Programming with Data

Session 5: Regression Forecasts with Seasonality

Dr. Wang Jiwei
Master of Professional Accounting

Application: Quarterly retail revenue

The question

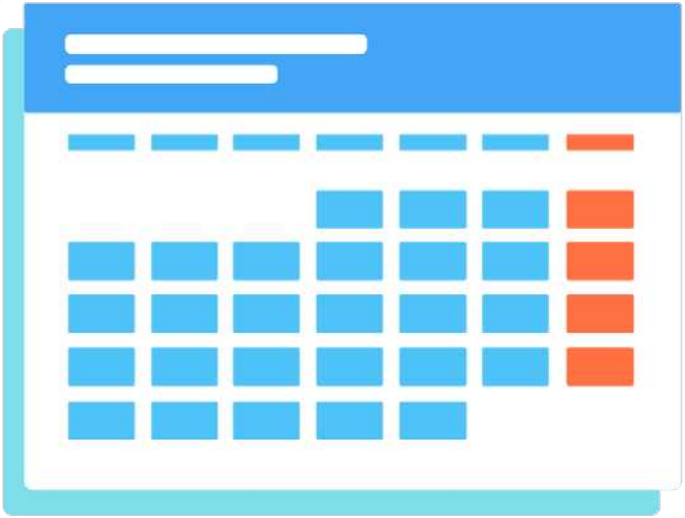
> How can we predict quarterly revenue for retail companies, leveraging our knowledge of such companies

- In aggregate
- By Store
- By department
- Consider time dimensions
    - What matters:
        - Last quarter?
        - Last year?
        - Other timeframes?
    - Cyclicality/Seasonality

Time matters a lot for retail

How to capture time effects?

- Autoregression
  - Regress y_t on earlier value(s) of itself
    - Last quarter, last year, etc.
- Controlling for time directly in the model
  - Essentially the same as fixed effects last week



Quarterly revenue prediction

The data

- From quarterly reports of US retail companies
- Two sets of firms:
  - US "Hypermarkets & Super Centers" `GICS`(gsubind): 30101040]
  - US "Multiline Retail" `GICS`(gind): 255030]
- Data from Compustat - Capital IQ > North America - Daily > Fundamentals Quarterly
  - datadate: all available (1962 to 2020 for this case)

Formalization

1. Question
   - How can we predict quarterly revenue for large retail companies?
2. Hypothesis (just the alternative ones)
   1. Current quarter revenue helps predict next quarter revenue
   2. 3 quarters ago revenue helps predict next quarter revenue (Year-over-year)
   3. Different quarters exhibit different patterns (seasonality)
   4. A long-run autoregressive model helps predict next quarter revenue
3. Research design
   - Use OLS for all the above -- t-tests for coefficients
   - Hold out sample (testing data): 2016-2020

Variable generation

```r
library(tidyverse)  # As always
library(plotly)  # interactive graphs
library(lubridate)  # import some sensible date functions

# Generate quarter over quarter growth "revtq_gr"
df <- df %>% group_by(gvkey) %>%
  mutate(revtq_gr = revtq / lag(revtq) - 1) %>% ungroup()

# Generate year-over-year growth "revtq_yoy"
df <- df %>% group_by(gvkey) %>%
  mutate(revtq_yoy = revtq / lag(revtq, 4) - 1) %>% ungroup()

# Generate first difference "revtq_d"
df <- df %>% group_by(gvkey) %>%
  mutate(revtq_d = revtq - lag(revtq)) %>% ungroup()

# Generate a proper date in R
# datadate (end of reporting period) is YYMMDDn8. (int 20200630)
# quarter() is to generate the calendar quarter based on date
# which may be different from company's fiscal quarter
df$date <- ymd(df$datadate)  # From lubridate
df$cqtr <- quarter(df$date)   # From lubridate
```

Date manipulation in R

- `ymd()` from `package:lubridate` is a handy way of converting date.
  - It also has `ydm()`, `mdy()`, `myd()`, `dmy()` and `dym()`
  - It can handle quarters, times, and date-times as well
  - Cheat sheet
  - It will convert the date format to the ISO 8601 international standard which expresses a day as "2001-02-03".
- `as.Date()` from the Base R can take a date formatted as "YYYY/MM/DD" and convert to a proper date value
  - You can convert other date types using the `format =` argument
    - e.g., "DD.MM.YYYY" is format code "%d.%m.%Y"
    - Full list of date codes
  - The default date format also follows ISO 8601.
  - The following code can do the same as `ymd()`

```r
# Generate a proper date in R
# Datadate is YYMMDDn8. (integer 20200630)
df$date <- as.Date(as.character(df$datadate), format = "%Y%m%d")
```

Example output

- The following shows some selective columns

conm	date	revtq	revtq_gr	revtq_yoy	revtq_d
ALLIED STORES	1962-04-30	156.5	NA	NA	NA
ALLIED STORES	1962-07-31	161.9	0.0345048	NA	5.4
ALLIED STORES	1962-10-31	176.9	0.0926498	NA	15.0
ALLIED STORES	1963-01-31	275.5	0.5573770	NA	98.6
ALLIED STORES	1963-04-30	171.1	-0.3789474	0.0932907	-104.4
ALLIED STORES	1963-07-31	182.2	0.0648743	0.1253860	11.1

```
## # A tibble: 6 x 5
##   conm          date       datadate  fqtr  cqtr
##   <chr>         <date>        <int> <int> <int>
## 1 ALLIED STORES 1962-04-30 19620430     1     2
## 2 ALLIED STORES 1962-07-31 19620731     2     3
## 3 ALLIED STORES 1962-10-31 19621031     3     4
## 4 ALLIED STORES 1963-01-31 19630131     4     1
## 5 ALLIED STORES 1963-04-30 19630430     1     2
## 6 ALLIED STORES 1963-07-31 19630731     2     3
```

Create 8 quarters (2 years) of lags

```r
# Brute force code for variable generation of quarterly data lags
df <- df %>%
  group_by(gvkey) %>%
  mutate(revtq_l1 = lag(revtq), revtq_l2 = lag(revtq, 2),
         revtq_l3 = lag(revtq, 3), revtq_l4 = lag(revtq, 4),
         revtq_l5 = lag(revtq, 5), revtq_l6 = lag(revtq, 6),
         revtq_l7 = lag(revtq, 7), revtq_l8 = lag(revtq, 8),
         revtq_gr1 = lag(revtq_gr), revtq_gr2 = lag(revtq_gr, 2),
         revtq_gr3 = lag(revtq_gr, 3), revtq_gr4 = lag(revtq_gr, 4),
         revtq_gr5 = lag(revtq_gr, 5), revtq_gr6 = lag(revtq_gr, 6),
         revtq_gr7 = lag(revtq_gr, 7), revtq_gr8 = lag(revtq_gr, 8),
         revtq_yoy1 = lag(revtq_yoy), revtq_yoy2 = lag(revtq_yoy, 2),
         revtq_yoy3 = lag(revtq_yoy, 3), revtq_yoy4 = lag(revtq_yoy, 4),
         revtq_yoy5 = lag(revtq_yoy, 5), revtq_yoy6 = lag(revtq_yoy, 6),
         revtq_yoy7 = lag(revtq_yoy, 7), revtq_yoy8 = lag(revtq_yoy, 8),
         revtq_d1 = lag(revtq_d), revtq_d2 = lag(revtq_d, 2),
         revtq_d3 = lag(revtq_d, 3), revtq_d4 = lag(revtq_d, 4),
         revtq_d5 = lag(revtq_d, 5), revtq_d6 = lag(revtq_d, 6),
         revtq_d7 = lag(revtq_d, 7), revtq_d8 = lag(revtq_d, 8)) %>%
  ungroup()
```

