ARIMA modeling of CO2 time series

Stone Jiang, Isaac Law, Mike Hardy

The purpose of this report is to apply regression and primarily ARIMA models to fitting and forecasting Carbon Dioxide (CO_2) levels from the 1960s to present (Feb. 2020). CO_2 levels since the 1960s display both a trend and seasonal component. Here, we perform:

1. rigorous Exploratory Data Analysis in R of both yearly and weekly data, including statistical testing for stationarity,
2. model selection based on residual analysis, in-sample fit, and pseudo-out-of-sample fit, for both yearly and weekly data, and for both non- and seasonally-adjusted data, and
3. forecasting and evaluation including confidence level generation.

The modeling package we use is fpp3 written by Rob J Hyndman and George Athanasopoulos:

https://otexts.com/fpp3

Yearly CO_2 data (of uptake in grass plants) is provided directly within R: https://www.rdocumentation.org/packages/datasets/versions/3.6.2/topics/CO2

Weekly CO_2 is provided by the National Oceanic and Atmospheric Administration (NOAA) and is available for free use by the public:

https://www.esrl.noaa.gov/gmd/ccgg/trends

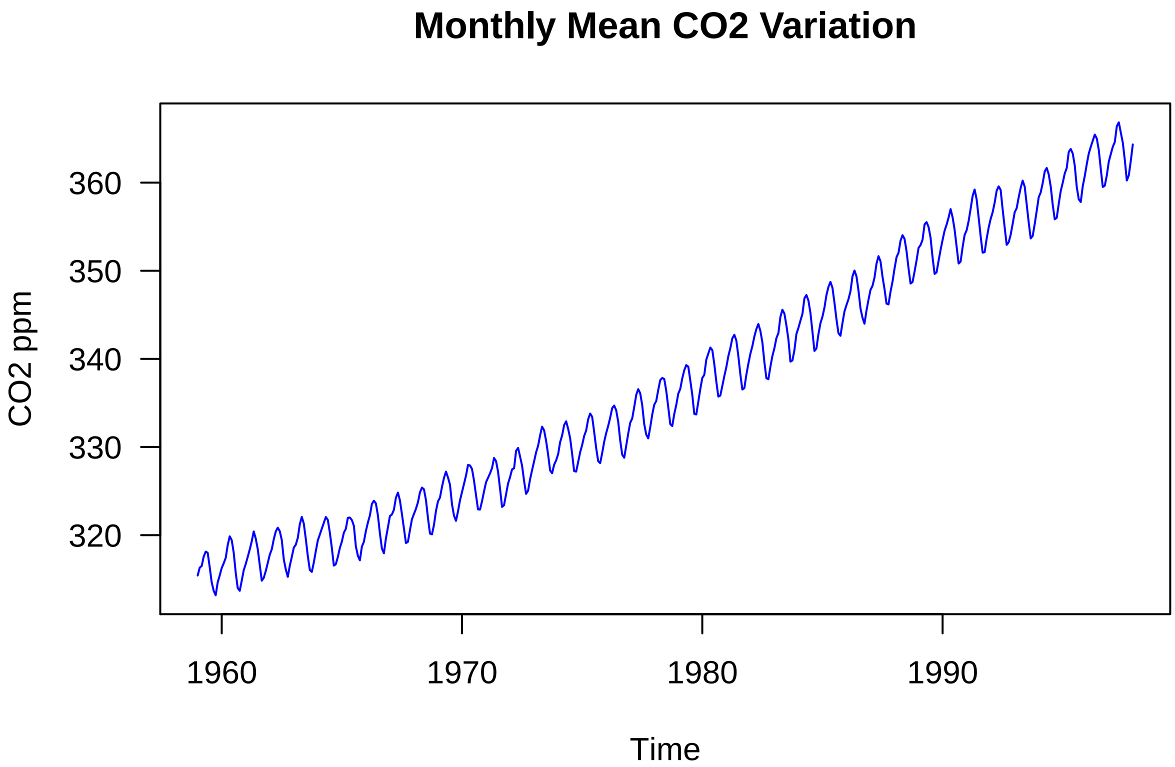
The Keeling Curve

In the 1950s, the geochemist Charles David Keeling observed a seasonal pattern in the amount of carbon dioxide present in air samples collected over the course of several years. He was able to attribute this pattern to the difference in the amount of land area and vegetation cover between the northern and southern hemipsheres, and the resulting variation in global rates of photosynthesis as the hemispheres' seasons alternated throughout the year.

In 1958 Keeling began continuous monitoring of atmospheric carbon dioxide concentrations from the Mauna Loa Observatory in Hawaii and soon observed a trend increase carbon dioxide levels in addition to the seasonal cycle. He was able to attribute this trend increase to growth in global rates of fossil fuel combustion. This trend has continued to the present.

The `co2` data set in R's `datasets` package (automatically loaded with base R) is a monthly time series of atmospheric carbon dioxide concentrations measured in ppm (parts per million) at the Mauna Loa Observatory from 1959 to 1997. The curve graphed by this data is known as the 'Keeling Curve'.

```r
plot(co2, ylab = expression("CO2 ppm"), col = 'blue', las = 1)
title(main = "Monthly Mean CO2 Variation")
```


Monthly Mean CO2 Variation
CO2 ppm
320
330
340
350
360
1960
1970
1980
1990
Time

Part 1: Exploratory Data Analysis

Formatting of inputs

We first perform EDA to discover the nature of the data series. We find no missing data, and that the data is in a `ts()` format, which we then convert into a `tsibble` object format for easier manipulation.

```
library(fpp3)
library(lubridate)
library(forecast)
library(tseries)
library(tidyr)
library(dplyr)
library(skimr)
library(patchwork)

select <- dplyr::select

#convert data to tsibble
co2.ts <- as_tsibble(co2)
head(co2.ts)
```

```
## # A tsibble: 6 x 2 [1M]
##      index value
##      <mth> <dbl>
## 1 1959 Jan  315.
## 2 1959 Feb  316.
## 3 1959 Mar  316.
## 4 1959 Apr  318.
## 5 1959 May  318.
## 6 1959 Jun  318
```

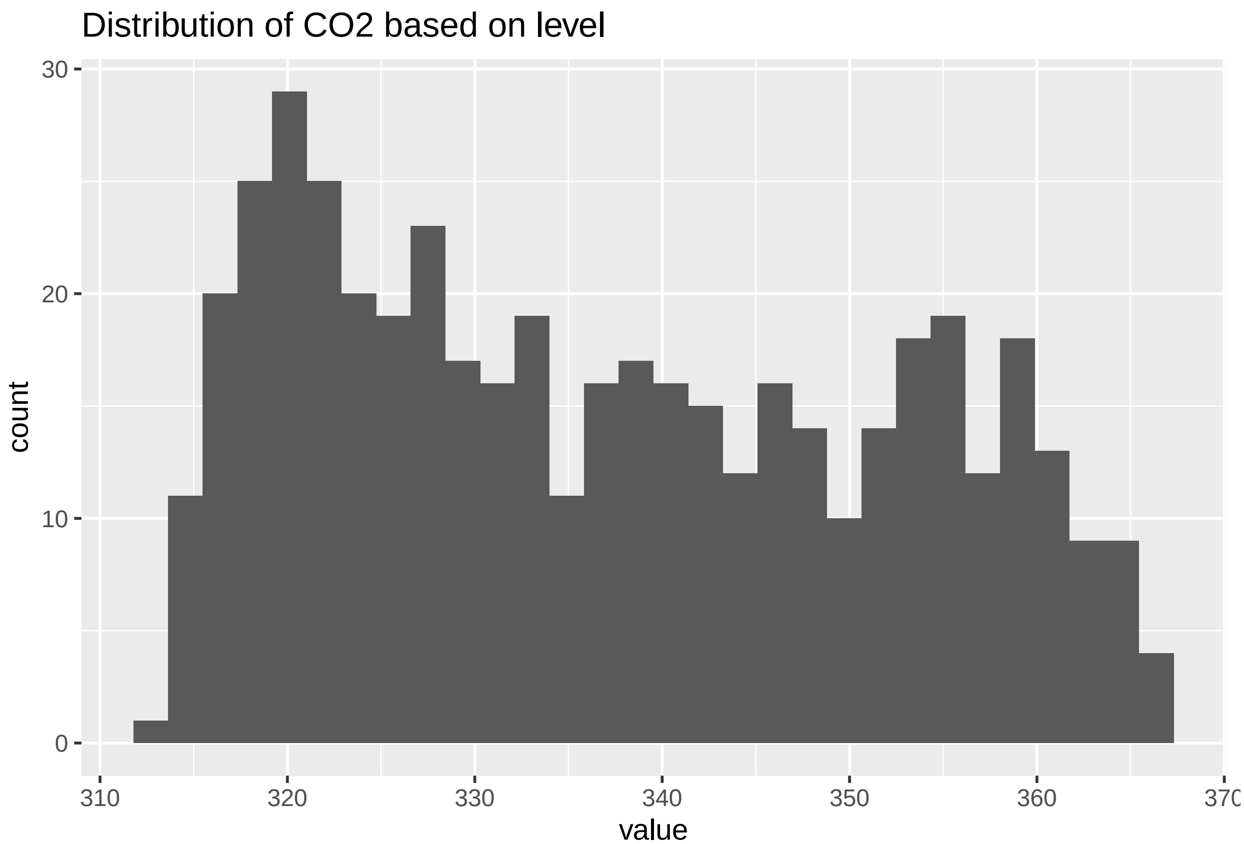
```
tail(co2.ts)
```

```
## # A tsibble: 6 x 2 [1M]
##      index value
##      <mth> <dbl>
## 1 1997 Jul  365.
## 2 1997 Aug  363.
## 3 1997 Sep  360.
## 4 1997 Oct  361.
## 5 1997 Nov  362.
## 6 1997 Dec  364.
```

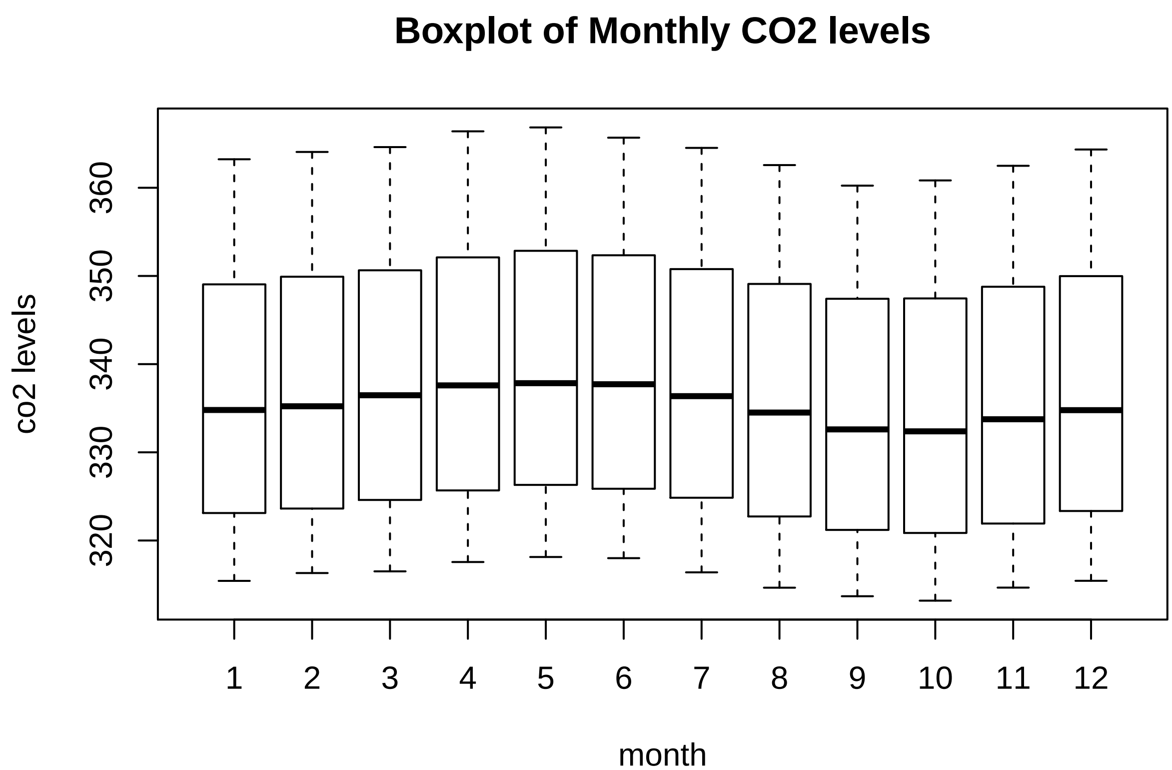
```
glimpse(co2.ts)
```

```
## Observations: 468
## Variables: 2
## $ index <mth> 1959 Jan, 1959 Feb, 1959 Mar, 1959 Apr, 1959 May, 1959 J...
## $ value <dbl> 315.42, 316.31, 316.50, 317.56, 318.13, 318.00, 316.39, ...
```

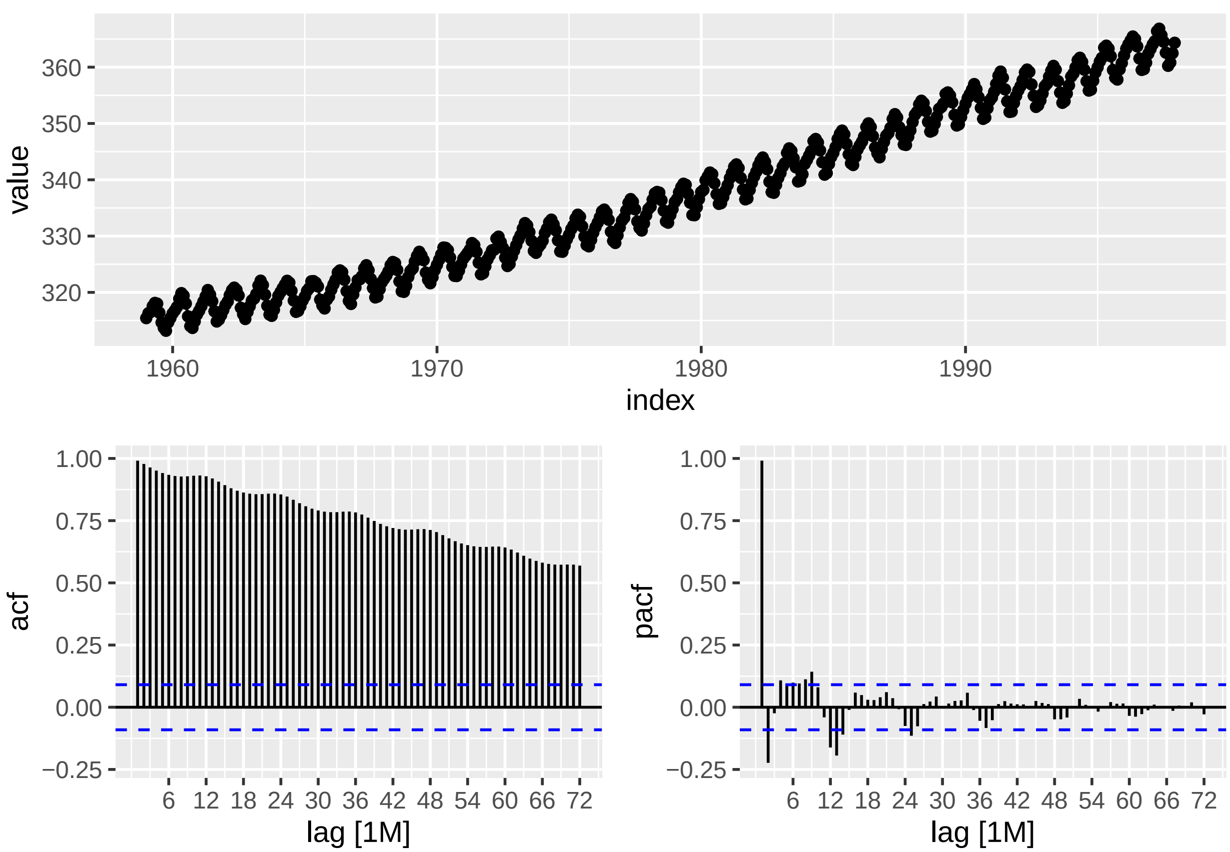
```
#Plot the histogram of the dataset
co2.ts %>% ggplot(aes(x = value)) + geom_histogram(bins=30) + ggtitle("Distribution of CO2 base
```




```
#Plot boxplot of the dataset grouped based on season
boxplot(co2 ~ cycle(co2), main="Boxplot of Monthly CO2 levels", xlab="month", ylab="co2 levels"
```




```
#Plot the time series, ACF, PACF, and histogram
co2.ts %>% gg_tsdisplay(value, plot_type = 'partial', lag_max = 72)
```



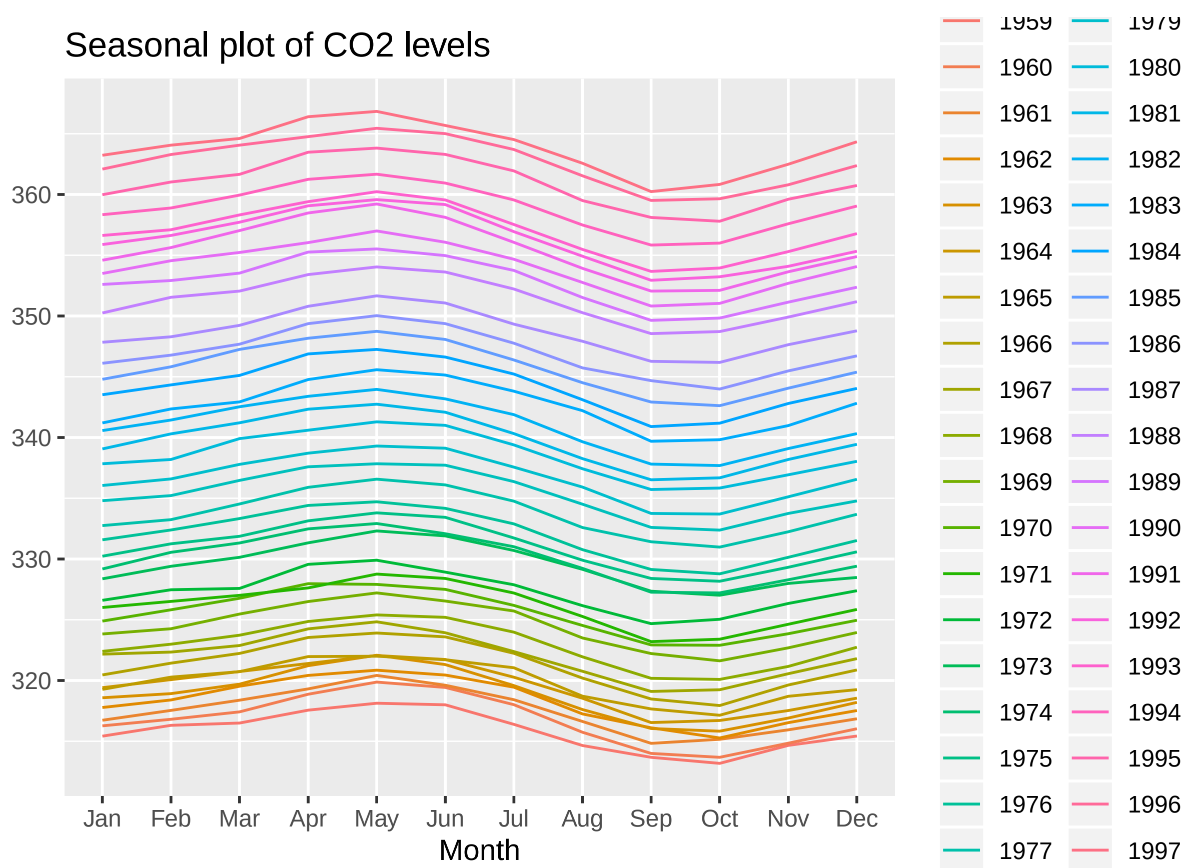
As suggested in the problem statement, there is a clear upward trend in CO_2 time series plot that looks mostly linear, although a higher order models could possibly generate a better in-sample fit. There is also strong seasonality as revealed by the zig-zag shape of the data and the boxplot. The boxplot shows that CO_2 levels are highest in the late spring early summer months (April - June) and lowest in late autumn. The size of the boxes for each month is similar, implying similar variation in CO_2 levels. The variance appears mostly constant, with only slightly larger zig-zigs around 1990 than 1970. The histogram reveals that, as we might expect, we do not have a normal distribution of values. However, there are no extreme outliers/irregular elements.

The `ACF` declination appears to slow, with apparent cyclic patterns, while the `PACF` shows significant partial autocorrelations at lags 1 and 2, as it oscillates toward 0. While some of the values are statistically significant, most values are not statistically significant past lag 2. We will need to explore the stability of this variance, the extent of the autocorrelations, the effects of seasonality on the model, and the growth rate of the trend.

Exploration of Variance

Since we have monthly data with what appears to be an annual seasonality, we can also visualize the annual relationship with a seasonal plot that shows variation across each month.

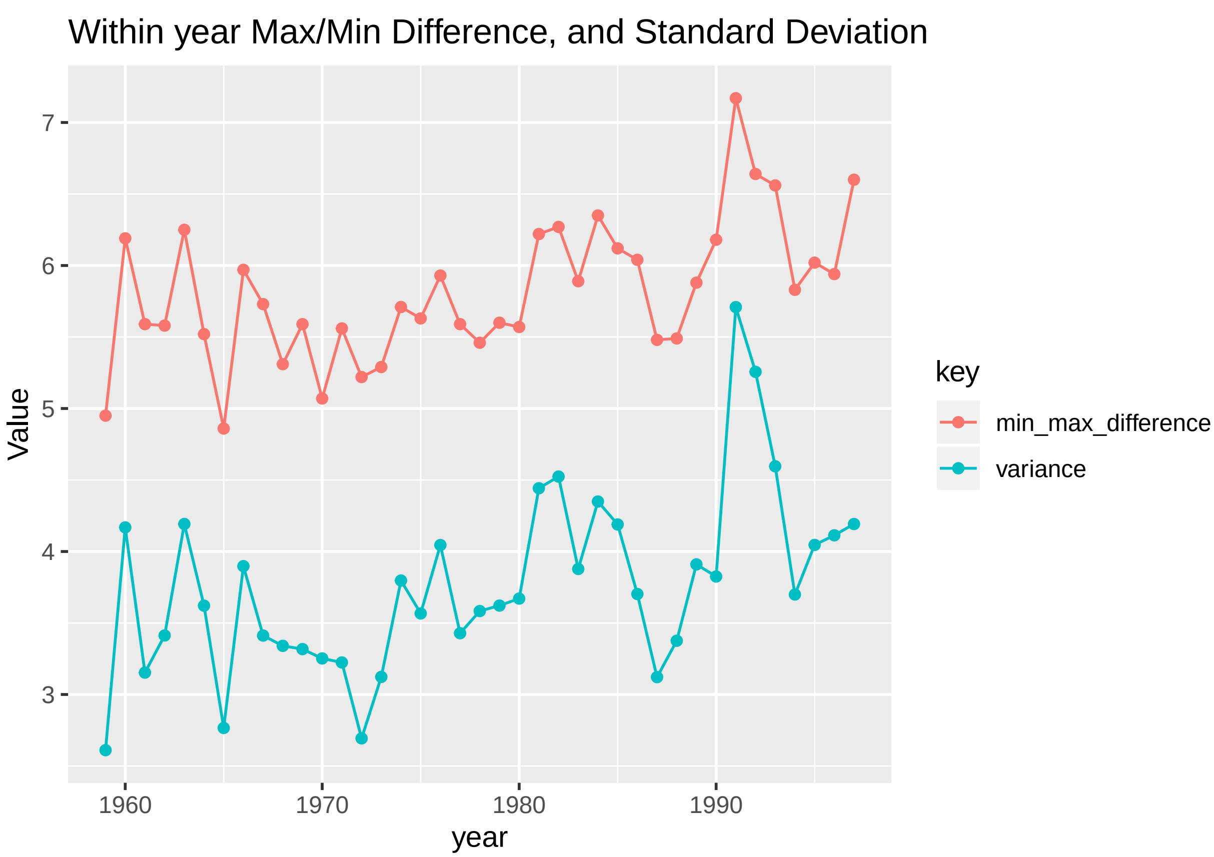
```r
ggseasonplot(co2) +
  ggtitle("Seasonal plot of CO2 levels")
```



The seasonality patterns appear constant across every month, with local maxima in the early Summer months May and June and and local minima around October as we saw in the boxplot, although the variation is small compared to the trend levels between years. The increase from year to year shows the same monotonically increasing trend seen in the Keeling Curve, as demonstrated by the fact that between Nov - Jan, the lines are increasing.

To look at stability of the variance, we will plot a series of the difference between the maximum and minimum values within each year, as well as the sample variance within each year. The former gives us a sense of the range of values taken on by each year, the latter is a direct a direct method of whether the variance is stable across years within our series.

```
co2.ts$year <- co2.ts$index %>% year()

#index_by is the equivalent of group_by in the temporal context
min_max_values <- co2.ts %>% index_by(year) %>% summarise(minv = min(value), maxv = max(value)
min_max_values <- min_max_values %>% mutate(min_max_difference = maxv - minv)

min_max_ggplot <- gather(min_max_values %>% dplyr::select(-minv, -maxv))
ggplot(data = min_max_ggplot, aes(x = year, y = value, color = key)) +
  geom_point() +
  geom_line()+
  labs(y = "Value",
       title = "Within year Max/Min Difference, and Standard Deviation")
```



The variance overall seems relatively stable. The only exception is in the year 1991, where there was a spike in the max CO_2 levels. This resulted in a larger variance, which comes back down in the following years. Overall, there should not be any concerns of increasing variance.

We can treat the sample variance series as a time series itself, representing the variance in the samples within each year. To statistically determine whether this series is stationary in the mean, we will apply the `Augmented Dickey-Fuller Test`:

```
adf.test(min_max_values$variance)
```

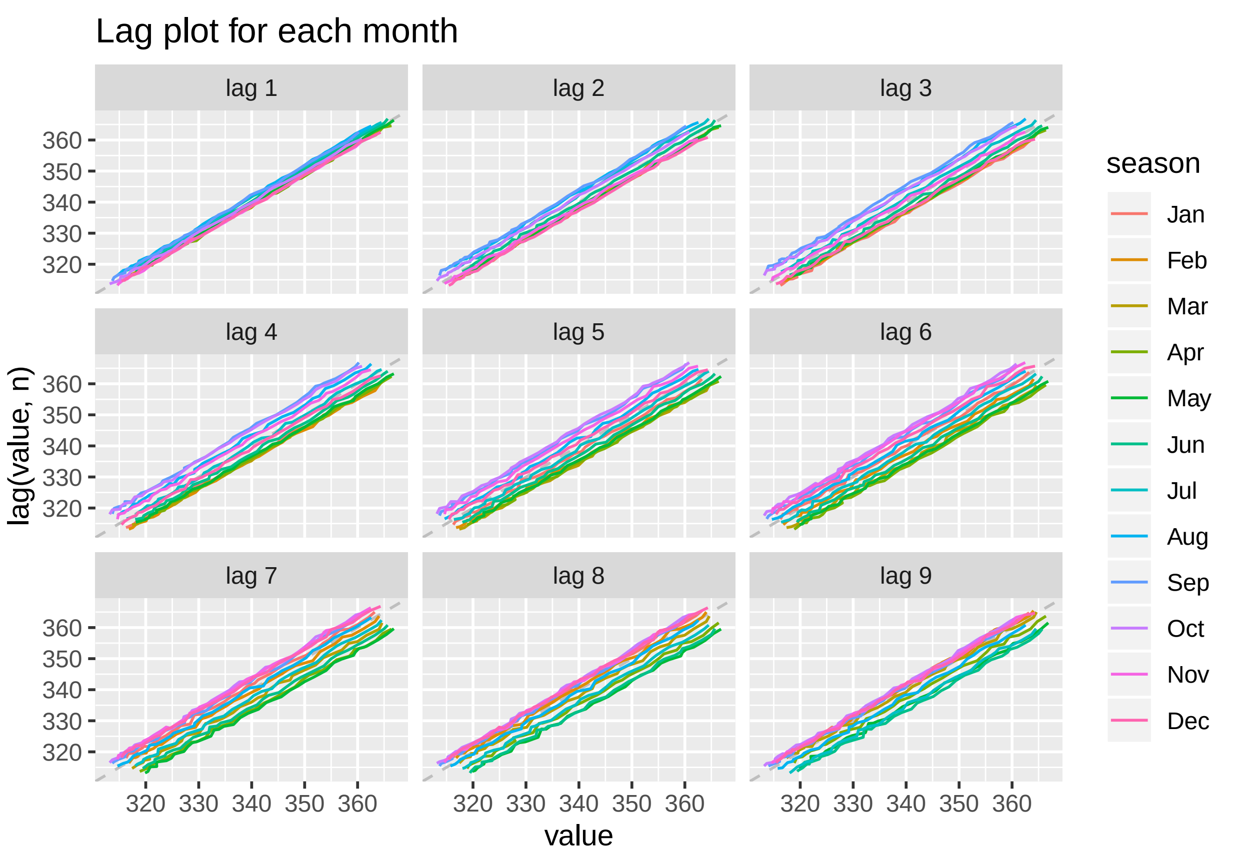
```
##
##  Augmented Dickey-Fuller Test
##
## data:  min_max_values$variance
## Dickey-Fuller = -3.8987, Lag order = 3, p-value = 0.02397
## alternative hypothesis: stationary
```

Since p-value < 0.05, we reject Ho that the model is non-stationary and assume that the variance is stationary.

