

ACCT 420: Advanced linear regression

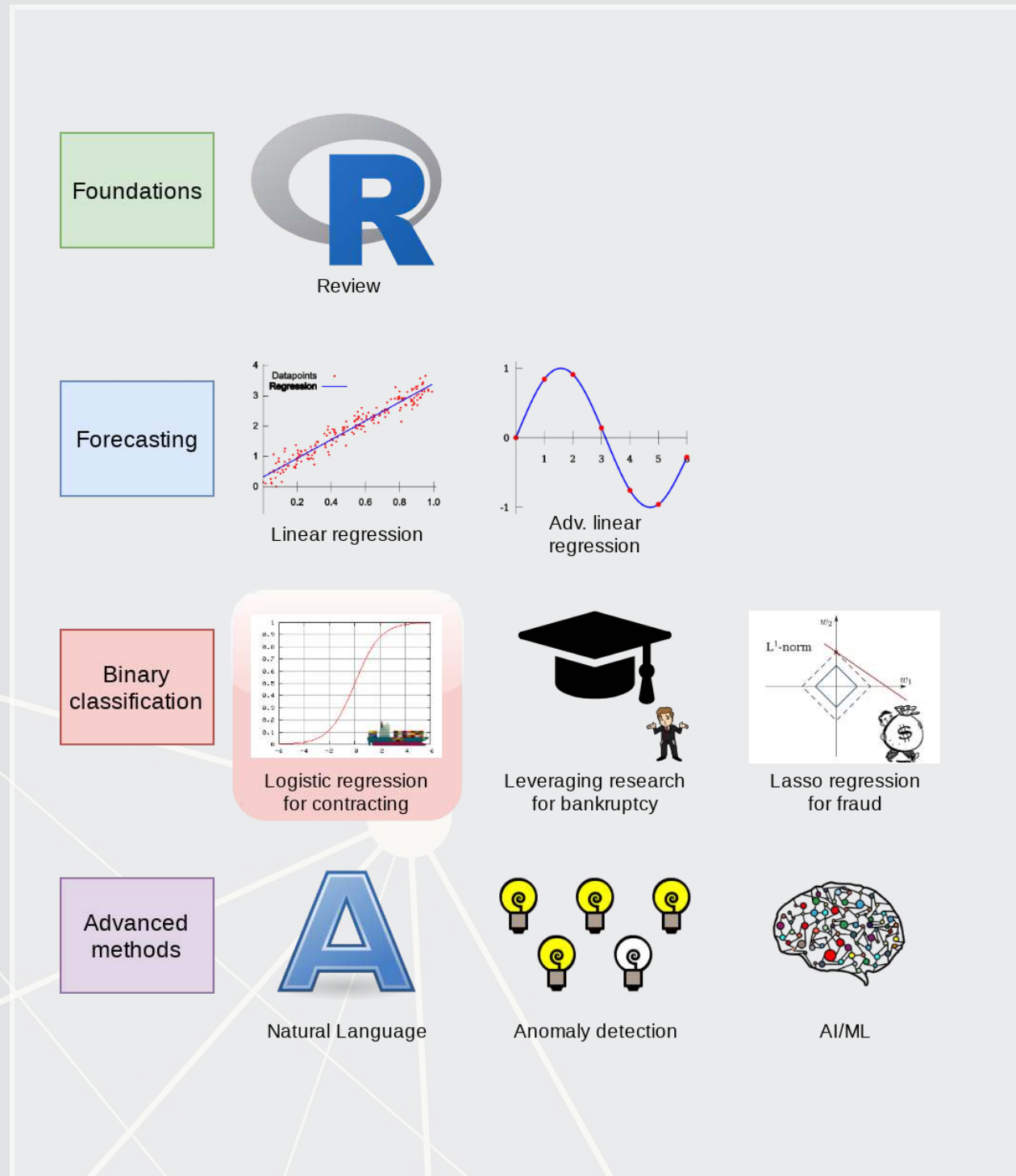
Session 3

Dr. Richard M. Crowley



Front matter

Learning objectives



- **Theory:**
 - Further understand stats treatments
 - Panel data
 - Time (seasonality)
- **Application:**
 - Using international data for our UOL problem
 - Predicting revenue quarterly and weekly
- **Methodology:**
 - Univariate
 - Linear regression (OLS)
 - Visualization

Datacamp

- Explore on your own
- No specific required class this week

Revisiting UOL with macro data

Macro data sources

- For Singapore: [Data.gov.sg](https://data.gov.sg)
 - Covers: Economy, education, environment, finance, health, infrastructure, society, technology, transport
- For real estate in Singapore: URA's REALIS system
 - Access through the library
- WRDS has some as well
- For US: data.gov, as well as many agency websites
 - Like [BLS](https://www.bls.gov) or the [Federal Reserve](https://www.federalreserve.gov)



Loading macro data

- Singapore business expectations data (from data.gov.sg)

```
# extract out Q1, finance only
expectations_avg <- expectations %>%
  filter(quarter == 1,
         level_2 == "Financial & Insurance") %>%
  group_by(year) %>%
  mutate(fin_sentiment=mean(value, na.rm=TRUE)) %>%
  slice(1)
# Keep only the first quarter
# Keep only financial responses
# Group data by year
# Calculate average
# Take only 1 row per group
```

- At this point, we can merge with our accounting data

What was in the macro data?

```
expectations %>%  
  arrange(level_2, level_3, desc(year)) %>% # sort the data  
  select(year, quarter, level_2, level_3, value) %>% # keep only these variables  
  datatable(options = list(pageLength = 5), rownames=FALSE) # display using DT
```

Show entries

Search:

year	quarter	level_2	level_3	value
2018	1	Accommodation & Food Services	Accommodation	-7
2018	2	Accommodation & Food Services	Accommodation	38
2017	1	Accommodation & Food Services	Accommodation	-15
2017	2	Accommodation & Food Services	Accommodation	27
2017	3	Accommodation & Food Services	Accommodation	11

Showing 1 to 5 of 846 entries

Previous

1

2

3

4

5

...

170

Next

dplyr makes merging easy

- For merging, use `dplyr`'s `*_join()` commands
 - `left_join()` for merging a dataset into another
 - `inner_join()` for keeping only matched observations
 - `outer_join()` for making all possible combinations
- For sorting, `dplyr`'s `arrange()` command is easy to use
 - For sorting in reverse, combine `arrange()` with `desc()`

Merging example

Merge in the finance sentiment data to our accounting data

```
# subset out our Singaporean data, since our macro data is Singapore-specific  
df_SG <- df_clean %>% filter(fic == "SGP")
```

```
# Create year in df_SG (date is given by datadate as YYYYMMDD)  
df_SG$year = round(df_SG$datadate / 10000, digits=0)
```

```
# Combine datasets  
# Notice how it automatically figures out to join by "year"  
df_SG_macro <- left_join(df_SG, expectations_avg[,c("year", "fin_sentiment")])
```

```
## Joining, by = "year"
```


Predicting with macro data

Building in macro data

- First try: Just add it in

```
macrol <- lm(revt_lead ~ revt + act + che + lct + dp + ebit + fin_sentiment,  
            data=df_SG_macro)  
library(broom)  
tidy(macrol)
```

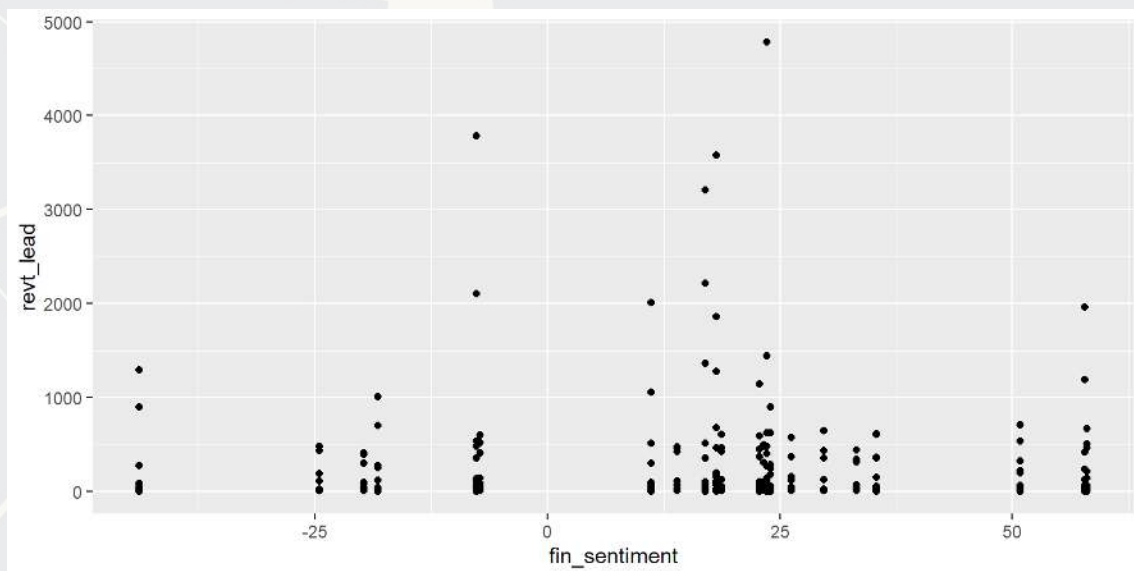
```
## # A tibble: 8 x 5  
##   term          estimate std.error statistic    p.value  
##   <chr>          <dbl>    <dbl>    <dbl>    <dbl>  
## 1 (Intercept)    24.0     15.9      1.50    0.134  
## 2 revt           0.497     0.0798    6.22    0.00000000162  
## 3 act           -0.102     0.0569   -1.79    0.0739  
## 4 che            0.495     0.167     2.96    0.00329  
## 5 lct            0.403     0.0903    4.46    0.0000114  
## 6 dp             4.54      1.63      2.79    0.00559  
## 7 ebit          -0.930     0.284    -3.28    0.00117  
## 8 fin_sentiment  0.122     0.472     0.259    0.796
```

It isn't significant. Why is this?