Create 8 quarters (2 years) of lags

```
# Custom function to generate a series of lags
library(rlang)
multi_lag <- function(df, lags, var, postfix="") {
  var <- enquo(var)
  quosures <- map(lags, ~quo(lag(!!var, !!.x))) %>%
    set_names(paste0(quo_text(var), postfix, lags))
  return(ungroup(mutate(group_by(df, gvkey), !!!quosures)))
}
# Generate lags "revtq_l#"
df <- multi_lag(df, 1:8, revtq, "_l")

# Generate changes "revtq_gr#"
df <- multi_lag(df, 1:8, revtq_gr)

# Generate year-over-year changes "revtq_yoy#"
df <- multi_lag(df, 1:8, revtq_yoy)

# Generate first differences "revtq_d#"
df <- multi_lag(df, 1:8, revtq_d)
```

- require more advanced understanding of `metaprogramming`, `advanced R`, `tidy evaluation`, and `quosure` concepts.
- `paste0()`: creates a string vector by concatenating all inputs
- `paste()`: same as `paste0()`, but with spaces added in between

Example output

conm	date	revtq	revtq_l1	revtq_gr1	revtq_yoy1	revtq_d1
ALLIED STORES	1962-04-30	156.5	NA	NA	NA	NA
ALLIED STORES	1962-07-31	161.9	156.5	NA	NA	NA
ALLIED STORES	1962-10-31	176.9	161.9	0.0345048	NA	5.4
ALLIED STORES	1963-01-31	275.5	176.9	0.0926498	NA	15.0
ALLIED STORES	1963-04-30	171.1	275.5	0.5573770	NA	98.6
ALLIED STORES	1963-07-31	182.2	171.1	-0.3789474	0.0932907	-104.4

Clean and holdout sample

```r
# Clean the data: Replace NaN, Inf, and -Inf with NA
df <- df %>%
  mutate_if(is.numeric, list(~replace(., !is.finite(.), NA)))

# Split into training and test datasets
# Training dataset: We'll use data released before 2016
train <- filter(df, year(date) < 2016)

# Test dataset: We'll use data released 2016 through 2020 (till 3Q2020)
test <- filter(df, year(date) >= 2016)
```

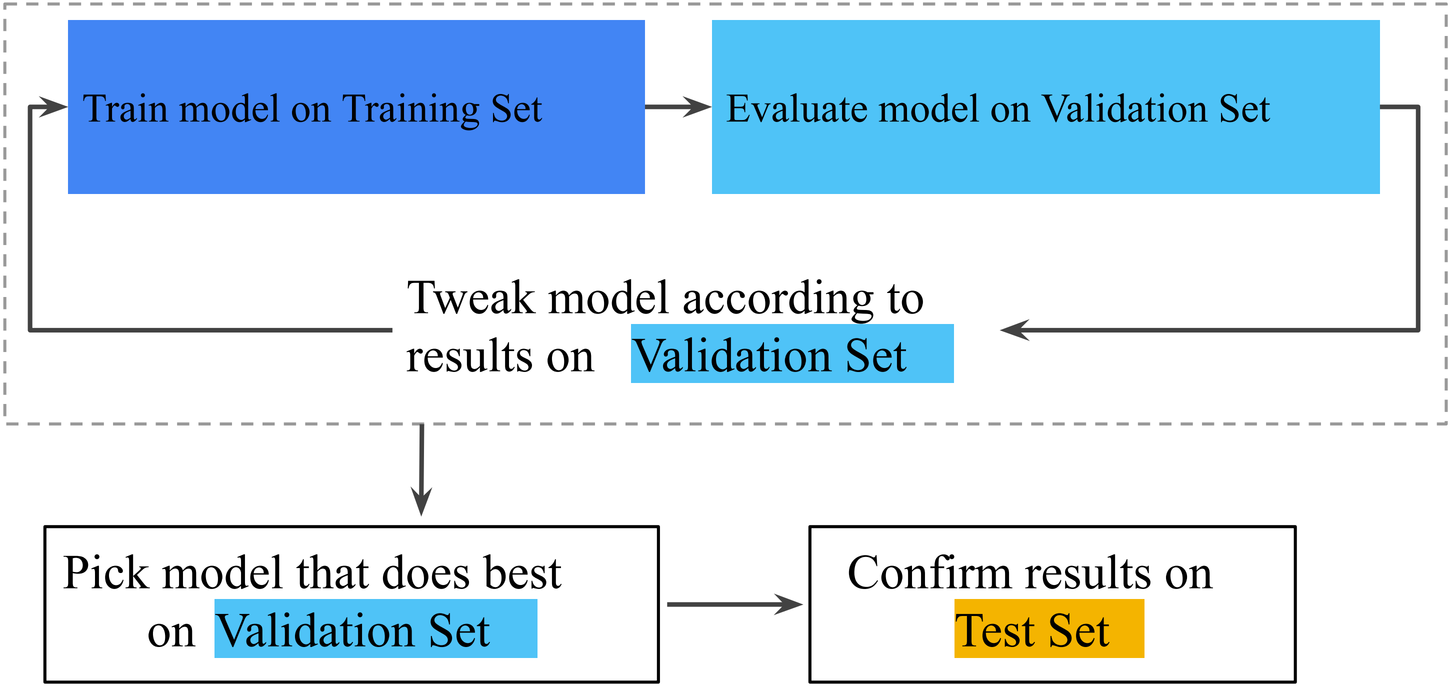
- Same cleaning function as last week:
  - Replaces all `NaN`, `Inf`, and `-Inf` with `NA`
- `year()` comes from `package:lubridate`

Training vs. test datasets

> train a model and test/validate it using the same set of data?

- We build analytics models for forecasting and other predictive purposes
- The key question: **could the model be generalized to new dataset?**
  - We need to have a new dataset to test how well the model performs
  - Existing data will be divided into training data and test data
- Training data will be used to train/build the model
  - It can be further divided into training set and validation set.
  - The validation set can be used to further tune the model (eg, detect overfitting problem), which helps get the most optimized model.
  - We will cover (cross) validation in a future topic
- Testing data will be used to test how well the model performs
  - In general, 80/20 rule (Pareto principle) will be applied to split the dataset