Exploration of Autocorrelational Relationships

We can look at the seasonal trend further by using a lag plot. The lag plot show CO_2 levels at time t, versus CO_2 levels at time $t - k$, for k ranging from $1 - 9$.

```
gg_lag(co2.ts, y = value) +
  ggtitle("Lag plot for each month")
```

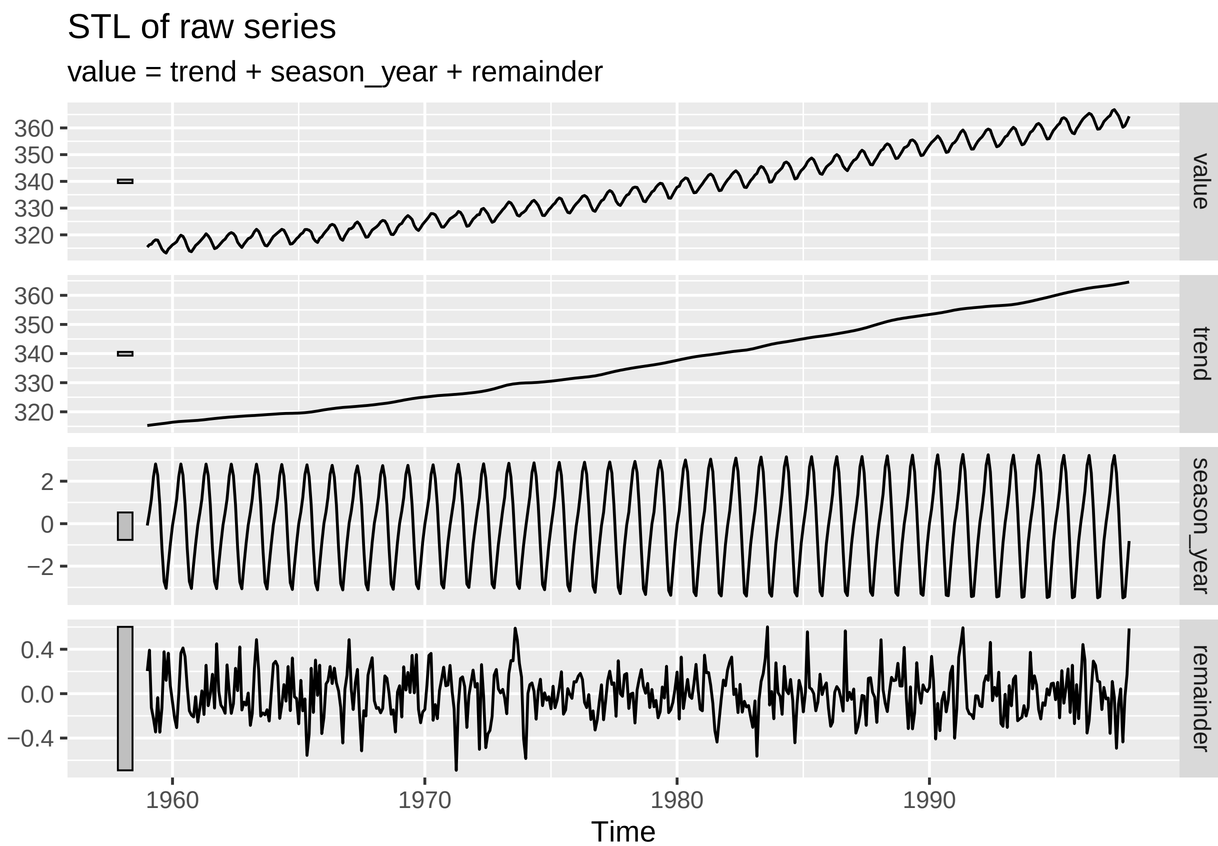


We can see that for all lags, the relationship between the y_t vs y_{t-k} for every month is appriximately linear. This bodes well for a roughly linear trend in the data. If the data were exponential for example, we would expect that for larger CO_2 levels corresponding to more recent years years, the slope would also increase. The spread for the different months is a result of shared CO_2 levels across months along with the oscillating nature of the seasonality.

Exploration of Seasonal Relationships

We next break the time series into a trend, seasonal, and remainder components. We will perform `STL` decomposition using default values for trend-cycle and seasonal windows. These are smoothing parameters that govern how quickly each of the components change. The seasonal window is by default set to 13, and trend-cycle window is determined based on the periodicity of the season.

```
co2.decomp <- co2.ts %>% model(STL(value)) %>%
  components()
co2.decomp %>% autoplot() + labs(title = 'STL of raw series', x = 'Time')
```



The seasonal component is periodic and the amplitude is increasing slightly as year increases. The trend is relatively smooth, suggesting that a simple linear regression could be adequate to model this trend. However, the growth in the trend appears to be non-constant, since the slope of the trend appears to be increasing slightly as time increases. This suggests that higher order polynomial terms might be required. There does not appear to be any irregular observations based on the residual series, which appears to be stationary overall with no extreme outliers.

We observe a slight increase in variance over time in our seasonal data; however, the mean looks stable. We can test for mean stationarity of the seasonal component and the remainder with the `Augmented Dickey-Fuller Test` at $\alpha = 0.05$:

```
adf.test(co2.decomp$season_year)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  co2.decomp$season_year
## Dickey-Fuller = -31.982, Lag order = 7, p-value = 0.01
## alternative hypothesis: stationary
```

```
adf.test(co2.decomp$remainder)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  co2.decomp$remainder
## Dickey-Fuller = -9.8633, Lag order = 7, p-value = 0.01
## alternative hypothesis: stationary
```

We have evidence to reject the null hypothesis and will assume that no transformation is required to address the variance.

Exploration of the Growth Rate

Though the growth rate looks mostly linear, visual inspections are often ambiguous. For example, one can argue that the growth rate seems to increase after around 1970, that the flattest region seems to be before 1965, or that the rate seems to slow in the early 1990s.

For a high-level view, we will calculate the percentage growth in the first 5 years, and compare with the growth rate over the entire series. The growth rate for the first 5 years is:

```
m12y1964 <- co2.ts %>% filter(month(co2.ts$index) == 12, year == 1964) %>% dplyr::select(value
m1y1959 <- co2.ts %>% filter(month(co2.ts$index) == 1, year == 1959) %>% dplyr::select(value)
pct.growth.5years <- (m12y1964$value - m1y1959$value) / m1y1959$value * 100
pct.growth.5years
```

```
## [1] 0.9923277
```

Now, we calculate what the growth would be at the end of the series, Dec. 1997, if this 5-year growth rate were constant. The data spans a total of 33 years, or 7.6 periods of 5-year growth. So:

```
prj.growth <- ((1+pct.growth.5years/100)^{7.6}-1)*100
prj.growth
```

```
## [1] 7.793284
```

The actual growth rate over the entire period is:

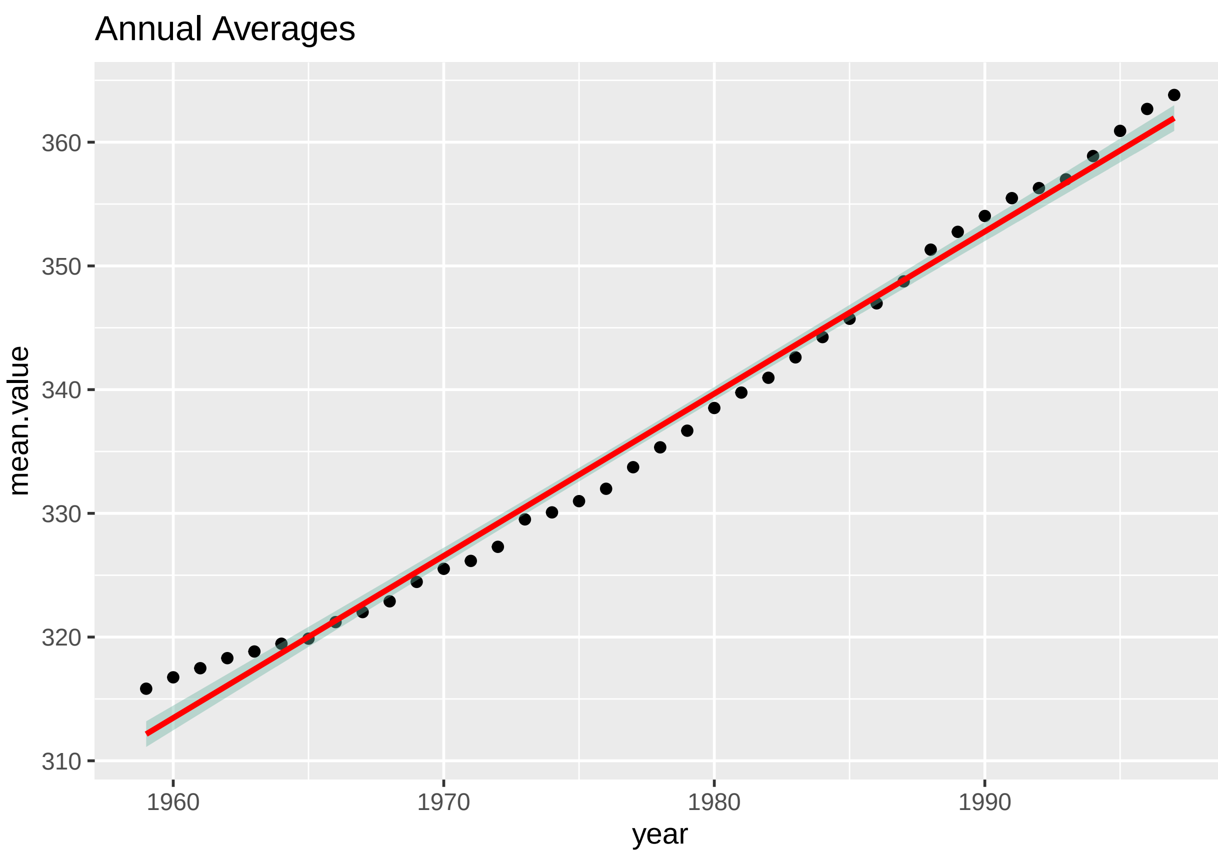
```
m12y1997 <- co2.ts %>% filter(month(co2.ts$index) == 12, year == 1997) %>% dplyr::select(value

pct.growth.total <- (m12y1997$value - m1y1959$value) / m1y1959$value * 100
pct.growth.total
```

```
## [1] 15.50948
```

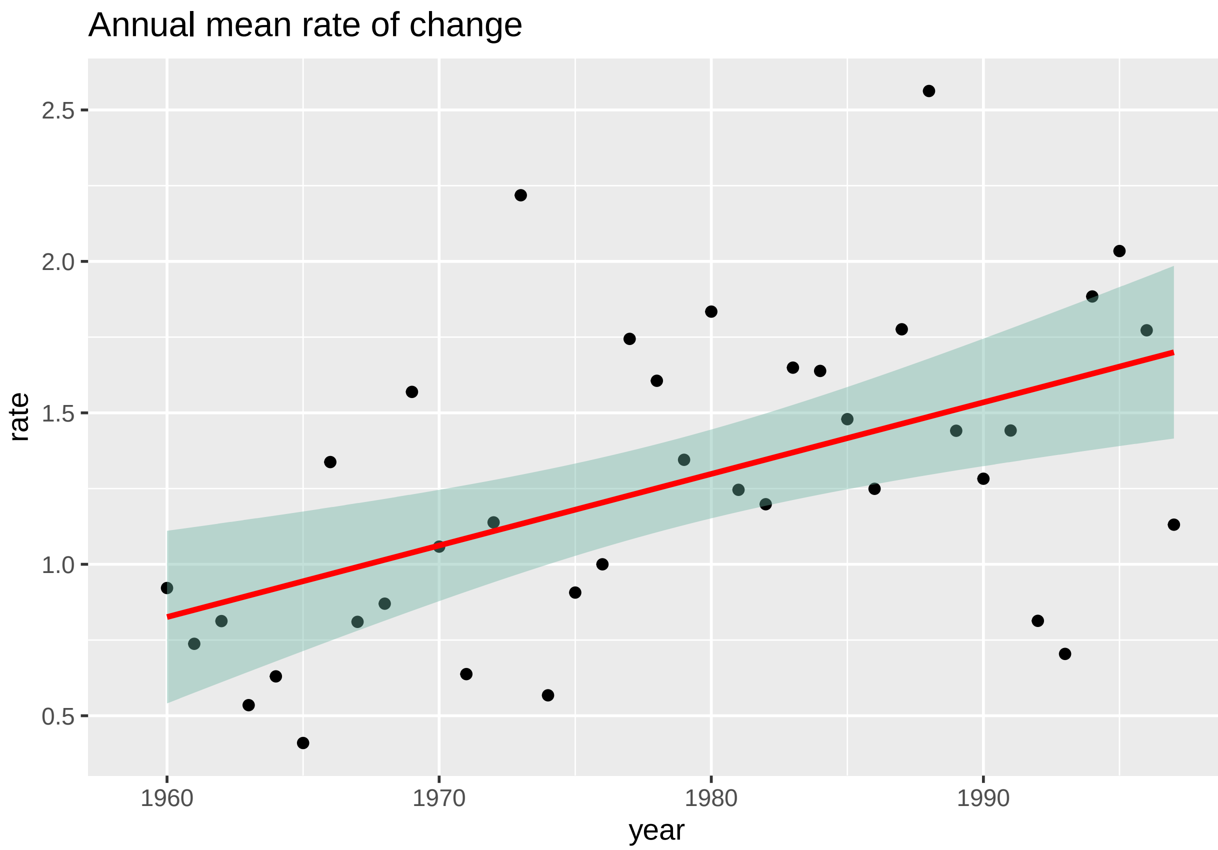
The actual growth rate over the entire period is almost double the 5-year growth rate projected over the entire series. This is evidence that the growth rate is not constant, but appears to increase with increasing time.

We can do a more robust inspection of these trends by plotting rates of change against time by taking the lag difference.

```
# Further examine the trend
year.average <- co2.ts %>%
  index_by(year) %>%
  summarise(mean.value = mean(value)) %>%
  mutate(rate = difference(mean.value)) %>%
  mutate(rate2 = difference(rate))


ggplot(year.average, aes(x=year, y=mean.value)) +
  geom_point() +
  geom_smooth(method=lm , color="red", fill="#69b3a2", se=TRUE) +
  ggtitle("Annual Averages")
```




```
ggplot(year.average, aes(x=year, y=rate)) +
  geom_point() +
  geom_smooth(method=lm , color="red", fill="#69b3a2", se=TRUE) +
  ggtitle("Annual mean rate of change")
```




```r
ggplot(year.average, aes(x=year, y=rate2)) +
  geom_point() +
  geom_smooth(method=lm , color="red", fill="#69b3a2", se=TRUE) +
  ggtitle("2nd Order rate of change")
```

The annual rate of change is mildly increasing although the variation between years is high, and the rate of this increase (2nd order rate of change) is relatively constant with a very slight downward slope. At this point, we have reason to suspect that there is at a quadratic relationship. A cubic model will also be explored.

Part 2: Fitting Linear Models

Using linear models, we examine the relationship between the time index and CO_2.

Defining Linear Models

We can model a linear regression model with TSLM. The linear model we are trying to fit is the following:

$$y = \beta_0 + \beta_1 * t + \epsilon$$

The quadratic model we are trying to fit is:

$$y = \beta_0 + \beta_1 * t + \beta_2 * t^2 + \epsilon$$

And finally, if we were to test a cubic model:

$$y = \beta_0 + \beta_1 * t + \beta_2 * t^2 + \beta_3 * t^3 + \epsilon$$

Fitting Linear Models

```
#Generate a new column that is row number and represents time index variable
co2.ts <- co2.ts %>% mutate(time_index = row_number())
linear.fit <- co2.ts %>% model(TSLM(value ~ trend()))

linear.fit %>% report()
```

```
## Series: value
## Model: TSLM
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -6.039885 -1.947575 -0.001671  1.911271  6.514852
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 3.115e+02  2.424e-01  1284.9   <2e-16 ***
## trend()     1.090e-01  8.958e-04   121.6   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.618 on 466 degrees of freedom
## Multiple R-squared: 0.9695,  Adjusted R-squared: 0.9694
## F-statistic: 1.479e+04 on 1 and 466 DF, p-value: < 2.22e-16
```

```
#Generate a quadratic fit
quad.fit <- co2.ts %>% model(TSLM(value ~ trend() + I(trend()^2)))
quad.fit %>% report()
```

```
## Series: value
## Model: TSLM
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.0195 -1.7120  0.2144  1.7957  4.8345
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.148e+02  3.039e-01 1035.65   <2e-16 ***
## trend()      6.739e-02  2.993e-03   22.52   <2e-16 ***
## I(trend()^2) 8.862e-05  6.179e-06   14.34   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.182 on 465 degrees of freedom
## Multiple R-squared: 0.9788,  Adjusted R-squared: 0.9787
## F-statistic: 1.075e+04 on 2 and 465 DF, p-value: < 2.22e-16
```

```r
#Generate a cubic fit
cubic.fit <- co2.ts %>% model(TSLM(value ~ trend() + I(trend()^2)+ I(trend()^3)))

cubic.fit %>% report()
```

```
## Series: value
## Model: TSLM
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.5786 -1.7299  0.2279  1.8073  4.4318
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)   3.163e+02  3.934e-01 804.008  < 2e-16 ***
## trend()       2.905e-02  7.256e-03   4.004 7.25e-05 ***
## I(trend()^2)  2.928e-04  3.593e-05   8.149 3.44e-15 ***
## I(trend()^3) -2.902e-07  5.036e-08  -5.763 1.51e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.11 on 464 degrees of freedom
## Multiple R-squared: 0.9802,  Adjusted R-squared: 0.9801
## F-statistic:  7674 on 3 and 464 DF, p-value: < 2.22e-16
```

```r
data.frame(Linear = t(glance(linear.fit)), Quadratic = t(glance(quad.fit)), Cubic = t(glance(cu
```

```
##                                  Linear                                  Quadratic
## .model        TSLM(value ~ trend()) TSLM(value ~ trend() + I(trend()^2))
## r_squared                  0.9694645                                 0.9788298
```

```
## adj_r_squared              0.969399                            0.9787387
## sigma2                      6.85425                            4.762267
## statistic                  14794.94                             10749.9
## p_value                           0                                   0
## df                                2                                   3
## log_lik                    -1113.48                           -1027.768
## AIC                        904.8343                             735.409
## AICc                       904.8861                            735.4954
## BIC                        917.2798                            752.0029
## CV                         6.888531                            4.794167
## deviance                    3194.08                            2214.454
## df.residual                     466                                 465
## rank                              2                                   3
##                                                        Cubic
## .model        TSLM(value ~ trend() + I(trend()^2) + I(trend()^3))
## r_squared                                          0.9802437
## adj_r_squared                                      0.9801159
## sigma2                                              4.453789
## statistic                                           7674.043
## p_value                                                    0
## df                                                         4
## log_lik                                            -1011.593
## AIC                                                 705.0603
## AICc                                                705.1902
## BIC                                                 725.8026
## CV                                                  4.490846
## deviance                                            2066.558
## df.residual                                              464
## rank                                                       4
```

```
#generate augmented fit
augmented.fit <- augment(linear.fit) %>% dplyr::select(index, value, linear.fitted = .fitted)
augmented.fit$quad.fitted <- augment(quad.fit)$.fitted
augmented.fit$cubic.fitted <- augment(cubic.fit)$.fitted
```

Interestingly, each of the models suggests that, when fit without seasonality, all of the variables are statistically significant: even the cubic value. If we were to use `AIC`, `AICc`, or `BIC` to guide our model selection, the cubic would be the model selected. However, there is nothing in nature that leads us to suspect that this model should be cubic. It is possible that utilizing this could contribute to over-fitting the model; further exploration is required.

The calculations result in the following model fits. **Note that t = 1 corresponds to the date of the first data point: 1959 Jan**

Linear

$$y = 311.5 + 0.109 * t$$

Quadratic

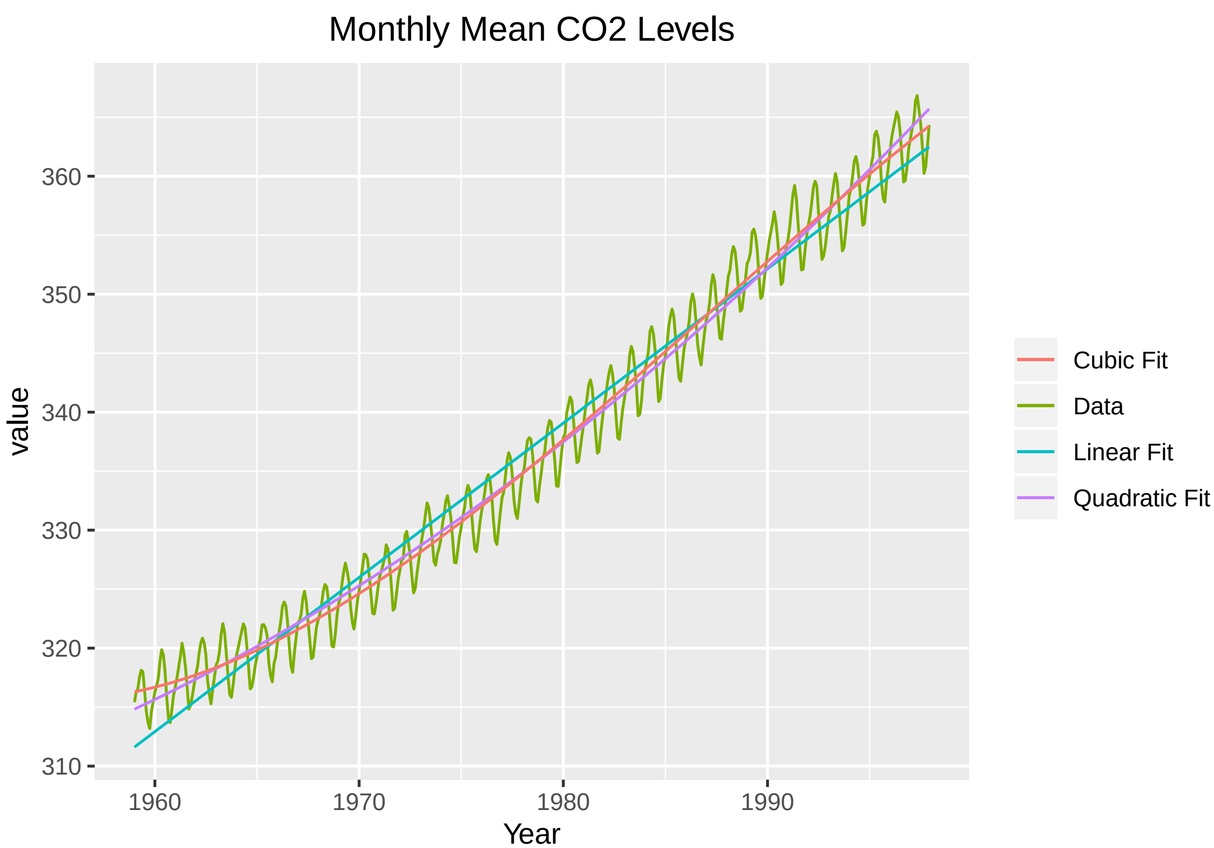
$$y = 314.8 + 0.0674 * t + 8.862 * 10^{-5} * t^2$$

Cubic

$$y = 316.3 + 0.02905 * t + 2.928 * 10^{-4} * t^2 - 2.902 * 10^{-7} * t^3$$

The fits are overlaid below:

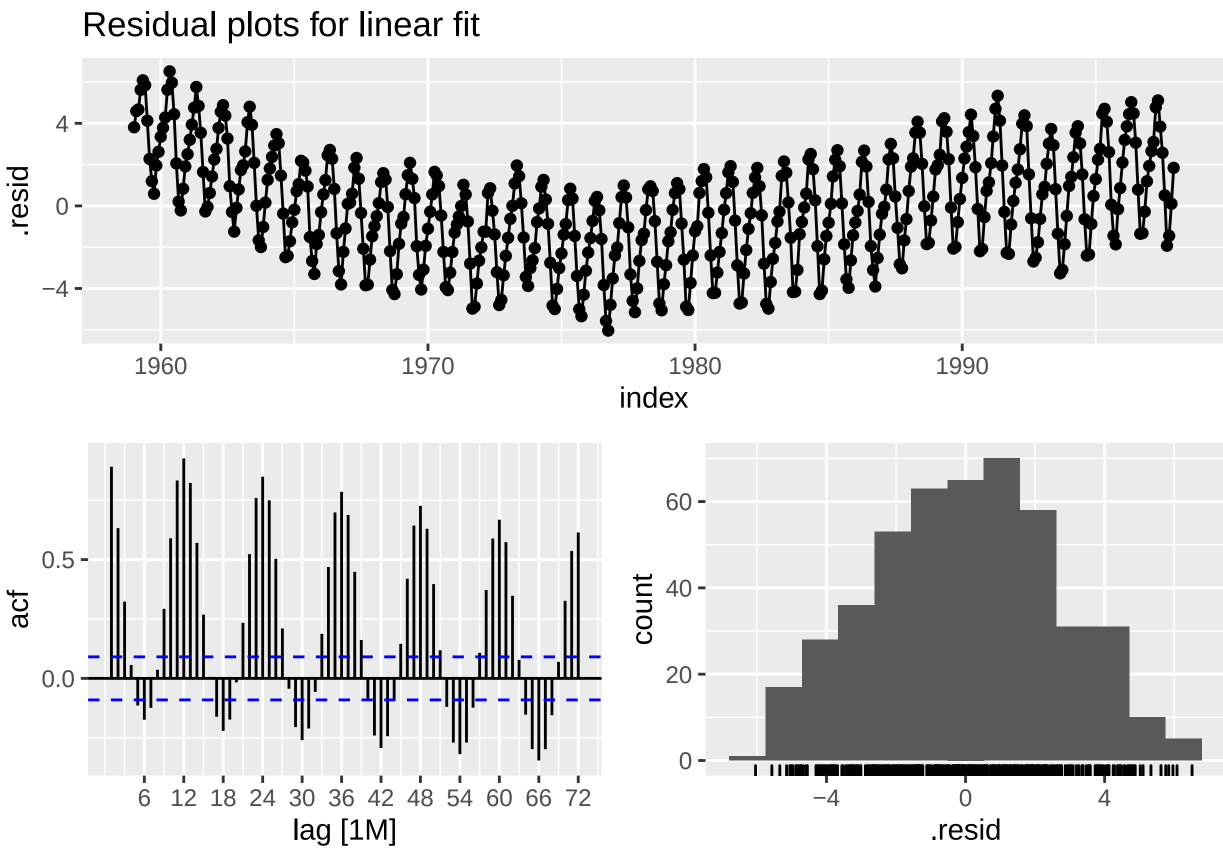
```r
#Generate a plot of fitted values
augmented.fit %>%
  ggplot(aes(x = index)) +
  geom_line(aes(y = value, colour = "Data")) +
  geom_line(aes(y = linear.fitted, colour = "Linear Fit")) +
  geom_line(aes(y = quad.fitted, colour = "Quadratic Fit")) +
  geom_line(aes(y = cubic.fitted, colour = "Cubic Fit")) +
  xlab("Year") + ylab("value") +
  ggtitle("Monthly Mean CO2 Levels") +
  theme(plot.title = element_text(hjust = 0.5))+
  guides(colour=guide_legend(title=NULL))
```



The quadratic fit appears to be a better fit than the linear fit, as it passes through the middle of the seasonal variation at almost every point. The linear fit clearly underfits the trend as toward the ends of the year intervals, the fit is quite poor. From visual inspection alone, the cubic is likely the best fit, but we remain skeptical of its relevance. We can now look at the residuals of these fits.

Linear Residuals

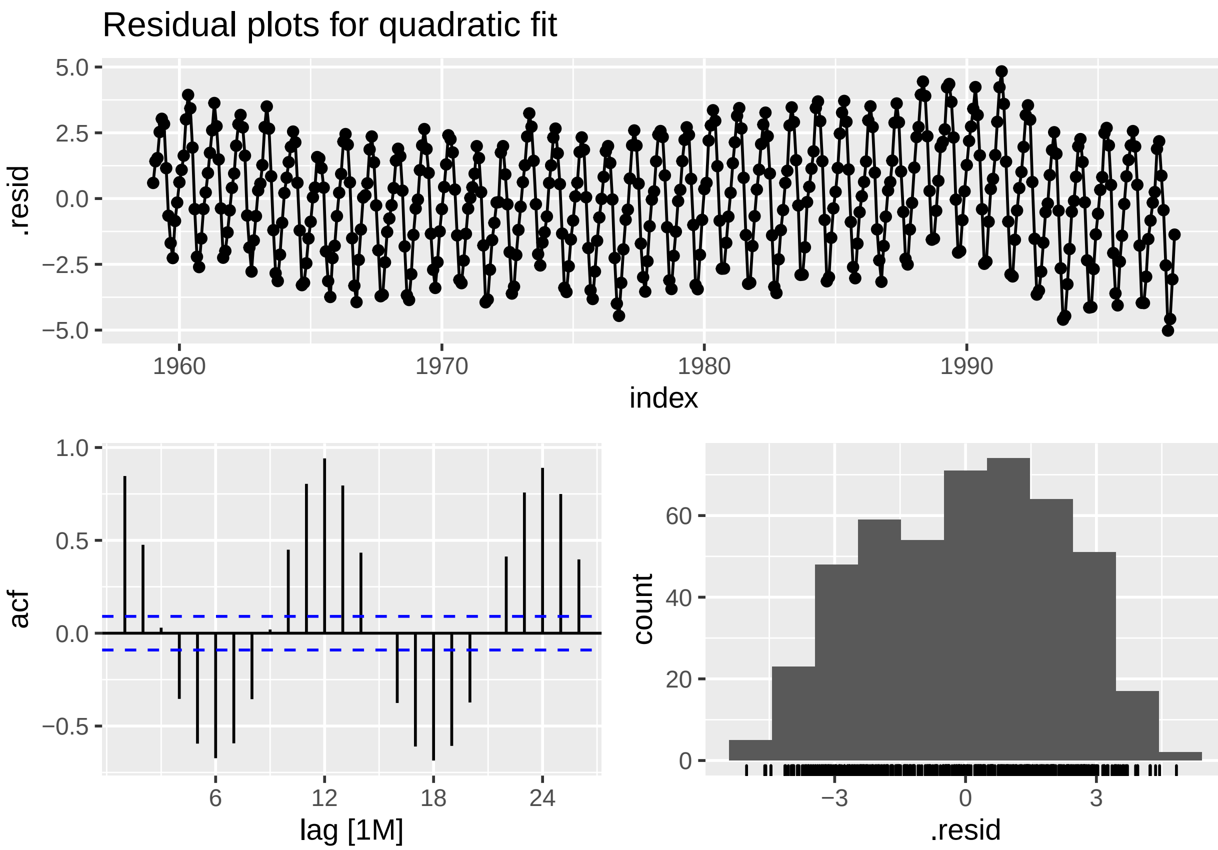
```r
gg_tsresiduals(linear.fit, lag_max = 72) +
  labs(title = "Residual plots for linear fit")
```



There is a clear pattern in the residuals versus time plot, which curves down in the middle of the fit. This indicates that the linear fit is poor, and the data might require a transformation, or higher order polynomial is required to fit the data. The ACF shows large autocorrelations that are oscillating between negative and positive, so the residuals are clearly not white noise.

