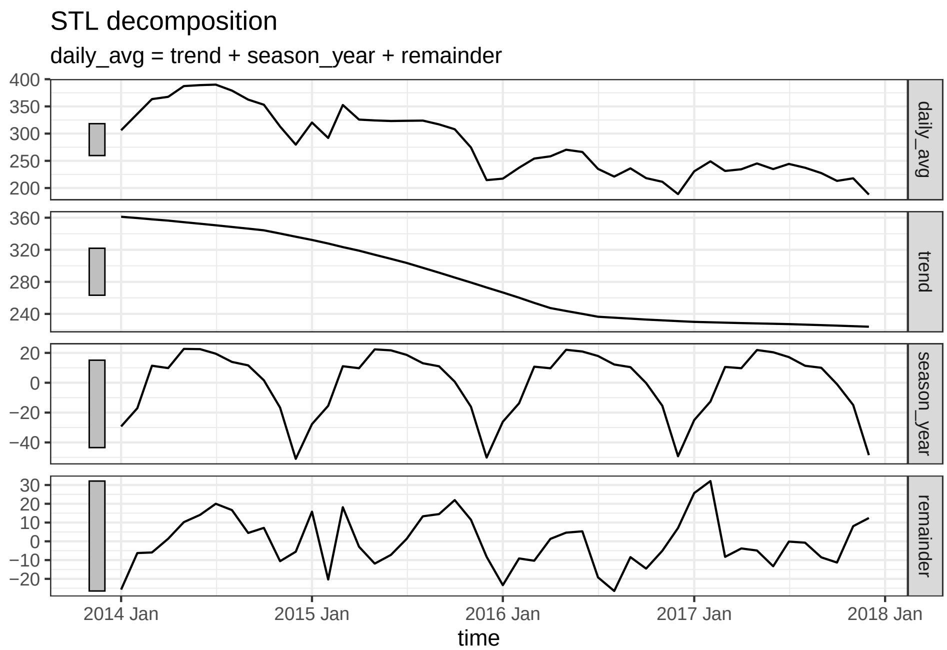
$$y_t = T_t + S_t + R_t$$

- T_t captures the *trend* (and possibly cycles)
- S_t captures the *seasonality* signal
- R_t is the *remainder*. This can also be termed *unexplained noise*.

There are several common approaches to estimate these terms. Later we will see how Facebook Prophet uses a similar additive structure. Here is how to implement the STL approach. STL is an acronym for *Seasonal and Trend decomposition using Loess*[2]:

```
decomp = arrest_ts %>% model(stl = STL(daily_avg))
components(decomp) %>% autoplot()
```

[2]Cleveland, R. B., Cleveland, W. S., McRae, J. E., & Terpenning, I. (1990). STL: A seasonal-trend decomposition. J. Off. Stat, 6(1), 3-73.



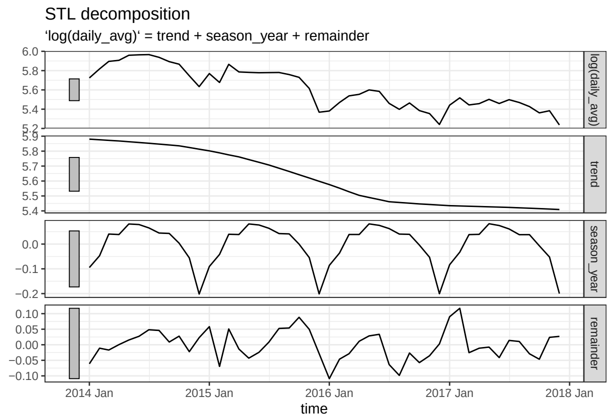
It is also common to consider a *multiplicative* structure:

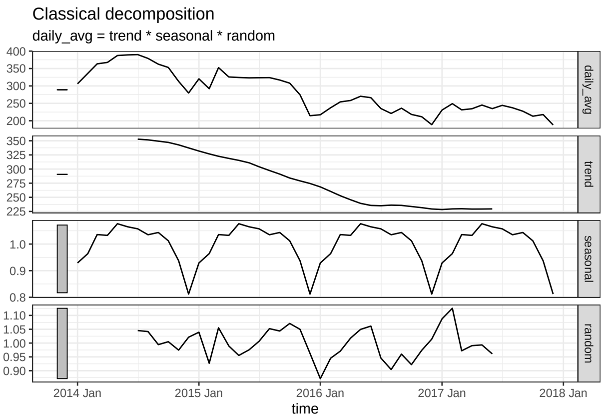
$$y_t = T_t \cdot S_t \cdot R_t$$

Although this changes the units for performance estimation, it is useful to consider that taking logs converts the multiplicative structure to something additive

$$\log y_t = \log T_t + \log S_t + \log R_t$$
$$y'_t = T'_t + S'_t + R'_t$$

```
#: multiplicative decompose by STL using logged values
decomp_mlog = arrest_ts %>% model(stl = STL(log(daily_avg)))
components(decomp_mlog) %>% autoplot()

#: multiplicative decomposition by "classical" methods
decomp_m = arrest_ts %>% model(
  classical_decomposition(daily_avg, type = "multiplicative")
)
components(decomp_m) %>% autoplot()
#> Warning: Removed 6 rows containing missing values (`geom_line()`).
```





3 Models

3.1 Regression

Regression with p predictor variables

$$\hat{y}_{t+h} = \hat{\beta}_{0,h} + \hat{\beta}_{1,h} X_{1,t} + \hat{\beta}_{2,h} X_{2,t} + \ldots + \hat{\beta}_{p,h} X_{p,t}$$

Notice that the coefficients $\hat{\beta}_{j,h}$ are a function of h; the effects of a predictor may change for different forecast horizons.