Workflow with training/test sets



Univariate stats

Univariate stats

- To get a better grasp on the problem, looking at univariate stats can help
  - Summary stats (using `summary()`)
  - Correlations using `cor()`
  - Plots using your preferred package such as `package:ggplot2`

```
summary(df[ , c("revtq", "revtq_gr", "revtq_yoy", "revtq_d", "fqtr")])
```

```
##      revtq             revtq_gr          revtq_yoy            revtq_d
##  Min.   :     0.00   Min.   :-1.0000   Min.   :-1.0000   Min.   :-24325.206
##  1st Qu.:    66.01   1st Qu.:-0.1091   1st Qu.: 0.0024   1st Qu.:   -20.260
##  Median :   312.59   Median : 0.0501   Median : 0.0704   Median :     4.548
##  Mean   :  2545.48   Mean   : 0.0625   Mean   : 0.1185   Mean   :    23.730
##  3rd Qu.:  1386.50   3rd Qu.: 0.2032   3rd Qu.: 0.1476   3rd Qu.:    60.146
##  Max.   :141671.00   Max.   :14.3333   Max.   :47.6600   Max.   : 16117.000
##  NA's   :394         NA's   :731       NA's   :1020      NA's   :704
##       fqtr
##  Min.   :1.000
##  1st Qu.:1.000
##  Median :2.000
##  Mean   :2.479
##  3rd Qu.:3.000
##  Max.   :4.000
##
```

ggplot2 for visualization

- The following slides will use some custom functions using `package:ggplot2`
- `package:ggplot2` has an odd syntax:
  - It doesn't use pipes (`%>%`), but instead adds everything together (`+`)

```
library(ggplot2)  # or tidyverse -- it's part of tidyverse
df %>%
  ggplot(aes(y = var_for_y_axis, x = var_for_y_axis)) +
  geom_point()  # scatterplot
```

- `aes()` is for aesthetics -- how the chart is set up
- Other useful aesthetics:
  - `group` = to set groups to list in the legend. Not needed if using the below though
  - `color` = to set color by some grouping variable. Put `factor()` around the variable if you want discrete groups, otherwise it will do a color scale (light to dark)
  - `shape` = to set shapes for points -- see here for a list

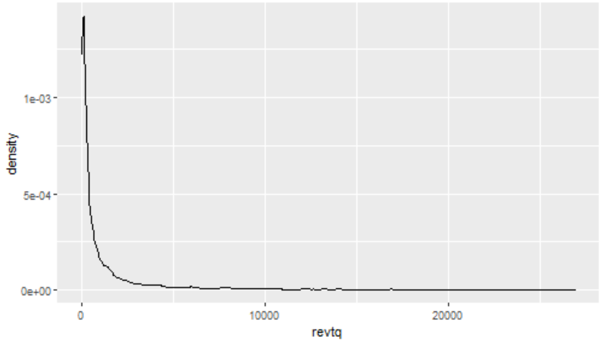
ggplot2 for visualization

```
library(ggplot2)  # or tidyverse -- it's part of tidyverse
df %>%
  ggplot(aes(y = var_for_y_axis, x = var_for_y_axis)) +
  geom_point()  # scatterplot
```

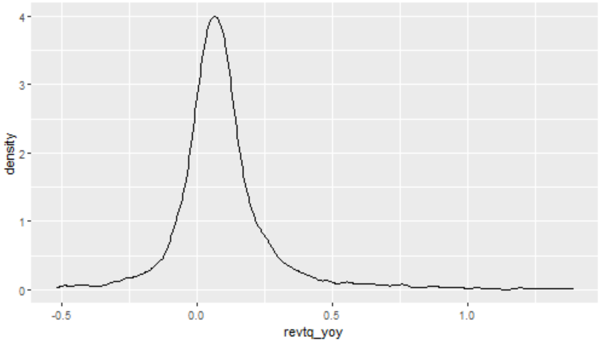
- `geom` stands for geometry -- any shapes, lines, etc. start with `geom`
- Other useful geoms:
  - `geom_line()`: makes a line chart
  - `geom_bar()`: makes a bar chart -- y is the height, x is the category
  - `geom_smooth(method = "lm")`: Adds a linear regression into the chart
  - `geom_abline(slope = 1)`: Adds a 45° line
- Add `xlab("Label text here")` to change the x-axis label
- Add `ylab("Label text here")` to change the y-axis label
- Add `ggtitle("Title text here")` to add a title
- Plenty more details in the 'Data Visualization Cheat Sheet'

Plotting: Distribution of revenue

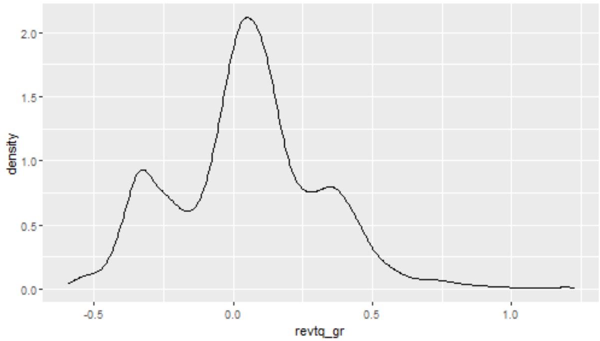
- (1) Revenue



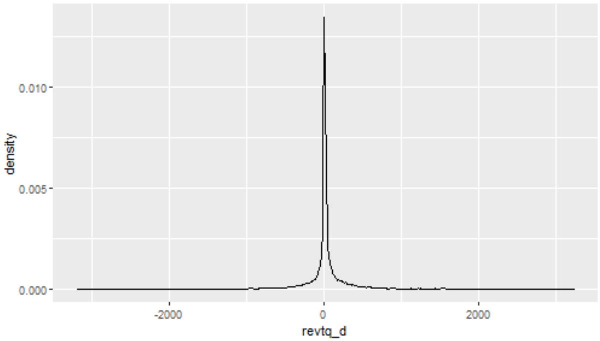
- (3) Year-over-year growth



- (2) Quarterly growth



- (4) First difference



What we learn from the graphs?

1. Revenue
   -
2. Quarterly growth
   -
3. Year-over-year growth
   -
4. First difference
   -

What we learn from the graphs?

1. Revenue

   - This is really skewed data -- a lot of small revenue quarters, but a significant amount of large revenue quarters in the tail
     - Potential fix: use `log(revtq)`?

2. Quarterly growth

   - Quarterly growth is reasonably close to normally distributed
     - Good for OLS

3. Year-over-year growth

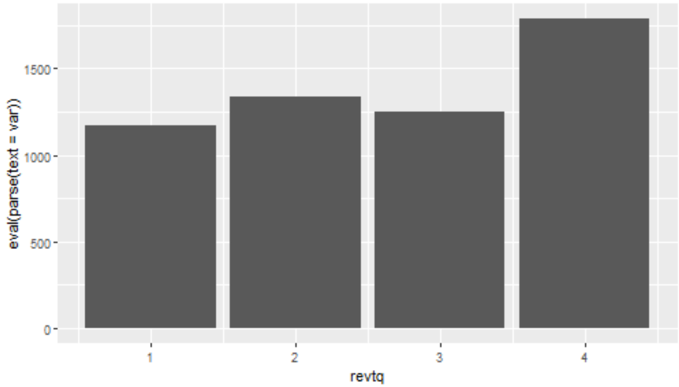
   - Year over year growth is reasonably close to normally distributed
     - Good for OLS

4. First difference

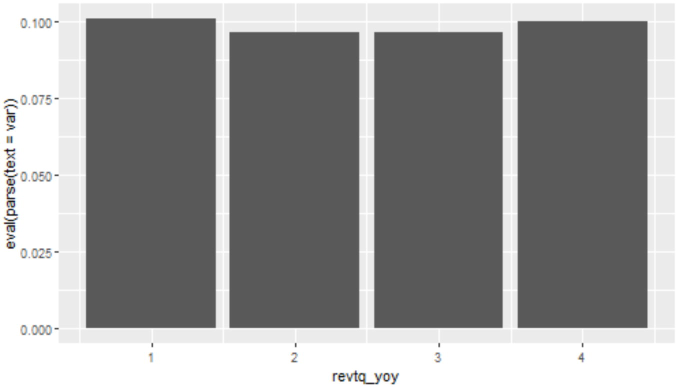
   - Reasonably close to normally distributed, with really long tails
     - Good enough for OLS