Quadratic Residuals

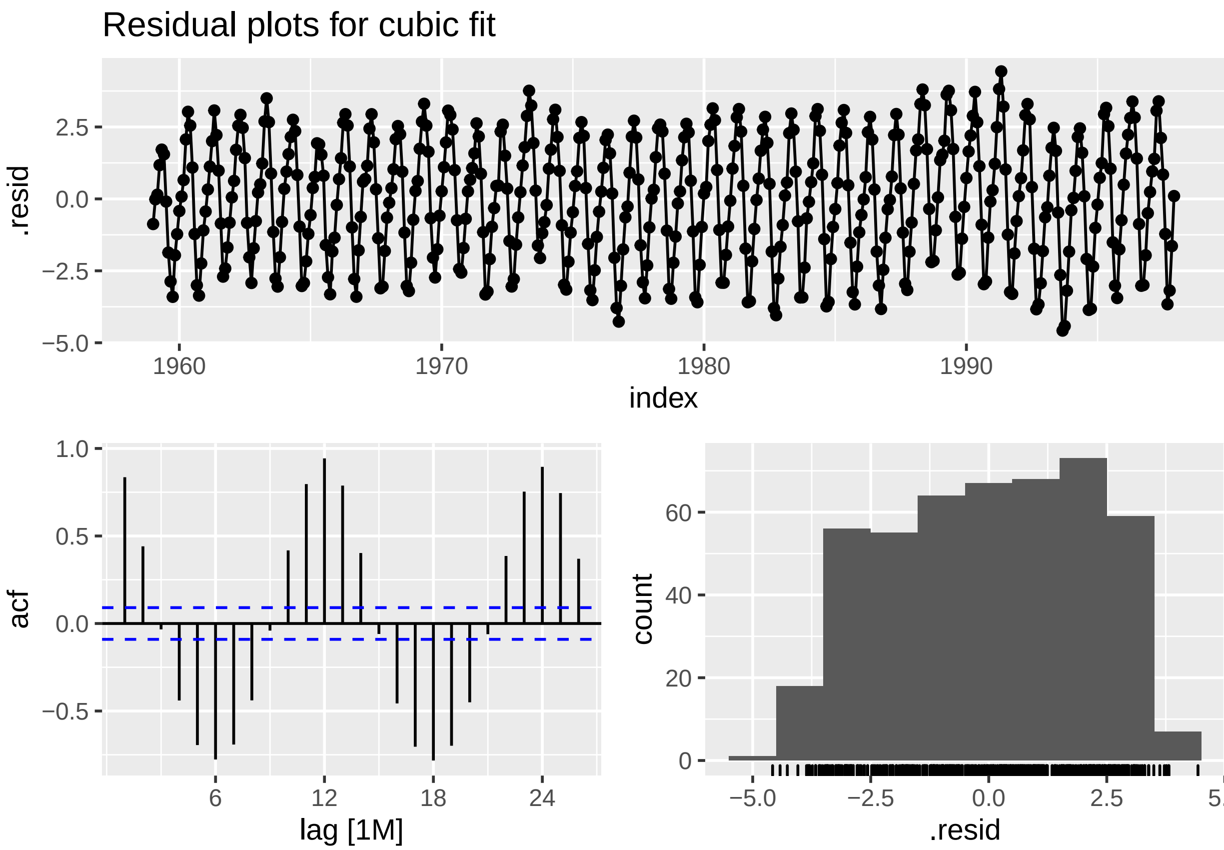
```
gg_tsresiduals(quad.fit) +
  labs(title = "Residual plots for quadratic fit")
```



The residual plot looks better as it is relatively flat. There appears to be slight fluctuations, but nothing major that should be of concern.

Cubic Residuals

```
gg_tsresiduals(cubic.fit) +
  labs(title = "Residual plots for cubic fit")
```



While the cubic seems to help the residuals, we remain skeptical of this value at present especially for forecasting, since we expect a second inflection point in the model, even though there's no evidence in the data to suggest this.

Rejection of Logarithmic Transformation

As discussed extensively in the EDA, a logarithmic transformation is not necessary since the trend does not appear to be exponential, and the variance of the series is stable. This indicates that we have an additive trend and season terms. This is supported by the regression analysis of the residuals. If there were an exponential trend, we would expect the residuals versus time plot to display a distinctive shape, such as an upward curve, which is not the case here. There does appear to be a wave-like pattern around 0, but this is due to autocorrelation of the residuals as seen in the `ACF` plot.

Adding Seasonal Factor as a Variable

In order to fit a trend with a monthly seasonal term, we need to encode 11 dummy variables in order to represent each of the 12 categories. January is treated as base level. `TSLM` does this automatically for us with the special `season()` function. The (quadratic) model to be fit is:

$$y = \beta_0 + \beta_1 * t + \beta_2 * t^2 + \beta_3 * \text{Feb} + \beta_4 \text{March} \ldots$$

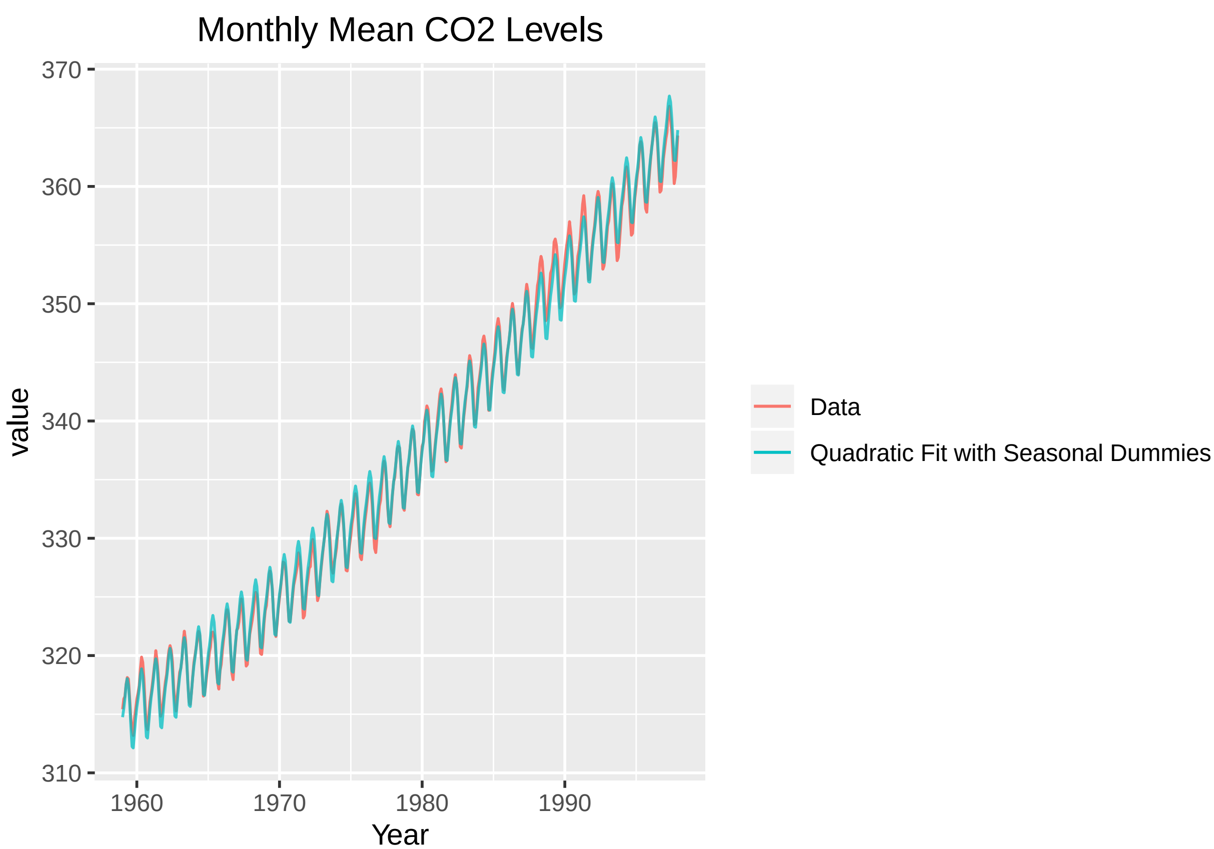
Quadratic with Seasons

```
quad.season.fit <- co2.ts %>% model(TSLM(value ~ trend() + I(trend()^2) + season()))
quad.season.fit %>% report
```

```
## Series: value
```

```
## Model: TSLM
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.99478 -0.54468 -0.06017  0.47265  1.95480
##
## Coefficients:
##                  Estimate Std. Error  t value Pr(>|t|)
## (Intercept)     3.147e+02  1.494e-01 2105.894  < 2e-16 ***
## trend()         6.763e-02  9.929e-04   68.114  < 2e-16 ***
## I(trend()^2)    8.865e-05  2.050e-06   43.242  < 2e-16 ***
## season()year2   6.642e-01  1.640e-01    4.051 5.99e-05 ***
## season()year3   1.407e+00  1.640e-01    8.582  < 2e-16 ***
## season()year4   2.538e+00  1.640e-01   15.480  < 2e-16 ***
## season()year5   3.017e+00  1.640e-01   18.400  < 2e-16 ***
## season()year6   2.354e+00  1.640e-01   14.357  < 2e-16 ***
## season()year7   8.331e-01  1.640e-01    5.081 5.50e-07 ***
## season()year8  -1.235e+00  1.640e-01   -7.531 2.75e-13 ***
## season()year9  -3.059e+00  1.640e-01  -18.659  < 2e-16 ***
## season()year10 -3.243e+00  1.640e-01  -19.777  < 2e-16 ***
## season()year11 -2.054e+00  1.640e-01  -12.526  < 2e-16 ***
## season()year12 -9.374e-01  1.640e-01   -5.717 1.97e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.724 on 454 degrees of freedom
## Multiple R-squared: 0.9977,  Adjusted R-squared: 0.9977
## F-statistic: 1.531e+04 on 13 and 454 DF, p-value: < 2.22e-16
```

```r
augment(quad.season.fit) %>%
  ggplot(aes(x = index)) +
  geom_line(aes(y = value, colour = "Data")) +
  geom_line(aes(y = .fitted, colour = "Quadratic Fit with Seasonal Dummies"), alpha = 0.75) +
  xlab("Year") + ylab("value") +
  ggtitle("Monthly Mean CO2 Levels") +
  theme(plot.title = element_text(hjust = 0.5))+
  guides(colour=guide_legend(title=NULL))
```



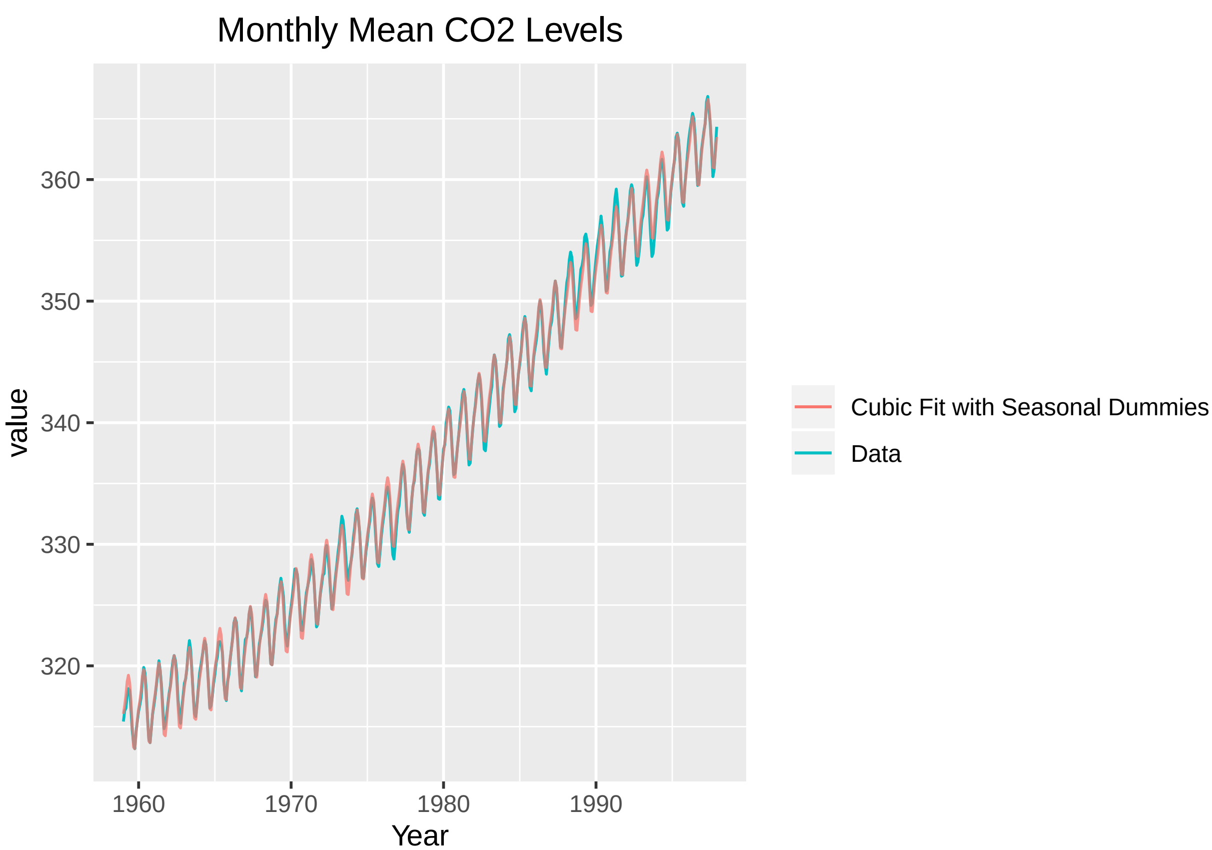
Cubic with Seasons

```
cubic.season.fit <- co2.ts %>% model(TSLM(value ~ trend() + I(trend()^2) + I(trend() ^ 3) + sea
cubic.season.fit %>% report
```

```
## Series: value
## Model: TSLM
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -1.5573094 -0.3312054  0.0008042  0.2880086  1.5039635
##
## Coefficients:
##                   Estimate Std. Error  t value Pr(>|t|)
## (Intercept)      3.160e+02  1.210e-01 2611.629  < 2e-16 ***
## trend()          3.275e-02  1.740e-03   18.827  < 2e-16 ***
## I(trend()^2)     2.744e-04  8.614e-06   31.850  < 2e-16 ***
## I(trend()^3)    -2.640e-07  1.207e-08  -21.863  < 2e-16 ***
## season()year2    6.700e-01  1.145e-01    5.852 9.32e-09 ***
## season()year3    1.419e+00  1.145e-01   12.390  < 2e-16 ***
## season()year4    2.555e+00  1.145e-01   22.319  < 2e-16 ***
## season()year5    3.040e+00  1.145e-01   26.550  < 2e-16 ***
## season()year6    2.383e+00  1.145e-01   20.811  < 2e-16 ***
## season()year7    8.678e-01  1.145e-01    7.578 2.00e-13 ***
## season()year8   -1.194e+00  1.145e-01  -10.429  < 2e-16 ***
```

```
## season()year9  -3.013e+00  1.145e-01  -26.311  < 2e-16 ***
## season()year10 -3.191e+00  1.145e-01  -27.860  < 2e-16 ***
## season()year11 -1.996e+00  1.145e-01  -17.428  < 2e-16 ***
## season()year12 -8.738e-01  1.145e-01   -7.628 1.41e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5056 on 453 degrees of freedom
## Multiple R-squared: 0.9989,  Adjusted R-squared: 0.9989
## F-statistic: 2.92e+04 on 14 and 453 DF, p-value: < 2.22e-16
```

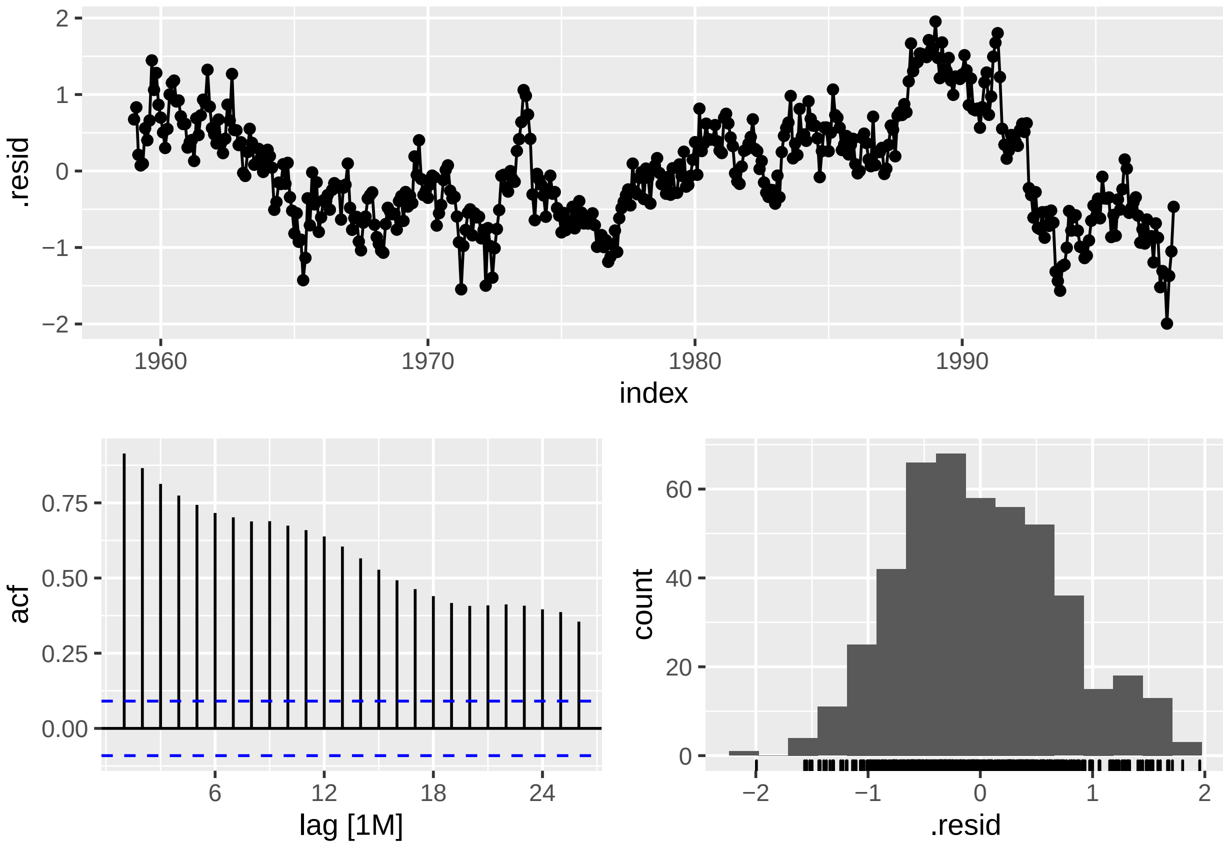
```r
augment(cubic.season.fit) %>%
  ggplot(aes(x = index)) +
  geom_line(aes(y = value, colour = "Data")) +
  geom_line(aes(y = .fitted, colour = "Cubic Fit with Seasonal Dummies"), alpha = 0.75) +
  xlab("Year") + ylab("value") +
  ggtitle("Monthly Mean CO2 Levels") +
  theme(plot.title = element_text(hjust = 0.5))+
  guides(colour=guide_legend(title=NULL))
```



The fit seems very good except toward the end of the curve, where the spikes in the fitted series is not as extreme as in the data. We need to evaluate the visual fit using residual plots. Every value in every model is statistically significant.

Seasonal Quadratic Residuals

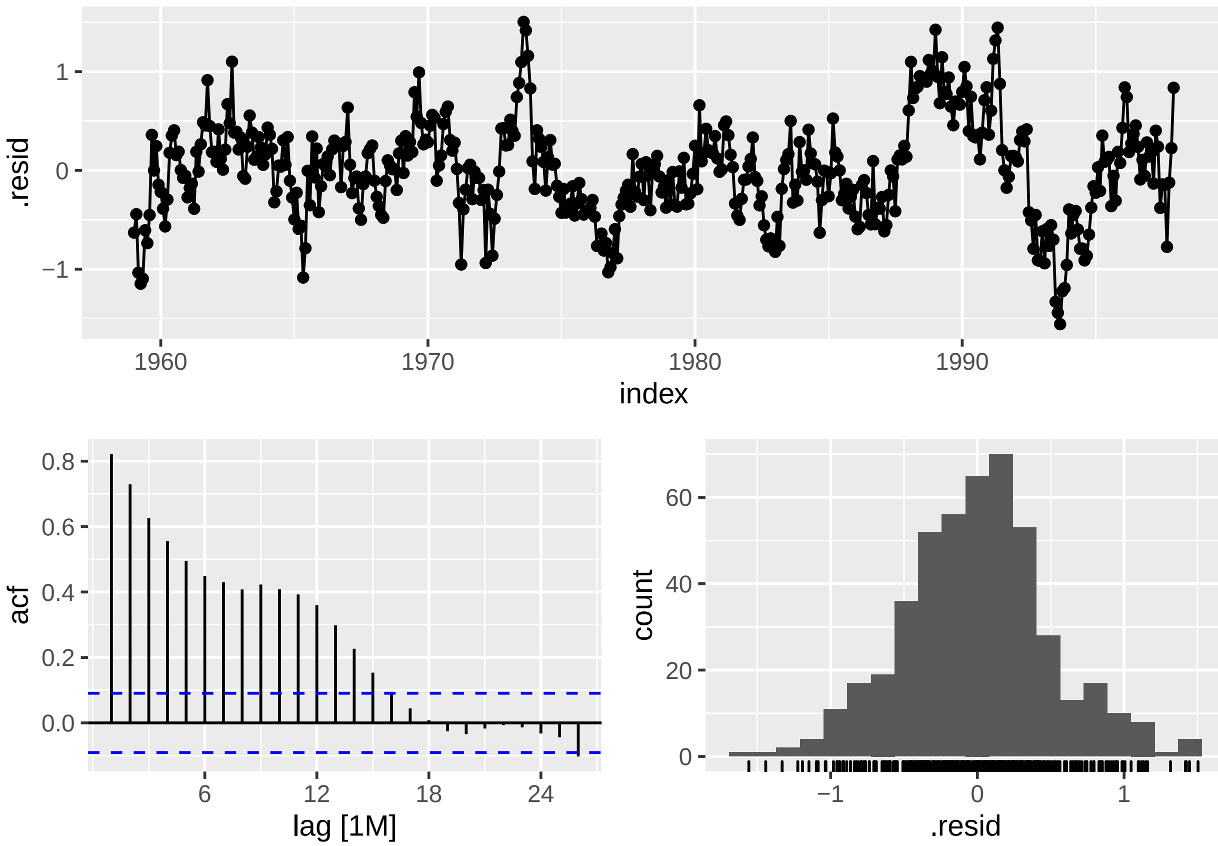
```r
quad.season.fit %>% gg_tsresiduals()
```



The observation is consistent in that there are larger fluctuations in the residuals toward the end of the series. However, the absolute values of the residuals are very small compared to the original values. The `ACF` is now delaying rather than oscillating.

Seasonal Cubic Residuals

```r
cubic.season.fit %>% gg_tsresiduals()
```



The cubic model also shows the larger fluctuations in the residuals toward the end of the series. However, the absolute values of the residuals are smaller than those in the quadratic. The `ACF` is now declining even faster.

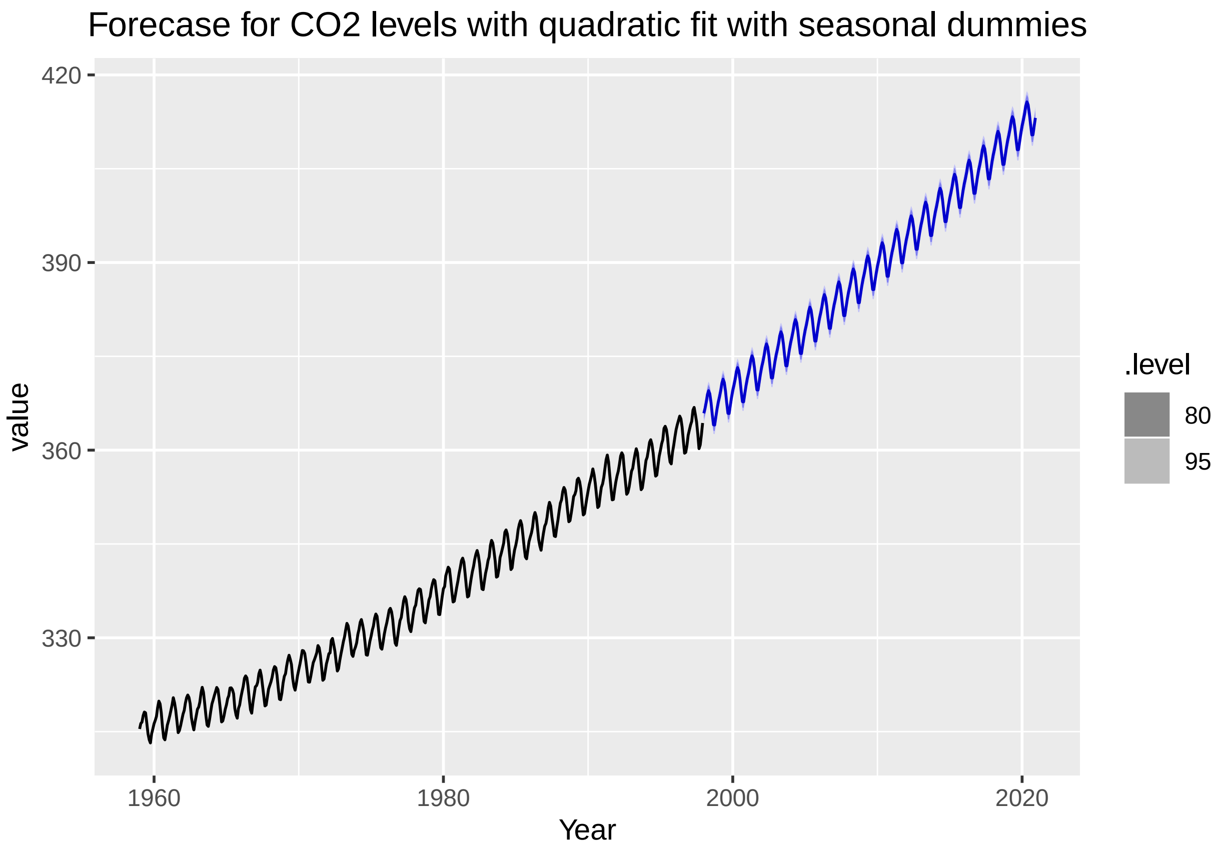
Forecasting

To generate the forecast, we first generate a dataframe with the desired time frame, which will range from Jan. 1998 (the first month after the data series ends) to Dec. 2020. With forecasting, it will will hopefully become apparent whether our cubic model is overfitting.

```
daterange <- as_tsibble(ts(start = c(1998, 1), end=c(2020, 12), frequency=12))

#For the time index, add 468 (the last value of the original series) to the row number
daterange <- daterange %>% mutate(time_index = row_number() + 468)
```

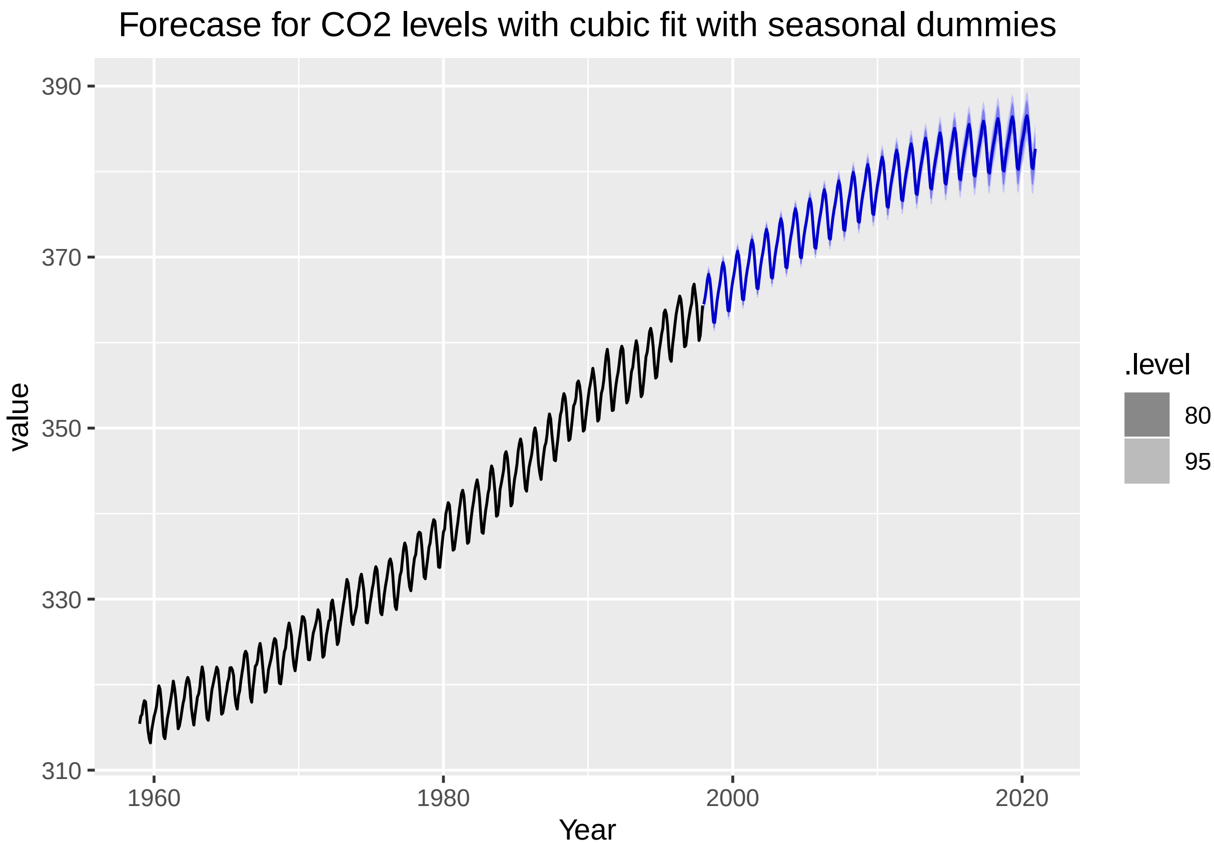
Quadratic Forecast

```
series.forecast <- quad.season.fit %>% forecast(h = 276)

series.forecast %>% autoplot(co2.ts)+
  xlab("Year") + ylab("value") +
  ggtitle("Forecase for CO2 levels with quadratic fit with seasonal dummies") +
  theme(plot.title = element_text(hjust = 0.5))+
  guides(colour=guide_legend(title=NULL))
```



Cubic Forecast

```
cubic.series.forecast <- cubic.season.fit %>% forecast(h = 276)

cubic.series.forecast %>% autoplot(co2.ts)+
  xlab("Year") + ylab("value") +
  ggtitle("Forecase for CO2 levels with cubic fit with seasonal dummies") +
  theme(plot.title = element_text(hjust = 0.5))+
  guides(colour=guide_legend(title=NULL))
```



By comparing the forecasts for both the quadratic and the cubic, we observe the cubic forecast exhibits a sharper change in trend than any behavior found within the original time series. Namely, the model predicts a second inflection point where the rate starts declining again, which is not supported by the original data. So while the cubic model was statistically significant and seemed to provide better in-sample fit that the quadratic, it appears to be over-fitting the original dataset. The quadratic predicts a more linear increase, which is consistent with the trend observed in the original data series. As a result, we trust the quadratic model for forecasting more than the cubic.

For our linear model, we will stick with our quadratic seasonal model.