3.1.1 AR(p) Autoregression models

Autoregressive model use lagged outcome values as the predictor variables:

$$\begin{aligned} \hat{y}_{t+1} &= \hat{\beta}_0 + \hat{\beta}_1 y_{t-1} + \hat{\beta}_2 y_{t-2} + \ldots + \hat{\beta}_p y_{t-p} \\ &= \hat{\beta}_0 + \sum_{j=1}^{p} \hat{\beta}_j y_{t-j} \end{aligned}$$

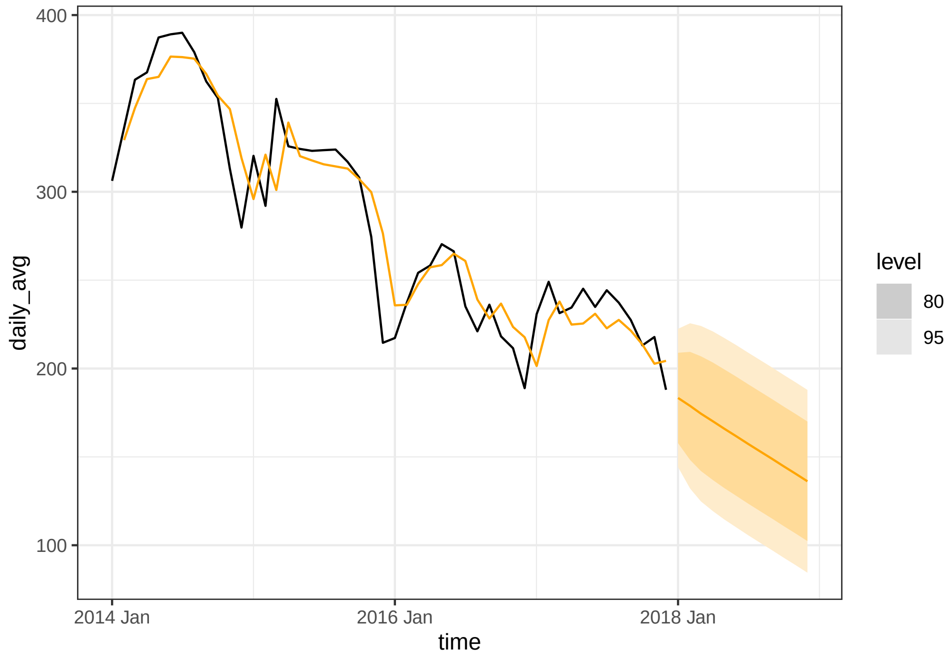
Here we'll mix autoregressive terms with a linear trend. `order()` specifies the number of AR terms to consider and `trend()` creates the linear trend. Note: higher order trends are not common in forecasting because of the possible extreme values when extrapolating.

```
fit1 = arrest_ts %>%
  model(AR(daily_avg ~ order(1:12) + trend()))

fit1 %>% report()
#> Series: daily_avg
#> Model: AR(1) w/ mean
#>
#> Coefficients:
#>    constant  trend()     ar1
#>       131.8   -1.458  0.6545
#>
#> sigma^2 estimated as 398.1
#> AIC = -98.6   AICc = -98.05   BIC = -92.98
```

The `fable::AR()` function uses AICc to select the number of AR terms (in this case only $p = 1$).

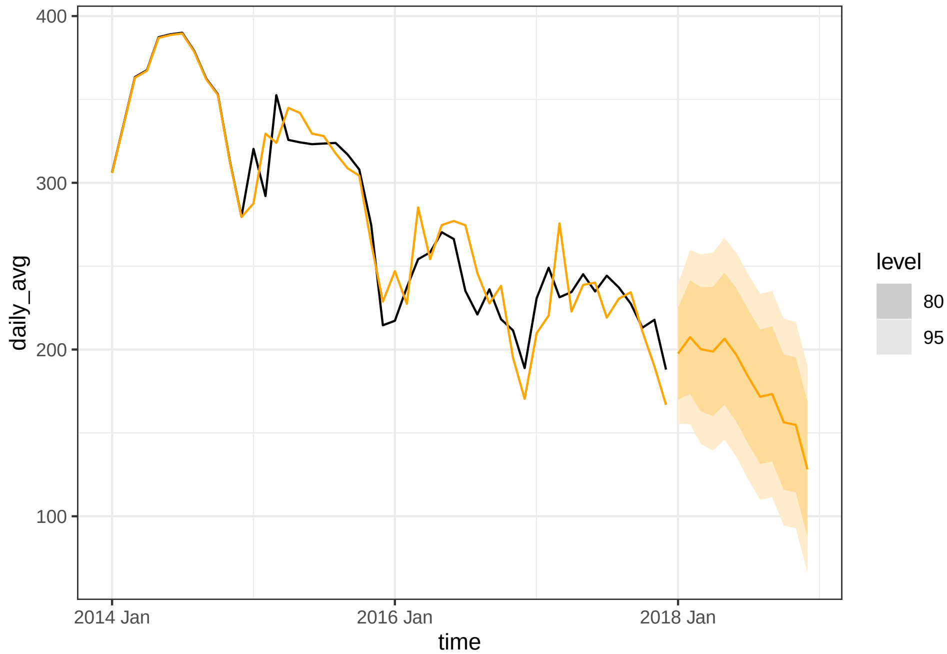
The forecasts can be obtained using the `forecast()` function.

```
fit1 %>%
  forecast(h=12) %>%
  autoplot(arrest_ts, fill="orange") +
  geom_line(data = augment(fit1), aes(time, .fitted), color="orange")
```



A better way to capture trends, autoregressive components is with a (possibly seasonal) ARIMA model. We're not going to cover ARIMA explictly, but it combines autoregressive terms, moving average terms, and differencing.

```
fit2 = arrest_ts %>%
  model(ARIMA(daily_avg))

fit2 %>% report()
#> Series: daily_avg
#> Model: ARIMA(1,0,0)(1,1,0)[12] w/ drift
#>
#> Coefficients:
#>          ar1     sar1  constant
#>       0.7340  -0.5017   -16.352
#> s.e.  0.1224   0.1528     3.753
#>
#> sigma^2 estimated as 461.3:  log likelihood=-162
#> AIC=332.1   AICc=333.4   BIC=338.4

fit2 %>%
  forecast(h=12) %>%
  autoplot(arrest_ts, fill="orange") +
  geom_line(data = augment(fit2), aes(time, .fitted), color="orange")
```



3.2 Prophet

The Facebook Prophet specifies an additive decomposition model

$$\hat{y}_t = \hat{T}_t + \hat{S}_t + \hat{H}_t$$

where T is trend, S is seasonality, and H is for holidays or special times.