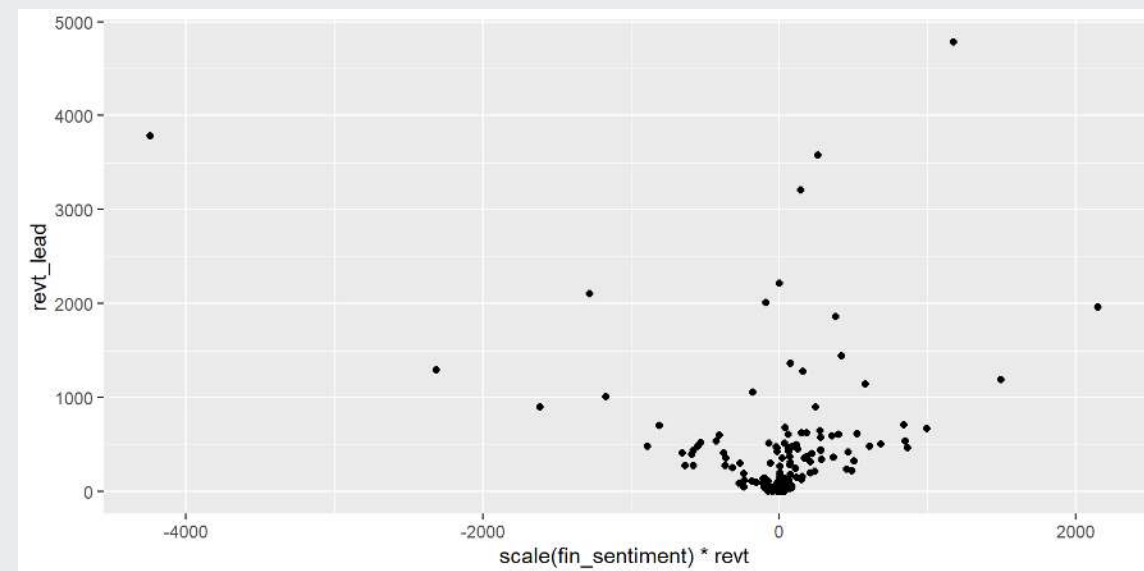
Scaling matters

- All of our firm data is on the same terms as revenue: dollars within a given firm
- But `fin_sentiment` is a constant scale...
 - Need to scale this to fit the problem
 - The current scale would work for revenue growth

```
df_SG_macro %>%  
  ggplot(aes(y=revt_lead,  
             x=fin_sentiment)) +  
  geom_point()
```



```
df_SG_macro %>%  
  ggplot(aes(y=revt_lead,  
             x=scale(fin_sentiment) * revt)) +  
  geom_point()
```



Scaled macro data

- Normalize and scale by revenue

```
# Scale creates z-scores, but returns a matrix by default. [,1] gives a vector
df_SG_macro$fin_sent_scaled <- scale(df_SG_macro$fin_sentiment)[,1]
macro3 <-
  lm(revt_lead ~ revt + act + che + lct + dp + ebit + fin_sent_scaled:revt,
     data=df_SG_macro)
tidy(macro3)
```

```
## # A tibble: 8 x 5
##   term                estimate std.error statistic    p.value
##   <chr>                <dbl>    <dbl>    <dbl>    <dbl>
## 1 (Intercept)          25.5      13.8      1.84 0.0663
## 2 revt                 0.490     0.0789     6.21 0.00000000170
## 3 act                -0.0677    0.0576    -1.18 0.241
## 4 che                 0.439     0.166     2.64 0.00875
## 5 lct                 0.373     0.0898     4.15 0.0000428
## 6 dp                  4.10      1.61     2.54 0.0116
## 7 ebit                -0.793    0.285    -2.78 0.00576
## 8 revt:fin_sent_scaled 0.0897    0.0332     2.70 0.00726
```

```
glance(macro3)
```

```
## # A tibble: 1 x 12
##   r.squared adj.r.squared sigma statistic    p.value    df logLik  AIC  BIC
##   <dbl>      <dbl> <dbl>    <dbl>    <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 0.847      0.844 215.    240. 1.48e-119    7 -2107. 4232. 4265.
## # ... with 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>
```


Model comparisons

```
baseline <-  
  lm(revt_lead ~ revt + act + che + lct + dp + ebit,  
     data=df_SG_macro[!is.na(df_SG_macro$fin_sentiment),])  
glance(baseline)
```

```
## # A tibble: 1 x 12  
##   r.squared adj.r.squared sigma statistic  p.value    df logLik  AIC  BIC  
##   <dbl>      <dbl> <dbl>    <dbl>    <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1     0.843      0.840  217.    273. 3.13e-119     6 -2111. 4237. 4267.  
## # ... with 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>
```

```
glance(macro3)
```

```
## # A tibble: 1 x 12  
##   r.squared adj.r.squared sigma statistic  p.value    df logLik  AIC  BIC  
##   <dbl>      <dbl> <dbl>    <dbl>    <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1     0.847      0.844  215.    240. 1.48e-119     7 -2107. 4232. 4265.  
## # ... with 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>
```

Adjusted R^2 and AIC are slightly better with macro data

Model comparisons

```
anova(baseline, macro3, test="Chisq")
```

```
## Analysis of Variance Table
##
## Model 1: revt_lead ~ revt + act + che + lct + dp + ebit
## Model 2: revt_lead ~ revt + act + che + lct + dp + ebit + fin_sent_scaled:revt
##   Res.Df      RSS Df Sum of Sq Pr(>Chi)
## 1     304 14285622
## 2     303 13949301  1     336321 0.006875 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Macro model definitely fits better than the baseline model!

Takeaway

1. Adding macro data can help explain some exogenous variation in a model
 - Exogenous meaning outside of the firms, in this case
2. Scaling is very important
 - Not scaling properly can suppress some effects from being visible

Interpretating the macro variable

- For every 1 S.D. increase in `fin_sentiment` (25.7 points)
 - Revenue stickiness increases by ~1.1%
- Over the range of data (-43.8 to 58)...
 - Revenue stickiness ranges from -1.8% to 2.4%

Scaling up our model, again

Building a model requires careful thought!

- What macroeconomic data makes sense to add to our model?

This is where having accounting and business knowledge comes in!

TEAMWORK

Validation: Is it better?

Validation

- Ideal:
 - Withhold the last year (or a few) of data when building the model
 - Check performance on *hold out sample*
 - This is *out of sample* testing
- Sometimes acceptable:
 - Withhold a random sample of data when building the model
 - Check performance on *hold out sample*

Estimation

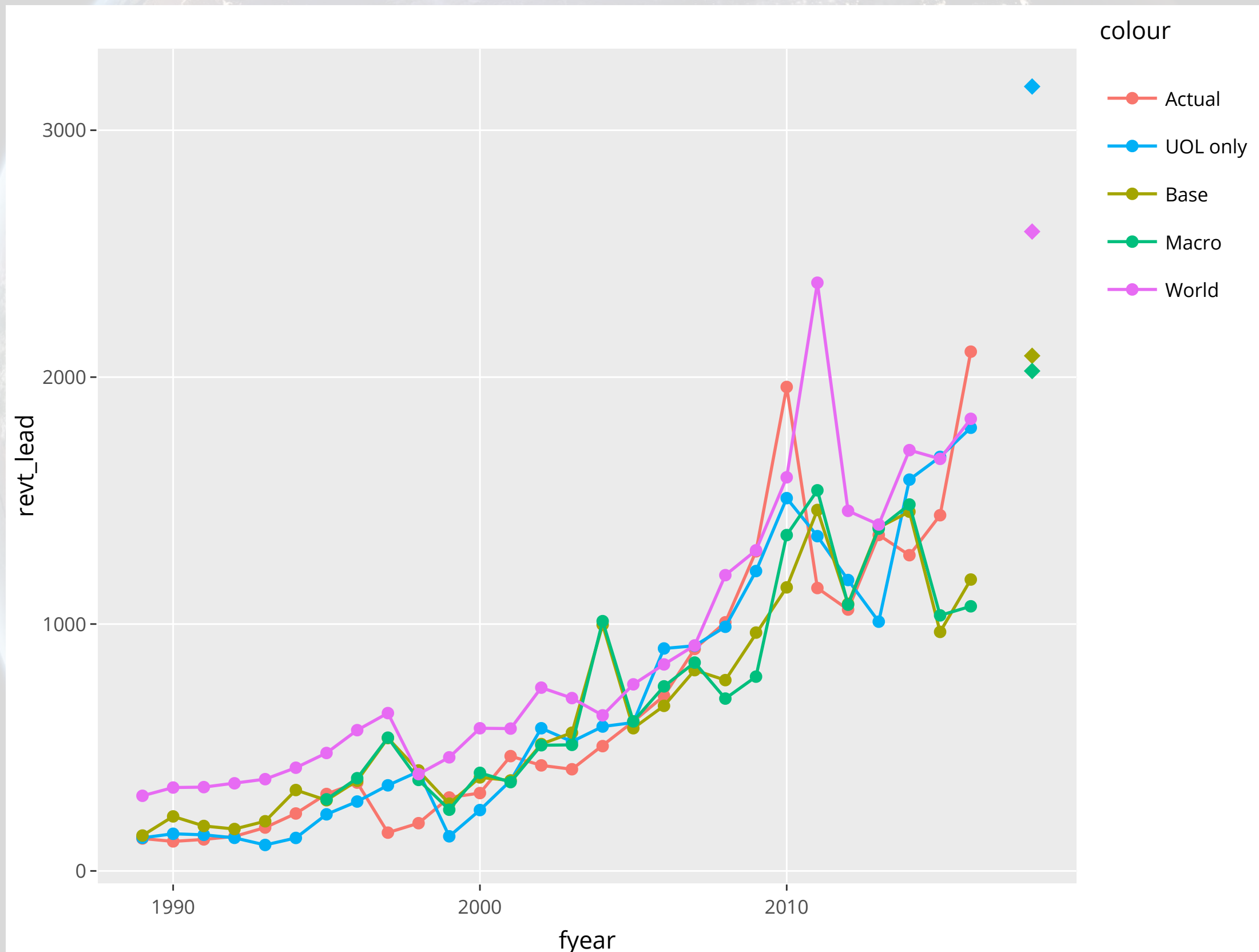
- As we never constructed a hold out sample, let's end by estimating UOL's *2018* year revenue
 - Announced in 2019

```
p_uol <- predict(forecast2, uol[uol$fyyear==2017,])
p_base <- predict(baseline,
  df_SG_macro[df_SG_macro$isin=="SG1S83002349" & df_SG_macro$fyyear==2017,])
p_macro <- predict(macro3,
  df_SG_macro[df_SG_macro$isin=="SG1S83002349" & df_SG_macro$fyyear==2017,])
p_world <- predict(forecast4,
  df_clean[df_clean$isin=="SG1S83002349" & df_clean$fyyear==2017,])
preds <- c(p_uol, p_base, p_macro, p_world)
names(preds) <- c("UOL 2018 UOL", "UOL 2018 Base", "UOL 2018 Macro",
  "UOL 2018 World")

preds
```

```
##      UOL 2018 UOL      UOL 2018 Base UOL 2018 Macro UOL 2018 World
##      3177.073          2086.437          2024.842          2589.636
```

Visualizing our prediction



In Sample Accuracy

```
# series vectors calculated here -- See appendix
rmse <- function(v1, v2) {
  sqrt(mean((v1 - v2)^2, na.rm=T))
}

rmse <- c(rmse(actual_series, uol_series), rmse(actual_series, base_series),
         rmse(actual_series, macro_series), rmse(actual_series, world_series))
names(rmse) <- c("UOL 2018 UOL", "UOL 2018 Base", "UOL 2018 Macro", "UOL 2018 World")
rmse
```