Part 3: Autoregressive Integrated Moving Average Models

ARIMA Selection Procedure

In order to choose an ARIMA model to fit to the CO_2 series, we will take the following steps:

1. Perform EDA by plotting the data. Identify any unusual observations.
2. If necessary, transform the data (e.g. using a Box-Cox transformation) to stabilize the variance.
3. If the data are non-stationary: take first differences of the data until the data are stationary.
4. Examine the `ACF`/`PACF`: Is an `AR(p)` or `MA(q)` model appropriate?
5. Try your chosen model(s), and use appropriate metrics to choose a model.
6. Model evaluation Check the residuals from your chosen model by plotting the `ACF` of the residuals, and doing a portmanteau test of the residuals. If they do not look like white noise, try a modified model.
7. Once the residuals look like white noise, calculate forecasts.

1. Perform EDA

The first step is ETSDA, which was already performed in parts 1 and extended in part 2.

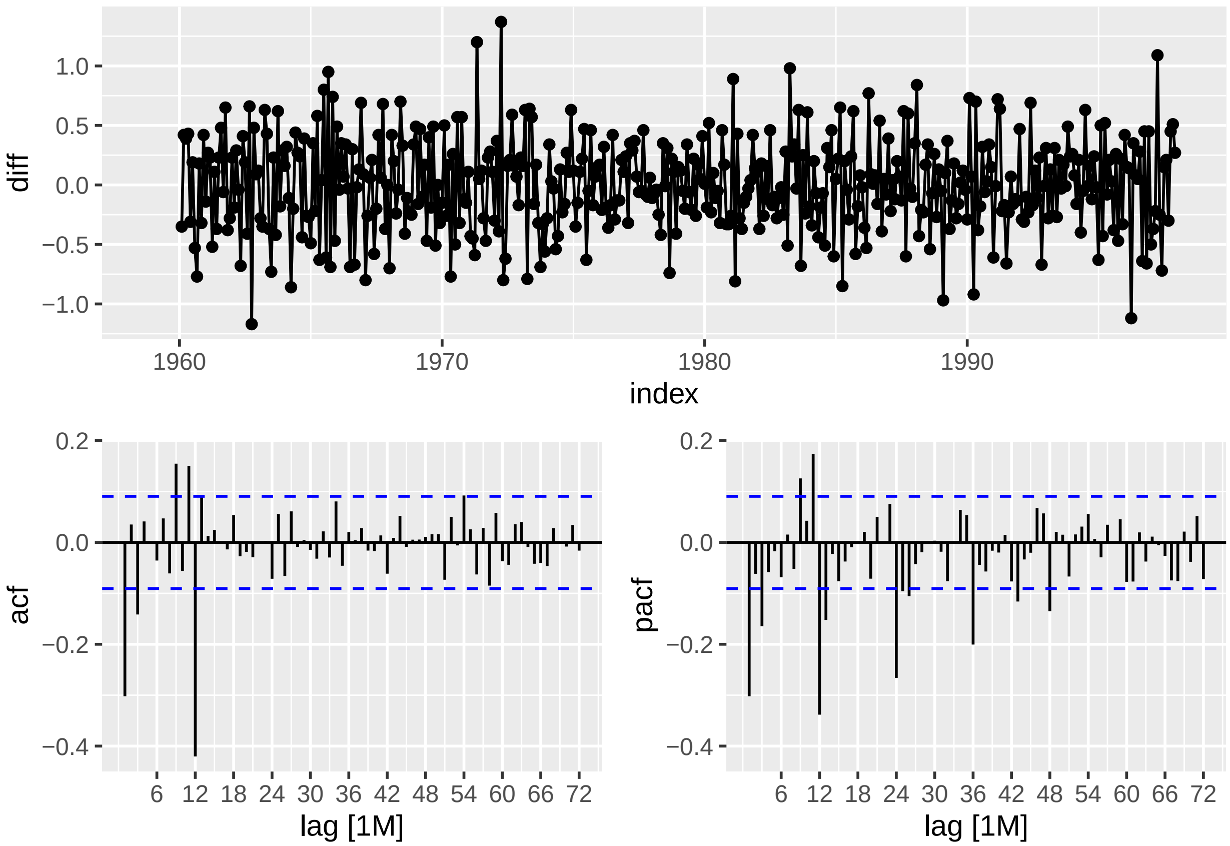
2. Apply Needed Transformations

In parts 1 and 2, we showed and discussed that we do need to stabilize the variance, and no transformations are needed.

3. Ensure Stationarity by Detrending with First Differences

Since the data is clearly non-stationary with strong seasonality, we evaluate differencing as a method to generate a stationary series. This can be examined with gg_tsdisplay. We start with first order differencing to remove the trend, and also first order seasonal differencing (at `frequency=12` for monthly data) to remove the seasonal trend. Because we saw how the cubic model over fit, we will be careful about adding higher orders of difference if the data fails to be stationary.

```
co2.ts$diff <- co2.ts$value %>% difference(12) %>% difference()
co2.ts %>% gg_tsdisplay(diff, plot_type = 'partial', lag_max = 72)
```



The differenced series looks quite stationary, with constant mean and constant variance. Therefore, it is likely we will need a `SARIMA` model with both first order differencing and first order seasonal differencing. We will verify stationarity with the Augmented Dickey-Fuller Test at a significance level of 0.05:

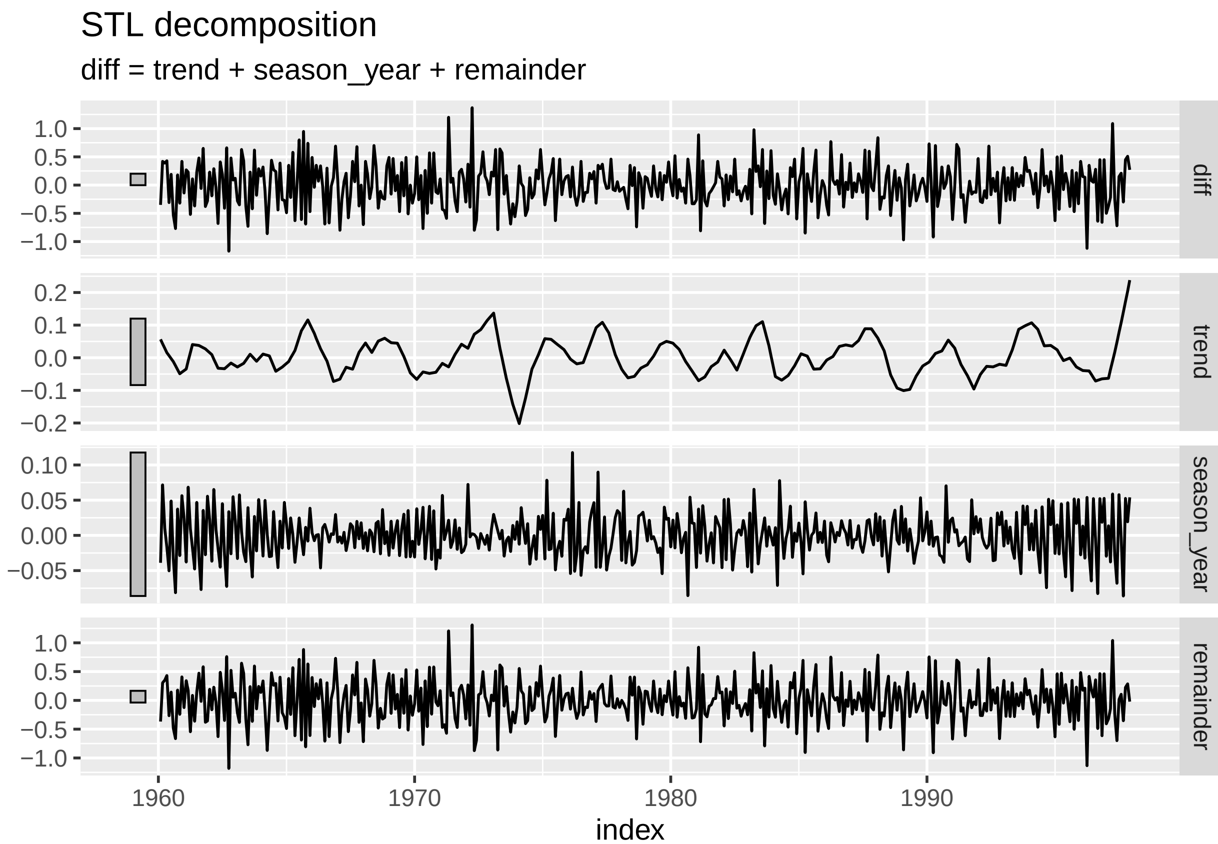
```
adf.test(co2.ts$diff[-1:-13])
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  co2.ts$diff[-1:-13]
## Dickey-Fuller = -8.9846, Lag order = 7, p-value = 0.01
## alternative hypothesis: stationary
```

Since p-value < 0.01, we reject H0 and conclude that our model is satisfactorily stationary.

We can also verify that the STL decomposition should not give us any clear trends or seasonality in our first order seasonal and non-seasonal differenced data, which is another indication of stationarity.

```
co2.ts[-1:-13,] %>% model(STL(diff)) %>% components() %>% autoplot()
```



4. Examine ACF and PACF

When examining the `ACF` and `PACF` of our stationary model, there are strong peaks in the `PACF` corresponding to multiples of 12, which we can adjust for by incorporating seasonal `AR` terms. Since there are signficant lags up to 4 seasonal periods, we may start with a seasonal `AR(4)` term. The significant spike in the `PACF` at lag 1 indicates that a non-seasonal `AR(1)` term might be useful, which makes sense as we would expect CO_2 from one month to physically lag in the atmosphere from one month to the next. As a result, we will first try the $ARIMA(1,1,0)(4,1,0)_{12}$ model. Analogously by looking at the `ACF`, a seasonal `MA(1)` term can account for the lag at 12, and a non-seasonal `MA(1)` term can account for the lag 1, so we will also try $ARIMA(0,1,1)(0,1,1)_{12}$ for comparison.

5. Model Using ACF/PACF Insights

To select the best model, we will use `AICc`, which is a measure of in-sample fit, penalized by a function of the number of parameters and sample size. We could also use BIC, but this tends to penalize the number of parameters more so than AICc. We will start with AICc, and if the number of lag terms found is too large, which will try again with BIC. AICc is a reasonable choice when we do not use out-of-sample fit as a measure. Since the series is so short, out-of-sample fits don't make as much sense because not enough data will be left for fitting the series.

```
mod.1 <- co2.ts %>% model(ARIMA(value ~ pdq(1,1,0) + PDQ(4,1,0)))
mod.2 <- co2.ts %>% model(ARIMA(value ~ pdq(0,1,1) + PDQ(0,1,1)))

mod.1 %>% glance
```

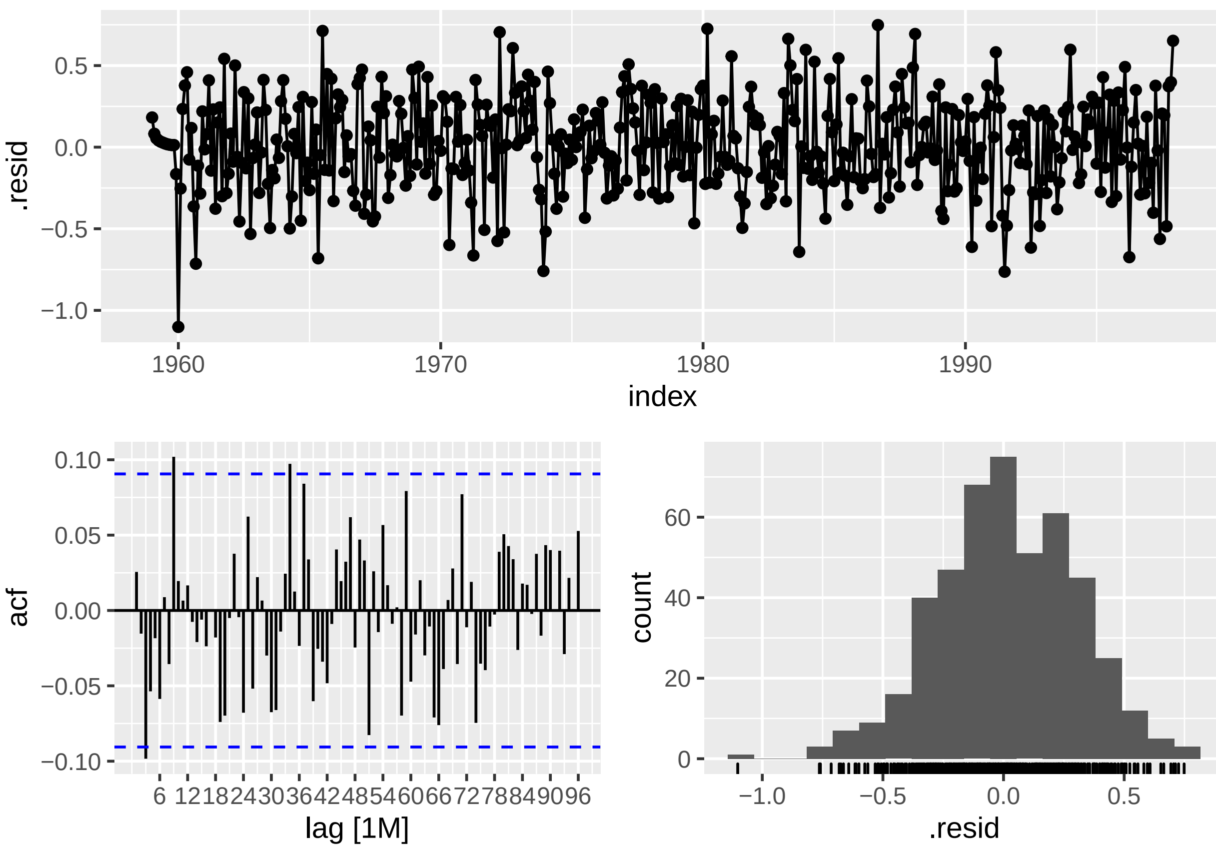
```
## # A tibble: 1 x 8
##   .model                   sigma2 log_lik   AIC  AICc   BIC ar_roots   ma_roots
##   <chr>                     <dbl>   <dbl> <dbl> <dbl> <dbl> <list>     <list>
## 1 ARIMA(value ~ pdq(1,~ 0.0958   -107.  226.  226.  251. <cpl [49~ <cpl [0~
```

```
mod.2 %>% glance
```

```
## # A tibble: 1 x 8
##   .model                   sigma2 log_lik   AIC  AICc   BIC ar_roots   ma_roots
##   <chr>                     <dbl>   <dbl> <dbl> <dbl> <dbl> <list>     <list>
## 1 ARIMA(value ~ pdq(0,~ 0.0858   -86.1  178.  178.  191. <cpl [0]> <cpl [1~
```

Based on the `AICc`, model 2 looks a bit better, and we check our assumptions with the `ACF`, and guide our iterations.

```
gg_tsresiduals(mod.2, lag_max = 96)
```



We see that our fit is pretty good for model 2, so our iterations based on the `ACF` will stay close to this model. Nevertheless, we need to explore these additional variations of this model, since the `ACF` and `PACF` plots suggests that the pattern is more complicated than having only MA terms:

- $ARIMA(1,1,1)(0,1,1)_{12}$
- $ARIMA(0,1,1)(1,1,1)_{12}$
- $ARIMA(1,1,1)(1,1,1)_{12}$
- $ARIMA(1,1,2)(1,1,1)_{12}$

```r
mods <- co2.ts %>% model(ARIMA(value ~ pdq(1,1,1) + PDQ(0,1,1)),
                      ARIMA(value ~ pdq(0,1,1) + PDQ(1,1,1)),
                      ARIMA(value ~ pdq(1,1,1) + PDQ(1,1,1)),
                      ARIMA(value ~ pdq(1,1,2) + PDQ(1,1,1)),
                      ARIMA(value ~ pdq(1,1,1) + PDQ(1,1,2)))
mods %>% glance()
```

```
## # A tibble: 5 x 8
##   .model                  sigma2 log_lik   AIC  AICc   BIC ar_roots  ma_roots
##   <chr>                    <dbl>   <dbl> <dbl> <dbl> <dbl> <list>    <list>
## 1 ARIMA(value ~ pdq(1,~ 0.0856   -85.0  178.  178.  195. <cpl [1]> <cpl [1~
## 2 ARIMA(value ~ pdq(0,~ 0.0860   -86.0  180.  180.  196. <cpl [12~ <cpl [1~
## 3 ARIMA(value ~ pdq(1,~ 0.0858   -84.9  180.  180.  200. <cpl [13~ <cpl [1~
## 4 ARIMA(value ~ pdq(1,~ 0.0856   -84.1  180.  180.  205. <cpl [13~ <cpl [1~
## 5 ARIMA(value ~ pdq(1,~ 0.0858   -84.4  181.  181.  205. <cpl [13~ <cpl [2~
```

For a more robust determination of our `ARIMA` model, we loop over parameters of pdq and PDQ. For d and D we will keep both differences at 1 since the EDA suggests that this generates a stationary series while for pq and PQ, we will check all values up to 4. Since we have the same differencing values for all models, we will evaluate based on the `AICc` , which will allow for internal model comparisons of in-sample fit.

Note that some models failed to find stationary coefficients due to insufficient differencing or the lack of inclusion of a constant term. We will ignore those specfications using the `tryCatch` block since they will not generate viable `ARIMA` models. Note that rather than running the scan each time, we've listed the best 10 models in the comments of the code, and $pqPQ$ parameters and the `AICc`.

```r
results2 <- data.frame(p = integer(), q = integer(), P = integer(), Q = integer(), AICc = doubl

fit_aicc <- function(p,q,P,Q) {

  mod.fit <- co2.ts %>% model(ARIMA(value ~ pdq(p,1,q) + PDQ(P,1,Q)))

  out <- tryCatch(
    {
      #If AICc cannot be found, then the model failed to converge
      AICc <- glance(mod.fit)$AICc
    },

    error = function(e) {
      return(NA)
    },

    warning = function(w) {
      return(NA)
    }
  )
```

```r
  if (!is.na(out)){
      return(data.frame(cbind(p=p,q=q,P=P,Q=Q,AICc=AICc)))
  }
  else {
      return(NA)
  }
}

run = FALSE
if (run) {
  for (p in seq(0,4)) {
    for (q in seq(0,4)) {
      for (P in seq(0,4)) {
        for (Q in seq(0,4)) {
            answer = fit_aicc(p,q,P,Q)

            if (!is.na(answer)){
              if (dim(answer)[1] == 1) {
                  results2 <- rbind(results2, answer)
              }
            }
        }
      }
    }
  }
  results2[which.min(results2$AICc), ]
}

#RESULTS from the scan
#> results2[order(results2$AICc), ] %>% head()
#    p q P Q     AICc
#103 1 1 2 2 172.9589
#100 1 1 1 3 172.9980
#35  0 1 2 2 173.6884
#31  0 1 1 3 173.7649
#53  0 2 2 2 174.2829
#50  0 2 1 3 174.3233

#Best model found by ARIMA based on AICc
mod.best <- co2.ts %>% model(ARIMA(value ~ pdq(1,1,1) + PDQ(2,1,2)))
mod.best %>% report()
```

```
## Series: value
## Model: ARIMA(1,1,1)(2,1,2)[12]
##
## Coefficients:
##          ar1      ma1     sar1     sar2     sma1     sma2
```

```
##        0.2652  -0.5950  0.9613  -0.1335  -1.8169  0.8564
## s.e.   0.1358   0.1154  0.0787   0.0603   0.0710  0.0622
##
## sigma^2 estimated as 0.08303:  log likelihood=-79.35
## AIC=172.71   AICc=172.96   BIC=201.55
```

```
mod.best %>% augment() %>% features(.resid, ljung_box)
```

```
## # A tibble: 1 x 3
##   .model                                       lb_stat lb_pvalue
##   <chr>                                          <dbl>     <dbl>
## 1 ARIMA(value ~ pdq(1, 1, 1) + PDQ(2, 1, 2)) 0.0000106     0.997
```

Our final model is similar to our original, but includes more autoregressive terms seasonally. Our best model is, therefore: $ARIMA(1,1,1)(2,1,2)_{12}$

$$(1-0.2652B)(1-0.9613B^{12}+0.1335B^{24})(1-B)(1-B^{12})y_t = (1-0.5950B)(1-1.8169B^{12}+0.8564B^{24})\omega_t$$

The characteristics of this model is that there is both order 1 seasonal and non-seasonal differencing by design, because the series generated by this process is stationary from the EDA. For both the AR and MA terms, there is 1 non-seasonal term, and 2 seasonal terms.

Checking Against the Built-In Algorithm

We compare our best model to that fit by `auto.arima`:

```
mod.auto.arima <- co2.ts %>% model(ARIMA(value))
mod.auto.arima %>% glance()
```

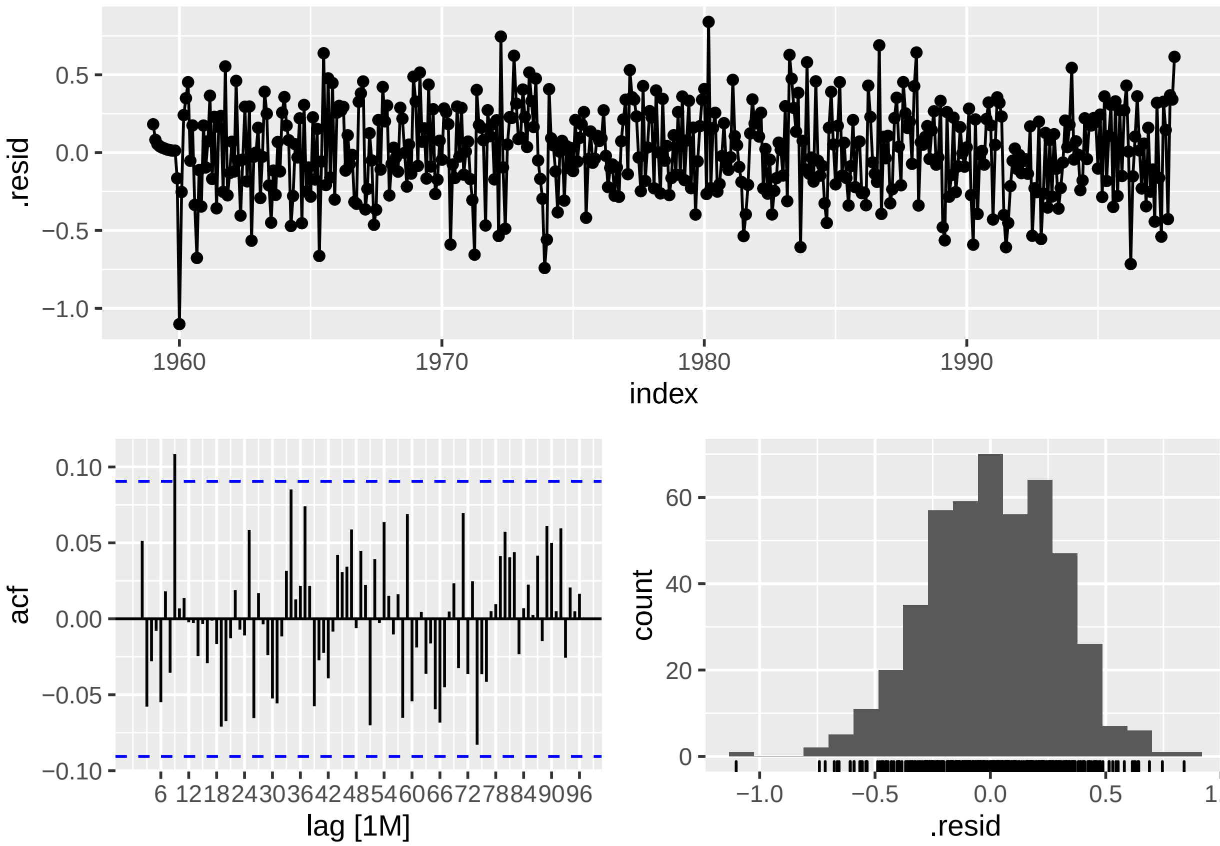
```
## # A tibble: 1 x 8
##   .model       sigma2 log_lik   AIC  AICc   BIC ar_roots   ma_roots
##   <chr>         <dbl>   <dbl> <dbl> <dbl> <dbl> <list>     <list>
## 1 ARIMA(value) 0.0858   -84.4  181.  181.  205. <cpl [13]> <cpl [25]>
```

The `auto.arima` function found a similar model: $ARIMA(1,1,1)(1,1,2)_{12}$

Check for White Noise Stationarity in the Residuals

We will now perform residual analysis of our best model:

```
gg_tsresiduals(mod.best, lag_max = 96)
```



The residuals look like white noise. The mean and variance both look constant across the series, with no obvious increases/decreases or fluctuations. There appears to be one outlier in the residual at Jan. 1960, but the outlier is not too extreme and there is only a single one.

There is also only a single significant `ACF` lag, but since our significance level for the confidence intervals is $\alpha = 0.05$, we would be surprised by 5% of the peaks showing statistical significance, so this single peak is also not surprising. To ensure that there are no significant autocorrelations, we will apply the Ljung-Box test at a significance level of $\alpha = 0.05$. The null hypothesis is that there are no significant autocorrelations accounting for a maximum of 96 lags:

```
mod.best %>% augment() %>% features(.resid, ljung_box, lag = 96)
```

```
## # A tibble: 1 x 3
##   .model                                     lb_stat lb_pvalue
##   <chr>                                        <dbl>     <dbl>
## 1 ARIMA(value ~ pdq(1, 1, 1) + PDQ(2, 1, 2))    82.6     0.833
```

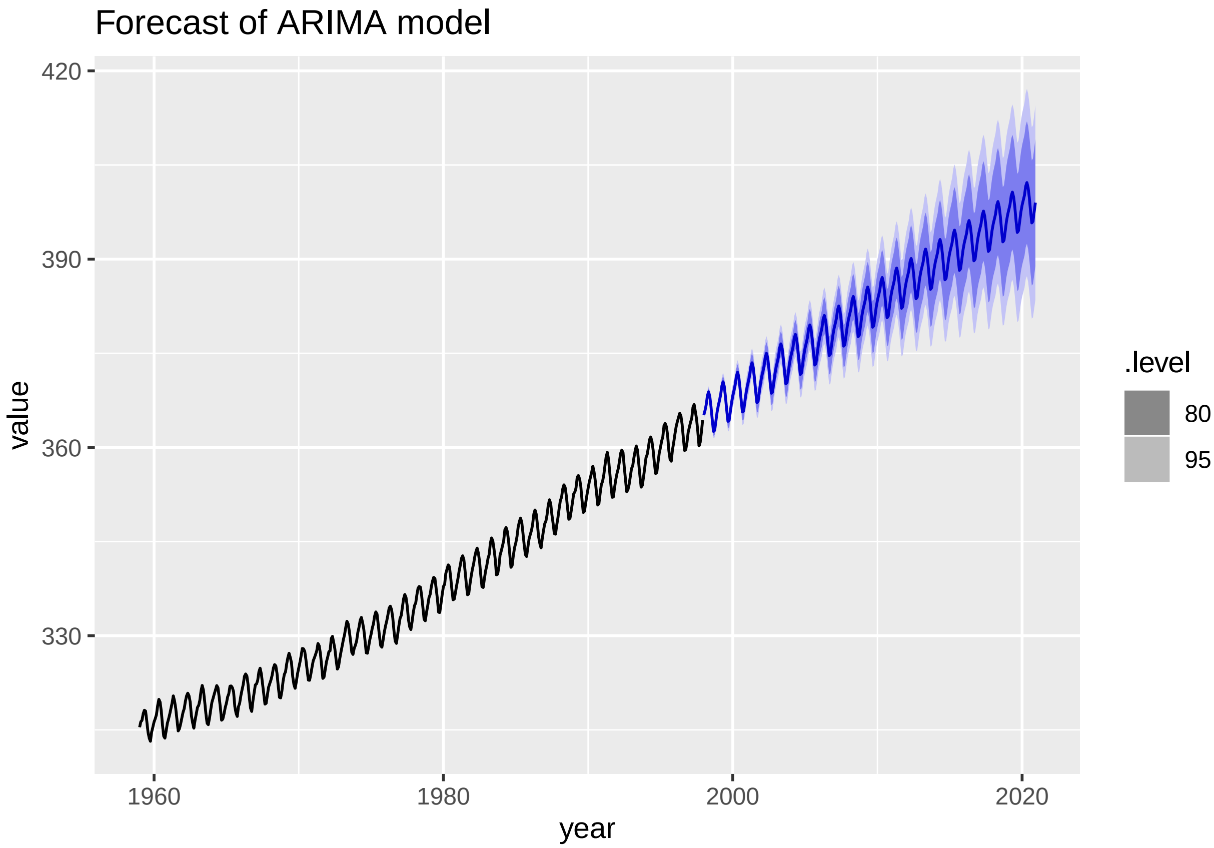
Even accounting for up to 96 lags, since $p > 0.05$, we fail to reject H_0 that there are no autocorrelations in the residuals. This, with the fact that we have constant mean and variance, suggests we have white noise as our residual series.

Calculating Forecasts

We are now ready to perform our forecast up to 2020:

```
arima.forecast <- forecast(mod.best, h=276)
arima.forecast %>% autoplot(co2.ts) +
```

```r
ggtitle("Forecast of ARIMA model") +
labs(x = 'year')
```



As forecasts extend outward, the confidence intervals also grow, suggesting less confidence in our model estimates. In contrast to the models created in part 2, we don't see very much quadratic trend, despite having taken the first difference in both our lag and seasonal lag.

Part 4: Analysis of Weekly Observations

The file `co2_weekly_mlo.txt` contains weekly observations of atmospheric carbon dioxide concentrations measured at the Mauna Loa Observatory from 1974 to 2020, published by the National Oceanic and Atmospheric Administration (NOAA). Convert these data into a suitable time series object, conduct a thorough EDA on the data, and address the problem of missing observations. Describe how the Keeling Curve evolved from 1997 to the present and compare current atmospheric CO2 levels to those predicted by your forecasts in Parts 2 and 3. Use the weekly data to generate a month-average series from 1997 to the present, and compare the overall forecasting performance of your models from Parts 2 and 3 over the entire period.