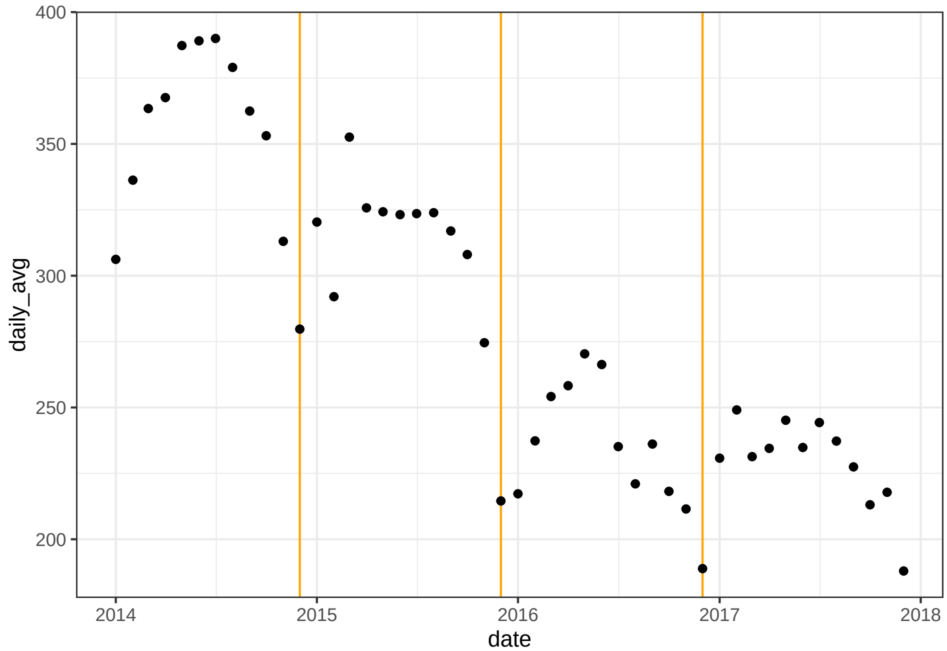
3.2.1 Trend Model

There are a few ways to include trends.

1. `growth` is *flat*, *linear*, or *logistic*.
2. The growth rate changes at a set of *change points*

This implies a join-point model for the trend. Here's an example of a linear trend. Suppose we specify change points at January of each year.

```
chgpts = seq(12, 36, by=12) # one at start of each year
arrest_ts %>%
  mutate(chgpt = index %in% !!chgpts) %>%
  ggplot(aes(date, daily_avg)) +
    geom_vline(data = . %>% filter(chgpt),
               aes(xintercept = date), color="orange") +
  geom_point()
```



To get a piecewise linear fit, we need to create a special model matrix that have columns (predictor variables) that are 0 for all times before the change point (s_j) and linear after the change point.

$$x_j(t) = (t - s_j)_+$$

This specifies the trend model of:

$$T(t) = \beta_0 + \sum_j x_j(t)\beta_j$$

```
data_train = arrest_ts %>%
  transmute(
    daily_avg,
    index,
    x = 1:n(),
    x1 = ifelse(index < chgpts[1], 0, index - chgpts[1]),
    x2 = ifelse(index < chgpts[2], 0, index - chgpts[2]),
    x3 = ifelse(index < chgpts[3], 0, index - chgpts[3]),
  )

print(data_train, n=25)
#> # A tsibble: 48 x 7 [1M]
#>         time daily_avg index     x    x1    x2    x3
#>        <mth>     <dbl> <int> <int> <dbl> <dbl> <dbl>
#>  1 2014 Jan       306.     1     1     0     0     0
#>  2 2014 Feb       336.     2     2     0     0     0
#>  3 2014 Mar       363.     3     3     0     0     0
#>  4 2014 Apr       368.     4     4     0     0     0
#>  5 2014 May       387.     5     5     0     0     0
#>  6 2014 Jun       389.     6     6     0     0     0
#>  7 2014 Jul       390      7     7     0     0     0
#>  8 2014 Aug       379.     8     8     0     0     0
#>  9 2014 Sep       362.     9     9     0     0     0
#> 10 2014 Oct       353.    10    10     0     0     0
#> 11 2014 Nov       313.    11    11     0     0     0
#> 12 2014 Dec       280.    12    12     0     0     0
```

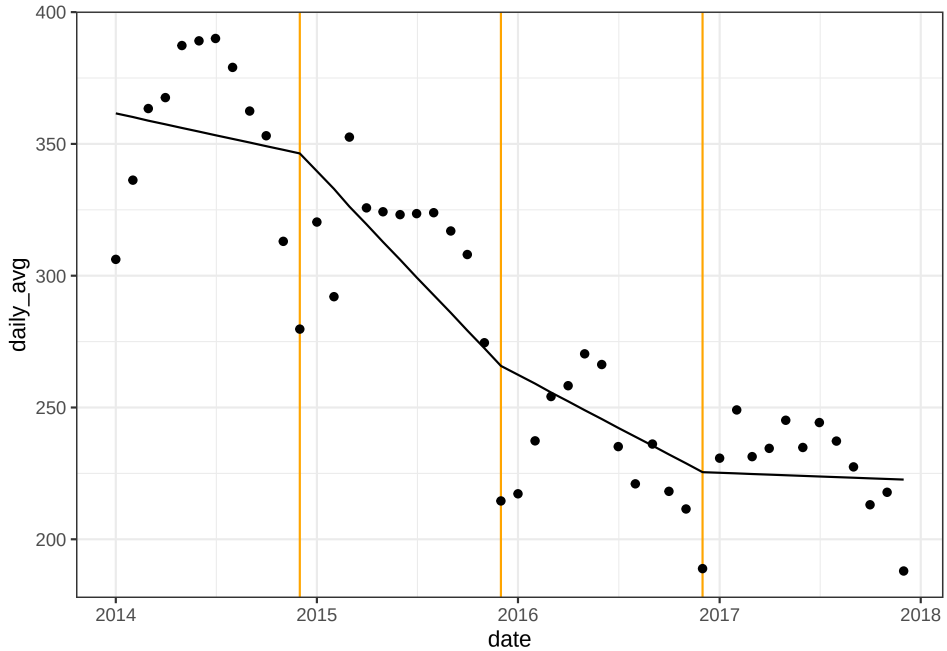
```
#> 13 2015 Jan      320.    13    13     1     0     0
#> 14 2015 Feb      292.    14    14     2     0     0
#> 15 2015 Mar      353.    15    15     3     0     0
#> 16 2015 Apr      326.    16    16     4     0     0
#> 17 2015 May      324.    17    17     5     0     0
#> 18 2015 Jun      323.    18    18     6     0     0
#> 19 2015 Jul      324.    19    19     7     0     0
#> 20 2015 Aug      324.    20    20     8     0     0
#> 21 2015 Sep      317.    21    21     9     0     0
#> 22 2015 Oct      308.    22    22    10     0     0
#> 23 2015 Nov      275.    23    23    11     0     0
#> 24 2015 Dec      215.    24    24    12     0     0
#> 25 2016 Jan      217.    25    25    13     1     0
#> # ... with 23 more rows
```