```
##      UOL 2018 UOL      UOL 2018 Base UOL 2018 Macro UOL 2018 World
##      175.5609      301.3161      344.9681      332.8101
```

Why is UOL the best for in sample?

Actual Accuracy

UOL posted a \$2.40B in revenue in 2018.

preds

##	UOL 2018	UOL	UOL 2018 Base	UOL 2018 Macro	UOL 2018 World
##	3177.073		2086.437	2024.842	2589.636

Why is the global model better? Consider UOL's business model ([2018 annual report](#))

Session 3's 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

More specifically...

- Consider time dimensions
 - What matters:
 - Last quarter?
 - Last year?
 - Other time frames?
 - Cyclicity

Time and OLS

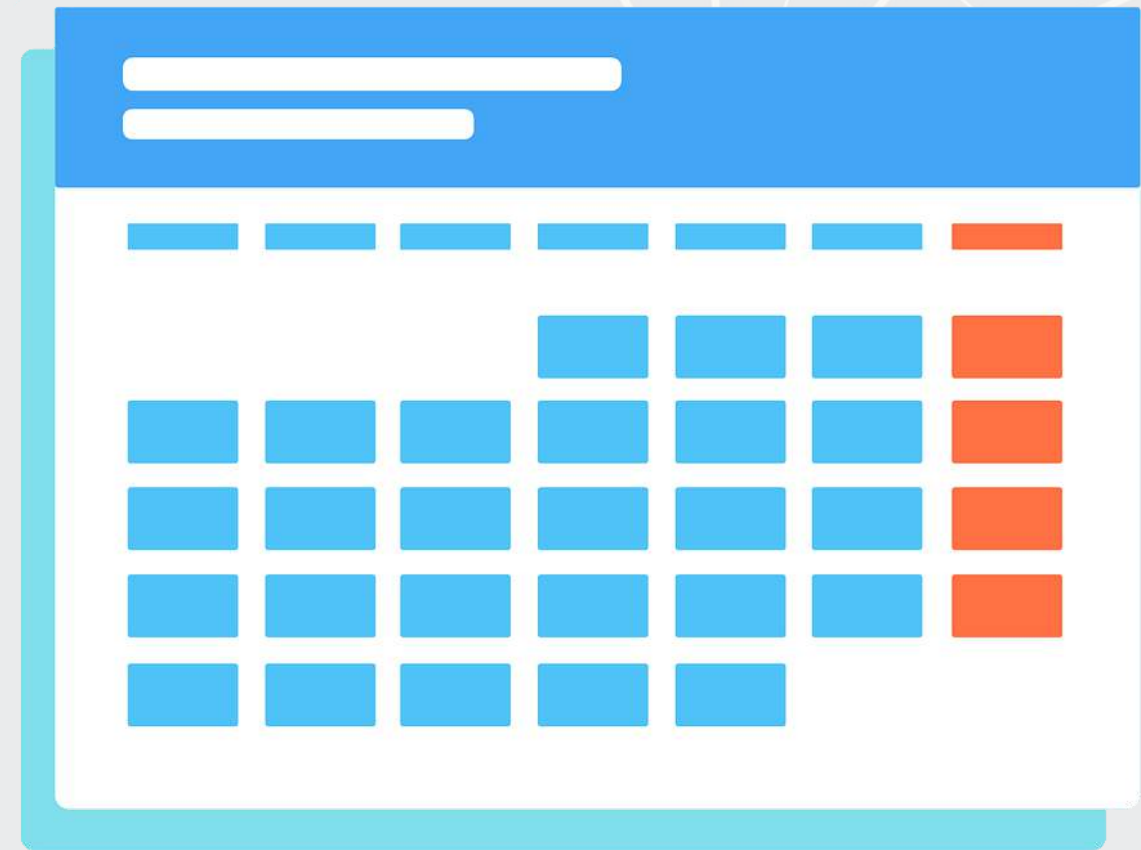
Time matters a lot for retail



Great Singapore Sale

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
- Two sets of firms:
 - US “Hypermarkets & Super Centers” (GICS: 30101040)
 - US “Multiline Retail” (GICS: 255030)
- Data from Compustat - Capital IQ > North America - Daily > Fundamentals Quarterly



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. Prediction

- Use OLS for all the above – t -tests for coefficients
- Hold out sample: 2015-2017

Variable generation

```
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
# Date was YYYYMMDDs10: YYYY/MM/DD, can be converted from text to date easily
df$date <- ymd(df$datadate) # From lubridate
df$qtr <- quarter(df$date) # From lubridate
```

- Use mutate for variables using lags
- ymd () from lubridate is a handy way of converting any date listing year, then month, than day.
 - It also has ydm (), mdy (), myd (), dmy () and dym ()
 - It can handle quarters, times, and date-times as well
 - [Cheat sheet](#)

Base R Date manipulation

- `as.Date()` 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
 - i.e., “DD.MM.YYYY” is format code “%d.%m.%Y”
 - [Full list of date codes](#)

Example output

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 3
##   conm      date      datadate
##   <chr>    <date>    <chr>
## 1 ALLIED STORES 1962-04-30 1962/04/30
## 2 ALLIED STORES 1962-07-31 1962/07/31
## 3 ALLIED STORES 1962-10-31 1962/10/31
## 4 ALLIED STORES 1963-01-31 1963/01/31
## 5 ALLIED STORES 1963-04-30 1963/04/30
## 6 ALLIED STORES 1963-07-31 1963/07/31
```


Create 8 quarters (2 years) of lags

```
# Equivalent brute force code for this is in the appendix
# 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)
```

- `paste0()`: creates a string vector by concatenating all inputs
- `paste()`: same as `paste0()`, but with spaces added in between
- `setNames()`: allows for storing a value and name simultaneously
- `mutate_at()`: is like `mutate` but with a list of functions

Example output

conm	date	revtq	revtq_l1	revtq_l2	revtq_l3	revtq_l4
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	156.5	NA	NA
ALLIED STORES	1963-01-31	275.5	176.9	161.9	156.5	NA
ALLIED STORES	1963-04-30	171.1	275.5	176.9	161.9	156.5
ALLIED STORES	1963-07-31	182.2	171.1	275.5	176.9	161.9

Clean and split into training and testing

```
# 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 testing data
# Training data: We'll use data released before 2015
train <- filter(df, year(date) < 2015)

# Testing data: We'll use data released 2015 through 2018
test <- filter(df, year(date) >= 2015)
```

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

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 `ggplot2`

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

```
##      revtq      revtq_gr      revtq_yoy      revtq_d
## Min.   : 0.00   Min.   : -1.0000   Min.   : -1.0000   Min.   : -24325.21
## 1st Qu.: 64.46   1st Qu.: -0.1112   1st Qu.: 0.0077   1st Qu.:  -19.33
## Median : 273.95   Median : 0.0505   Median : 0.0740   Median :    4.30
## Mean   : 2439.38   Mean   : 0.0650   Mean   : 0.1273   Mean   :   22.66
## 3rd Qu.: 1254.21   3rd Qu.: 0.2054   3rd Qu.: 0.1534   3rd Qu.:   55.02
## Max.   :136267.00   Max.   :14.3333   Max.   :47.6600   Max.   : 15495.00
## NA's   :367       NA's   :690       NA's   :940       NA's   :663
##
##      qtr
## Min.   :1.000
## 1st Qu.:2.000
## Median :3.000
## Mean   :2.503
## 3rd Qu.:3.000
## Max.   :4.000
##
```

ggplot2 for visualization

- The next slides will use some custom functions using `ggplot2`
- `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_x_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](#)

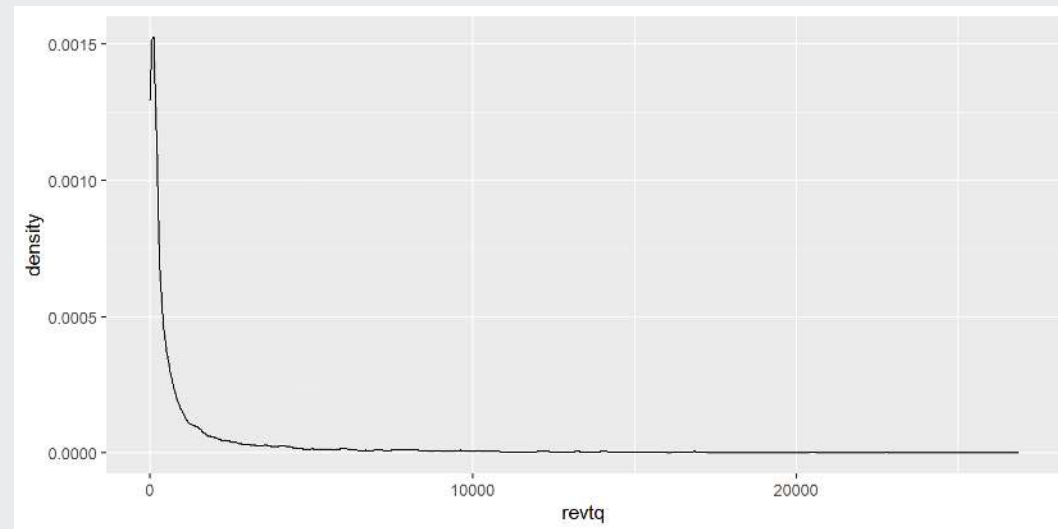
ggplot2 for visualization

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

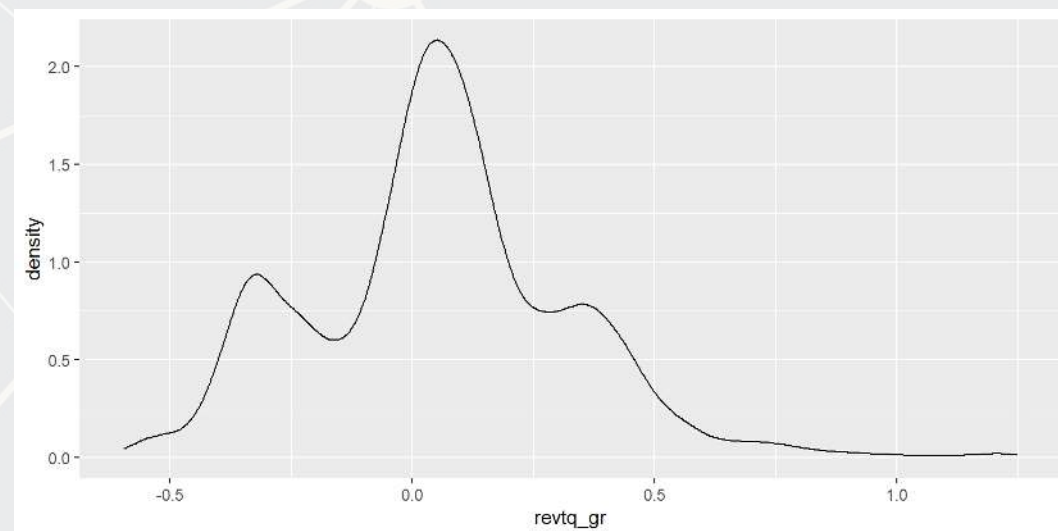
- geom stands for geometry – any shapes, lines, etc. start with geom
- Other useful geoms:
 - `geom_line()`: makes a line chart
 - `geom_bar()`: makes a bar chart – y is the height, x is the category
 - `geom_smooth(method="lm")`: Adds a linear regression into the chart
 - `geom_abline(slope=1)`: Adds 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’](#) on eLearn