Plotting: Mean revenue by quarter

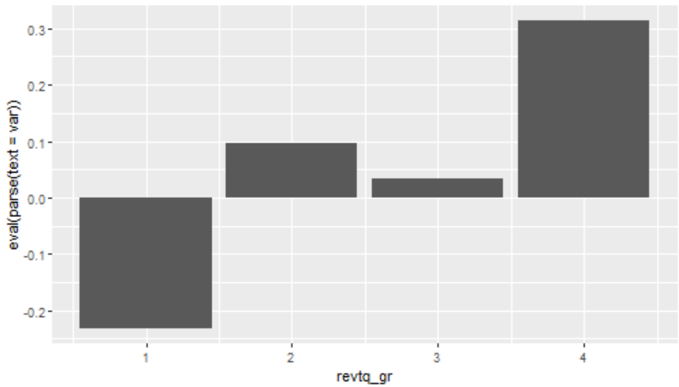
(1) Revenue



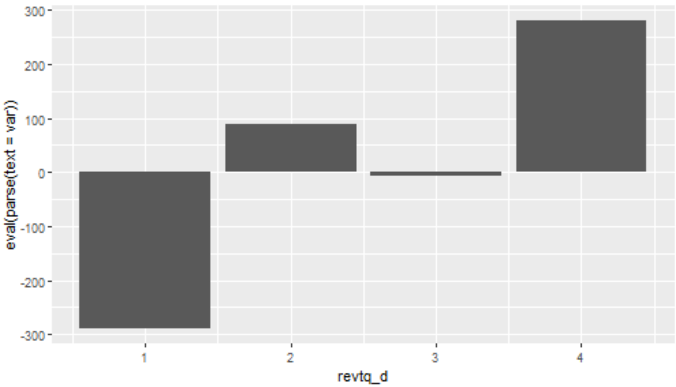
(3) Year-over-year growth



(2) Quarterly growth



(4) First difference



What we learn from the graphs?

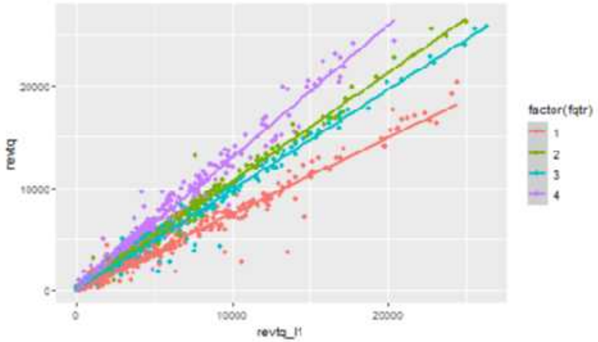
1. Revenue
    -
2. Quarterly growth
    -
3. Year-over-year growth
    -
4. First difference
    -

What we learn from the graphs?

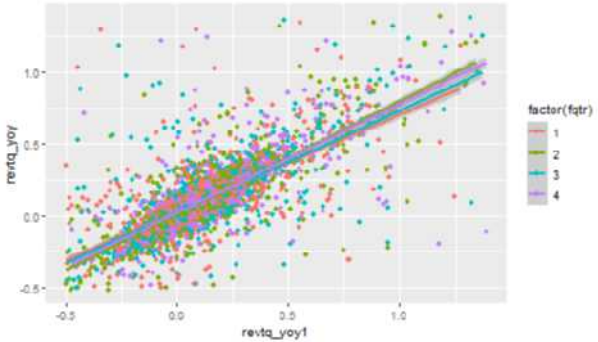
1. Revenue
   - Revenue seems cyclical!
2. Quarterly growth
   - Definitely cyclical!
3. Year-over-year growth
   - Year over year difference is less cyclical -- more constant
4. First difference
   - Definitely cyclical!

Plotting: Revenue vs lag by quarter

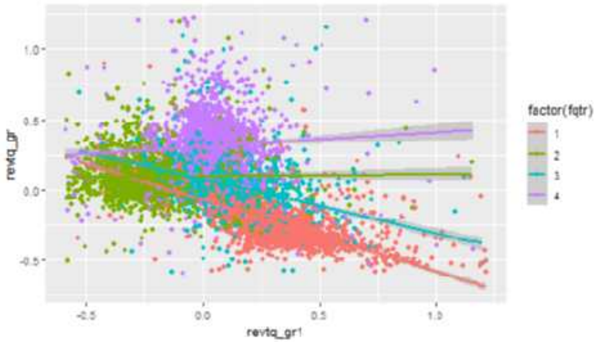
- (1) Revenue



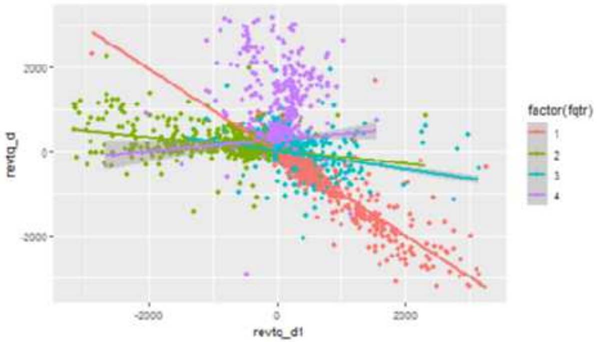
- (3) Year-over-year growth



- (2) Quarterly growth



- (4) First difference



What we learn from the graphs?

1. Revenue
   - Revenue is really linear! But each quarter has a distinct linear relation.
2. Quarterly growth
   - All over the place. Each quarter appears to have a different pattern though. Quarters will matter.
3. Year-over-year growth
   - Linear but noisy.
4. First difference
   - Again, all over the place. Each quarter appears to have a different pattern though. Quarters will matter.

Correlation matrices

```
cor(train[,c("revtq","revtq_l1","revtq_l2","revtq_l3","revtq_l4")],
    use = "complete.obs") # delete row if with NA
```

```
##              revtq  revtq_l1  revtq_l2  revtq_l3  revtq_l4
## revtq    1.0000000 0.9917996 0.9939751 0.9907381 0.9973540
## revtq_l1 0.9917996 1.0000000 0.9917016 0.9938476 0.9901821
## revtq_l2 0.9939751 0.9917016 1.0000000 0.9916042 0.9932811
## revtq_l3 0.9907381 0.9938476 0.9916042 1.0000000 0.9910049
## revtq_l4 0.9973540 0.9901821 0.9932811 0.9910049 1.0000000
```

```
cor(train[,c("revtq_gr","revtq_gr1","revtq_gr2","revtq_gr3","revtq_gr4")],
    use = "complete.obs")
```

```
##              revtq_gr   revtq_gr1   revtq_gr2   revtq_gr3   revtq_gr4
## revtq_gr   1.00000000 -0.33021570  0.06675942 -0.23736085  0.65335232
## revtq_gr1 -0.33021570  1.00000000 -0.32597810  0.06581984 -0.22955824
## revtq_gr2  0.06675942 -0.32597810  1.00000000 -0.33452265  0.07215056
## revtq_gr3 -0.23736085  0.06581984 -0.33452265  1.00000000 -0.32429873
## revtq_gr4  0.65335232 -0.22955824  0.07215056 -0.32429873  1.00000000
```

> Retail revenue has really high autocorrelation! Concern for multicolinearity. Revenue growth is less autocorrelated and oscillates.