Fit a least-squares model using these special predictors:

```
fit_piecewise_linear = lm(daily_avg ~ x + x1 + x2 + x3, data = data_train)
summary(fit_piecewise_linear)
#>
#> Call:
#> lm(formula = daily_avg ~ x + x1 + x2 + x3, data = data_train)
#>
#> Residuals:
#>    Min     1Q Median     3Q    Max
#> -66.64 -17.46   5.73  20.72  36.71
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept)   362.97      15.76   23.04   <2e-16 ***
#> x              -1.38       1.86   -0.74    0.460
#> x1             -5.33       2.94   -1.82    0.076 .
#> x2              3.36       2.52    1.33    0.191
#> x3              3.12       2.74    1.14    0.261
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 27.2 on 43 degrees of freedom
#> Multiple R-squared:  0.807,  Adjusted R-squared:  0.789
#> F-statistic:   45 on 4 and 43 DF,  p-value: 7.91e-15
```

Which gives the desired piecewise linear fit

```
arrest_ts %>%
  mutate(chgpt = index %in% !!chgpts) %>%
  ggplot(aes(date, daily_avg)) +
  geom_vline(data = . %>% filter(chgpt),
             aes(xintercept = date), color="orange") +
  geom_point() +
  geom_line(
    data = augment(fit_piecewise_linear, arrest_ts),
    aes(y = .fitted)
    )
```



The Prophet model allows automatic detection of change points (and manual specification). The idea is to specify lots of potential change points and fit a lasso model to remove the unnecessary change points.

For this example, I'll specify a change point at each month and fit a lasso model to remove most of the change points.

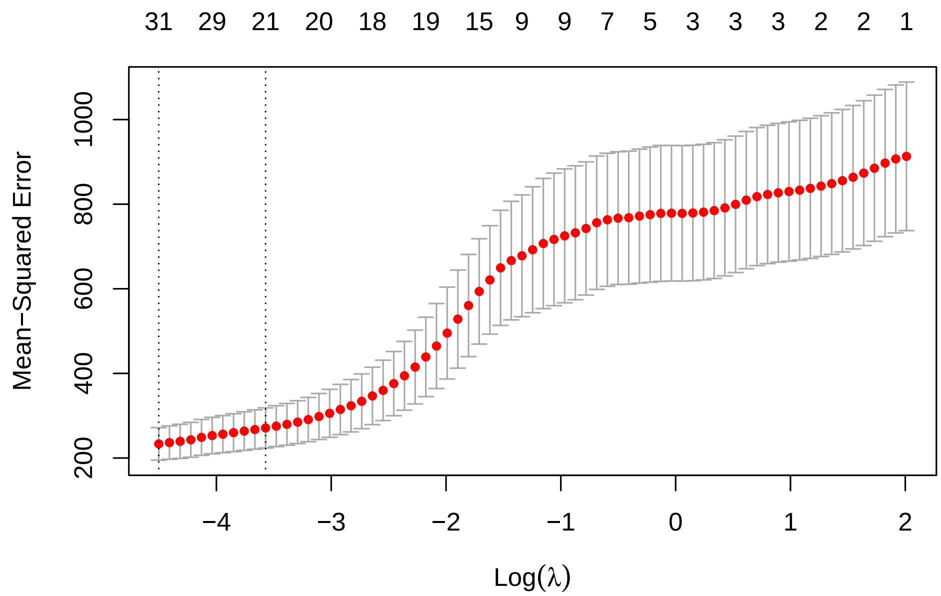
```
#: function to generate df of change point predictor variables
get_chgpt_pred <- function(x, chgpts){
  nm_chgpts = str_c("chgpt_", chgpts)
  map_df(chgpts %>% set_names(nm_chgpts),
      ~ ifelse(x < ., 0, x - .)  )
}

#: add the chgpt predictor variables, convert to matrix for glmnet
chgpts = 5:44  # don't consider changes near edges
X_chgpts = arrest_ts %>% as_tibble() %>%
  mutate(
    x = 1:n(),
    get_chgpt_pred(index, chgpts)
  ) %>%
  select(x, starts_with("chgpt")) %>%
  as.matrix()
Y = arrest_ts$daily_avg
```

Fitting the lasso model.

- Not best practice to use straight cross-validation with time series data, but trying to illustrate the general idea without getting into time series cv details.
- Here we have to be especially careful to avoid capturing seasonality (since there are not yet seasonal components added). So I'll manually set lambda to only keep around 3 changepoints.