Plotting: Distribution of revenue

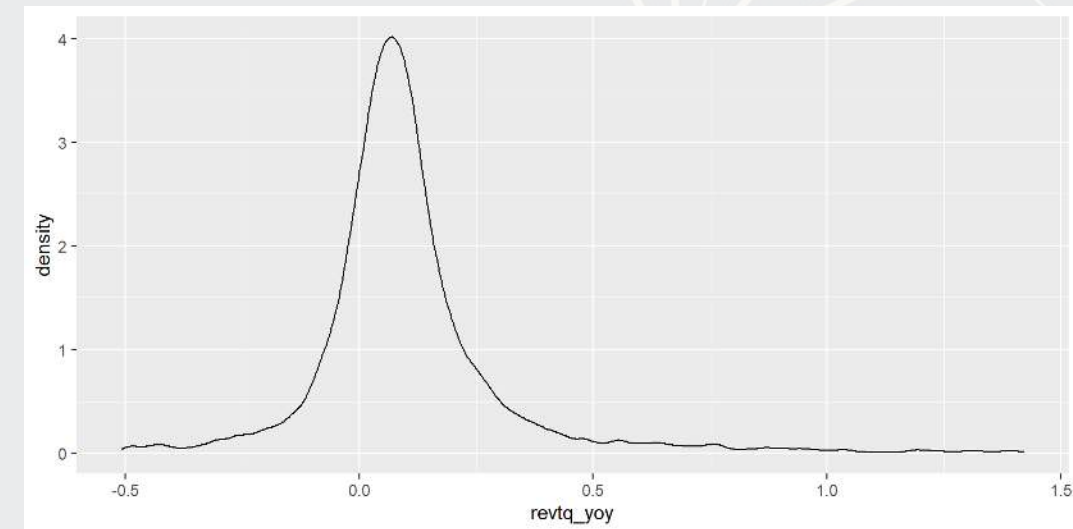
1. Revenue



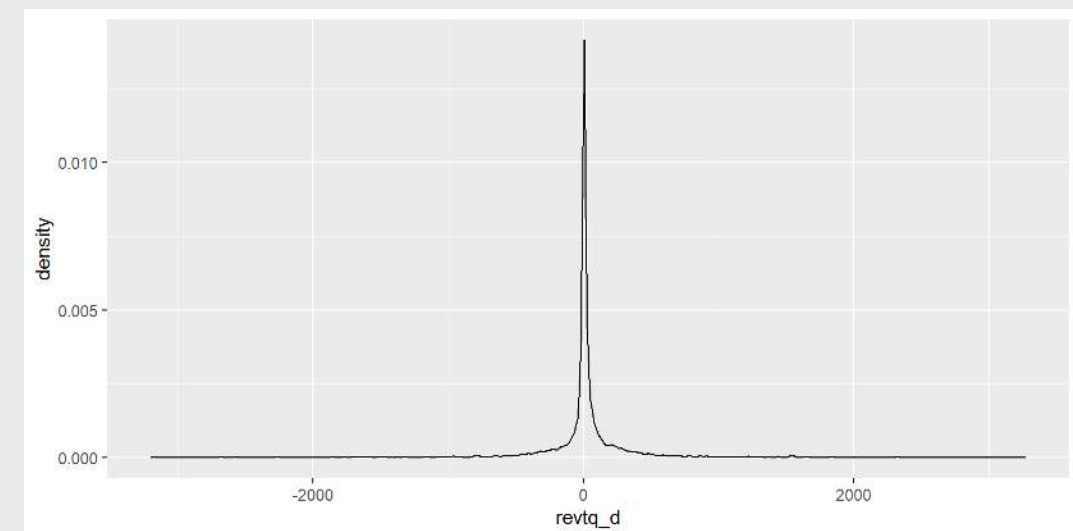
2. Quarterly growth



3. Year-over-year growth



4. First difference



What do we learn from these graphs?

1. Revenue



2. Quarterly growth



3. Year-over-year growth

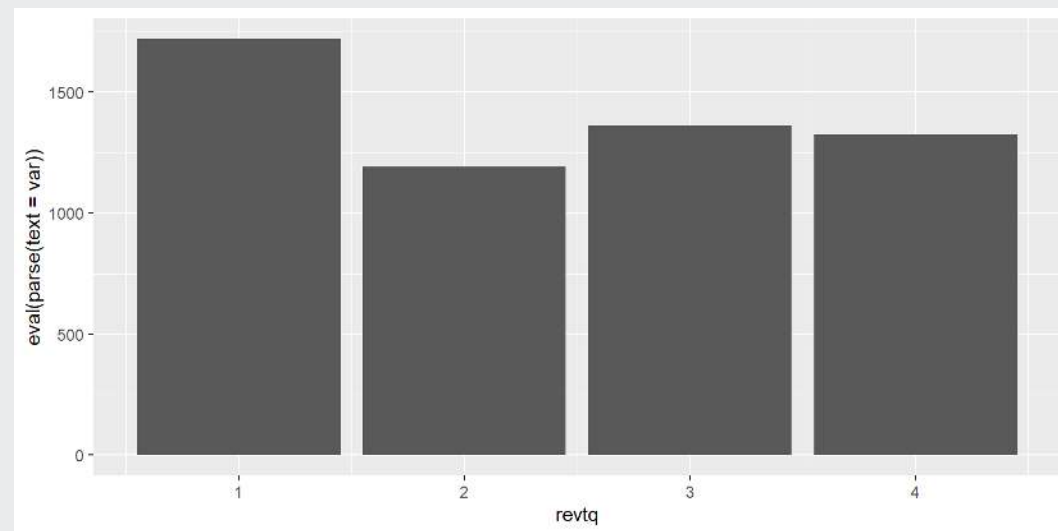


4. First difference

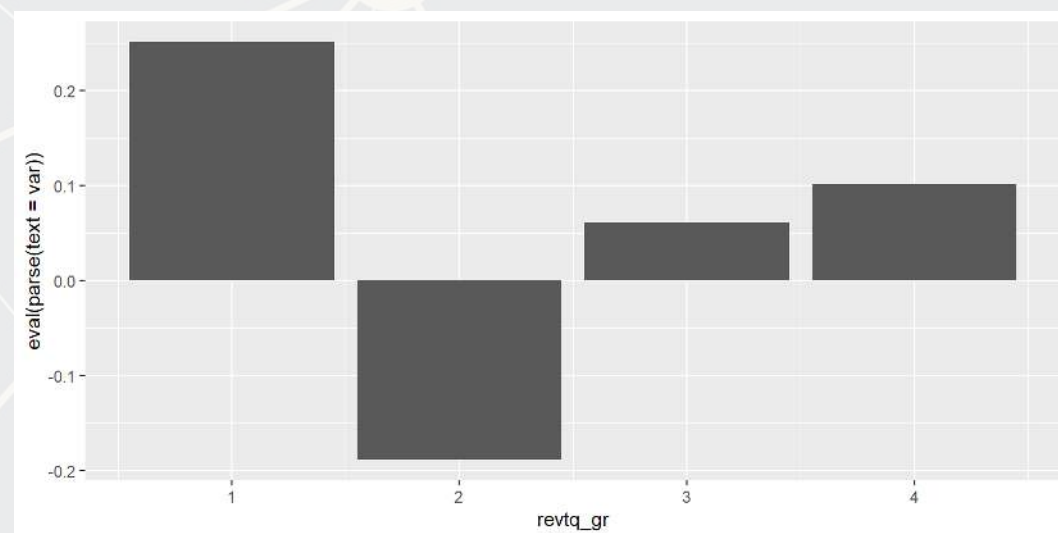


Plotting: Mean revenue by quarter

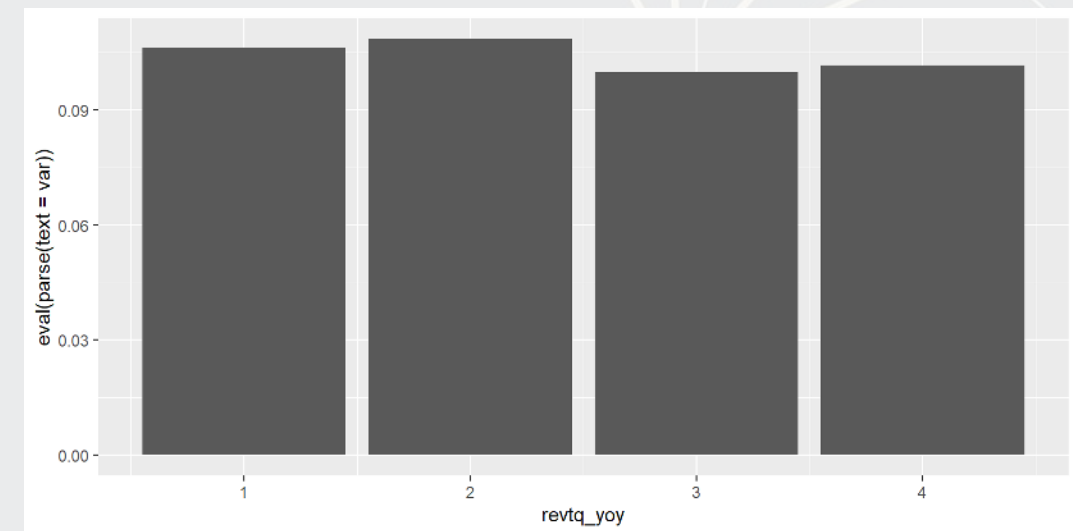
1. Revenue



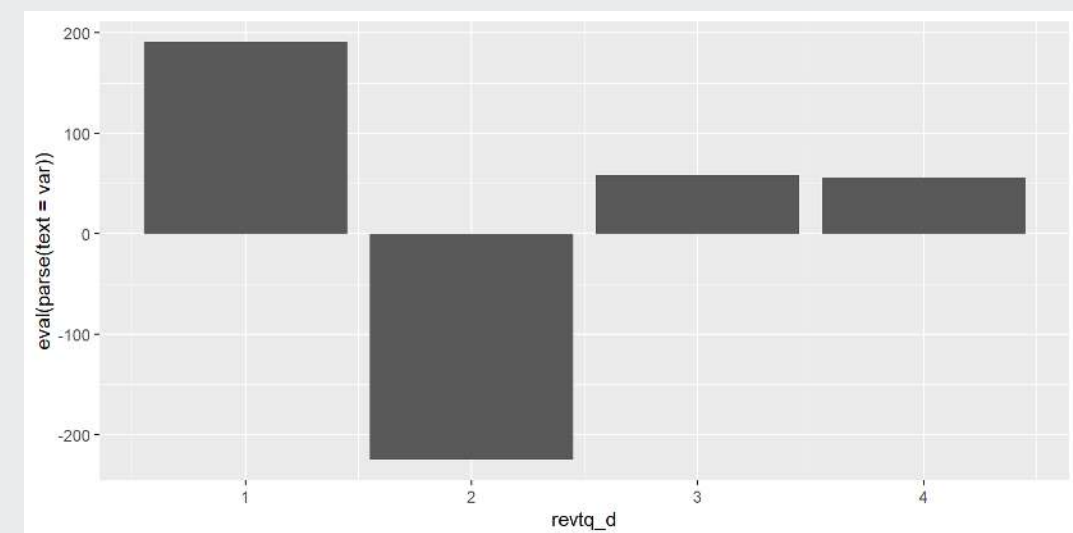
2. Quarterly growth



3. Year-over-year growth



4. First difference



What do we learn from these graphs?