Correlation matrices

```
cor(train[,c("revtq_yoy","revtq_yoy1","revtq_yoy2","revtq_yoy3","revtq_yoy4")],
    use="complete.obs")
```

```
##             revtq_yoy revtq_yoy1 revtq_yoy2 revtq_yoy3 revtq_yoy4
## revtq_yoy  1.0000000  0.6588642  0.4183968  0.4216933  0.1805950
## revtq_yoy1 0.6588642  1.0000000  0.5802585  0.3731204  0.3546604
## revtq_yoy2 0.4183968  0.5802585  1.0000000  0.5921796  0.3738081
## revtq_yoy3 0.4216933  0.3731204  0.5921796  1.0000000  0.5710053
## revtq_yoy4 0.1805950  0.3546604  0.3738081  0.5710053  1.0000000
```

```
cor(train[,c("revtq_d","revtq_d1","revtq_d2","revtq_d3","revtq_d4")],
    use="complete.obs")
```

```
##               revtq_d   revtq_d1   revtq_d2   revtq_d3   revtq_d4
## revtq_d   1.0000000 -0.6203336  0.3300007 -0.6075689  0.9165429
## revtq_d1 -0.6203336  1.0000000 -0.6171063  0.3311438 -0.5872559
## revtq_d2  0.3300007 -0.6171063  1.0000000 -0.6209104  0.3152248
## revtq_d3 -0.6075689  0.3311438 -0.6209104  1.0000000 -0.5908631
## revtq_d4  0.9165429 -0.5872559  0.3152248 -0.5908631  1.0000000
```

> Year over year change fixes the multicollinearity issue. First difference oscillates like quarter over quarter growth.

R Practice

- This practice will look at predicting Walmart's quarterly revenue using:
  - 1 lag
  - Cyclicality
- Practice using:
  - `mutate()`
  - `lm()`
  - `package:ggplot2`
- Do the exercises in today's practice file
  - R Practice

Forecasting

1 period models

- 1 Quarter lag
  - We saw a very strong linear pattern here earlier

```
mod1 <- lm(revtq ~ revtq_l1, data = train)
```

- Quarter and year lag
  - Year-over-year seemed pretty constant

```
mod2 <- lm(revtq ~ revtq_l1 + revtq_l4, data = train)
```

- 2 years of lags
  - Other lags could also help us predict

```
mod3 <- lm(revtq ~ revtq_l1 + revtq_l2 + revtq_l3 + revtq_l4 +
             revtq_l5 + revtq_l6 + revtq_l7 + revtq_l8, data = train)
```

- 2 years of lags, by observation quarter
  - Take into account cyclicality observed in bar charts

```
mod4 <- lm(revtq ~ (revtq_l1 + revtq_l2 + revtq_l3 + revtq_l4 +
             revtq_l5 + revtq_l6 + revtq_l7 + revtq_l8):factor(fqtr),
           data = train)
```

Quarter lag

```
summary(mod1)
```

```
##
## Call:
## lm(formula = revtq ~ revtq_l1, data = train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -24399.7    -35.8    -13.0     36.3  15314.7
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept) 17.299837  12.991379   1.332    0.183
## revtq_l1     1.001776   0.001474 679.753   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1151 on 8294 degrees of freedom
##   (702 observations deleted due to missingness)
## Multiple R-squared:  0.9824,    Adjusted R-squared:  0.9824
## F-statistic: 4.621e+05 on 1 and 8294 DF,  p-value: < 2.2e-16
```

Quarter and year lag

```
summary(mod2)
```

```
##
## Call:
## lm(formula = revtq ~ revtq_l1 + revtq_l4, data = train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -20224.4    -21.6     -7.4     17.8   9320.8
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 8.740416   6.900972   1.267    0.205
## revtq_l1    0.225726   0.005434  41.540   <2e-16 ***
## revtq_l4    0.816635   0.005650 144.532   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 594.5 on 7855 degrees of freedom
##   (1140 observations deleted due to missingness)
## Multiple R-squared:  0.9955,    Adjusted R-squared:  0.9955
## F-statistic: 8.753e+05 on 2 and 7855 DF,  p-value: < 2.2e-16
```

2 years of lags

```
summary(mod3)
```

```
##
## Call:
## lm(formula = revtq ~ revtq_l1 + revtq_l2 + revtq_l3 + revtq_l4 +
##     revtq_l5 + revtq_l6 + revtq_l7 + revtq_l8, data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4854.9   -14.8    -5.7     8.0  5868.9
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  6.173259   4.176286   1.478   0.1394
## revtq_l1     0.785242   0.011881  66.095  < 2e-16 ***
## revtq_l2     0.106283   0.015152   7.015 2.52e-12 ***
## revtq_l3    -0.026460   0.014771  -1.791   0.0733 .
## revtq_l4     0.931266   0.011653  79.915  < 2e-16 ***
## revtq_l5    -0.779892   0.012756 -61.141  < 2e-16 ***
## revtq_l6    -0.079794   0.015819  -5.044 4.67e-07 ***
## revtq_l7     0.006604   0.015313   0.431   0.6663
## revtq_l8     0.065782   0.011621   5.660 1.57e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 343.8 on 7196 degrees of freedom
##   (1793 observations deleted due to missingness)
## Multiple R-squared:  0.9986,    Adjusted R-squared:  0.9986
## F-statistic: 6.536e+05 on 8 and 7196 DF,  p-value: < 2.2e-16
```

2 years of lags, by observation quarter

```
summary(mod4)
```

```
##
## Call:
## lm(formula = revtq ~ (revtq_l1 + revtq_l2 + revtq_l3 + revtq_l4 +
##     revtq_l5 + revtq_l6 + revtq_l7 + revtq_l8):factor(fqtr),
##     data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6141.4   -14.6     0.3    15.7  4980.3
##
## Coefficients:
##                         Estimate Std. Error t value Pr(>|t|)
## (Intercept)             -0.42798    3.89557  -0.110 0.912521
## revtq_l1:factor(fqtr)1   0.50358    0.02104  23.934  < 2e-16 ***
## revtq_l1:factor(fqtr)2   1.11831    0.02231  50.121  < 2e-16 ***
## revtq_l1:factor(fqtr)3   0.81435    0.02848  28.591  < 2e-16 ***
## revtq_l1:factor(fqtr)4   0.89057    0.02585  34.456  < 2e-16 ***
## revtq_l2:factor(fqtr)1   0.25042    0.03399   7.367 1.94e-13 ***
## revtq_l2:factor(fqtr)2  -0.09685    0.02387  -4.057 5.02e-05 ***
## revtq_l2:factor(fqtr)3   0.21067    0.03883   5.425 5.97e-08 ***
## revtq_l2:factor(fqtr)4   0.27270    0.03498   7.797 7.25e-15 ***
## revtq_l3:factor(fqtr)1   0.07270    0.03563   2.040 0.041349 *
## revtq_l3:factor(fqtr)2  -0.01645    0.03468  -0.474 0.635234
## revtq_l3:factor(fqtr)3  -0.02509    0.02361  -1.063 0.287895
## revtq_l3:factor(fqtr)4  -0.20644    0.03805  -5.426 5.96e-08 ***
```