First we load the data, and identified that missing data is represented by -999.99. In order to generate a continuous time series, we will fill this data in by interpolating between the other values of the series. First, we see where the missing values are.

```
#Make date time for
col.names <- c('yr', 'mon', 'day', 'decimal', 'ppm', 'num.days', '1 yr ago',  '10 yr ago',  's
dat <- read.delim('data/co2_weekly_mlo.txt', comment.char = '#', header = F, sep = "")
colnames(dat) <- col.names
dat$Date <- make_datetime(dat$yr, dat$mon, dat$day) %>% as.Date()
dat$week <- week(dat$Date)

Hmisc::describe(dat)
```

```
## dat
##
##  11  Variables      2388  Observations
## ---------------------------------------------------------------------------
## yr
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##     2388        0       47        1     1997    15.26     1976     1978
##      .25      .50      .75      .90      .95
##     1985     1997     2008     2015     2017
##
## lowest : 1974 1975 1976 1977 1978, highest: 2016 2017 2018 2019 2020
## ---------------------------------------------------------------------------
## mon
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##     2388        0       12    0.993    6.539    3.971        1        2
##      .25      .50      .75      .90      .95
##        4        7       10       11       12
##
## Value          1     2     3     4     5     6     7     8     9    10
## Frequency    203   185   199   193   201   198   204   203   198   203
## Proportion 0.085 0.077 0.083 0.081 0.084 0.083 0.085 0.085 0.083 0.085
##
## Value         11    12
## Frequency    197   204
## Proportion 0.082 0.085
```

```
## ---------------------------------------------------------------------------
## day
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##     2388        0       31    0.999    15.71    10.16        2        4
##      .25      .50      .75      .90      .95
##        8       16       23       28       29
##
## lowest :  1  2  3  4  5, highest: 27 28 29 30 31
## ---------------------------------------------------------------------------
## decimal
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##     2388        0     2388        1     1997    15.26     1977     1979
##      .25      .50      .75      .90      .95
##     1986     1997     2009     2016     2018
##
## lowest : 1974.380 1974.399 1974.418 1974.437 1974.456
## highest: 2020.051 2020.070 2020.089 2020.108 2020.127
## ---------------------------------------------------------------------------
## ppm
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##     2388        0     2112        1    355.4     49.4    332.2    335.7
##      .25      .50      .75      .90      .95
##    346.4    363.9    385.9    401.4    407.4
##
## Value      -1000   320   340   360   380   400   420
## Frequency     20    45   638   669   522   435    59
## Proportion 0.008 0.019 0.267 0.280 0.219 0.182 0.025
## ---------------------------------------------------------------------------
## num.days
##        n  missing distinct     Info     Mean      Gmd
##     2388        0        8    0.899    5.858    1.384
##
## Value          0     1     2     3     4     5     6     7
## Frequency     20    12    39    92   178   382   653  1012
## Proportion 0.008 0.005 0.016 0.039 0.075 0.160 0.273 0.424
## ---------------------------------------------------------------------------
## 1 yr ago
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##     2388        0     2029        1    326.3    101.5    330.4    334.2
##      .25      .50      .75      .90      .95
##    344.9    361.8    384.1    398.5    404.6
##
## Value      -1000   320   340   360   380   400   420
## Frequency     69    45   643   669   520   424    18
## Proportion 0.029 0.019 0.269 0.280 0.218 0.178 0.008
## ---------------------------------------------------------------------------
## 10 yr ago
##        n  missing distinct     Info     Mean      Gmd      .05      .10
```

```
##     2388        0     1609    0.988    49.23    488.1  -1000.0  -1000.0
##      .25      .50      .75      .90      .95
##    330.7    348.9    366.6    379.8    384.4
##
## Value      -1000   330   340   350   360   370   380   390
## Frequency    542   195   336   336   342   284   250   103
## Proportion 0.227 0.082 0.141 0.141 0.143 0.119 0.105 0.043
## ---------------------------------------------------------------------------
## since 1800
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##     2388        0     2012        1    77.76    44.63    51.92    55.68
##      .25      .50      .75      .90      .95
##    66.38    83.28   106.20   120.90   127.12
##
## Value      -1000    50    60    70    80    90   100   110   120   130
## Frequency     20   196   329   320   371   276   257   243   202   174
## Proportion 0.008 0.082 0.138 0.134 0.155 0.116 0.108 0.102 0.085 0.073
## ---------------------------------------------------------------------------
## Date
##        n  missing distinct
##     2388        0     2388
##
## lowest : 1974-05-19 1974-05-26 1974-06-02 1974-06-09 1974-06-16
## highest: 2020-01-19 2020-01-26 2020-02-02 2020-02-09 2020-02-16
## ---------------------------------------------------------------------------
## week
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##     2388        0       53        1    26.65     17.4        3        6
##      .25      .50      .75      .90      .95
##       14       27       40       47       50
##
## lowest :  1  2  3  4  5, highest: 49 50 51 52 53
## ---------------------------------------------------------------------------
```

Based on the tabular form of the data, there are no missing values. However, we see that there are extreme values of -1000 in the ppm, which are placeholders representing missing values. We will take a look at these now.

```
dat %>% filter(ppm == -999.99)
```

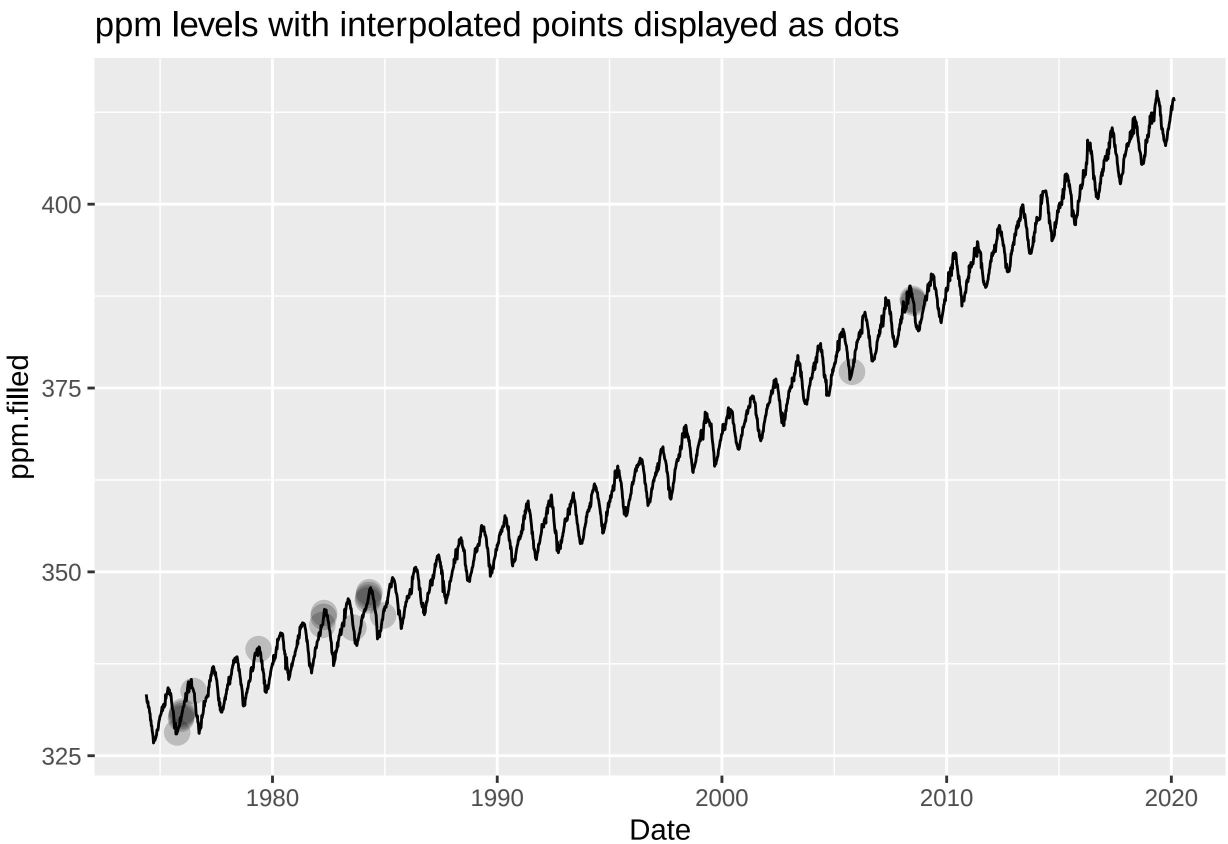
```
##     yr mon day  decimal     ppm num.days 1 yr ago 10 yr ago since 1800
## 1 1975  10   5 1975.760 -999.99        0   326.98   -999.99    -999.99
## 2 1975  12   7 1975.933 -999.99        0   329.32   -999.99    -999.99
## 3 1975  12  14 1975.952 -999.99        0   329.67   -999.99    -999.99
## 4 1975  12  21 1975.971 -999.99        0   329.96   -999.99    -999.99
## 5 1975  12  28 1975.990 -999.99        0   330.27   -999.99    -999.99
## 6 1976   6  27 1976.488 -999.99        0   333.05   -999.99    -999.99
## 7 1979   5  20 1979.382 -999.99        0   337.80   -999.99    -999.99
## 8 1982   3  21 1982.218 -999.99        0   342.37   -999.99    -999.99
```

```
## 9  1982   4  11 1982.275 -999.99        0   342.66   -999.99    -999.99
## 10 1982   4  18 1982.294 -999.99        0   342.66   -999.99    -999.99
## 11 1983   8   7 1983.599 -999.99        0   340.74   -999.99    -999.99
## 12 1984   4   1 1984.250 -999.99        0   344.65   -999.99    -999.99
## 13 1984   4   8 1984.269 -999.99        0   345.05   -999.99    -999.99
## 14 1984   4  15 1984.288 -999.99        0   345.53   -999.99    -999.99
## 15 1984   4  22 1984.307 -999.99        0   345.74   -999.99    -999.99
## 16 1984  12   2 1984.919 -999.99        0   342.48    328.90    -999.99
## 17 2005  10  16 2005.790 -999.99        0   374.50    358.07    -999.99
## 18 2008   6  29 2008.493 -999.99        0   385.30    368.07    -999.99
## 19 2008   7   6 2008.512 -999.99        0   385.15    368.71    -999.99
## 20 2008   7  13 2008.531 -999.99        0   384.21    367.57    -999.99
##          Date week
## 1  1975-10-05   40
## 2  1975-12-07   49
## 3  1975-12-14   50
## 4  1975-12-21   51
## 5  1975-12-28   52
## 6  1976-06-27   26
## 7  1979-05-20   20
## 8  1982-03-21   12
## 9  1982-04-11   15
## 10 1982-04-18   16
## 11 1983-08-07   32
## 12 1984-04-01   14
## 13 1984-04-08   15
## 14 1984-04-15   16
## 15 1984-04-22   17
## 16 1984-12-02   49
## 17 2005-10-16   42
## 18 2008-06-29   26
## 19 2008-07-06   27
## 20 2008-07-13   28
```

While there appears to be entire months of data missing, the values that are missing are spead out across many different years. Most are concentrated before 1984. To fill this in, we can interpolate using the tsclean function. This function will replace missing values and extreme outliers (-999.999) by interpolating values around the missing/outlier data. For example, if a single value is missing, then the interpolation is the average of the two neighbors. If there are multiple values missing in a row, the function will linearly interpolate between the two neighbors for all values. We will perform this action, then plot the data to make sure the replacement was reasonable. Note that we also need to fill num.days. We will do this by taking the average of the two neighbors, and replacing all values in the range of missing values with this average.

```r
#Use tsclean
dat$ppm.filled <- tsclean(dat$ppm)

#For num days, we will take average of closest neighbors used for interpolation
#For example:
```

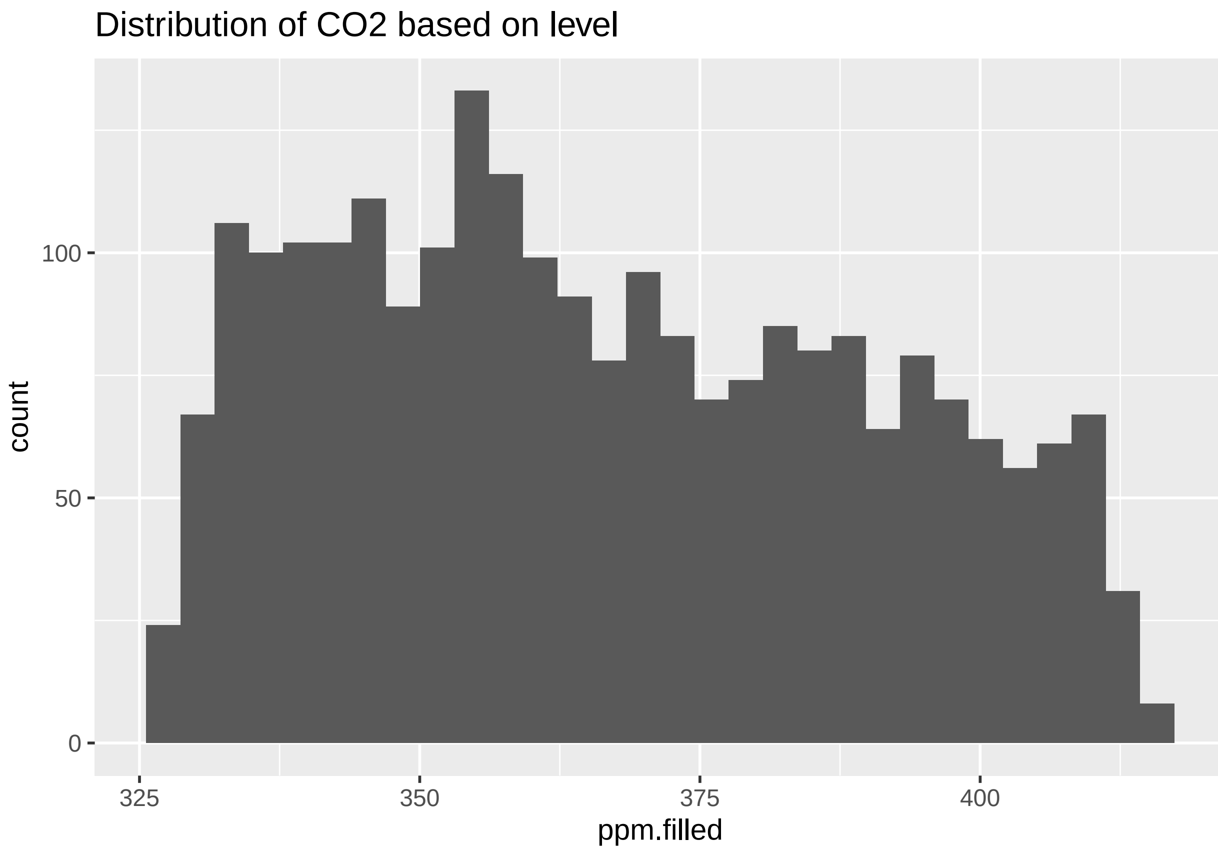
```r
#    7 NA 7 becomes 7 7 7
#    7 NA NA 7 becomes 7 7 7 7
#    5 NA NA NA NA 7 becomes 5 6 6 6 6 7
num.days <- dat$num.days

findNextValue <- function(ind){
    ind <- ind + 1
    while (TRUE) {
        if (dat$num.days[ind] > 0){
            return (dat$num.days[ind])
        }
        ind <- ind + 1
    }
}

num.day.prev <- 0
for (i in seq(1,length(num.days))) {
  if (num.days[i] == 0) {
    num.days[i] <- (num.day.prev + findNextValue(i))/2
  }
  else {
    num.day.prev <- num.days[i]
  }
}

#Fill in num.days as well
dat$num.days.filled <- num.days

#Generate a column so we can color points that were interpolated differently.
dat.ppm.missing <- dat[dat$ppm < 0, ]

ggplot(dat, aes(x = Date, y = ppm.filled)) +
  geom_line()+
  geom_point(data = dat.ppm.missing, aes(x = Date, y = ppm.filled), size = 4, alpha = 0.2)+
  ggtitle("ppm levels with interpolated points displayed as dots")
```



We can see that the interpolation did a pretty good job filling in the points, capturing both the trend and seasonality. The locations are where we'd expect them to be if we filled in these values manually.

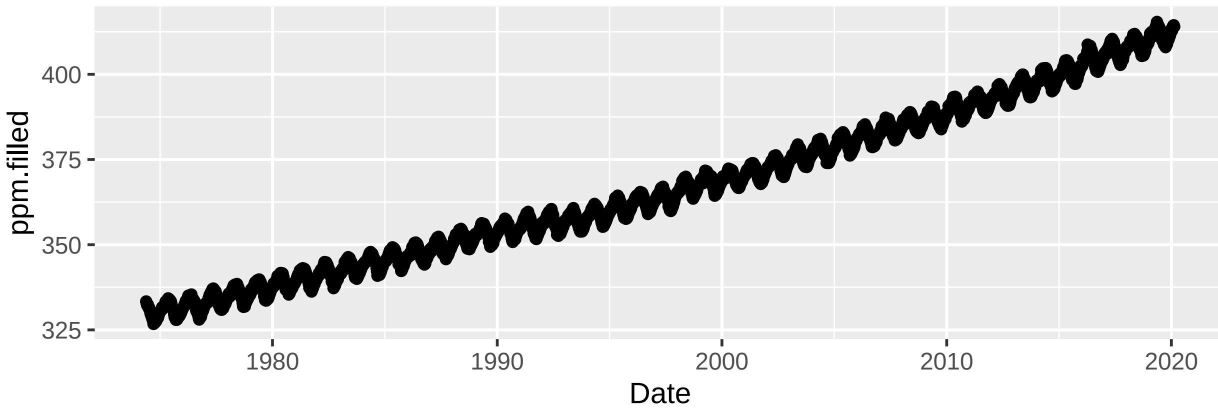
For EDA of weekly data, we will start with a histogram of the filled values:

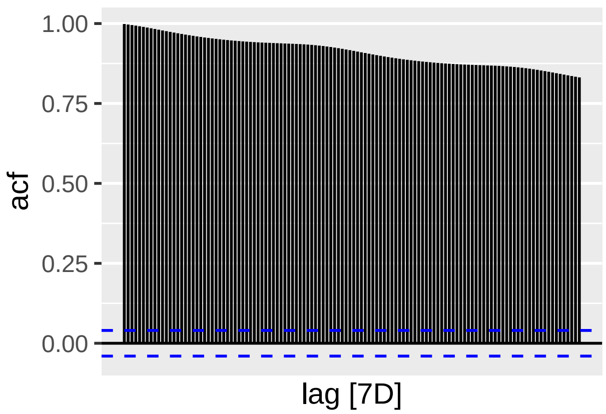
```
dat %>% ggplot(aes(x = ppm.filled)) + geom_histogram(bins=30) + ggtitle("Distribution of CO2 ba
```

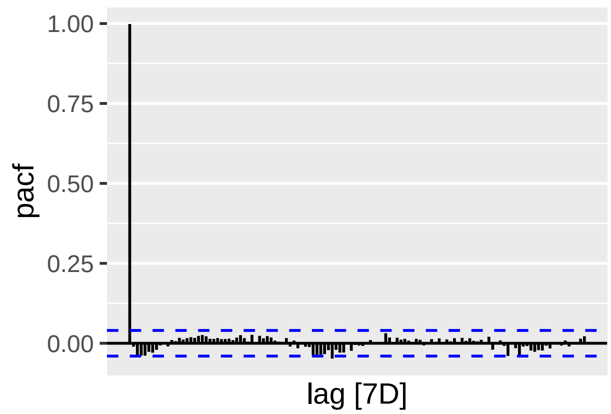


We see that the values are not close to normal, but are closer to uniform that the original series in part 3. We now look at a time series plot with ACF and PACF:

```
#Plot the time series, ACF and PACF
dat.weekly <- as_tsibble(dat, index = Date)
dat.weekly %>% gg_tsdisplay(ppm.filled, plot_type = 'partial', lag_max = 120)
```



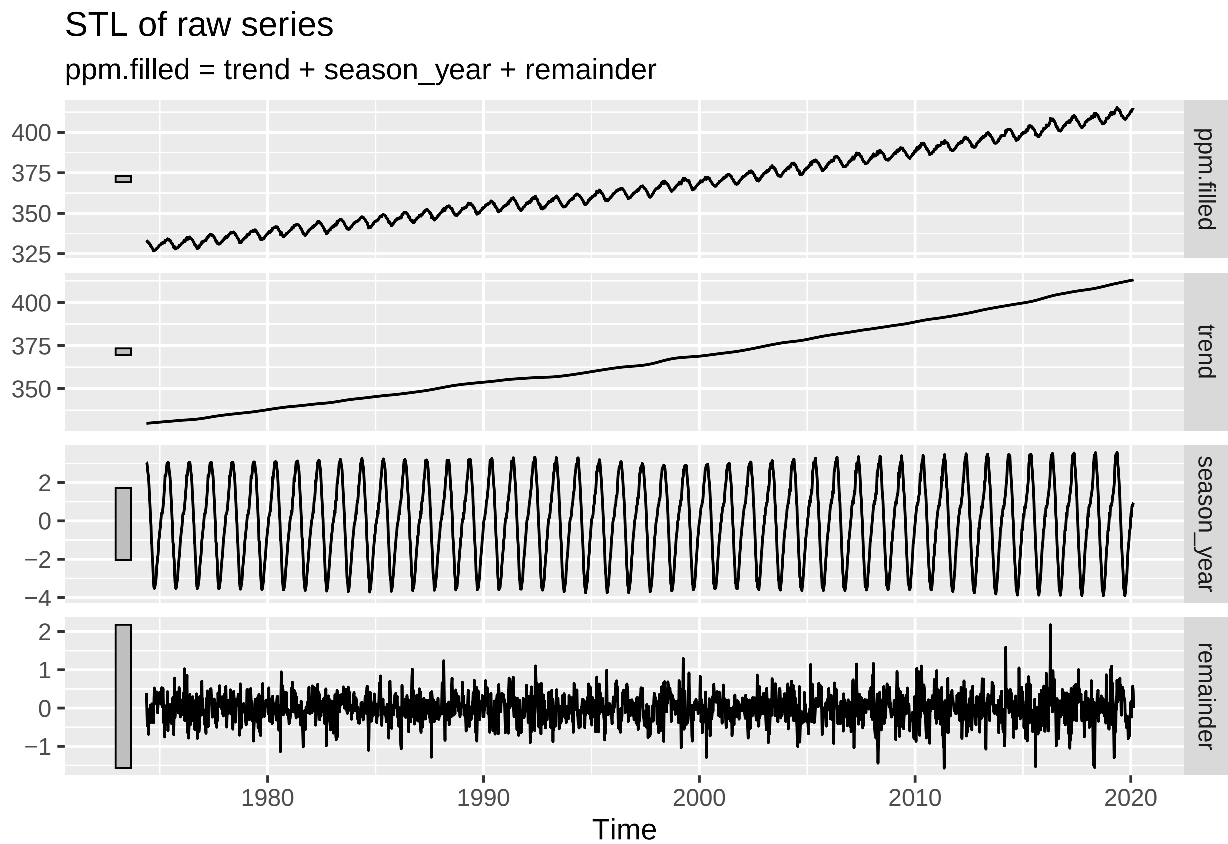




Based on the time series plot of weekly data, we can see that we have a fairly linear trend with mostly constant variance. There are small fluctuations in the variance from 1993 to a little past 2000, but not significant enough requiring any adjustments. The ACF shows very strong autocorrelations, even up to lags of 120, whereas the PACF sharply cuts off to 0 after the first lag. Next, we look at the growth rate in more detail.

We now look at the STL decomposition into trend, seasonal, and remainder components.

```
dat.decomp <- dat.weekly %>% model(STL(ppm.filled)) %>%
  components()
dat.decomp %>% autoplot() + labs(title = 'STL of raw series', x = 'Time')
```



The seasonality appears in a yearly cycle, so it will make sense during modeling time to consider yearly periods. The trend looks mostly linearly, displaying less of the quadratic growth observed in the shorter, monthly data from the previous problem. As a result, the Keeling curve appears to have extended linearly from 1997 to the present. The seasonal variation looks roughly constant, and slightly smaller in amplitude after 1997 to present compared to the 1990s. However, looking at the curve as a whole, there still appears curvature especially around year 1995, so a higher order polynomial might be the best deterministic trend for regression.

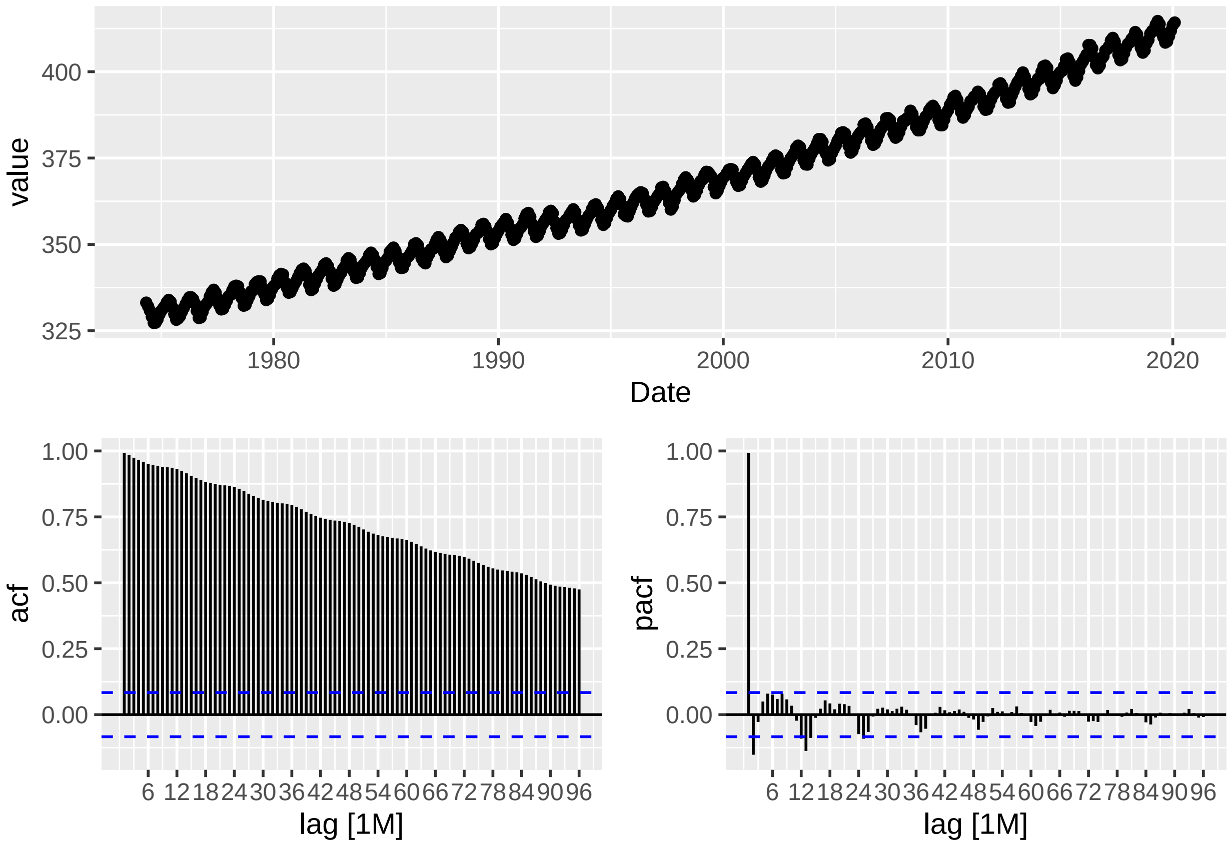
Before comparing forecast performance of our previous models, we now perform the monthly aggregation using a weighted average.

```
#First, multiply ppm by num.days observed to get total ppm over those days (each week)
#Group by yr and month objects in order to aggregate by month
#Using summarise, we calculate a weighted sum of ppm as sum of total ppm for each week within

dat.monthly <- dat %>%
  mutate(ppm.days = ppm.filled * num.days.filled) %>%
  group_by(yr, mon) %>%
  dplyr::summarise(ppm.mon = sum(ppm.days)/sum(num.days.filled)) %>%
  data.frame()

dat.monthly$time <- make_datetime(dat.monthly$yr, dat.monthly$mon) %>% as.Date()
dat.mon <- tsibble(Date = yearmonth(dat.monthly$time), value = dat.monthly$ppm.mon)
```

```
## Using `Date` as index variable.
```

```r
dat$Date <- make_datetime(dat$yr, dat$mon, dat$day) %>% as.Date()
dat <- as_tsibble(dat, index = Date)

#Display monthly data
gg_tsdisplay(dat.mon, plot_type = 'partial', lag_max = 96)
```

```
## Plot variable not specified, automatically selected `y = value`
```



The current values (as of Feb. 2020) from NOAA and our forecasts are below. For our polynomial fit, we will use our quadratic model. Note that due to overfitting and poor forecast performance (second inflection point), we will not look at the cubic model from Part 2, and due to underfitting (failing to capture the curvature), we will not look at our linear model from Part 1. We are also using the middle of the month (2/16/2020) as the value for February, 2020 (our previous models made only monthly forecasts, and not weekly).

```r
get_ci <- function(.distribution) {
    hilo.crap <- hilo(.distribution)
    lower <- hilo.crap$.lower
    upper <- hilo.crap$.upper
    return (c(lower, upper) %>% matrix(ncol = 2, byrow = FALSE))
}

current.value <- dat %>% filter(Date == make_datetime(2020, 02, 16))
current.value <- current.value$ppm.filled
```

```r
current.arima <- arima.forecast %>% filter(year(index) == 2020, month(index) == 2)
current.quad <- series.forecast %>% filter(year(index) == 2020, month(index) == 2)

arima.diff <- current.arima$value - current.value
quad.diff <- current.quad$value - current.value

forecasts <- rbind(quadratic = c(current.quad$value,
                                 get_ci(current.quad$.distribution), quad.diff),
                   arima     = c(current.arima$value,
                               get_ci(current.arima$.distribution), arima.diff))

colnames(forecasts) <- c('value', '95lower', '95upper', 'difference')
forecasts
```

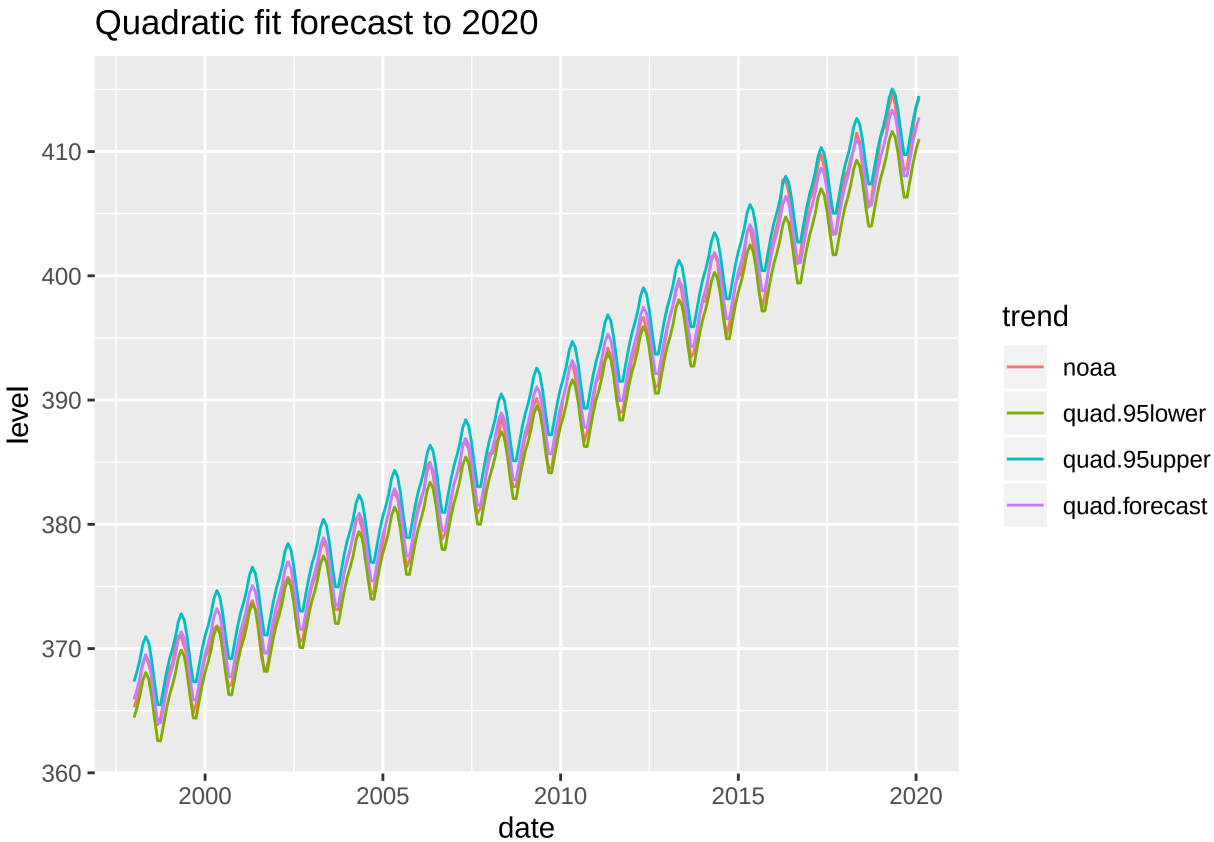
```
##              value  95lower  95upper difference
## quadratic 412.7438 411.0151 414.4726  -1.266199
## arima     399.4278 384.7568 414.0987 -14.582247
```

It looks like the current value on Feb. 2020 day 16 was 414.01. The value from the quadratic forecast was closer at 412.74 than the ARIMA forecast, which was at 399.43. However, the CI for ARIMA contains this value, whereas the quadratic forecast barely misses it, due to the fact that the ARIMA CI is much larger than the quadratic forecast.