1. Revenue



2. Quarterly growth



3. Year-over-year growth

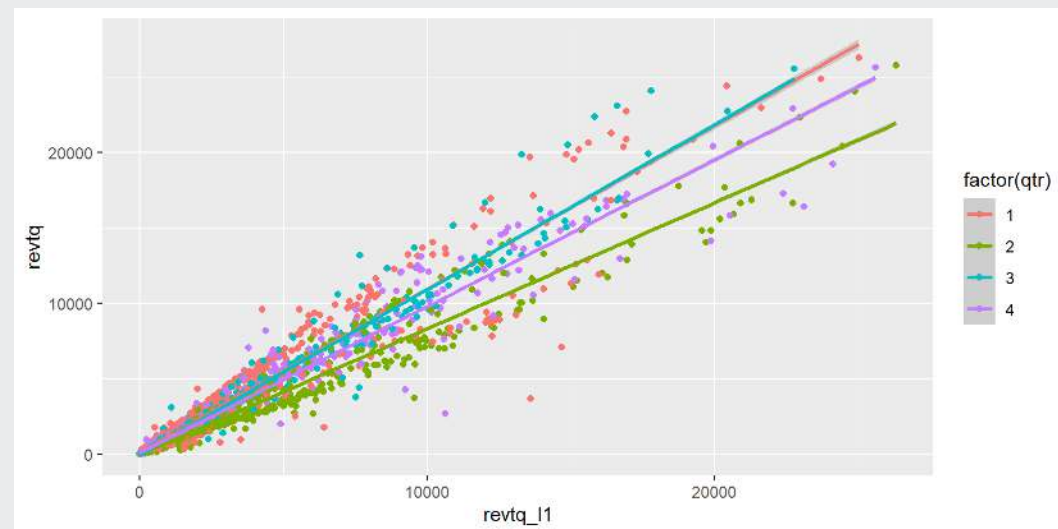


4. First difference

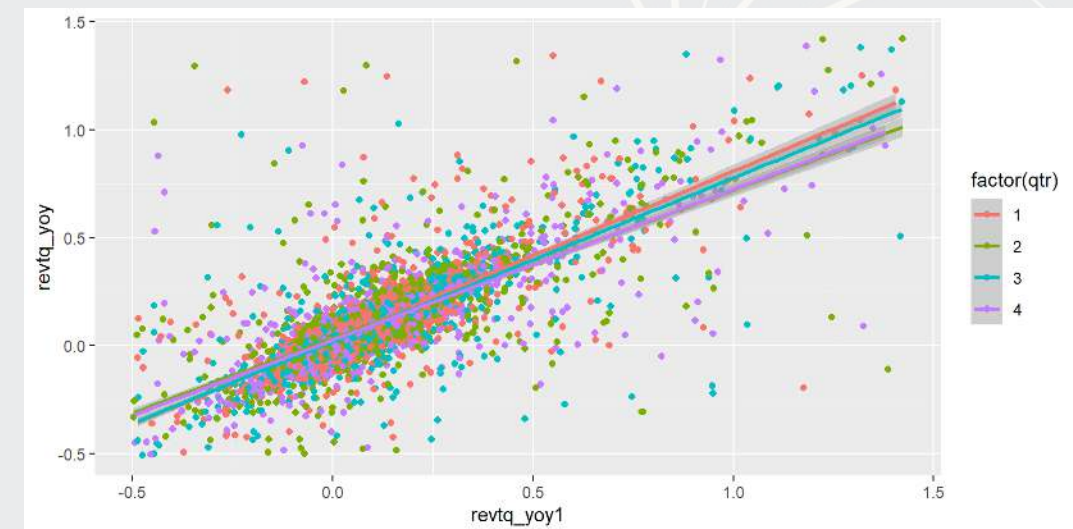


Plotting: Revenue vs lag by quarter

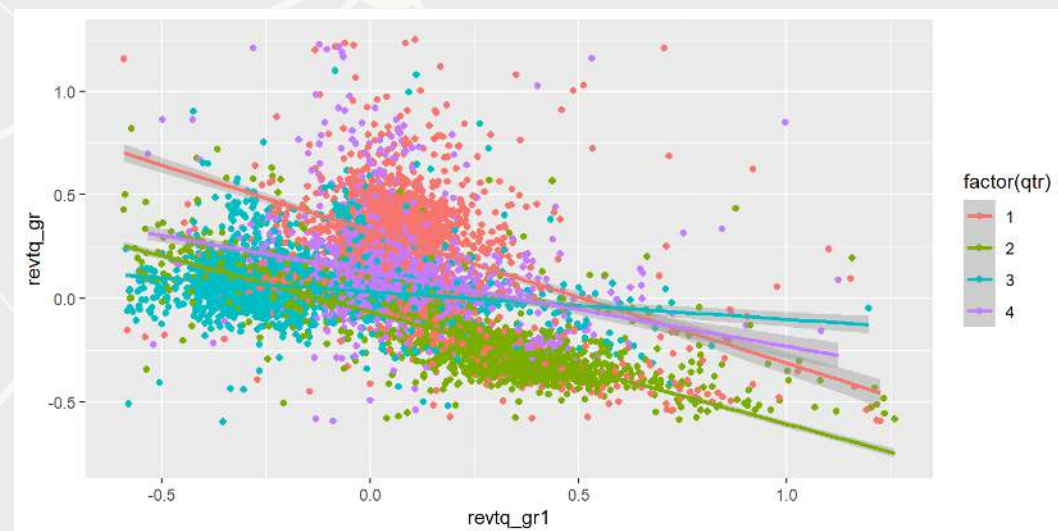
1. Revenue



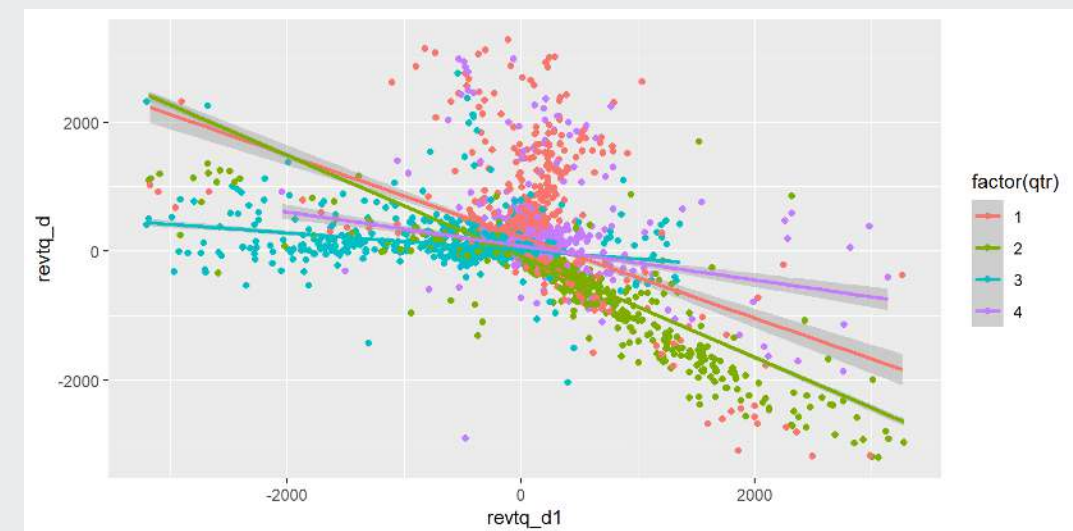
3. Year-over-year growth



2. Quarterly growth



4. First difference



What do we learn from these 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")
```

```
##           revtq  revtq_l1  revtq_l2  revtq_l3  revtq_l4  
## revtq      1.0000000  0.9916167  0.9938489  0.9905522  0.9972735  
## revtq_l1  0.9916167  1.0000000  0.9914767  0.9936977  0.9898184  
## revtq_l2  0.9938489  0.9914767  1.0000000  0.9913489  0.9930152  
## revtq_l3  0.9905522  0.9936977  0.9913489  1.0000000  0.9906006  
## revtq_l4  0.9972735  0.9898184  0.9930152  0.9906006  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  -0.32291329  0.06299605  -0.22769442  0.64570015  
## revtq_gr1 -0.32291329  1.00000000  -0.31885020  0.06146805  -0.21923630  
## revtq_gr2  0.06299605  -0.31885020  1.00000000  -0.32795121  0.06775742  
## revtq_gr3 -0.22769442  0.06146805  -0.32795121  1.00000000  -0.31831023  
## revtq_gr4  0.64570015  -0.21923630  0.06775742  -0.31831023  1.00000000
```

Retail revenue has really high autocorrelation! Concern for multicollinearity. 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 0.6554179 0.4127263 0.4196003 0.1760055  
## revtq_yoy1 0.6554179 1.0000000 0.5751128 0.3665961 0.3515105  
## revtq_yoy2 0.4127263 0.5751128 1.0000000 0.5875643 0.3683539  
## revtq_yoy3 0.4196003 0.3665961 0.5875643 1.0000000 0.5668211  
## revtq_yoy4 0.1760055 0.3515105 0.3683539 0.5668211 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 -0.6181516 0.3309349 -0.6046998 0.9119911  
## revtq_d1 -0.6181516 1.0000000 -0.6155259 0.3343317 -0.5849841  
## revtq_d2 0.3309349 -0.6155259 1.0000000 -0.6191366 0.3165450  
## revtq_d3 -0.6046998 0.3343317 -0.6191366 1.0000000 -0.5864285  
## revtq_d4 0.9119911 -0.5849841 0.3165450 -0.5864285 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
 - Cyclicalty
- Practice using:
 - `lm()`
 - `ggplot2`
- Do the exercises in today's practice file
 - [R Practice](#)
 - Short link: rmc.link/420r3

Forecasting

1 period models

1. 1 Quarter lag

- We saw a very strong linear pattern here earlier

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

2. Quarter and year lag

- Year-over-year seemed pretty constant

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

3. 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)
```

4. 2 years of lags, by observation quarter

- Take into account cyclicity observed in bar charts

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


Quarter lag

`summary(mod1)`

```
##
## Call:
## lm(formula = revtq ~ revtq_l1, data = train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -24438.7   -34.0    -11.7    34.6   15200.5
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 15.639975  13.514877   1.157   0.247
## revtq_l1    1.003038   0.001556 644.462 <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1152 on 7676 degrees of freedom
## (662 observations deleted due to missingness)
## Multiple R-squared:  0.9819, Adjusted R-squared:  0.9819
## F-statistic: 4.153e+05 on 1 and 7676 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
## -20245.7   -18.4    -4.4    19.1   9120.8
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  5.444986   7.145633   0.762   0.446
## revtq_l1     0.231759   0.005610  41.312 <2e-16 ***
## revtq_l4     0.815570   0.005858 139.227 <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 592.1 on 7274 degrees of freedom
## (1063 observations deleted due to missingness)
## Multiple R-squared:  0.9954, Adjusted R-squared:  0.9954
## F-statistic: 7.94e+05 on 2 and 7274 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
## -5005.6   -12.9    -3.7     9.3   5876.3
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4.02478    4.37003   0.921  0.3571
## revtq_l1     0.77379    0.01229  62.972 < 2e-16 ***
## revtq_l2     0.10497    0.01565   6.707 2.16e-11 ***
## revtq_l3    -0.03091    0.01538  -2.010  0.0445 *
## revtq_l4     0.91982    0.01213  75.800 < 2e-16 ***
## revtq_l5    -0.76459    0.01324 -57.749 < 2e-16 ***
## revtq_l6    -0.08080    0.01634  -4.945 7.80e-07 ***
## revtq_l7     0.01146    0.01594   0.719  0.4721
## revtq_l8     0.07924    0.01209   6.554 6.03e-11 ***
## ---
```