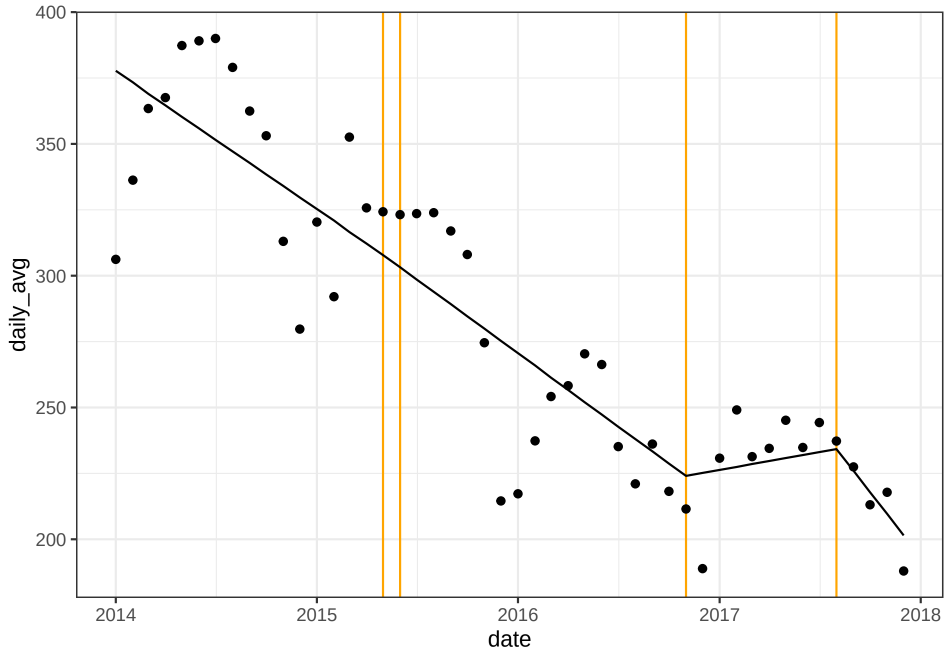
```
library(glmnet)
set.seed(2023)
fit_lasso = cv.glmnet(X_chgpts, Y, alpha = 1, penalty.factor = c(0, rep(1, length(chgpts) )))
plot(fit_lasso)
lam = 1
```

```
coef(fit_lasso, s = lam) %>% as.matrix %>% as_tibble(rownames = "var") %>%
  filter(s1 != 0 )
#> # A tibble: 6 x 2
#>   var               s1
#>   <chr>          <dbl>
#> 1 (Intercept) 382.
#> 2 x            -4.37
#> 3 chgpt_17     -0.247
#> 4 chgpt_18     -0.0395
#> 5 chgpt_35      5.79
#> 6 chgpt_44     -9.31
```



And the plot

```
chgpts_est =
  coef(fit_lasso, s = lam) %>%
  as.matrix %>% as_tibble(rownames = "var") %>%
  filter(s1 != 0, var != "(Intercept)") %>% pull(var) %>%
  str_remove("chgpt_") %>% as.integer

fitted = predict(fit_lasso, s = lam, newx = X_chgpts)

arrest_ts %>%
  mutate(chgpt = index %in% !!chgpts_est) %>%
  ggplot(aes(date, daily_avg)) +
  geom_vline(data = . %>% filter(chgpt),
             aes(xintercept = date), color="orange") +
  geom_point() +
  geom_line(y = fitted)
```



For the logistic growth model the trend (with piecewise change points) is:

$$T(t) = \frac{C(t)}{1 + \exp(\beta_0 + \sum_j x_j(t)\beta_j)}$$

where $C(t)$ is the *carrying capacity* of the system.

3.2.2 Seasonality

Patterns that repeat at a regular frequency are termed seasonal features.

The Prophet model uses a set of Fourier basis functions to capture seasonality. The Fourier basis functions are represented by a pair of $\sin$ and $\cos$ functions scaled by order k and period P.

$$X_{kc}(t) = \cos\left(2\pi t\frac{k}{P}\right)$$
$$X_{ks}(t) = \sin\left(2\pi t\frac{k}{P}\right)$$

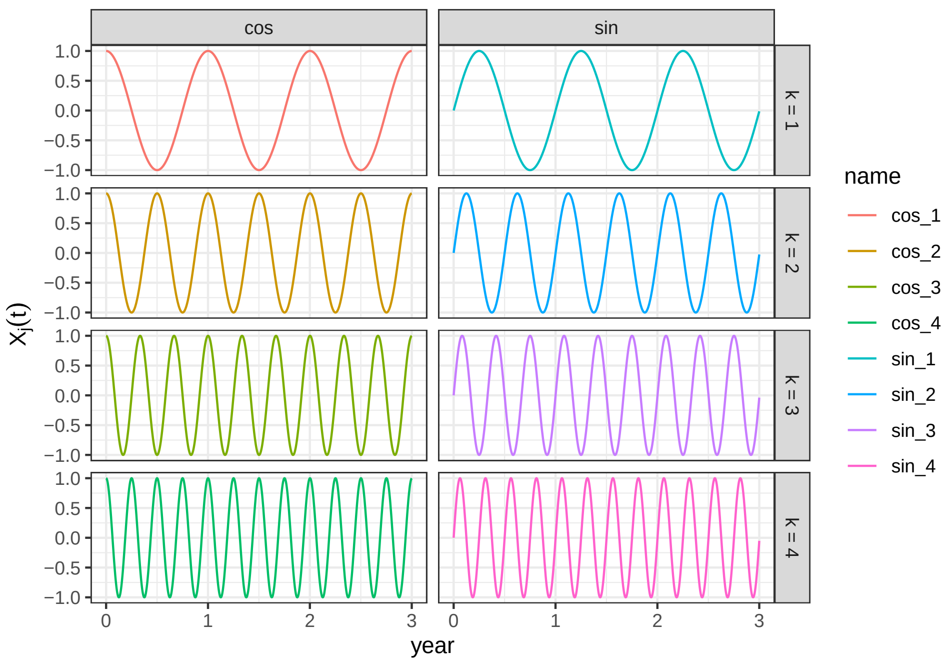
For daily observed data and yearly seasonality, the first few predictors are

$$X_{1c}(t) = \cos\left(2\pi t\frac{1}{365.25}\right)$$
$$X_{1s}(t) = \sin\left(2\pi t\frac{1}{365.25}\right)$$
$$X_{2c}(t) = \cos\left(2\pi t\frac{2}{365.25}\right)$$
$$X_{2s}(t) = \sin\left(2\pi t\frac{2}{365.25}\right)$$
$$X_{3c}(t) = \cos\left(2\pi t\frac{3}{365.25}\right)$$
$$X_{3s}(t) = \sin\left(2\pi t\frac{3}{365.25}\right)$$