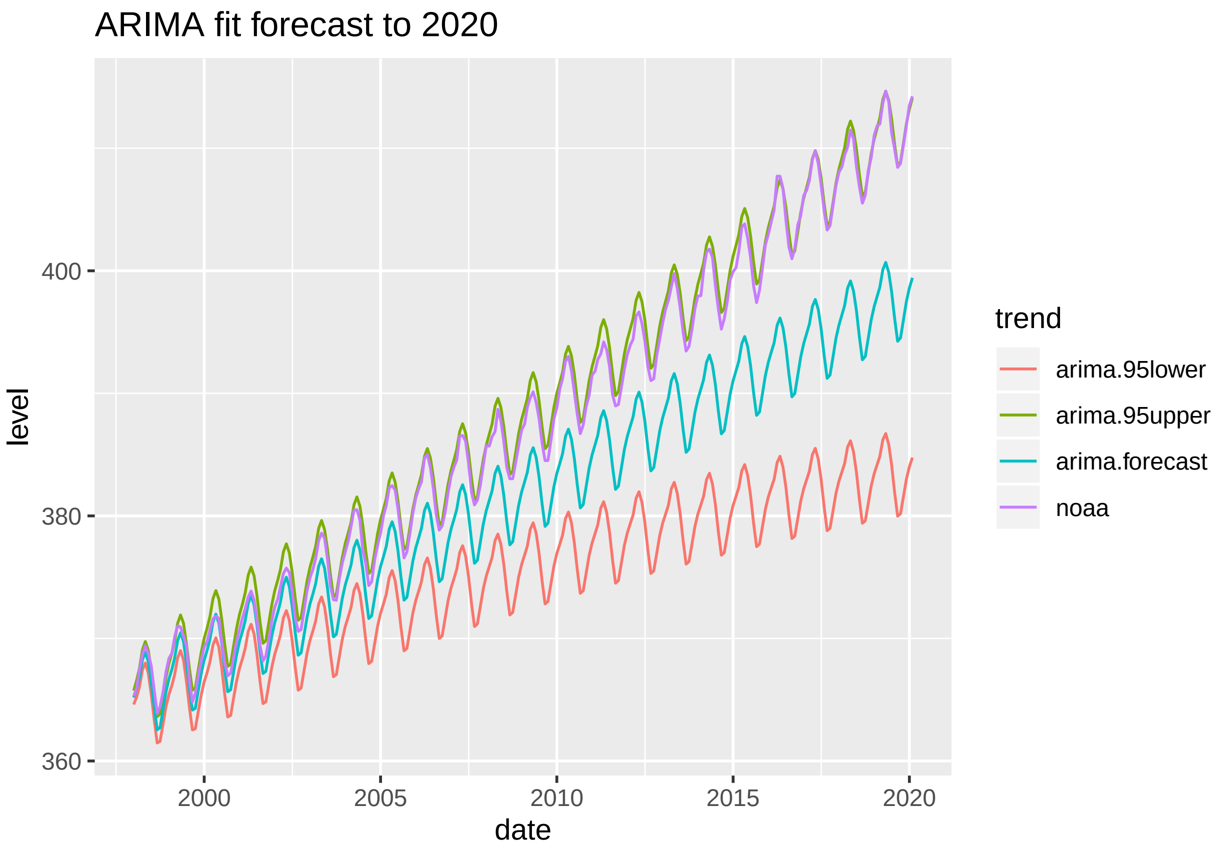
To compare the NOAA curve with our forecasts, we first plot both the quadratic forecast and ARIMA forecast on the same plot.

```r
rmse <- function(series1, series2) {
    return (sqrt(sum((series1 - series2)**2)/length(series1)))
}
dat.mon.compat <- dat.mon %>% filter(year(Date) >= 1998)

plot.arima <- data.frame(date = arima.forecast[1:266,]$index,
                      arima.forecast = arima.forecast[1:266,]$value,
                      arima.95lower = t(get_ci(arima.forecast[1:266,]$.distribution))[1,],
                      arima.95upper = t(get_ci(arima.forecast[1:266,]$.distribution))[2,],
                      noaa = dat.mon.compat$value) %>% gather(trend, level, -date)

plot.quad <- data.frame(date = series.forecast[1:266,]$index,
                      quad.forecast = series.forecast[1:266,]$value,
                      quad.95lower = t(get_ci(series.forecast[1:266,]$.distribution))[1,],
                      quad.95upper = t(get_ci(series.forecast[1:266,]$.distribution))[2,],
                      noaa = dat.mon.compat$value) %>% gather(trend, level, -date)

quad.plot <- ggplot(data = plot.quad, aes(x = date, y = level, color = trend)) +
  ggtitle("Quadratic fit forecast to 2020") +
  geom_line()

arima.plot <- ggplot(data = plot.arima, aes(x = date, y = level, color = trend)) +
```

```
  ggtitle("ARIMA fit forecast to 2020") +
  geom_line()

quad.plot
```




```
arima.plot
```

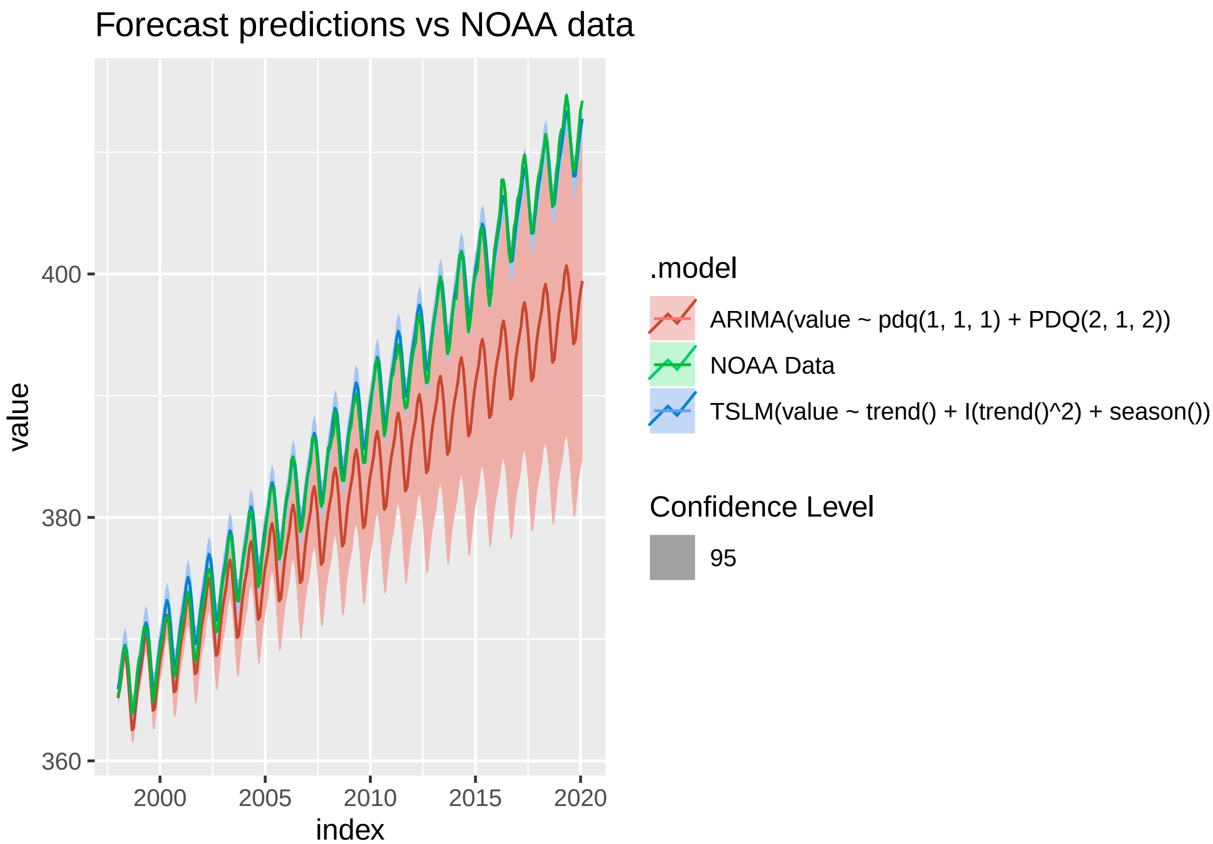



```
print(rbind(quad.rmse = rmse(dat.mon.compat$value,series.forecast[1:266,]$value),
            arima.rmse = rmse(dat.mon.compat$value,arima.forecast[1:266,]$value)))
```

```
##                  [,1]
## quad.rmse  0.8012229
## arima.rmse 7.0985541
```

The seasonal quadratic value rmse is much better compared to the rmse of the ARIMA forecasted estimates. The reason is that ARIMA is generating a linear trend in forecasting. From earlier EDA, we saw that the CO2 increase in quadratic time speed, this explains whey ARIMA model generate lower forecasting values compare to quadratic time model and actual values. We show the predictions again on a second plot, where the confidence intervals are colored rather than as a separate series.

```
dat.mon.compat <- dat.mon %>% filter(year(Date) >= 1998)
plot_noaa <- data.frame(noaa = dat.mon.compat$value, data = 'NOAA Data')

autoplot(rbind(arima.forecast[1:266,], series.forecast[1:266,]), level = 95) +
  geom_line(data = plot_noaa, aes(x = arima.forecast[1:266, ]$index, y = noaa, color = data)) +
  labs(level = 'Confidence Level',
       title = "Forecast predictions vs NOAA data")
```



Part 5: Seasonally vs. Non-seasonally Adjusted Models

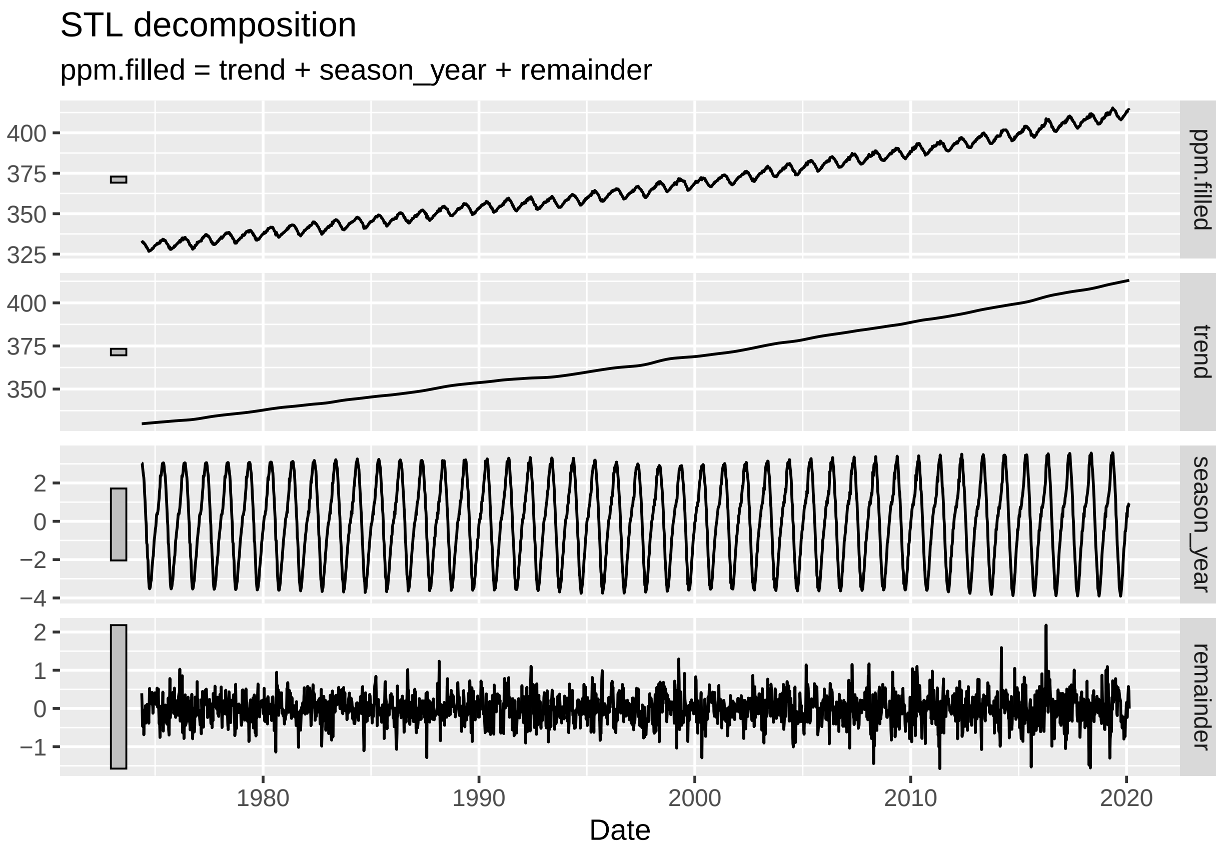
Here, we seasonally adjust the weekly NOAA data, and split both seasonally-adjusted (SA) and non-seasonally-adjusted (NSA) series into training and test sets, using the last two years of observations as the test sets. For both SA and NSA series, fit ARIMA models using all appropriate steps. Measure and discuss how your models perform in-sample and (psuedo-) out-of-sample, comparing candidate models and explaining your choice. In addition, fit a polynomial time-trend model to the seasonally-adjusted series and compare its performance to that of your ARIMA model.

The steps we will take come from sync lectures:

1. Plot the data. Identify any unusual observations.
2. If necessary, transform the data (e.g. using a Box-Cox transformation) to stabilize the variance.
3. If the data are non-stationary: take first differences of the data until the data are stationary.
4. Examine the ACF/PACF: Is an AR(p) or MA(q) model appropriate?
5. Try your chosen model(s), and use appropriate metrics to choose a model.
6. Model evaluation Check the residuals from your chosen model by plotting the ACF of the residuals, and doing a portmanteau test of the residuals. If they do not look like white noise, try a modified model.
7. Once the residuals look like white noise, calculate forecasts.

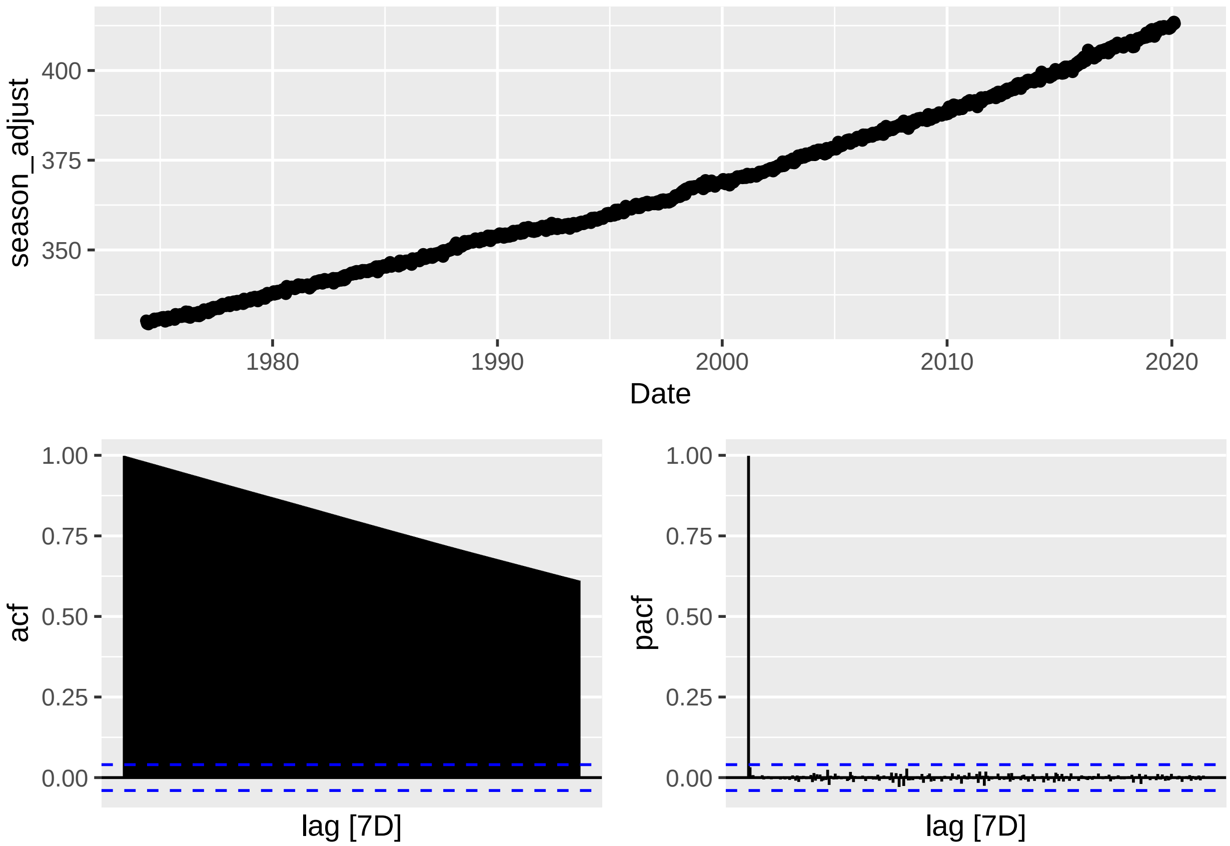
First, we will seasonally adjust the weekly data using STL.

```
stl <- dat %>% model(STL(ppm.filled)) %>% components()
stl %>% autoplot()
```



The seasonally adjusted series, along with its ACF and PACF are displayed below.

```
stl %>% gg_tsdisplay(season_adjust, plot_type = 'partial', lag_max = 300)
```

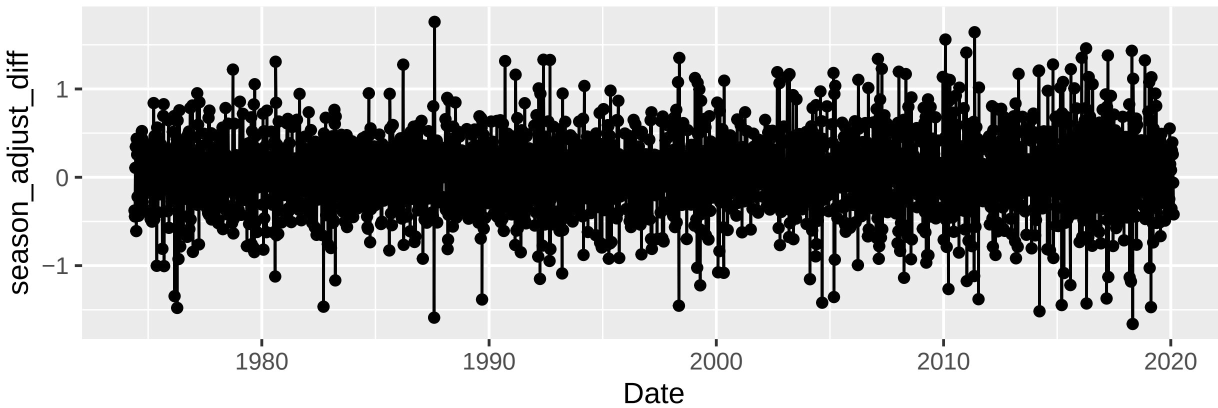


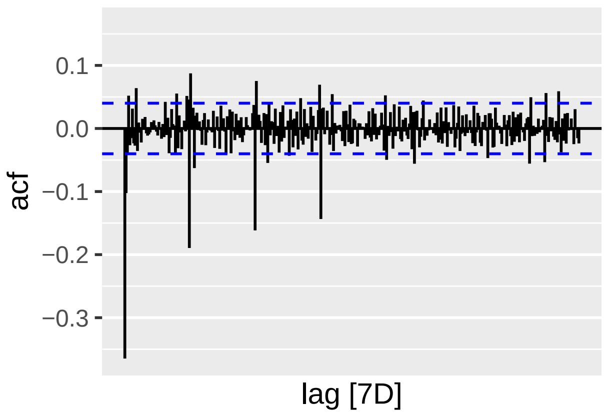
Clearly, there's an upward trend, so we will observe the first order difference of the data. Since the data is already seasonally-adjusted, we will only first look at the first order non-seasonal difference.

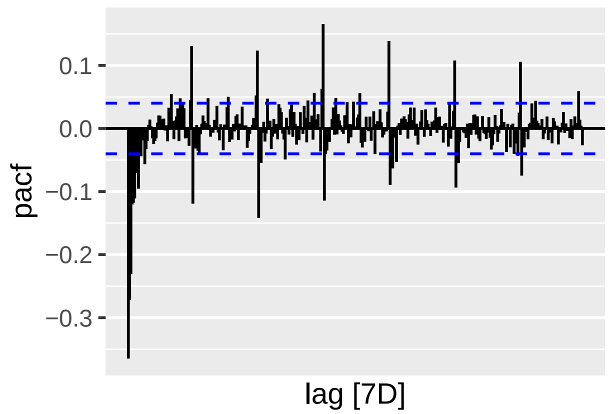
```
#First order difference works very well
stl <- stl %>% mutate(season_adjust_diff = season_adjust %>% difference())
gg_tsdisplay(stl, season_adjust_diff, plot='partial', lag_max = 360)
```

```
## Warning: Removed 1 rows containing missing values (geom_path).
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```







The series look stationary both in terms of the mean and the variance. The series clearly fluctuates without a trend around the mean, and the variance does not seem to be either increasing or decreasing as we increase the year. Therefore, no transformations on the data are needed. To verify stationarity in the mean, we can conduct the Augmented Dickey-Fuller Test test at a significance level of 0.05:

```
adf.test(stl$season_adjust_diff[-1])
```

```
## Warning in adf.test(stl$season_adjust_diff[-1]): p-value smaller than
## printed p-value
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  stl$season_adjust_diff[-1]
## Dickey-Fuller = -18.858, Lag order = 13, p-value = 0.01
## alternative hypothesis: stationary
```

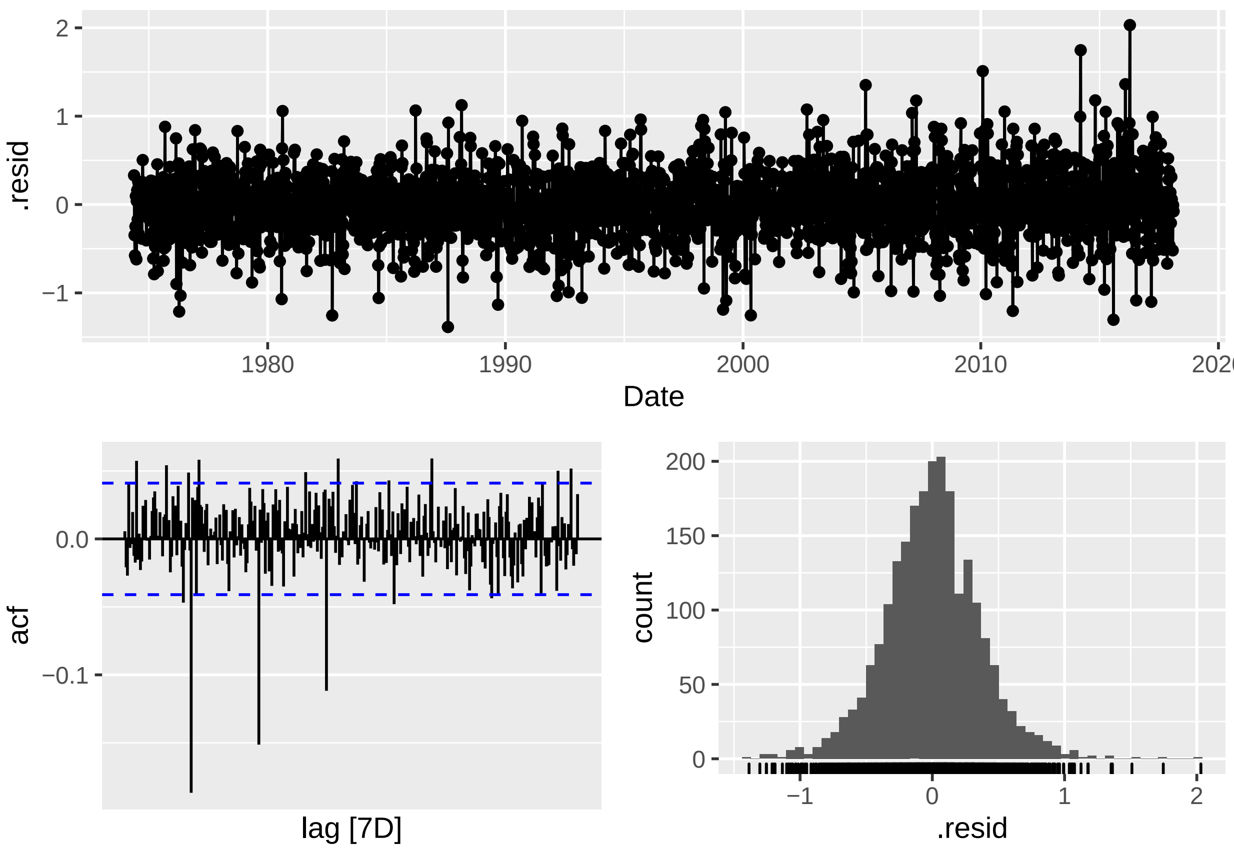
Based on the ACF/PACF plots, the PACF seems to slowly decay, while the ACF cuts off quicker at a lag of 2. This would indicate that an MA model might be suitable. However, the ACF/PACF is too complex to be simplified in such a way, so we will scan a range of models. Both plots also display strong seasonality at 52 weeks (1 year), and other yearly lags, indicating that there is a strong yearly seasonality. We will take this into account in our ARIMA modeling, where we look for have non-zero PQ terms at the 52 week frequencies.

We will also now divide up the data into train and test sets. Since 2020 ends in February 16, we will set aside 2018-2-16 - end as test set.

```
#split train/test
stl.train <- stl %>% filter(Date <= as.Date("2018-02-16"))
stl.test <- stl %>% filter(Date > as.Date("2018-02-16"))
```

First, we see what model auto ARIMA gives us:

```
mod.sea.auto <- stl.train %>% model(ARIMA(season_adjust))
gg_tsresiduals(mod.sea.auto, lag_max = 350)
```




```
mod.sea.auto %>% report()
```

```
## Series: season_adjust
## Model: ARIMA(1,1,1) w/ drift
##
## Coefficients:
##          ar1      ma1  constant
##       0.2226  -0.8371    0.0265
## s.e.  0.0262   0.0140    0.0012
##
## sigma^2 estimated as 0.1326:  log likelihood=-931.52
## AIC=1871.04   AICc=1871.05   BIC=1893.97
```

```
mod.sea.auto %>% augment() %>% features(.resid, ljung_box, lag = 36)
```

```
## # A tibble: 1 x 3
##   .model              lb_stat lb_pvalue
##   <chr>                 <dbl>     <dbl>
## 1 ARIMA(season_adjust)    42.1     0.223
```

There are still strong autocorrelations in the ACF of the residuals plot from the auto ARIMA model at bi-yearly frequencies, suggesting we could perhaps do better. The model also fails the Ljung-Box test, since p << 0.01 (we reject the null hypothesis that the series is white noise).

We will check the out of sample performance:

```
mod.sea.auto.forecast <- mod.sea.auto %>%forecast(h=104)
RMSE.auto <- sqrt(mean((mod.sea.auto.forecast$season_adjust - stl.test$season_adjust)**2))
```

```
## Warning in mod.sea.auto.forecast$season_adjust - stl.test$season_adjust:
## longer object length is not a multiple of shorter object length
```

```
RMSE.auto
```

```
## [1] 1.369598
```

Also, the auto ARIMA detected a mean in a series, which it modeled with a constant term c. We find that the mean of the seasonally adjusted differenced series is around 0.035:

```
mean(stl$season_adjust_diff[-1])
```

```
## [1] 0.03467717
```

We can either remove the mean from the series and model without a constant, or keep it in and have the model find the appropriate constant. We will go with the latter.

We will now scan multiple models, and record the AICc and RMSE. We will emphasize RMSE, the out-of-sample fit, as being more important than AICc in this case, since having a test set is usually a better way to select a model than any in-sample fit metrics. Based on the EDA, we will always set d=1, and D=0.