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(qtr), data = train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -5316.5   -12.2     0.9    15.7   5283.2
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)    -0.45240     4.32484  -0.105  0.916692
## revtq_l1:factor(qtr)1  0.91094     0.02610  34.896 < 2e-16 ***
## revtq_l1:factor(qtr)2  0.67361     0.02376  28.355 < 2e-16 ***
## revtq_l1:factor(qtr)3  0.97588     0.02747  35.525 < 2e-16 ***
## revtq_l1:factor(qtr)4  0.65106     0.02216  29.377 < 2e-16 ***
## revtq_l2:factor(qtr)1  0.05733     0.02872   1.996  0.045997 *
## revtq_l2:factor(qtr)2  0.14708     0.03410   4.313  1.64e-05 ***
## revtq_l2:factor(qtr)3  0.02910     0.02976   0.978  0.328253
## revtq_l2:factor(qtr)4  0.36807     0.03468  10.614 < 2e-16 ***
## revtq_l3:factor(qtr)1 -0.09063     0.03717  -2.438  0.014788 *
```


Testing out of sample

- RMSE: Root mean square Error
- RMSE is very affected by outliers, and a bad choice for noisy data where you are OK with missing a few outliers here and there
 - Doubling error *quadruples* the penalty

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

- MAE: Mean absolute error
- MAE is measures average accuracy with no weighting
 - Doubling error *doubles* the penalty

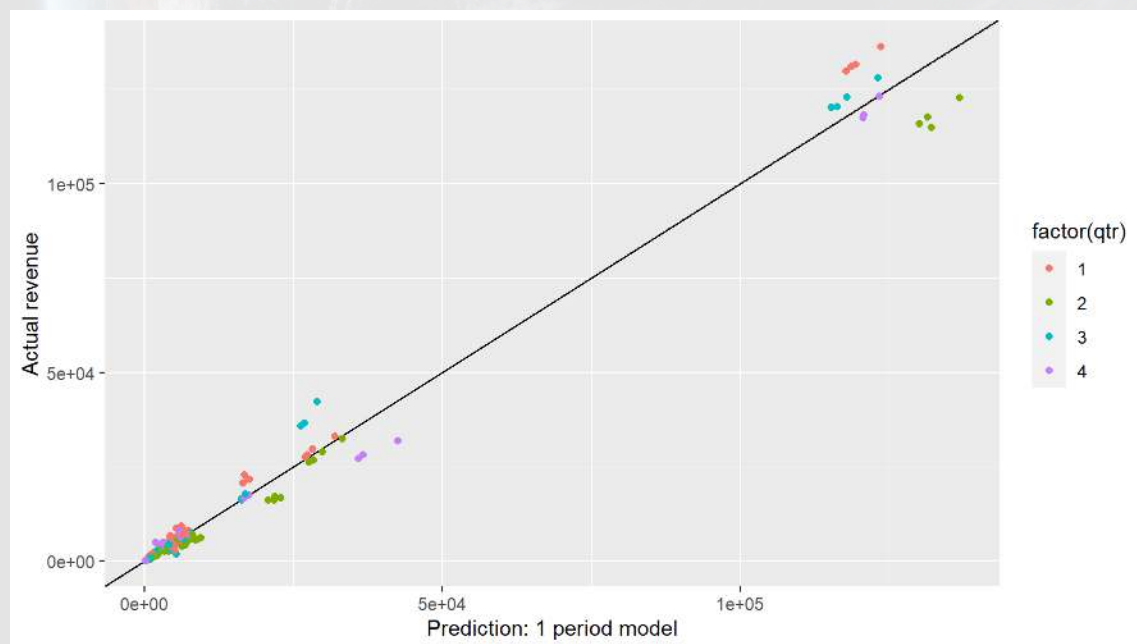
```
mae <- function(v1, v2) {  
  mean(abs(v1-v2), na.rm=T)  
}
```

Both are commonly used for evaluating OLS out of sample

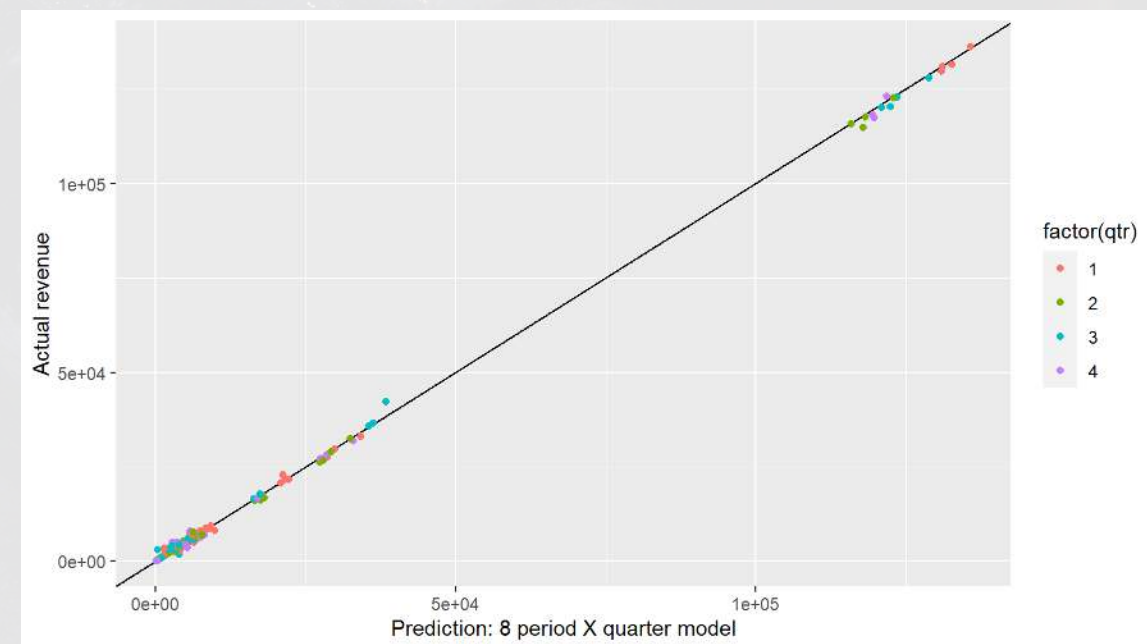
Testing out of sample

	adj_r_sq	rmse_in	mae_in	rmse_out	mae_out