Testing out of sample

- RMSE: Root Mean Square Error
- RMSE is very affected by outliers, and a bad choice for noisy data that you are OK with missing a few outliers here and there
  - Doubling error *quadruples* that part of the error

```
rmse <- function(v1, v2) {
  sqrt(mean((v1 - v2)^2, na.rm = TRUE))
}
```

- MAE: Mean Absolute Error
- MAE is measures average accuracy with no weighting
  - Doubling error *doubles* that part of the error

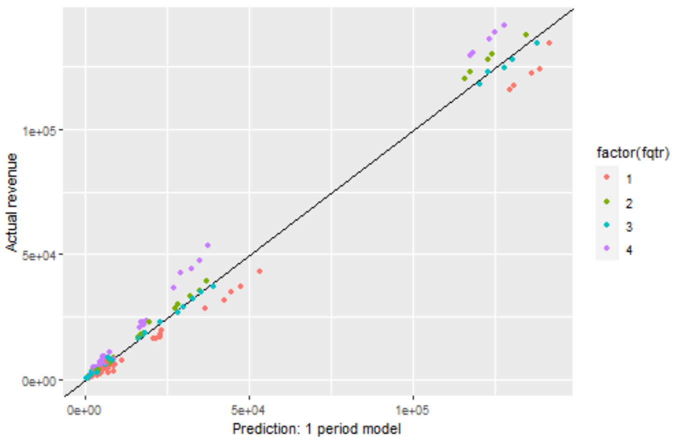
```
mae <- function(v1, v2) {
  mean(abs(v1-v2), na.rm = TRUE)
}
```

> Both are commonly used for evaluating OLS out of sample

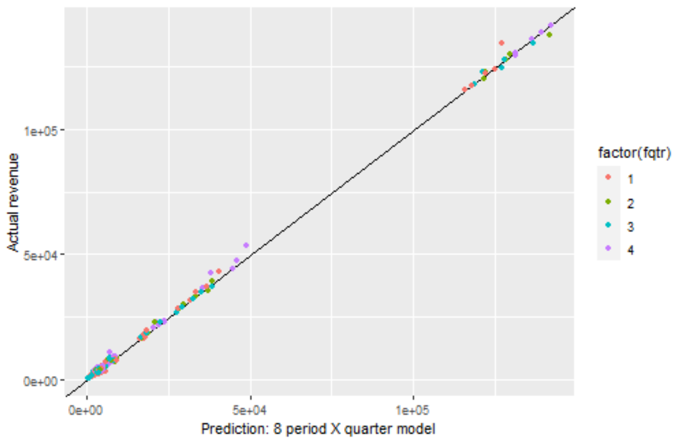
Testing out of sample

	adj_r_sq	rmse_in	mae_in	rmse_out	mae_out
1 period	0.9823645	1151.0560	323.82144	2916.3430	1223.4301
1 and 4 periods	0.9955321	594.4151	157.48397	1143.8276	553.5204
8 periods	0.9986241	343.5646	94.98273	764.7114	362.1292
8 periods w/ quarters	0.9989338	301.9370	92.26997	757.4591	354.6585

1 quarter model



8 period model, by quarter

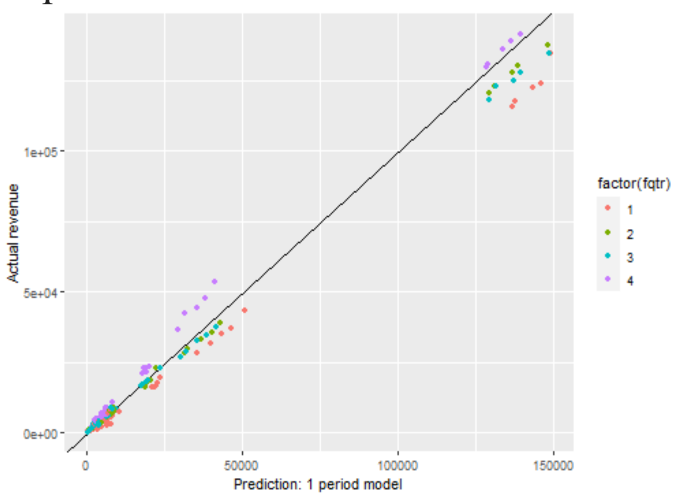


What about for revenue growth?

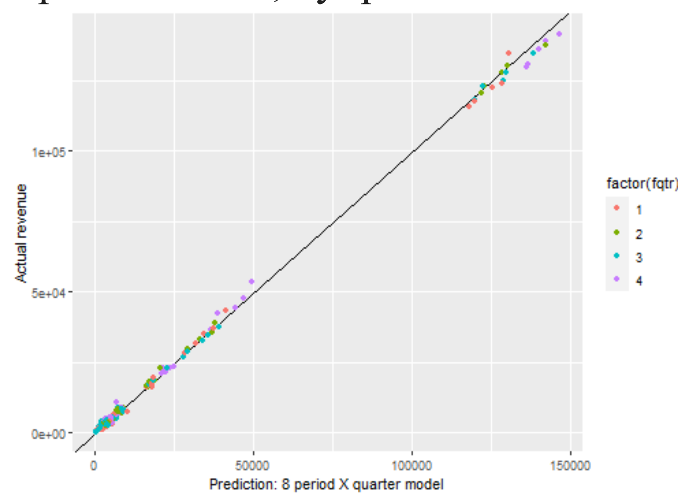
Backing out a revenue prediction, $revt_t = (1 + growth_t) \times revt_{t-1}$

	adj_r_sq	rmse_in	mae_in	rmse_out	mae_out
1 period	0.0955220	1110.5010	307.8361	3202.2234	1338.9696
1 and 4 periods	0.4497703	530.0174	152.8021	1355.5009	631.5524
8 periods	0.6788386	463.3719	123.3965	1165.7280	530.6755
8 periods w/ quarters	0.7720057	381.7661	99.5676	986.1408	452.1947

1 quarter model



8 period model, by quarter

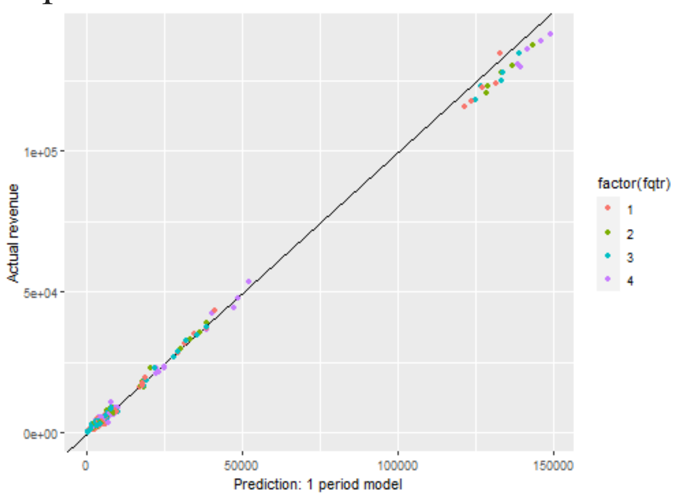


What about for YoY growth?