Here is a visual:

```
#: function to manually calculate Fourier basis function
fourier_series <- function(x, k = 4, period = 12){
  fourier_k <- function(k){
    tibble(
      !!str_c("cos_",k) := cos(2*pi*x*k/period),
      !!str_c("sin_",k) := sin(2*pi*x*k/period)
    )
  }
  map(1:k, fourier_k) %>% bind_cols()
}
```

```
tibble(x = seq(0, 365*3, length=1000)) %>%
  mutate(fourier_series(x, k = 4, period = 365.25)) %>%
  pivot_longer(-x) %>%
  separate(name, into = c("trig", "k"), sep = "_", remove = FALSE) %>%
  arrange(k, trig) %>% mutate(k = str_c("k = ", k)) %>%
  ggplot(aes(x, value, color=name)) + geom_line() +
  facet_grid(k~trig) +
  scale_x_continuous(breaks = seq(0, 365*5, by=365), labels = 0:5) +
  labs(x = "year", y = expression(X[j](t)))
```



Let X_P be the set of Fourier basis functions for period P. Prophet allows multiple seasonal components, e.g.,

$$\hat{S}(t) = X_{P_1}(t)\hat{\beta}_{P_1} + X_{P_2}(t)\hat{\beta}_{P_2}$$

where $P_1 = 365.25$ incorporates yearly seasonality and $P_2 = 7$ weekly. Prophet uses a ridge penalty on the Fourier coefficients.

As a concrete example, here are the the first few Fourier predictors for the Chicago Arrest data (using a period of $P = 12$ months per year).

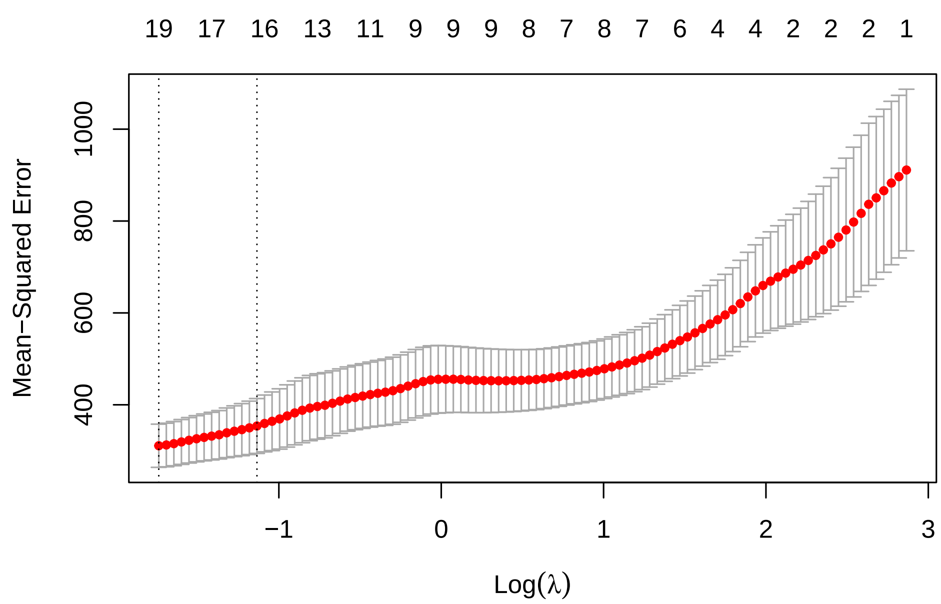
```
arrest_ts %>%
  summarize(fourier_series(index, k = 4, period = 12))
```

Let's merge these seasonal predictors with the changepoint trend model and use lasso to estimate all coefficients.

time	cos_1	sin_1	cos_2	sin_2	cos_3	sin_3	cos_4	sin_4
2014 Jan	0.866	0.500	0.5	0.866	0	1	-0.5	0.866
2014 Feb	0.500	0.866	-0.5	0.866	-1	0	-0.5	-0.866
2014 Mar	0.000	1.000	-1.0	0.000	0	-1	1.0	0.000
2014 Apr	-0.500	0.866	-0.5	-0.866	1	0	-0.5	0.866
2014 May	-0.866	0.500	0.5	-0.866	0	1	-0.5	-0.866
2014 Jun	-1.000	0.000	1.0	0.000	-1	0	1.0	0.000
2014 Jul	-0.866	-0.500	0.5	0.866	0	-1	-0.5	0.866
2014 Aug	-0.500	-0.866	-0.5	0.866	1	0	-0.5	-0.866
2014 Sep	0.000	-1.000	-1.0	0.000	0	1	1.0	0.000
2014 Oct	0.500	-0.866	-0.5	-0.866	-1	0	-0.5	0.866
2014 Nov	0.866	-0.500	0.5	-0.866	0	-1	-0.5	-0.866
2014 Dec	1.000	0.000	1.0	0.000	1	0	1.0	0.000

```
X_chgpts_season = cbind(
  X_chgpts,
  X_season = arrest_ts %>%
    summarize(fourier_series(index, k = 4, period = 12)) %>%
    as_tibble() %>% select(-time) %>% as.matrix()
)
```

```
library(glmnet)
set.seed(2023)
pen_fac = rep(1, ncol(X_chgpts_season)); pen_fac[1] = 0
fit_lasso2 = cv.glmnet(x=X_chgpts_season, y=Y, alpha = 1, penalty.factor = pen_fac)
plot(fit_lasso2)
```