1 period	0.9818514	1151.3535	322.73819	2947.3619	1252.5196
1 and 4 periods	0.9954393	591.9500	156.20811	1400.3841	643.9823
8 periods	0.9985643	345.8053	94.91083	677.6218	340.8236
8 periods w/ quarters	0.9987376	323.6768	94.07378	633.8951	332.0945

1 quarter model



8 period model, by quarter

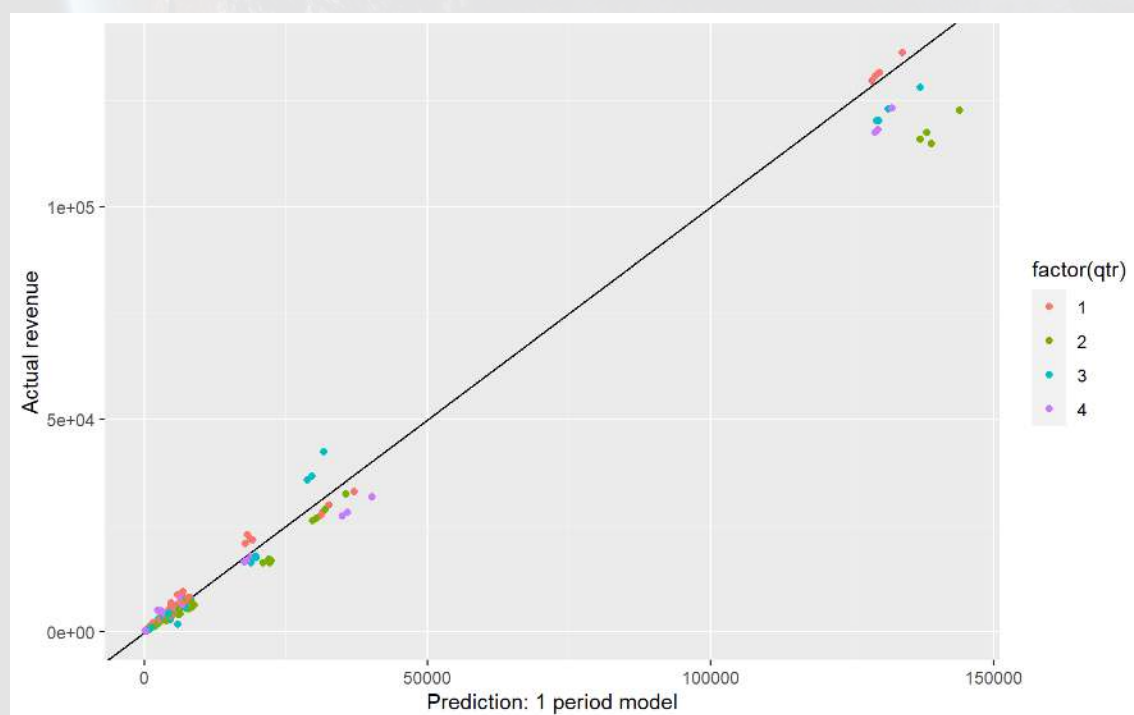


What about for revenue growth?

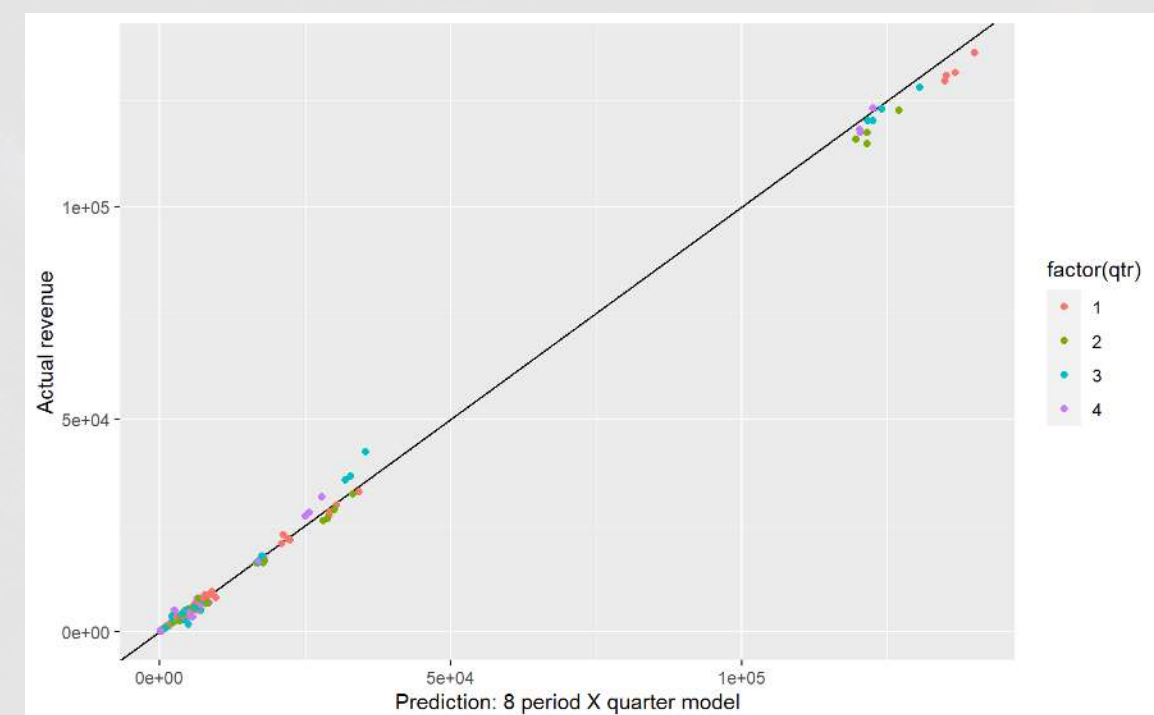
$$rev_t = (1 + growth_t) \times rev_{t-1}$$

	adj_r_sq	rmse_in	mae_in	rmse_out	mae_out
1 period	0.0910390	1106.3730	308.4833	3374.728	1397.6541
1 and 4 periods	0.4398456	530.6444	154.1509	1447.035	679.3536
8 periods	0.6761666	456.2551	123.3407	1254.201	584.9709
8 periods w/ quarters	0.7547897	423.7594	113.6537	1169.282	537.2325

1 quarter model



8 period model, by quarter

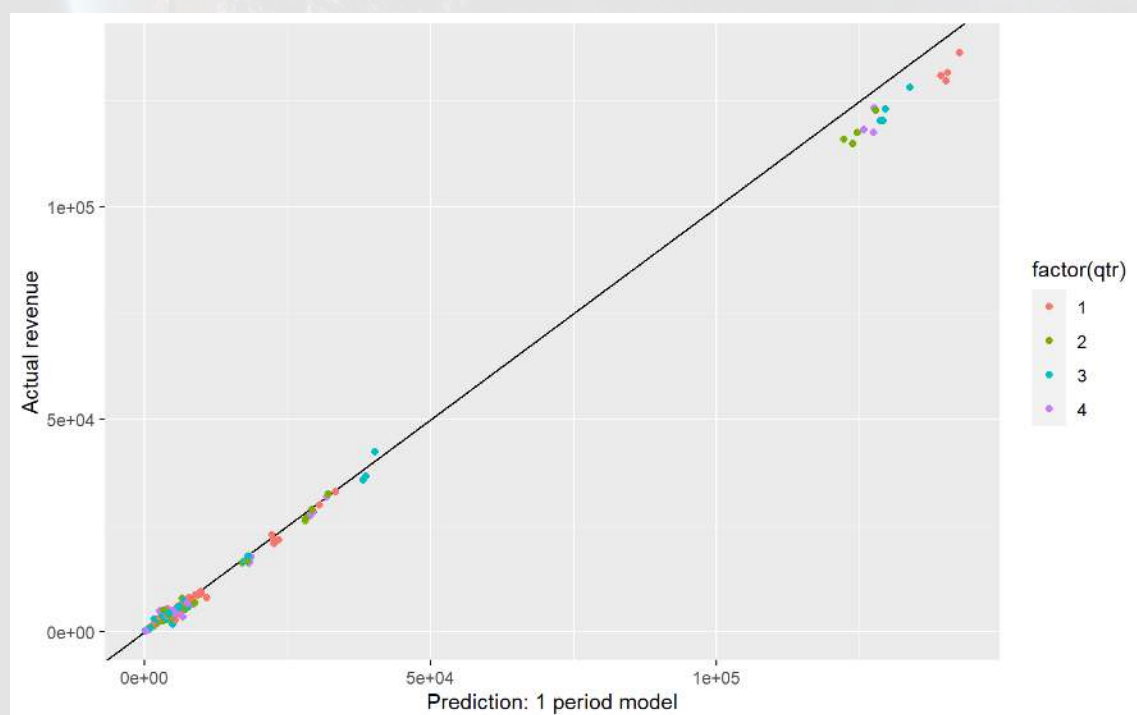


What about for YoY revenue growth?

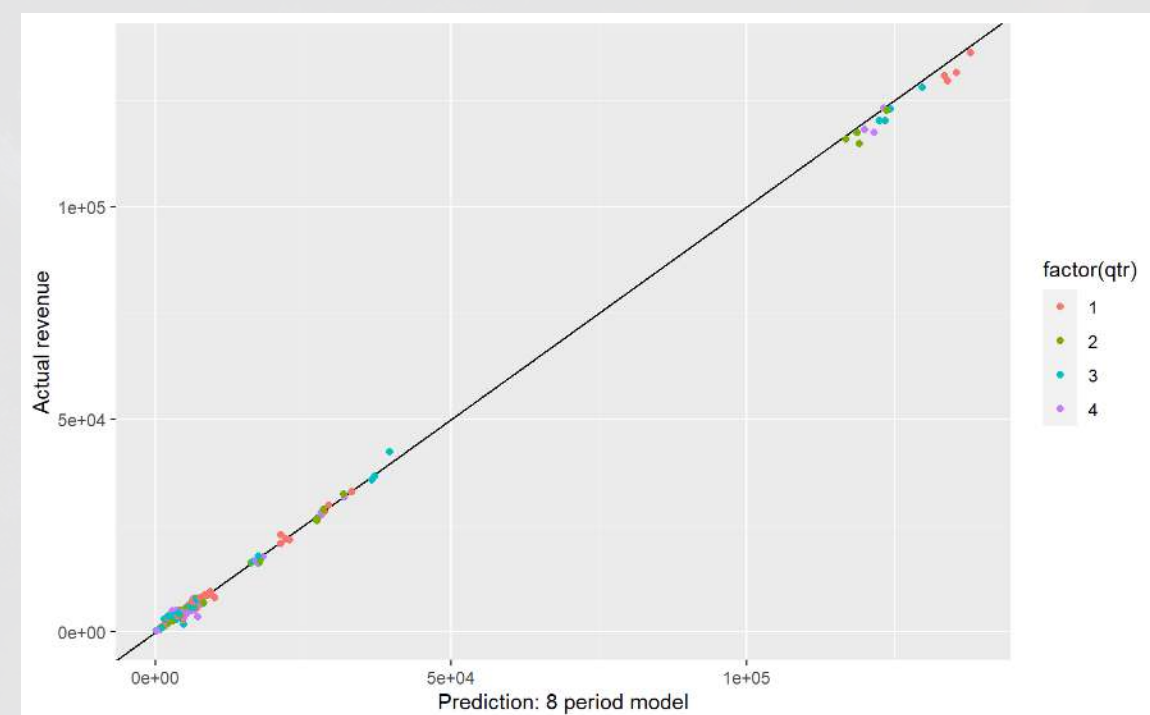
$$rev_t = (1 + yoy_growth_t) \times rev_{t-4}$$

	adj_r_sq	rmse_in	mae_in	rmse_out	mae_out
1 period	0.4370372	513.3264	129.2309	1867.4957	798.0327
1 and 4 periods	0.5392281	487.6441	126.6012	1677.4003	731.2841
8 periods	0.5398870	384.2923	101.0104	822.0065	403.5445
8 periods w/ quarters	0.1040702	679.9093	187.4486	1330.7890	658.4296

1 quarter model



8 period model

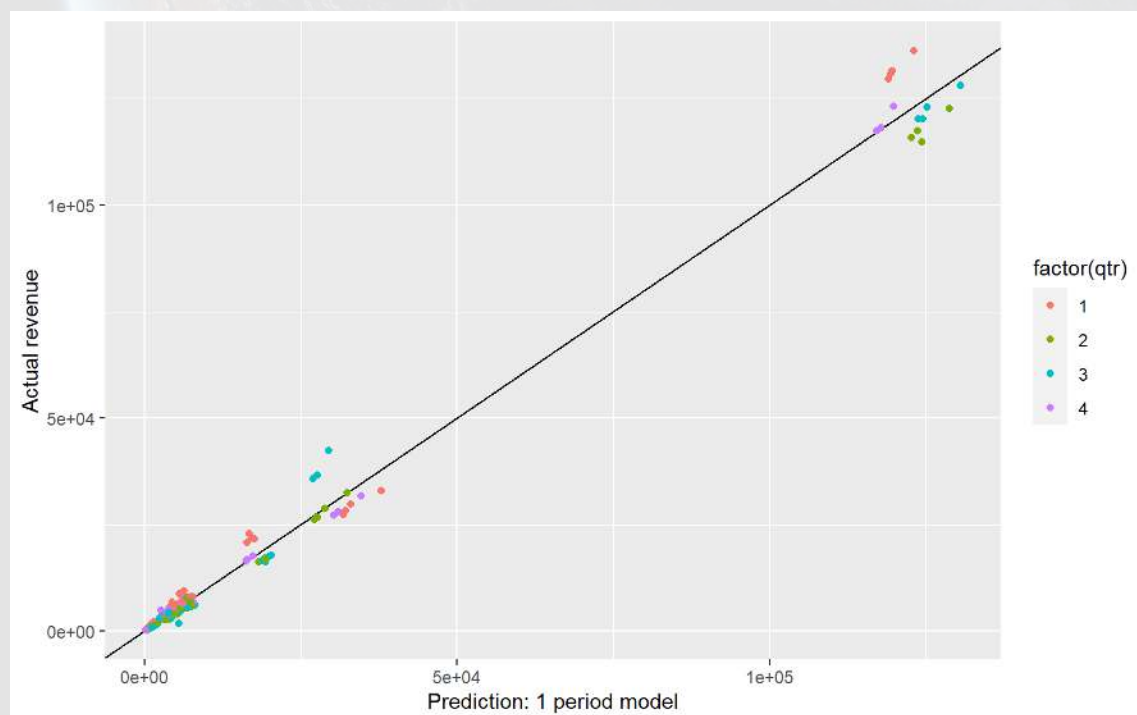


What about for first difference?

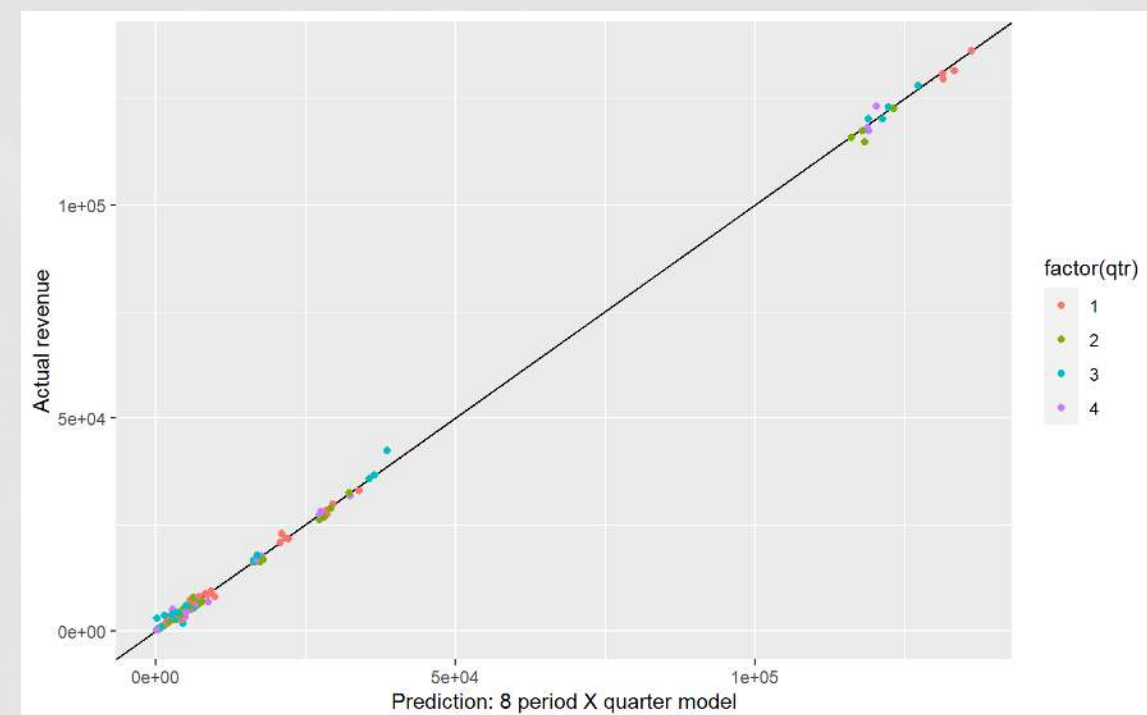
$$rev_t = change_t + rev_{t-1}$$

	adj_r_sq	rmse_in	mae_in	rmse_out	mae_out
1 period	0.3532044	896.7969	287.77940	2252.7605	1022.0960
1 and 4 periods	0.8425348	454.8651	115.52694	734.8120	377.5281