```
#Function defined previously
fit_aicc <- function(p,d,q,P,D,Q) {

  out <- tryCatch(
    {
      #Try to fit model
      mod.fit <- stl.train %>% model(ARIMA(season_adjust ~ 1 + pdq(p,d,q) + PDQ(P,D,Q, period =
      #If AICc cannot be found, then the model failed to converge
      AICc <- glance(mod.fit)$AICc
      #Get the RMSE by first forecasting 104 weeks
      mod.forecast <- mod.fit %>%forecast(h=104)
      RMSE <- sqrt(mean((mod.forecast$season_adjust - stl.test$season_adjust)**2))
    },

    error = function(e) {
      return(NA)
    },
```

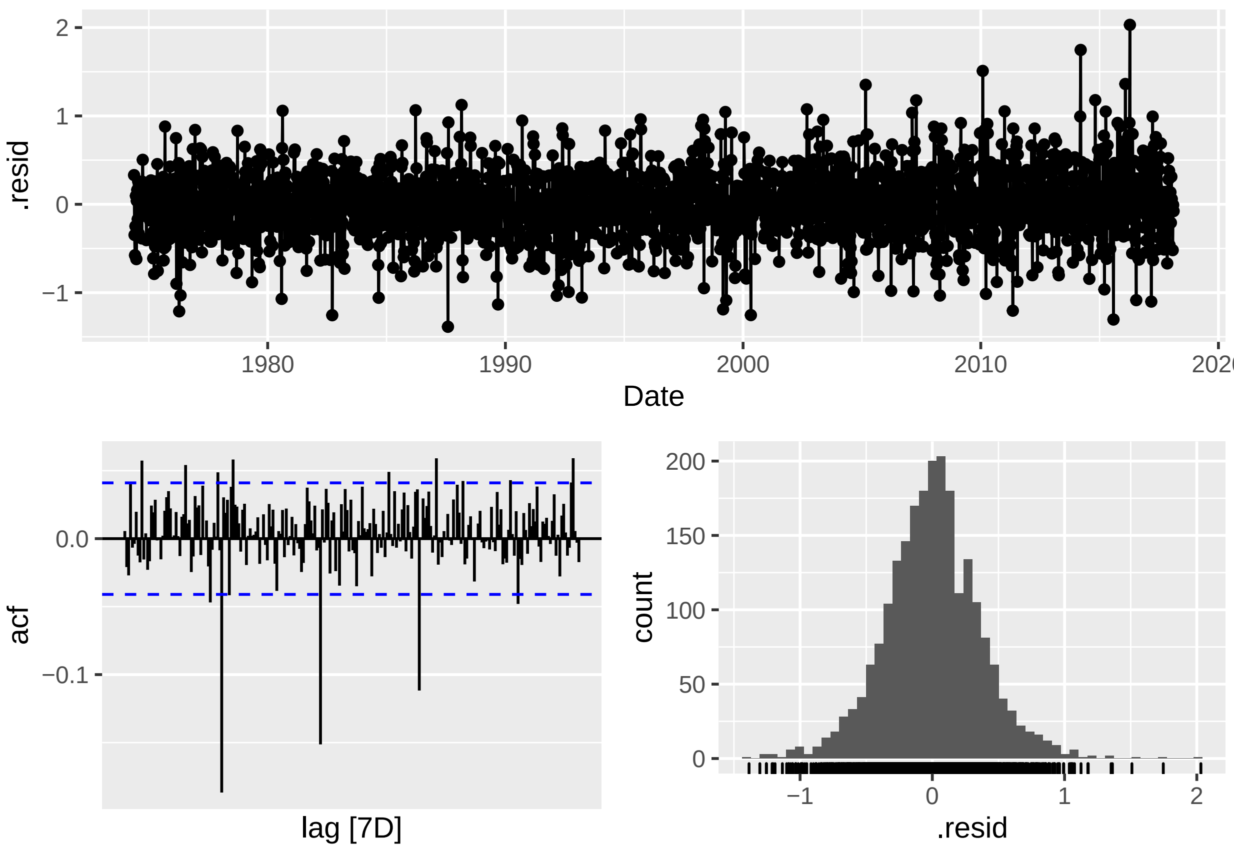
```
    warning = function(w) {
      return(NA)
    }
  )
  if (!is.na(out)){
      answer = data.frame(cbind(p=p,d=d,q=q,P=P,D=D,Q=Q,AICc=AICc,RMSE=RMSE))
      return(answer)
  }
  else {
      return(NA)
  }
}

results3 <- data.frame(p = integer(), d = integer(), q = integer(), P = integer(), D = integer

run = FALSE
if (run) {
  for (p in seq(1,4)) {
    for (q in seq(1,4)) {
      for (d in seq(1,1)) {
        for (P in seq(0,1)) {
          for (D in seq(0,0)) {
            for (Q in seq(0,1)) {

#    for (cond in 1:dim(conditions)[1]) {
        answer = fit_aicc(p,d,q,P,D,Q)
        if (!is.na(answer[1])){
            if (dim(answer)[1] == 1) {
                results3 <- rbind(results3, answer)
            }

        #Just for progress
        }
        print(tail(results3, 1))
            }}}}}}
    }

#Results from the scan. Will not be re-run each time because it's too slow
#> head(results3[order(results3$RMSE), ])
#   p  d  q  P  D  Q      AICc     RMSE
#13 1  1  1  0  0  0  1870.906 1.292273
#5  0  1  1  0  0  0  1937.059 1.322440
#3  0  1  0  1  0  0  2535.480 1.326527
#7  0  1  1  1  0  0  1852.844 1.342996
#9  1  1  0  0  0  0  2311.881 1.357979
#11 1  1  0  1  0  0  2212.195 1.358994
```

Since d and D are set to be the same for all models, we can directly compare the AICc as a measure of in-sample fit. It seems like even though auto ARIMA did not find the best model based on AICc, it did find the best model out of the set we scanned based on out-of-sample RMSE. This is typically a better metric because it measures model generalization, which is desirable for making forecasts. We will therefore take this model with parameters found by looping and auto ARIMA.

```
#Best model found by ARIMA based on AICc
mod.season.adj.best <- stl.train %>% model(ARIMA(season_adjust ~ 1 + pdq(1,1,1) + PDQ(0,0,0,pe
mod.season.adj.best %>% report()
```

```
## Series: season_adjust
## Model: ARIMA(1,1,1) w/ drift
##
## Coefficients:
##          ar1      ma1  constant
##       0.2226  -0.8371    0.0265
## s.e.  0.0262   0.0140    0.0012
##
## sigma^2 estimated as 0.1326:  log likelihood=-931.52
## AIC=1871.04   AICc=1871.05   BIC=1893.97
```

```
mod.season.adj.best %>% gg_tsresiduals(lag_max = 240)
```




```
mod.season.adj.best %>% augment() %>% features(.resid, ljung_box, lag = 51)
```

```
## # A tibble: 1 x 3
```

```
##   .model                                                  lb_stat lb_pvalue
##   <chr>                                                     <dbl>     <dbl>
## 1 ARIMA(season_adjust ~ 1 + pdq(1, 1, 1) + PDQ(0, 0, 0, ~    64.0     0.104
```

```r
mod.season.adj.best %>% augment() %>% features(.resid, ljung_box, lag = 52)
```

```
## # A tibble: 1 x 3
##   .model                                                  lb_stat lb_pvalue
##   <chr>                                                     <dbl>     <dbl>
## 1 ARIMA(season_adjust ~ 1 + pdq(1, 1, 1) + PDQ(0, 0, 0, ~    146.  8.56e-11
```

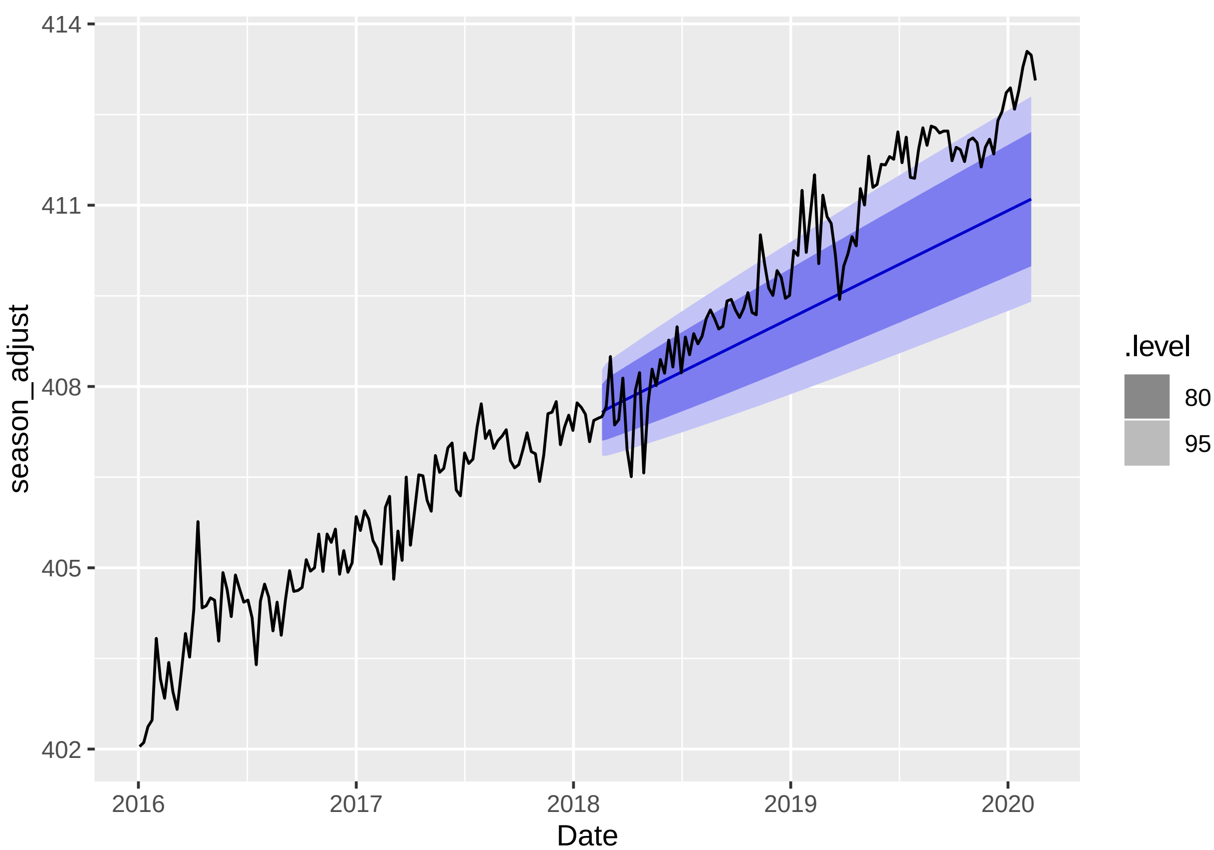
Our model:

$$(1 - 0.2226B)(1 - B)y_t = 0.0265 + (1 - 0.837B)\omega_t$$

We see that at lag 51, the Ljung-box says that the series is white noise (or at least fail to reject the null hypothesis), but if we look up to lags of 1 year, we fail the test with a very large p-value. As a result, we still have strong seasonality in the residual series. Even with seasonally adjusted data, for high frequency data, it is difficult to completely eliminate seasonality.

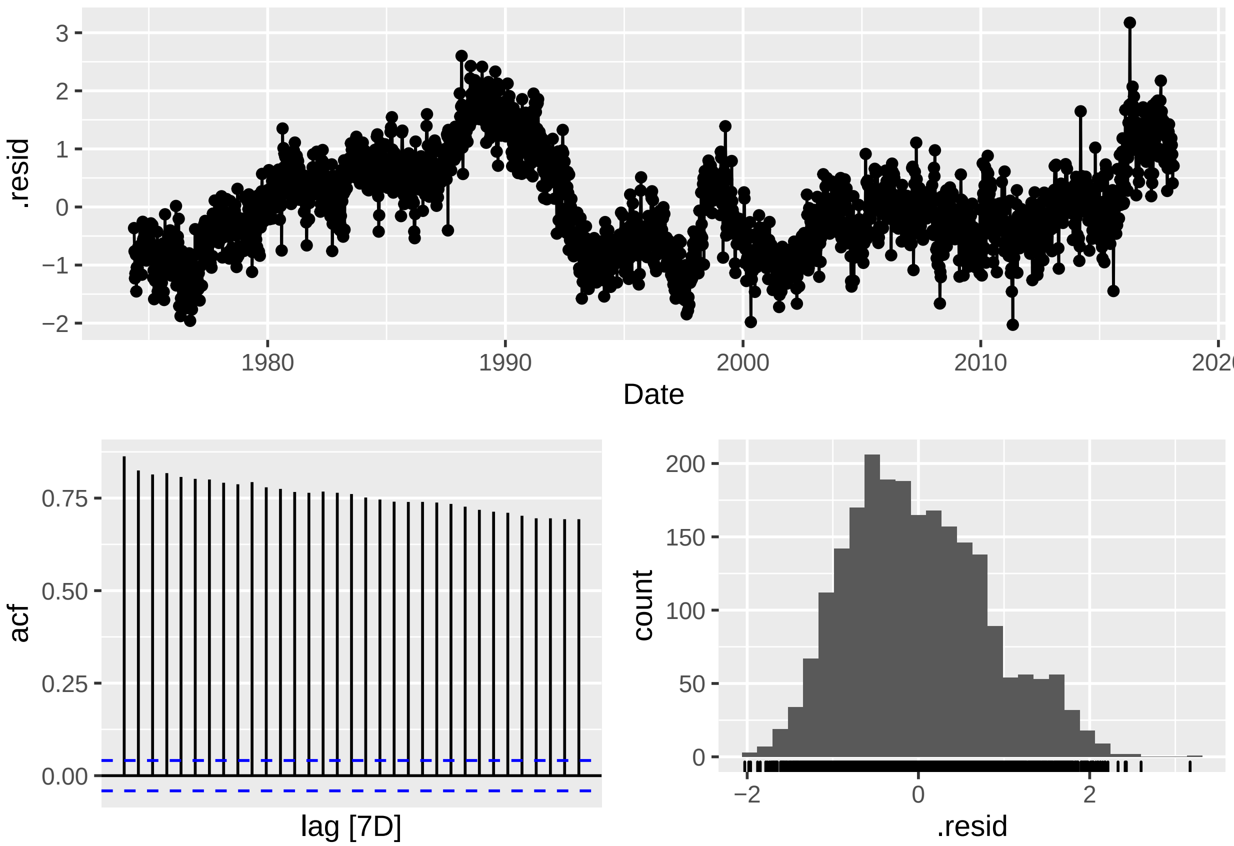
We now generate a forecast plot onto our original series. We also show recent years from which the data was fitted.

```r
stl.recent <- stl %>% filter(year(Date) > 2015)
mod.sa.forecast <- mod.season.adj.best %>%forecast(h=104)
autoplot(mod.sa.forecast) +
  geom_line(data = stl.recent, aes(x = Date, y = season_adjust))
```



We now compare the ARIMA forecast to quadratic time fit. This is the curve we found to be best for the original series, and could be used to explain the slight curvature in overall trend as found in the EDA.

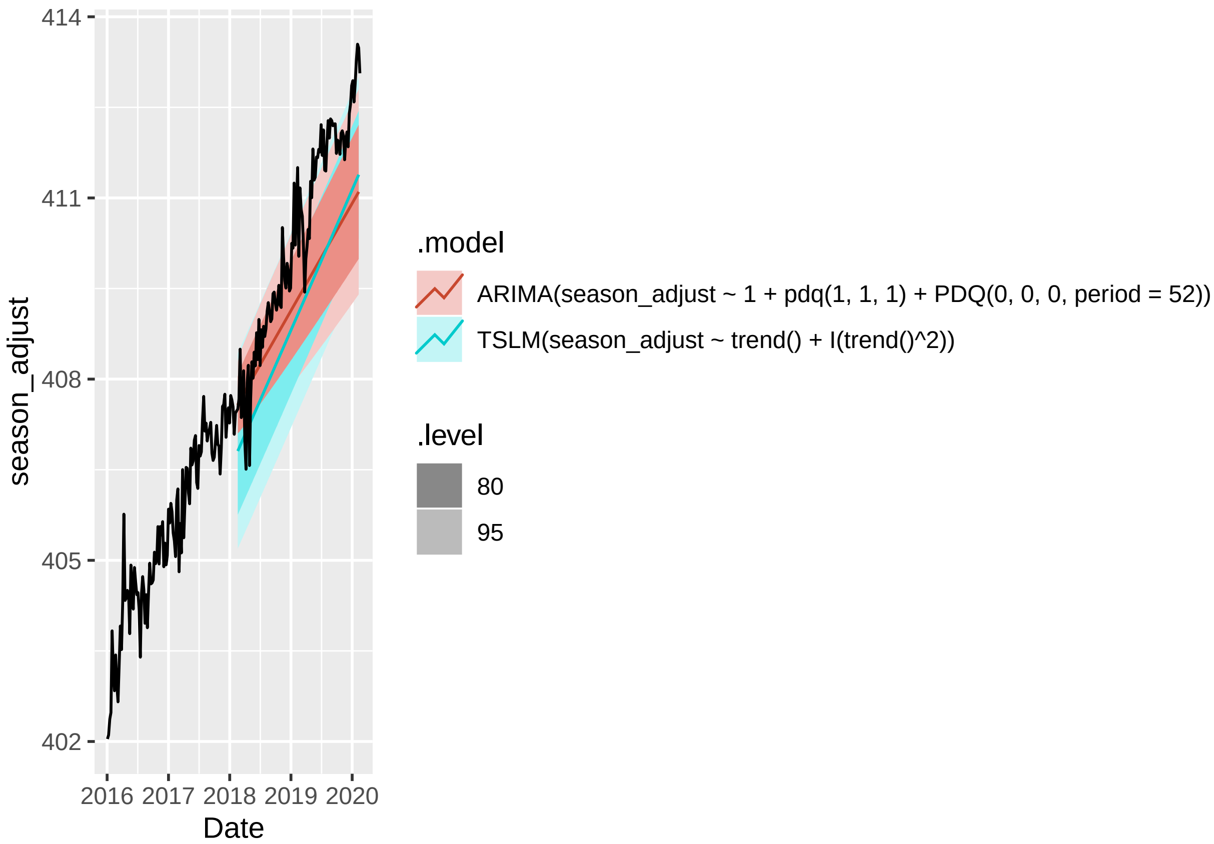
```
#With quadratic time trend
quad.season.adj.fit <- stl.train %>% model(TSLM(season_adjust ~ trend() + I(trend()^2)))

gg_tsresiduals(quad.season.adj.fit)
```




```
quad.season.adj.fit %>% augment() %>% features(.resid, ljung_box)
```

```
## # A tibble: 1 x 3
##   .model                                      lb_stat lb_pvalue
##   <chr>                                         <dbl>     <dbl>
## 1 TSLM(season_adjust ~ trend() + I(trend()^2))   1702.         0
```

```
quad.forecast <- quad.season.adj.fit %>% forecast(h=104)
autoplot(rbind(mod.sa.forecast, quad.forecast)) +
  geom_line(data = stl.recent, aes(x = Date, y = season_adjust))
```



RMSE is the best metric to use for comparison, as it is measure of generalization error. In this case, ARIMA is slightly better than for quadratic fit:

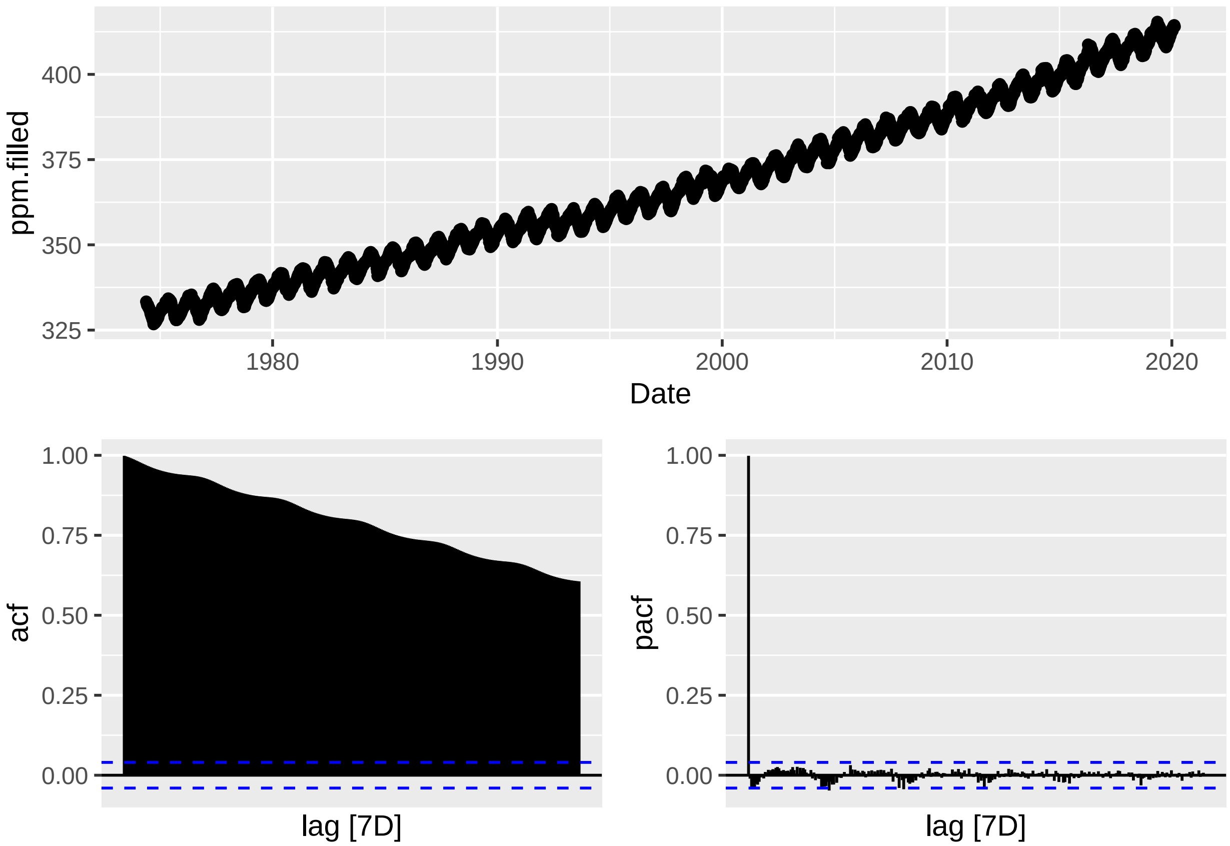
```
rbind(quad.rmse = mean((quad.forecast$season_adjust - stl.test$season_adjust) ** 2) %>% sqrt()
      arima.rmse = mean((mod.sa.forecast$season_adjust - stl.test$season_adjust) ** 2) %>% sqr
```

```
## Warning in quad.forecast$season_adjust - stl.test$season_adjust: longer
## object length is not a multiple of shorter object length

## Warning in mod.sa.forecast$season_adjust - stl.test$season_adjust: longer
## object length is not a multiple of shorter object length

##                [,1]
## quad.rmse  1.495876
## arima.rmse 1.369598
```

Non-seasonally adjusted series For the original series, we will now perform a similar exercise with non-seasonally adjusted data.

The original series, along with its ACF and PACF are displayed below.

```
dat %>% gg_tsdisplay(ppm.filled, plot_type = 'partial', lag_max = 300)
```

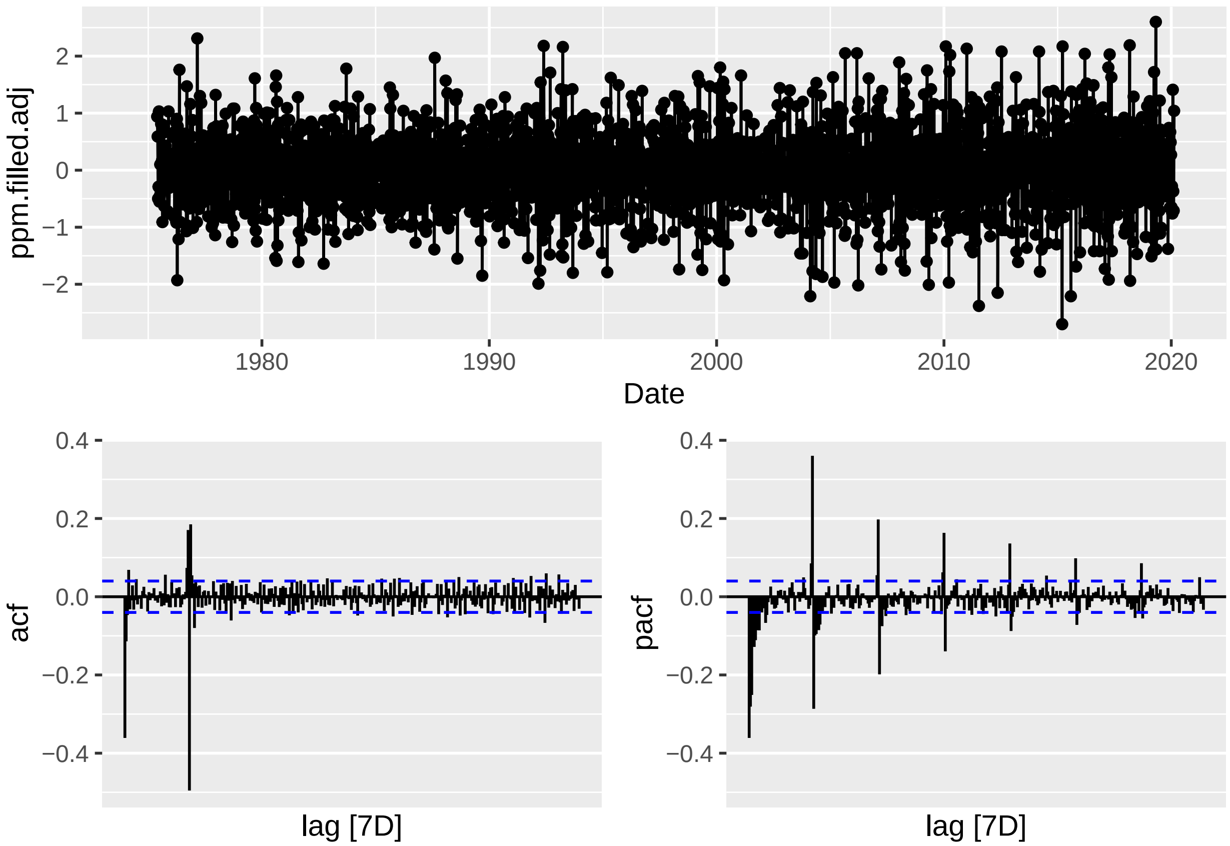


Clearly, there's an upward trend, and there is clear seasonality, so we will observe the first order difference of the data along with first order seasonal difference.

```
#First order difference + seasonal difference works very well
dat <- dat  %>% mutate(ppm.filled.adj = ppm.filled %>% difference() %>% difference(52))
gg_tsdisplay(dat, ppm.filled.adj, plot='partial', lag_max = 360)
```

```
## Warning: Removed 53 rows containing missing values (geom_path).
```

```
## Warning: Removed 53 rows containing missing values (geom_point).
```



The series look stationary both in terms of the mean and the variance. The series clearly fluctuates without a trend around the mean, and the variance does not seem to be either increasing or decreasing as we increase the year. Therefore, no transformations on the data are needed. To verify stationarity in the mean, we can conduct the Augmented Dickey-Fuller Test test at a significance level of 0.05:

```r
adf.test(dat$ppm.filled.adj[-1:-53])
```

```
## Warning in adf.test(dat$ppm.filled.adj[-1:-53]): p-value smaller than
## printed p-value

##
##  Augmented Dickey-Fuller Test
##
## data:  dat$ppm.filled.adj[-1:-53]
## Dickey-Fuller = -18.789, Lag order = 13, p-value = 0.01
## alternative hypothesis: stationary
```

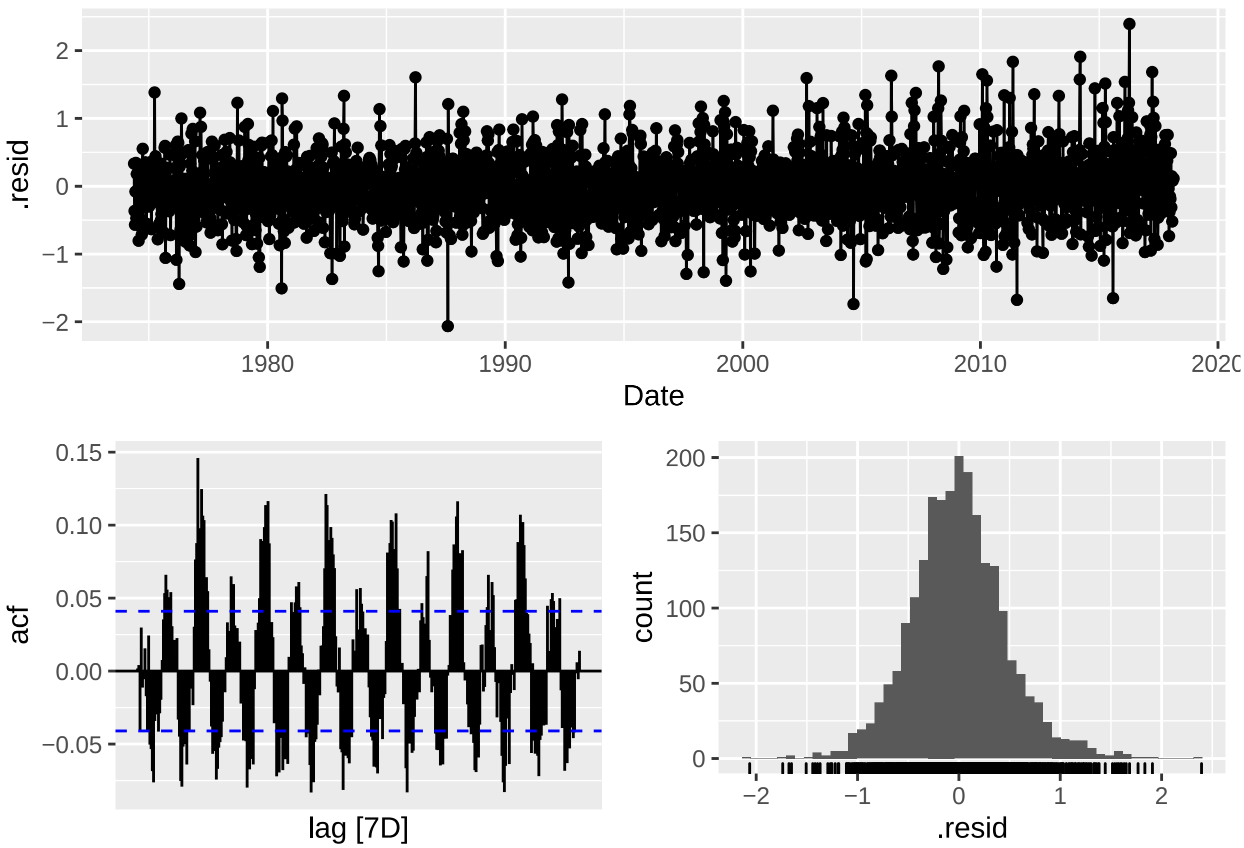
Since we have p-value < 0.01, we reject H0 and state that we have a stationary series.

Based on the ACF/PACF plots, the PACF seems to decay slowly, while the ACF cuts off quicker. Both though still show significant seasonal lag of 1 year. The ACF/PACF is too complex to be simplified using only MA or AR models, so we will scan a range of models.

We now divide the series into train and test sets, and first we see what model auto ARIMA gives us:

```r
#split train/test
dat$Date <- make_datetime(dat$yr, dat$mon, dat$day) %>% as.Date()
dat.train <- dat %>% filter(Date <= as.Date("2018-02-18"))
dat.test <- dat %>% filter(Date > as.Date("2018-02-18"))
```

```r
mod.nonsea.auto <- dat.train %>% model(ARIMA(ppm.filled))
gg_tsresiduals(mod.nonsea.auto, lag_max = 360)
```




```r
mod.nonsea.auto %>% report()
```

```
## Series: ppm.filled
## Model: ARIMA(2,1,4) w/ drift
##
## Coefficients:
##          ar1      ar2      ma1     ma2      ma3      ma4  constant
##       1.9394  -0.9577  -2.1963  1.4816  -0.1710  -0.1112     6e-04
## s.e.  0.0082   0.0079   0.0214  0.0501   0.0522   0.0248     0e+00
##
## sigma^2 estimated as 0.224:  log likelihood=-1529.68
## AIC=3075.36   AICc=3075.42   BIC=3121.22
```

```r
mod.nonsea.auto %>% augment() %>% features(.resid, ljung_box, lag = 360)
```

```
## # A tibble: 1 x 3
##   .model            lb_stat lb_pvalue
```

```
##   <chr>                <dbl>     <dbl>
## 1 ARIMA(ppm.filled)   2416.         0
```

There are still strong autocorrelations in the ACF of the residuals plot from the auto ARIMA even at lag 6. The model also fails the Ljung-Box test, since p << 0.01 (we reject the null hypothesis that the series is white noise).

The RMSE for this forcast is shown below

```
mod.forecast <- mod.nonsea.auto %>%forecast(h=104)
RMSE <- sqrt(mean((mod.forecast$ppm.filled - dat.test$ppm.filled)**2))
RMSE
```

```
## [1] 2.720996
```

Like the seasonally-adjusted series, we will scan models based on AICc and RMSE. Note again we will set d = 1 for the scan. The same is true for the sAR1 term, where any value other than 1 or 1 will fail.

We scan some models and capture their AICc and RMSE to the test set.