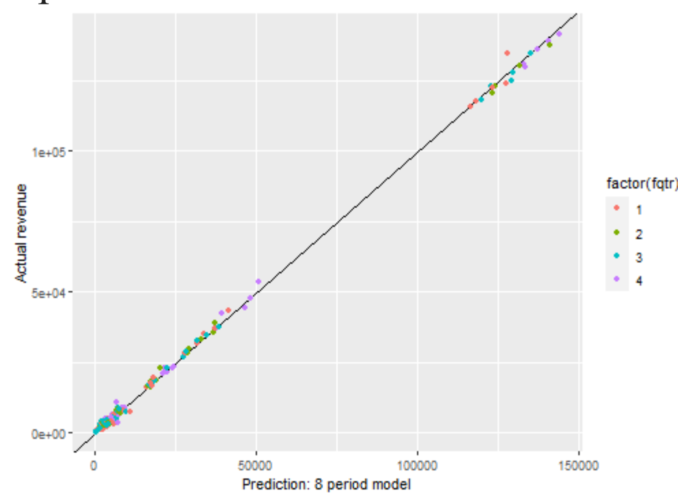
Backing out a revenue prediction, $revt_t = (1 + yoy\_growth_t) \times revt_{t-4}$

	adj_r_sq	rmse_in	mae_in	rmse_out	mae_out
1 period	0.4376253	520.7532	129.1364	1570.5401	695.8093
1 and 4 periods	0.5378241	495.5506	127.3290	1400.2662	642.0383
8 periods	0.5430590	383.6760	101.1748	863.9954	425.6484
8 periods w/ quarters	0.1462837	705.8313	193.7847	1214.8656	620.3688

1 quarter model



8 period model

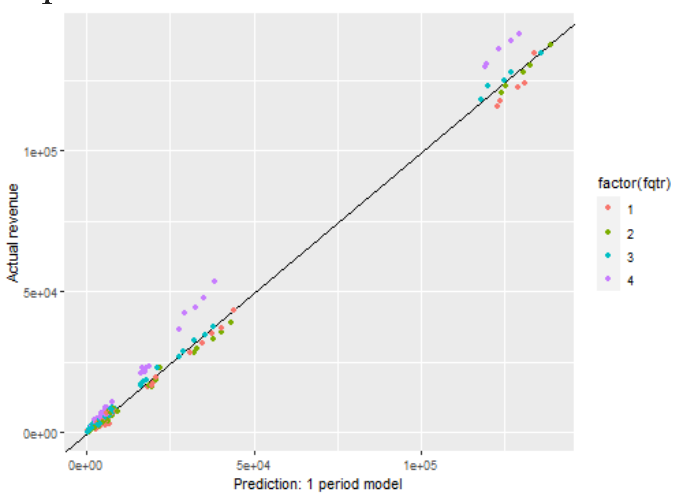


What about for first difference?

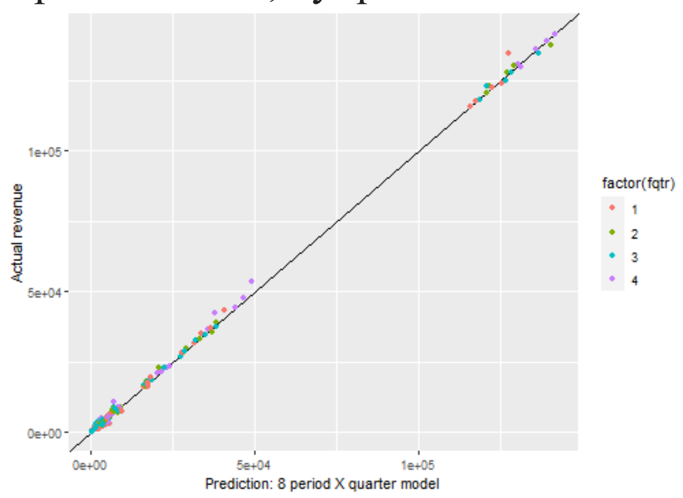
Backing out a revenue prediction, $revt_t = change_t + revt_{t-1}$

	adj_r_sq	rmse_in	mae_in	rmse_out	mae_out
1 period	0.3578089	896.1441	286.47866	2247.2158	986.9519
1 and 4 periods	0.8502591	444.9570	113.00284	860.6968	411.8824
8 periods	0.9242547	329.4611	95.17826	764.8854	348.4883
8 periods w/ quarters	0.9383434	296.7399	88.32380	731.1697	343.4773

1 quarter model



8 period model, by quarter



Takeaways

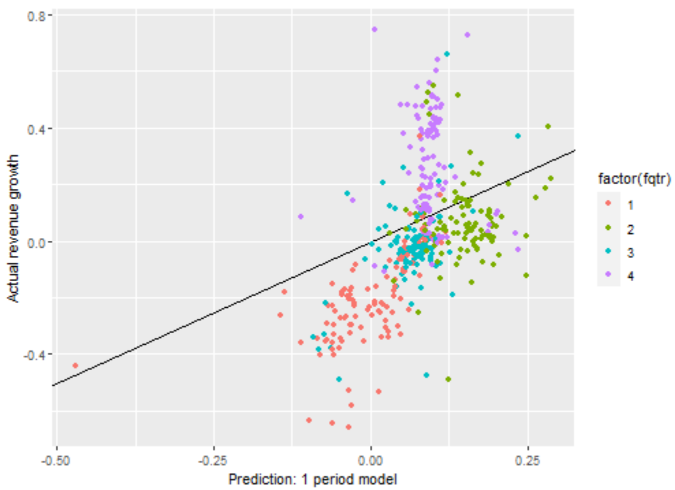
1. The first difference model works about as well as the revenue model at predicting next quarter revenue
   - From earlier, it doesn't suffer (as much) from multicollinearity either
     - This is why time series analysis is often done on first differences
       - Or second differences (difference in differences)
2. The other models perform pretty well as well
3. Extra lags generally seems helpful when accounting for cyclicality
4. Regressing by quarter helps a bit, particularly with revenue growth

What about for revenue growth?

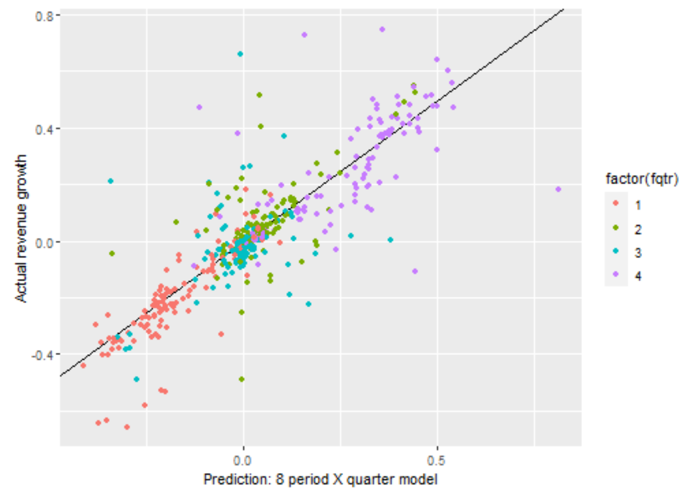
Predicting quarter over quarter revenue growth itself

	adj_r_sq	rmse_in	mae_in	rmse_out	mae_out
1 period	0.0955220	0.3436252	0.2073042	0.2087555	0.1663210
1 and 4 periods	0.4497703	0.2611941	0.1103827	0.1373419	0.0947553
8 periods	0.6788386	0.1737244	0.0848606	0.1269428	0.0801675
8 periods w/ quarters	0.7720057	0.1461233	0.0762027	0.1267874	0.0758181

1 quarter model



8 period model, by quarter

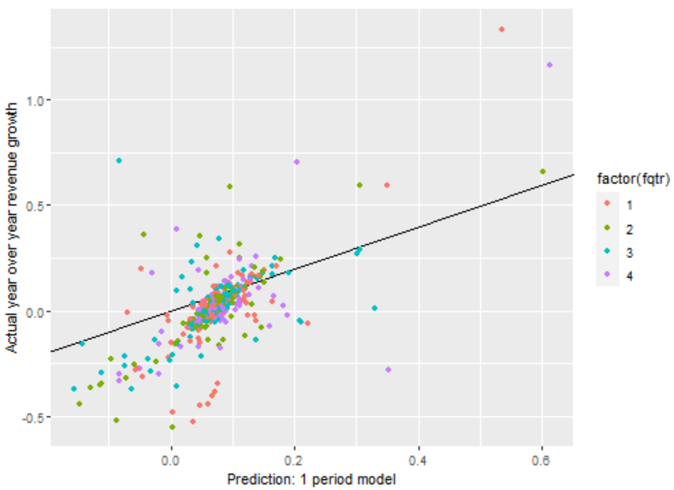


What about for YoY growth?