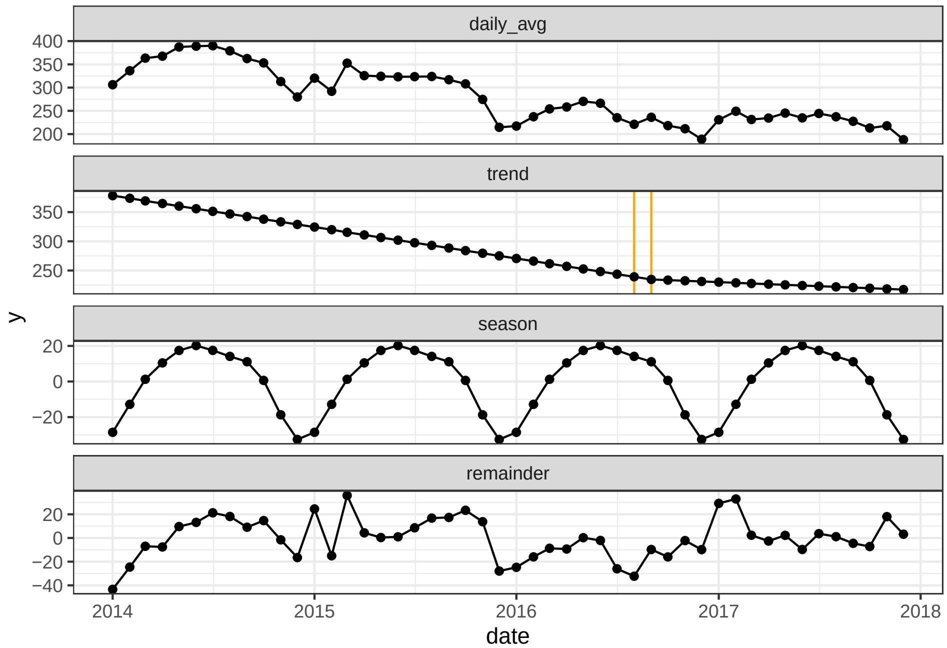



```
# Again, selecting lambda to minimize number of change points
lam = exp(.5)
coef(fit_lasso2, s = lam) %>% as.matrix %>% as_tibble(rownames = "var") %>%
  filter(s1 != 0 )
#> # A tibble: 9 x 2
#>   var                  s1
```

```
#>   <chr>            <dbl>
#> 1 (Intercept) 383.
#> 2 x             -4.48
#> 3 chgpt_32       0.00103
#> 4 chgpt_33       3.32
#> 5 cos_1        -23.7
#> 6 sin_1         -4.94
#> # ... with 3 more rows
```

And the plot

```
#: estimated change points
chgpts_est =
  coef(fit_lasso2, s = lam) %>%
  as.matrix %>% as_tibble(rownames = "var") %>%
  filter(s1 != 0, var != "(Intercept)") %>% pull(var) %>%
  str_remove("chgpt_") %>% as.integer %>% na.omit()

#: get seasonal and trend components
beta = coef(fit_lasso2, s = lam) %>% as.matrix %>% as_tibble(rownames = "var")

beta_season = beta %>%
  mutate(s1 = ifelse(str_detect(var, "sin|cos"), s1, 0))

beta_trend = beta %>%
  mutate(s1 = ifelse(str_detect(var, "sin|cos"), 0, s1))

components = tibble(
  season = cbind(1, X_chgpts_season) %*% beta_season$s1 %>% as.numeric,
  trend = cbind(1, X_chgpts_season) %*% beta_trend$s1 %>% as.numeric,
  fitted = predict(fit_lasso2, s = lam, newx = X_chgpts_season)[,1]
)

#: plot
arrest_ts %>%
  bind_cols(components) %>%
  mutate(remainder = daily_avg - fitted) %>%
  pivot_longer(
    c(season, trend, remainder, daily_avg), names_to = "component", values_to = "y"
  ) %>%
  mutate(chgpt = component == "trend" & index %in% !!chgpts_est) %>%
  ggplot(aes(date, y)) +
  geom_vline(data = . %>% filter(chgpt),
             aes(xintercept = date), color="orange") +
  geom_point() + geom_line() +
  facet_wrap(~factor(component, c("daily_avg", "trend", "season", "remainder")),
             ncol=1, scales = "free_y")
```



3.2.3 Holidays and Special Days

Special days, like holidays, often have observations that don't match the trend and seasonal patterns. In essence, these days produce outliers. But if we have a list of the special days, then we can easily estimate their effects.

Prophet let's the user specify special days with a table, e.g.,

Holiday	Country	Date
Thanksgiving	US	26 Nov 2015
Thanksgiving	US	24 Nov 2016
Thanksgiving	US	23 Nov 2017
Thanksgiving	US	22 Nov 2018
Christmas	*	25 Dec 2015
Christmas	*	25 Dec 2016
Christmas	*	25 Dec 2017
Christmas	*	25 Dec 2018

and a ridge penalized parameter is estimated for each unique holiday. Mathematically, predictor variables are simple indicators $Z(t) = [\mathbb{1}(t \in H_1), \mathbb{1}(t \in H_2), \ldots, \mathbb{1}(t \in H_p)]$ where H_j is the jth holiday.

$$\hat{H}(t) = Z(t)\hat{\beta}_Z$$

Some other thoughts:

1. The effects of a holiday may span several time periods. For example, the weekend following Thanksgiving or before Christmas may also have unusual observations. Instead of an indicator function, a function that spans several time periods and is a pre-specified shape can capture such patterns.

2. A technique called intervention analysis is used to model changes or shocks to a time series. Think of policy changes or major events. In the Chicago Arrest data, a new police superintendent started in April 2016 following protests (close to change point), the 2014 shooting of Michael Brown in Ferguson, MO inspired protests across the county, etc.