```
#Function defined previously
fit_aicc <- function(p,d,q,P,D,Q) {

  out <- tryCatch(
    {
      #Try to fit model
      mod.fit <- dat.train %>% model(ARIMA(ppm.filled ~ 1 + pdq(p,d,q) + PDQ(P,D,Q, period = 52
      #If AICc cannot be found, then the model failed to converge
      AICc <- glance(mod.fit)$AICc

      #Get the RMSE by first forecasting 104 weeks
      mod.forecast <- mod.fit %>%forecast(h=104)
      RMSE <- sqrt(mean((mod.forecast$ppm.filled - dat.test$ppm.filled)**2))
    },

    error = function(e) {
      return(NA)
    },

    warning = function(w) {
      return(NA)
    }
  )
  if (!is.na(out)){
      return(data.frame(cbind(p=p,d=d,q=q,P=P,D=D,Q=Q,AICc=AICc,RMSE=RMSE)))
  }
  else {
      return(NA)
  }
}
```

```r
results4 <- data.frame(p = integer(), d = integer(), q = integer(), P = integer(), D = integer

run = FALSE
if (run) {
  for (p in seq(0,3)) {
    for (q in seq(2,5)) {
      for (d in seq(1,1)) {
        for (P in seq(0,1)) {
          for (D in seq(0,1)) {
            for (Q in seq(0,2)) {

        answer = answer = fit_aicc(p,d,q,P,D,Q)
        if (!is.na(answer[1])){
            if (dim(answer)[1] == 1) {
                results4 <- rbind(results4, answer)
            }
        }
        #Just for progress
        print(dim(results4))
        }}}}}}
}

#RESULTS
#results4[order(results4$RMSE),]
#   p d q P D Q     AICc      RMSE
#7  0 1 2 1 1 2 2557.081 0.6758327
#22 3 1 4 1 0 1 2725.591 0.7677650
#17 2 1 4 1 0 1 2733.933 0.8915470
#13 2 1 3 1 0 1 2749.908 0.9030012
#3  1 1 3 1 0 0 3280.112 2.0719085
#25 3 1 5 1 0 0 3267.984 2.0754739
#6  1 1 4 1 0 0 3259.468 2.0834647
#9  1 1 5 1 0 0 3260.517 2.0854689
#2  1 1 3 0 0 1 3289.197 2.1473776
```

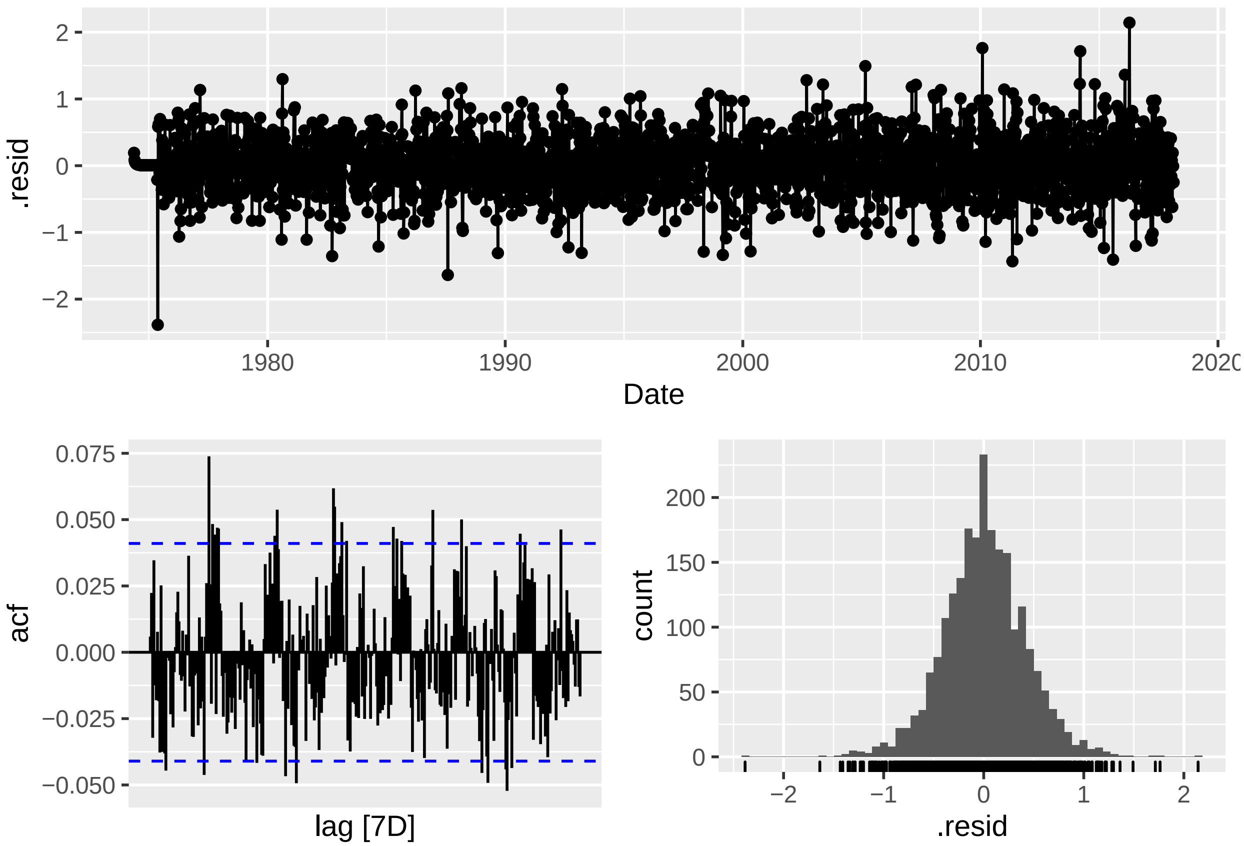
Our scan found models that are much better than auto ARIMA in terms of AICc and RMSE. The best model in terms of RMSE is also the best one in terms of AICc, which is is 0,1,2,1,1,2.

```r
#Best model found by ARIMA based on AICc
mod.nonseason.adj.best <- dat.train %>% model(ARIMA(ppm.filled ~ pdq(0,1,2) + PDQ(1,1,2, perio
mod.nonseason.adj.best %>% report()
```

```
## Series: ppm.filled
## Model: ARIMA(0,1,2)(1,1,2)[52]
##
## Coefficients:
##          ma1      ma2    sar1     sma1    sma2
##      -0.5872  -0.1679  0.9730  -1.8661  0.8812
```

```
## s.e.   0.0210   0.0210  0.0129   0.0199  0.0178
##
## sigma^2 estimated as 0.1775:  log likelihood=-1272.76
## AIC=2557.52   AICc=2557.56   BIC=2591.79
```

```r
mod.nonseason.adj.best %>% gg_tsresiduals(lag_max = 360)
```




```r
mod.nonseason.adj.best %>% augment() %>% features(.resid, ljung_box, lag = 36)
```

```
## # A tibble: 1 x 3
##   .model                                              lb_stat lb_pvalue
##   <chr>                                                 <dbl>     <dbl>
## 1 ARIMA(ppm.filled ~ pdq(0, 1, 2) + PDQ(1, 1, 2, period ~    39.7     0.308
```

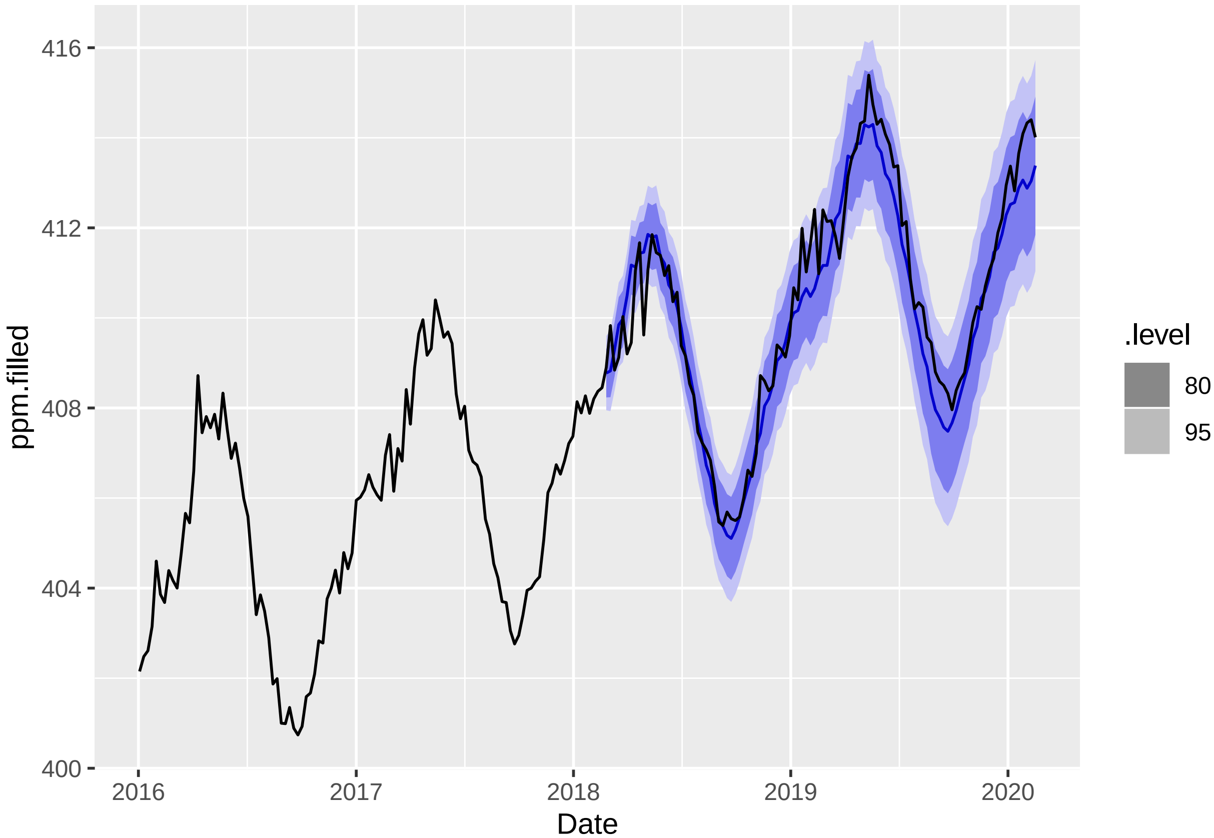
Our model:

$$(1 - 0.9730B^{52})(1 - B)(1 - B^{52})y_t = (1 - 0.5872B - 0.1679B^2)(1 - 1.8661B^{52} + 0.8812B^{104})\omega_t$$

The residuals are still highly correlated, so it is not surprising that the Ljung-box test fails. It is difficult to eliminate seasonality entirely, especially with high frequency data (such as the weekly data we have here). Therefore, we will proceed with forecasting.

We now generate a forecast plot onto our original series. We also show recent years from which the data was fitted. The forecast generally looks very good.

```r
dat.recent <- dat %>% filter(year(Date) > 2015)
mod.nonseason.forecast <- mod.nonseason.adj.best %>%forecast(h=104)
autoplot(mod.nonseason.forecast) +
 geom_line(data = dat.recent, aes(x = Date, y = ppm.filled))
```




```r
RMSE <- sqrt(mean((mod.nonseason.forecast$ppm.filled - dat.test$ppm.filled)**2))
RMSE
```

```
## [1] 0.6758327
```

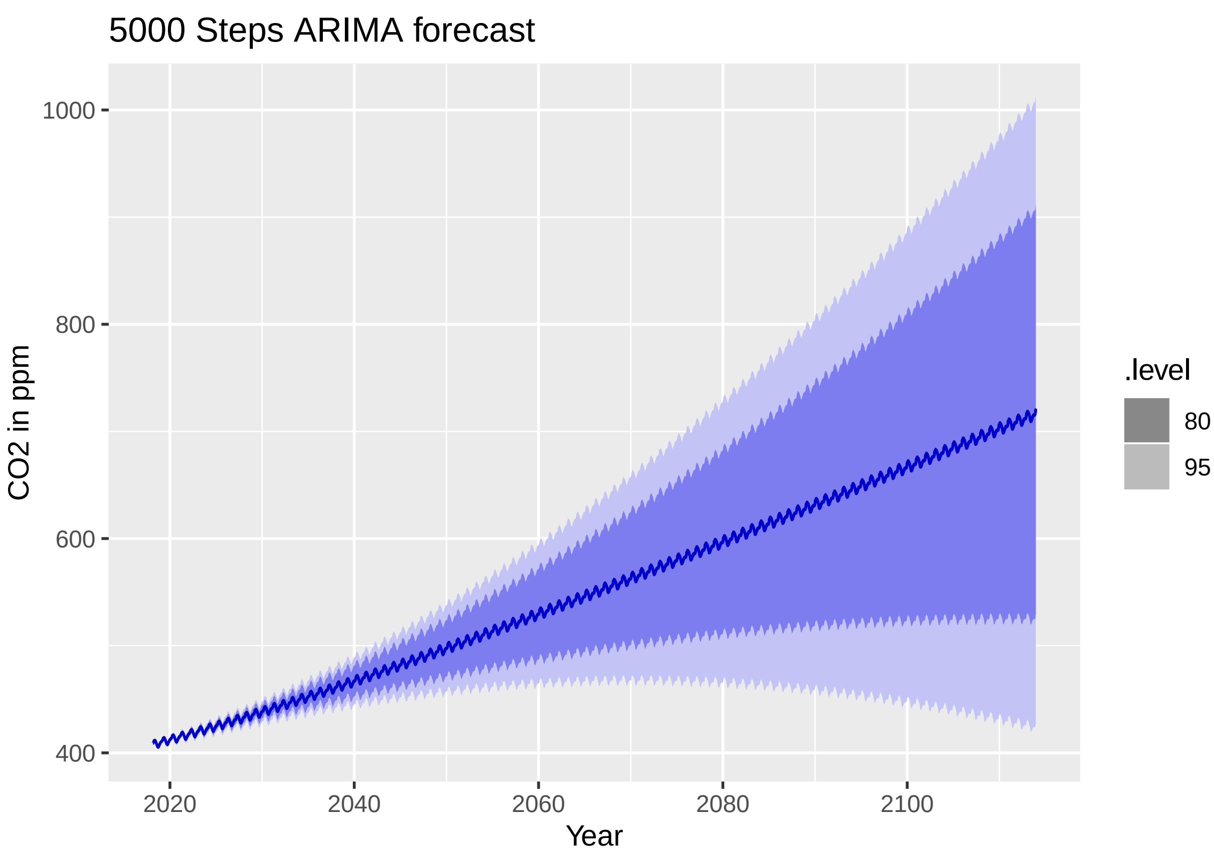
*

*Part 6 (3 points)**

Generate predictions for when atmospheric CO2 is expected to be at 420 ppm and 500 ppm levels for the first and final times (consider prediction intervals as well as point estimates in your answer). Generate a prediction for atmospheric C02 levels in the year 2100. How confident are you that these are accurate predictions?

We will forecast our model from Part 5, by generating a 5000 steps ahead forecast. We will use the 95% confidence interval in discussing first and last times. Note that the 80% confidence interval will have times that always fall between the point estimate and 95% CI. Based on the forecast above, the lower 80% CI never reaches 500, the same behavior at the 95% CI (see below).

```r
#Get ci from the pointless "hilo" R class
get_ci <- function(.distribution) {
    hilo.crap <- hilo(.distribution)
    lower <- hilo.crap$.lower
    upper <- hilo.crap$.upper
    return (c(lower, upper) %>% matrix(ncol = 2, byrow = FALSE))
}

#Perform 5000 steps forecast and search for range of value of our interest
fc <- mod.nonseason.adj.best %>% forecast(h=5000)
autoplot(fc) + ggtitle("5000 Steps ARIMA forecast") +
  ylab("CO2 in ppm") + xlab("Year")
```



```
fc.df = data.frame(fc)

names <- colnames(fc.df)
names <- append(names, c('95CI.lower', '95CI.upper'))

#Get the CIs into the dataframe
fc.df <- cbind(fc.df, get_ci(fc.df$.distribution))
colnames(fc.df) <- names
fc.df <- fc.df %>% dplyr::select(Date, ppm.filled, '95CI.lower', '95CI.upper')
#head(fc.df[fc.df$ppm.fill >= 420 & fc.df$ppm.fill < 500,][,c("Date", "ppm.filled")],5)
#tail(fc.df[fc.df$ppm.fill >= 420 & fc.df$ppm.fill < 500,][,c("Date", "ppm.filled")],5)
```

Since the periods are cyclic, there will be multiple points where 420/500 is crossed by the CIs and the point estimate. We will write some code to extract all such dates.

```
#Returns all indices for which the threshold value is crossed over by a particular field in th
get_indices_crossover <- function(field, threshold){
  thresholds <- c()
  current_idx <- 0
  prev_below <- TRUE

  for (row in 1:nrow(fc.df)) {
    if (fc.df[row, field] < threshold){
      current_idx <- row
      prev_below <- TRUE }

    else if(fc.df[row, field] > threshold) {
      if (prev_below){
        thresholds <- append(thresholds, current_idx)
      }
      prev_below <- FALSE
    }

  }
  return(thresholds)

}

ppm.filled.420 <- get_indices_crossover('ppm.filled', 420)
lower.ci.420 <- get_indices_crossover('95CI.lower', 420)
upper.ci.420 <- get_indices_crossover('95CI.upper', 420)
ppm.filled.500 <- get_indices_crossover('ppm.filled', 500)
lower.ci.500 <- get_indices_crossover('95CI.lower', 500)
upper.ci.500 <- get_indices_crossover('95CI.upper', 500)


#Demostrate that we've pulled out values for which crossover occurs
for (threshold in ppm.filled.420) {
```

```r
  print(fc.df[threshold:(threshold+1), ])
}
```

```
##           Date ppm.filled 95CI.lower 95CI.upper
## 213 2022-03-20   419.6319   415.9868   423.2771
## 214 2022-03-27   420.2128   416.5550   423.8707
##           Date ppm.filled 95CI.lower 95CI.upper
## 255 2023-01-08   419.9566   415.8131   424.1002
## 256 2023-01-15   420.3355   416.1807   424.4902
##           Date ppm.filled 95CI.lower 95CI.upper
## 298 2023-11-05   419.6599   414.9702   424.3495
## 299 2023-11-12   420.3253   415.6237   425.0270
##           Date ppm.filled 95CI.lower 95CI.upper
## 343 2024-09-15   419.8745   414.5935   425.1554
## 344 2024-09-22   420.0514   414.7575   425.3452
```

The first time that the point estimate, lower and upper CIs crosses 420 is below:

```r
fc.df[upper.ci.420[1]:(upper.ci.420[1]+1), ]
```

```
##           Date ppm.filled 95CI.lower 95CI.upper
## 159 2021-03-07   416.4713   413.4639   419.4786
## 160 2021-03-14   416.9850   413.9651   420.0049
```

```r
fc.df[ppm.filled.420[1]:(ppm.filled.420[1]+1), ]
```

```
##           Date ppm.filled 95CI.lower 95CI.upper
## 213 2022-03-20   419.6319   415.9868   423.2771
## 214 2022-03-27   420.2128   416.5550   423.8707
```

```r
fc.df[lower.ci.420[1]:(lower.ci.420[1]+1), ]
```

```
##           Date ppm.filled 95CI.lower 95CI.upper
## 270 2023-04-23   423.8680   419.5269   428.2091
## 271 2023-04-30   424.3592   420.0052   428.7132
```

Based on our forecast, the upper 95% CI first reaches 420ppm between March 7th, 2021, and March 14th, 2021. Our point estimate first reaches 420ppm between March 20th, 2022 and March 27th, 2020, and the lower 95% CI first reaches 420ppm between April 23rd, 2023 and April 30th, 2023. These estimates spans more than two years of time.

The last time that the point estimate, lower and upper CIs crosses 420 is below:

```r
fc.df[tail(upper.ci.420, 1):(tail(upper.ci.420, 1) + 1), ]
```

```
##           Date ppm.filled 95CI.lower 95CI.upper
## 243 2022-10-16   415.9680   411.9605   419.9755
## 244 2022-10-23   416.2782   412.2592   420.2972
```

```r
fc.df[tail(ppm.filled.420, 1):(tail(ppm.filled.420, 1) + 1), ]
```

```
##           Date ppm.filled 95CI.lower 95CI.upper
## 343 2024-09-15   419.8745   414.5935   425.1554
```

```
## 344 2024-09-22   420.0514   414.7575   425.3452
```

```r
fc.df[tail(lower.ci.420, 1):(tail(lower.ci.420, 1)+1), ]
```

```
##            Date ppm.filled 95CI.lower 95CI.upper
## 4974 2113-06-18   709.4702   419.7512   999.1892
## 4975 2113-06-25   710.0201   420.1957   999.8446
```

Based on our forecast, the upper 95% CI last reaches 420ppm between Oct. 16th, 2022, and Oct. 23rd, 2022. Our point estimate last reaches 420ppm between Sept. 15th, 2024 and Sept. 22th, 2024, and the lower 95% CI first reaches 420ppm between Oct. 18th, 2026 and Oct. 25th, 2026. These estimates spans almost four years of time. Put together with when the forecast first reaches 420ppm, our estimates suggest that the ppm levels will cycle and reach 420ppm around 4 times (based on the point estimate) between March 2021 and Oct 2026.

We repeat the exercise for 500ppm. First, we note that the lower 95CI never reaches 500ppm in our model. As demonstrated by the forecast plot above, the lower CI reaches a peak at around 460ppm, and starts decreasing again, due to the fact that the confidence intervals increases as the length of forecast increases.

The first time that the point estimate and upper CIs crosses 500ppm is below:

```r
fc.df[upper.ci.500[1]:(upper.ci.500[1]+1), ]
```

```
##            Date ppm.filled 95CI.lower 95CI.upper
## 1253 2042-02-23   474.6746   449.3666   499.9826
## 1254 2042-03-02   475.5082   450.1674   500.8490
```

```r
fc.df[ppm.filled.500[1]:(ppm.filled.500[1]+1), ]
```

```
##            Date ppm.filled 95CI.lower 95CI.upper
## 1624 2049-04-04   499.7648   461.1183   538.4113
## 1625 2049-04-11   500.1531   461.4659   538.8404
```

Based on our forecast, the upper 95% CI first reaches 500ppm between Feb 23rd, 2024, and March 2nd, 2024. Our point estimate first reaches 500ppm between April 4th, 2049 and April 11th, 2049. These estimates span over 25 years.

The last time that the point estimate and upper CIs crosses 500 is below:

```r
fc.df[tail(upper.ci.500, 1):(tail(upper.ci.500, 1) + 1), ]
```

```
##            Date ppm.filled 95CI.lower 95CI.upper
## 1288 2042-10-26   473.4388   447.0076   499.8701
## 1289 2042-11-02   473.7017   447.2390   500.1644
```

```r
fc.df[tail(ppm.filled.500, 1):(tail(ppm.filled.500, 1) + 1), ]
```

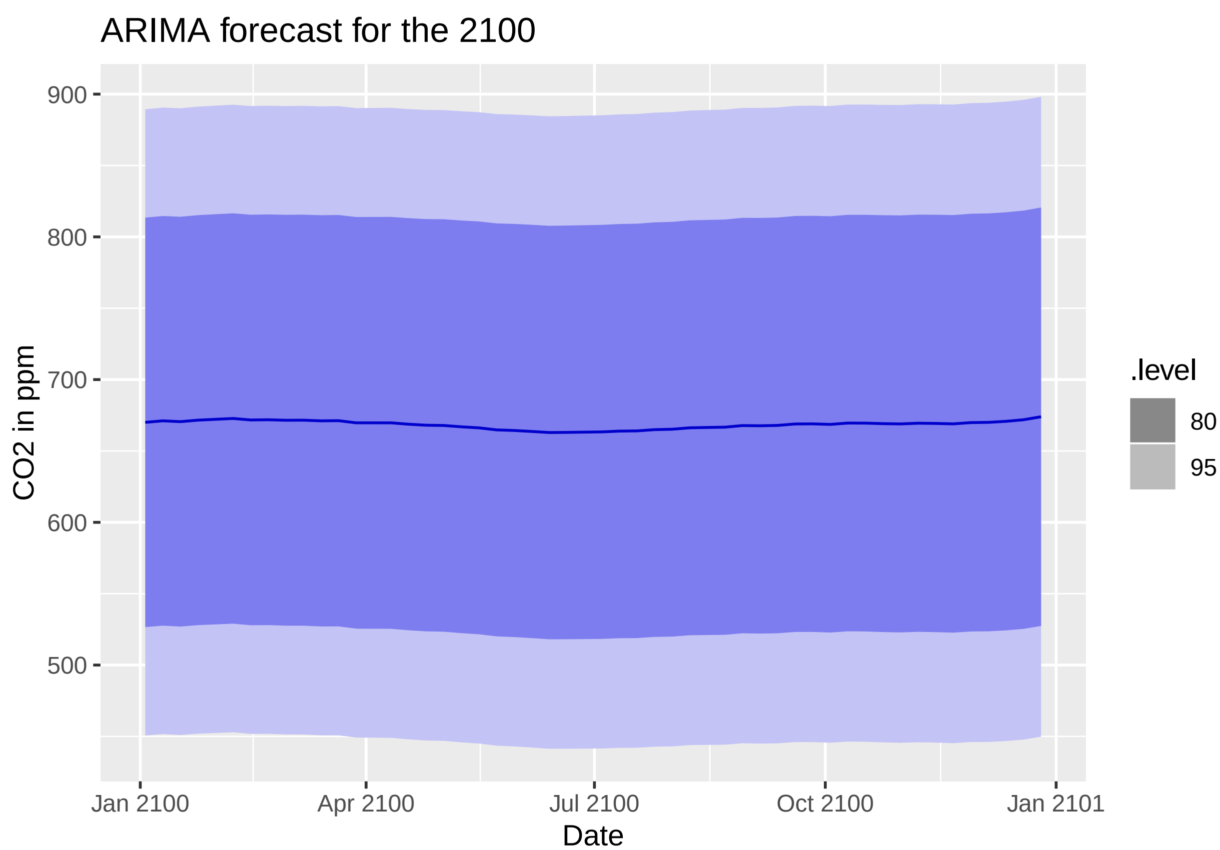
```
##            Date ppm.filled 95CI.lower 95CI.upper
## 1754 2051-10-01   499.9946   455.9473   544.0419
## 1755 2051-10-08   500.8621   456.7728   544.9514
```

Based on our forecast, the upper 95% CI last reaches 500ppm between Oct. 26th, 2042, and Nov. 2nd, 2042. Our point estimate last reaches 500ppm between Oct. 1st, 2051 and Oct. 8th, 2051.

These estimates spans almost 9 years of time. Put together with when the forecast first reaches 500ppm, our estimates suggest that the ppm levels will cycle and reach 500ppm around 4 times (based on the point estimate) between Feb 2024 and Oct 2051. The large range of time demonstrates the uncertainly in the estimates.

We will now look at the portion of the forecast in the year 2100.

```
# forecast value to year 2100
fc.2100 <- fc %>% filter(year(fc$Date) == 2100)
fc.df.2100 <- fc.df[year(fc.df$Date) == '2100',]
autoplot(fc.2100) + ggtitle("ARIMA forecast for the 2100") +
  ylab("CO2 in ppm") + xlab("Date")
```



First, in the middle of the year between June and July, we have the following values:

```
fc.df.2100[month(fc.df.2100$Date) == 7 | month(fc.df.2100$Date) == 6 , ]
```

```
##            Date ppm.filled 95CI.lower 95CI.upper
## 4294 2100-06-06   663.6753   442.2318   885.1187
## 4295 2100-06-13   662.9141   441.3766   884.4515
## 4296 2100-06-20   663.0224   441.3909   884.6539
## 4297 2100-06-27   663.1937   441.4682   884.9191
## 4298 2100-07-04   663.3716   441.5522   885.1910
## 4299 2100-07-11   663.9142   442.0009   885.8274
## 4300 2100-07-18   664.0732   442.0662   886.0803
## 4301 2100-07-25   664.9423   442.8414   887.0432
```

The point estimate between June and July is around 663, and lower CI is around 441, and upper CI 885. For Jun 27, 2100, we estimate with 95% confidence the CO2 levels to be 441.5 to 884.9. This is a very large range for the confidence interval.

We now calculate the mean estimate of this year:

```
#calculate mean value of CO2 level in year 2100
mean(fc.df.2100$ppm.filled)
```

```
## [1] 668.4283
```

Mean of forecast value in year 2100 is around 668 ppm. From forecast plot, 95% confidence interval is wide for all estimates, ranging from ~440 ppm to ~900 ppm. The model was constructed with 44 years data, and so performing forecast that is 80 years ahead is difficult. Due to the large confidence interval, we are not confident on these forecast values.

*This work was done as part of the W271 - Statistical Methods for Discrete Response, Time Series, and Panel Data course under the U.C. Berkeley Master of Information and Data Science program.