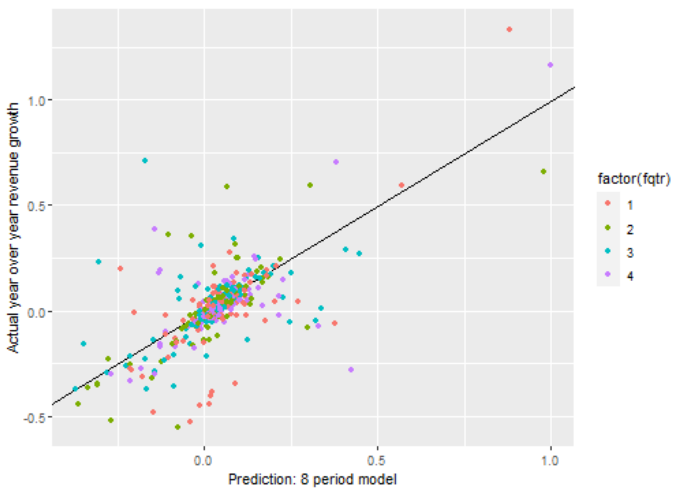
Predicting YoY revenue growth itself

	adj_r_sq	rmse_in	mae_in	rmse_out	mae_out
1 period	0.4376253	0.3022800	0.1085684	0.1511589	0.1006249
1 and 4 periods	0.5378241	0.2389085	0.0993933	0.1493757	0.0967341
8 periods	0.5430590	0.1881716	0.0750616	0.1358365	0.0753768
8 periods w/ quarters	0.1462837	0.2935877	0.1373069	0.1866005	0.1137764

1 quarter model



8 period model

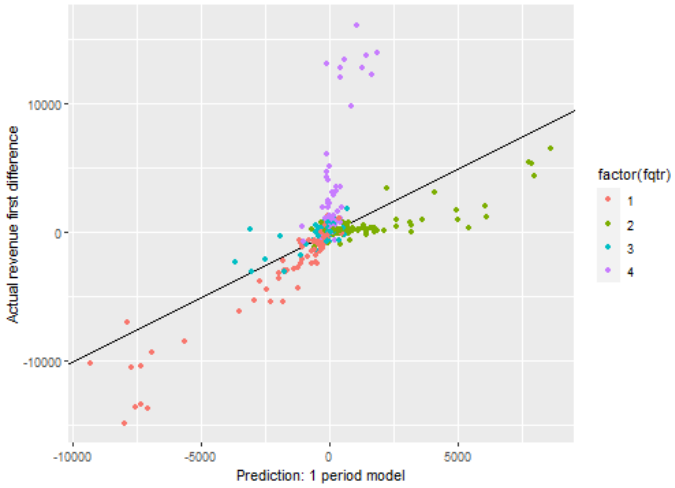


What about for first difference?

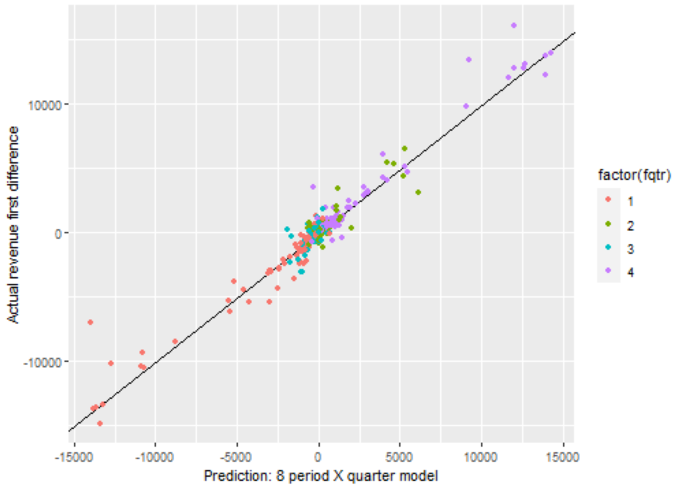
Predicting first difference in revenue itself

	adj_r_sq	rmse_in	mae_in	rmse_out	mae_out
1 period	0.3578089	896.1441	286.47866	2247.2158	986.9519
1 and 4 periods	0.8502591	444.9570	113.00284	860.6968	411.8824
8 periods	0.9242547	329.4611	95.17826	764.8854	348.4883
8 periods w/ quarters	0.9383434	296.7399	88.32380	731.1697	343.4773

1 quarter model



8 period model, by quarter



Summary of Session 5

For next week

- Try to replicate the code for this session
- How is your group project?
- Datacamp
  - Practice a bit more to keep up to date
    - Using R more will make it more natural
- Case: Walmart Store Sales Forecasting

R Coding Style Guide

Style is subjective and arbitrary but it is important to follow a generally accepted style if you want to share code with others. I suggest the The tidyverse style guide which is also adopted by Google with some modification

- Highlights of **the tidyverse style guide**:
  - *File names*: end with .R
  - *Identifiers*: variable_name, function_name, try not to use "." as it is reserved by Base R's S3 objects
  - *Line length*: 80 characters
  - *Indentation*: two spaces, no tabs (RStudio by default converts tabs to spaces and you may change under global options)
  - *Spacing*: x = 0, not x=0, no space before a comma, but always place one after a comma
  - *Curly braces {}*: first on same line, last on own line
  - *Assignment*: use `<-`, not = nor `->`
  - *Semicolon(;)*: don't use, I used once for the interest of space
  - *return()*: Use explicit returns in functions: default function return is the last evaluated expression
  - *File paths*: use relative file path "../../filename.csv" rather than absolute path "C:/mydata/filename.csv". Backslash needs `\\`

R packages used in this slide

This slide was prepared on 2021-09-06 from Session_5s.Rmd with R version 4.1.1 (2021-08-10) Kick Things on Windows 10 x64 build 18362 😄.

The attached packages used in this slide are:

```
##      rlang  lubridate    plotly    forcats    stringr      dplyr      purrr
##   "0.4.11"   "1.7.10"  "4.9.4.1"   "0.5.1"    "1.4.0"    "1.0.7"    "0.3.4"
##      readr      tidyr     tibble    ggplot2  tidyverse kableExtra      knitr
##    "2.0.1"    "1.1.3"    "3.1.3"    "3.3.5"    "1.3.1"    "1.3.4"     "1.33"
```