3.2.4 Peyton Manning Wikipedia

The Prophet Quickstart Guide illustrates usage on modeling the *log daily page views for the Wikipedia page of Peyton Manning*.

```
library(tidyverse)
url = 'https://raw.githubusercontent.com/facebook/prophet/main/examples/example_wp_log_peyton_manning
manning = read_csv(url)
head(manning)
```

```
#> # A tibble: 6 x 2
#>   ds             y
#>   <date>     <dbl>
#> 1 2007-12-10  9.59
#> 2 2007-12-11  8.52
#> 3 2007-12-12  8.18
#> 4 2007-12-13  8.07
#> 5 2007-12-14  7.89
#> 6 2007-12-15  7.78
```

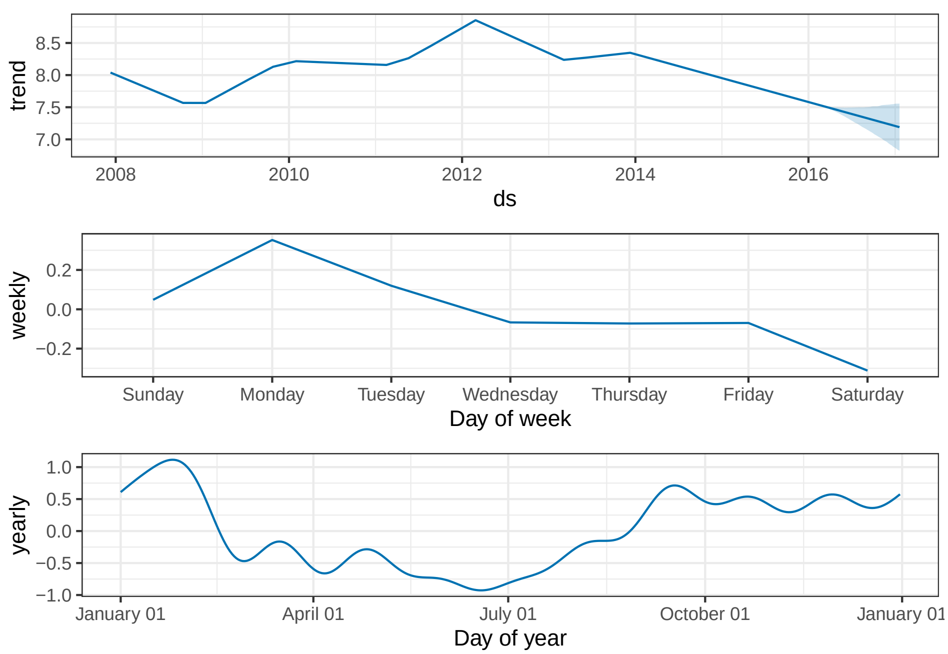
Default fitting

```
library(prophet)
fit = prophet::prophet(manning)         # fit the default prophet model
```

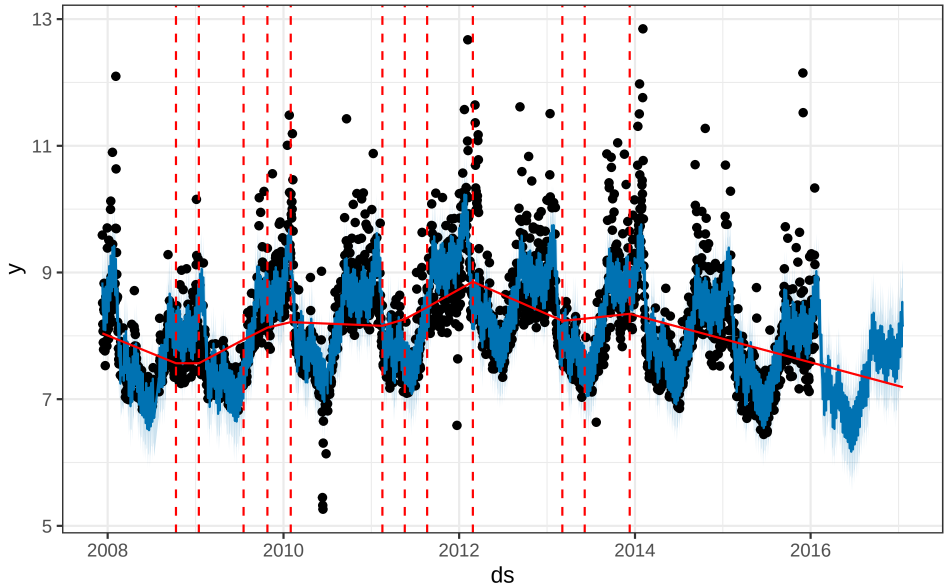
To forecast, we need to extend the data to future periods (here 365 days)

```
manning_future = make_future_dataframe(fit, periods = 365)
forecast = predict(fit, manning_future)
```

```
prophet_plot_components(fit, forecast)
```



```
plot(fit, forecast) + add_changepoints_to_plot(fit)
```



3.2.4.1 Holidays Holidays and other special days can be added. Here we add the dates associated with NFL playoffs and Superbowl. And we will add a special seasonality term that allows a different day-of-week effect when the NFL is in season.

```
#: playoffs
playoffs = tibble(
  holiday = 'playoff',
  ds = as.Date(c('2008-01-13', '2009-01-03', '2010-01-16',
                 '2010-01-24', '2010-02-07', '2011-01-08',
                 '2013-01-12', '2014-01-12', '2014-01-19',
                 '2014-02-02', '2015-01-11', '2016-01-17',
                 '2016-01-24', '2016-02-07')),
  lower_window = 0,
  upper_window = 1
)
#: superbowl
superbowls = tibble(
  holiday = 'superbowl',
  ds = as.Date(c('2010-02-07', '2014-02-02', '2016-02-07')),
  lower_window = 0,
  upper_window = 1
)
#: holidays
holidays = bind_rows(playoffs, superbowls)
```

```
#: NFL Season
is_nfl_season <- function(ds) {
  dates <- as.Date(ds)
  month <- as.numeric(format(dates, '%m'))
  return(month > 8 | month < 2)
}
manning$in_season = is_nfl_season(manning$ds)
```

```
#: specify the seasonality details, add holidays, add NFL season
fit_2 = prophet::prophet(fit = FALSE,
  yearly.seasonality = 10,
```

```
  weekly.seasonality = 3,
  holidays = holidays) %>%
  add_country_holidays("US") %>%
  add_seasonality("nfl_season", period = 7, fourier.order = 3, condition.name = "in_season") %>%
  fit.prophet(manning)
```

```
manning_future = make_future_dataframe(fit_2, periods = 365) %>%
  mutate(in_season = is_nfl_season(ds)) # need to add specials to future data
forecast = predict(fit_2, manning_future)
```

```
prophet_plot_components(fit_2, forecast)
```