8 periods	0.9220849	333.0054	95.95924	651.4967	320.0567
8 periods w/ quarters	0.9312580	312.2140	88.24559	661.4063	331.0617

1 quarter model



8 period model, by quarter



Takeaways

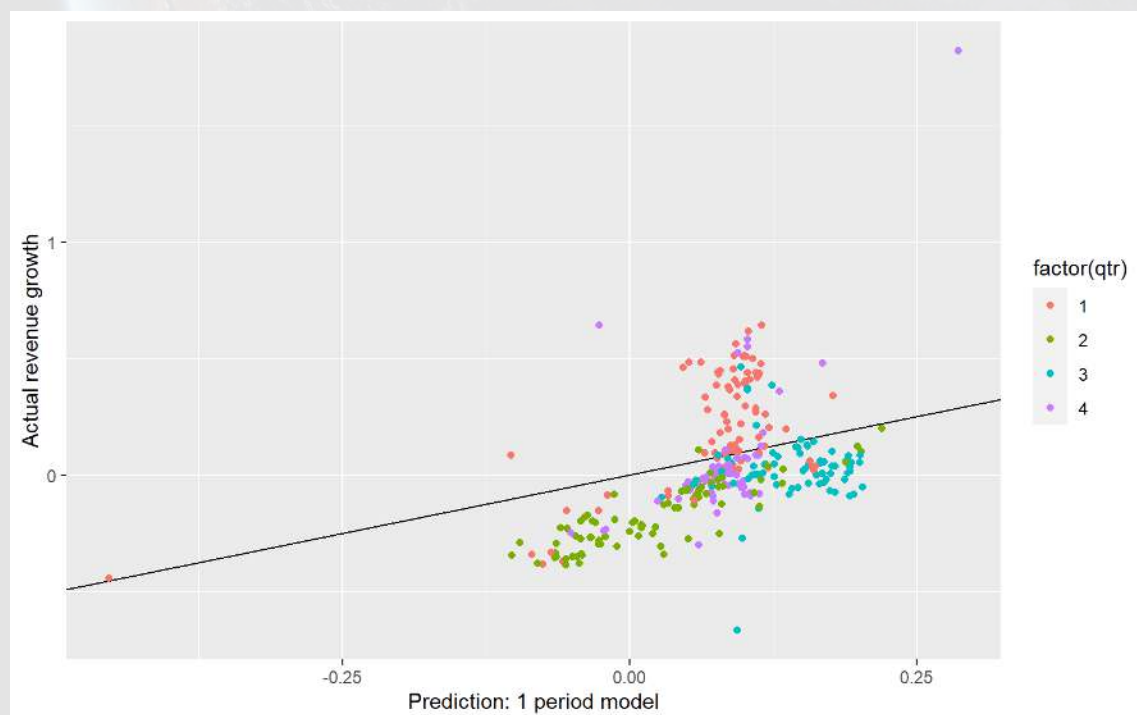
1. The first difference model works about as well as the revenue model at predicting next quarter revenue
 - From earlier, it doesn't suffer (as much) from multicollinearity either
 - This is why time series analysis is often done on first differences
 - Or second differences (difference in differences)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

What about for revenue growth?

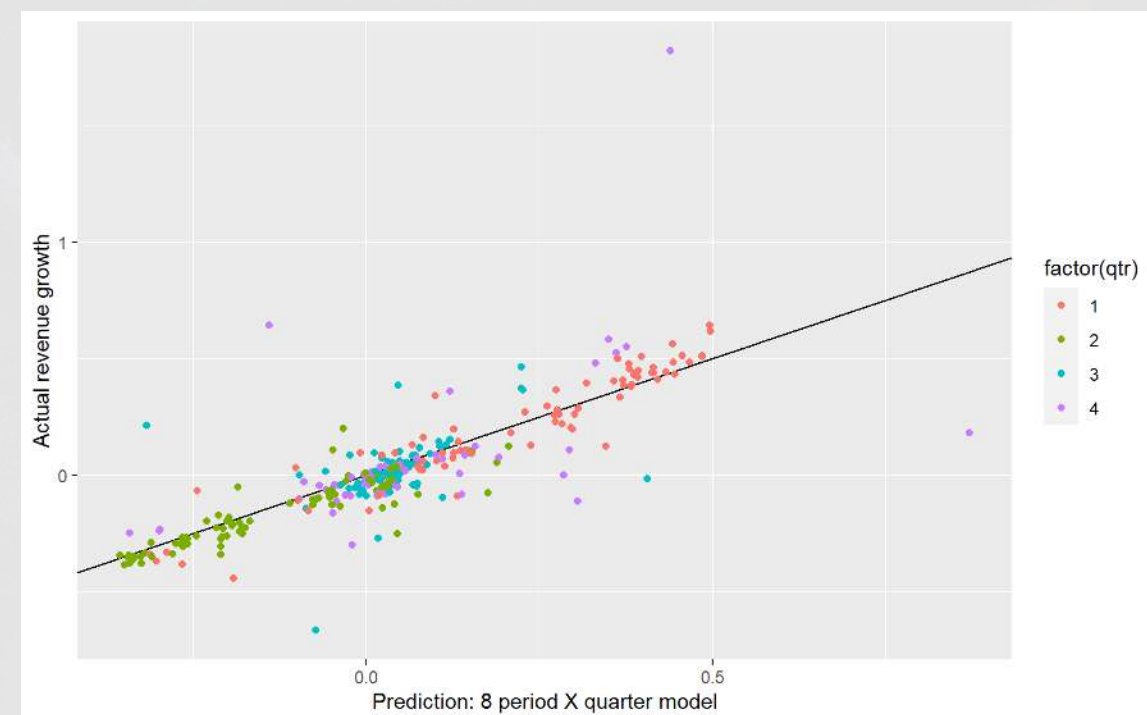
Predicting quarter over quarter revenue growth itself

	adj_r_sq	rmse_in	mae_in	rmse_out	mae_out
1 period	0.0910390	0.3509269	0.2105219	0.2257396	0.1750580
1 and 4 periods	0.4398456	0.2681899	0.1132003	0.1597771	0.0998087
8 periods	0.6761666	0.1761825	0.0867347	0.1545298	0.0845826
8 periods w/ quarters	0.7547897	0.1530278	0.0816612	0.1433094	0.0745658

1 quarter model



8 period model, by quarter

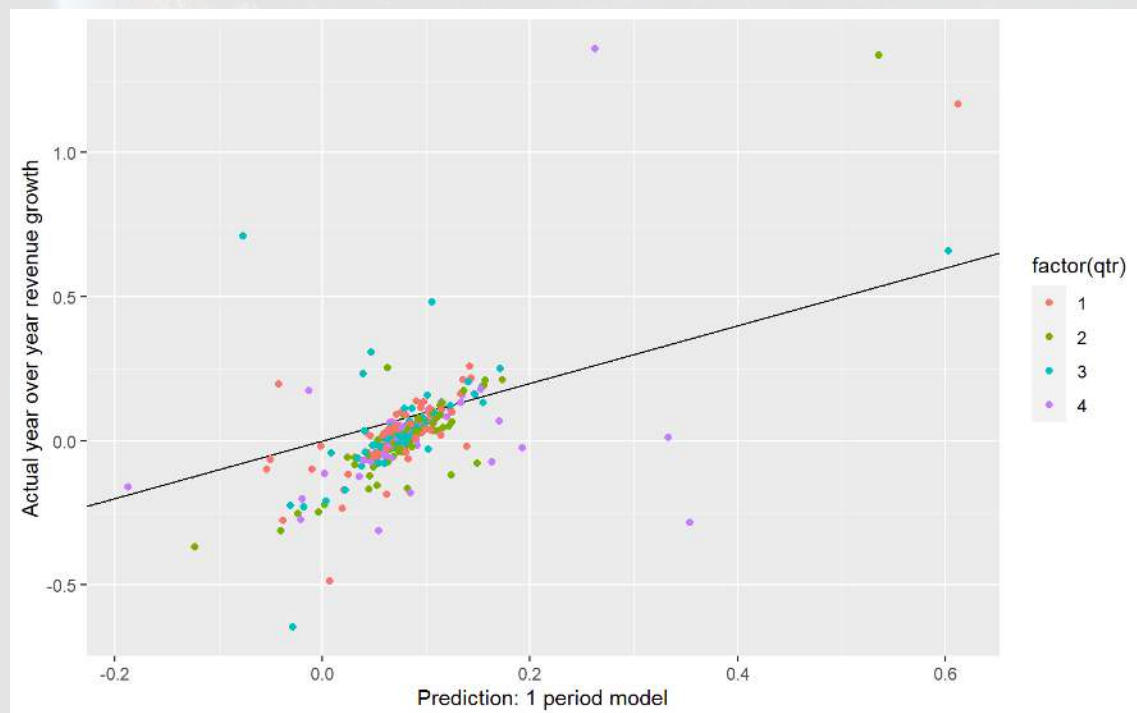


What about for YoY revenue growth?

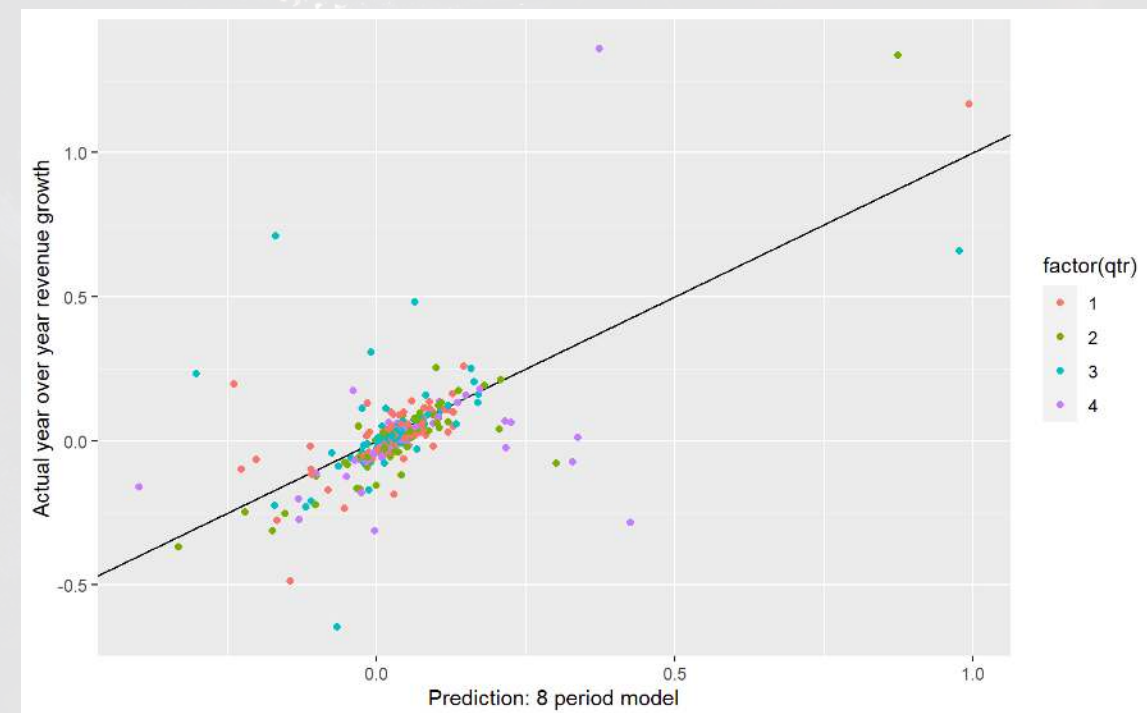
Predicting YoY revenue growth itself

	adj_r_sq	rmse_in	mae_in	rmse_out	mae_out
1 period	0.4370372	0.3116645	0.1114610	0.1515638	0.0942544
1 and 4 periods	0.5392281	0.2451749	0.1015699	0.1498755	0.0896079
8 periods	0.5398870	0.1928940	0.0764447	0.1346238	0.0658011
8 periods w/ quarters	0.1040702	0.2986735	0.1380062	0.1960325	0.1020124

1 quarter model



8 period model

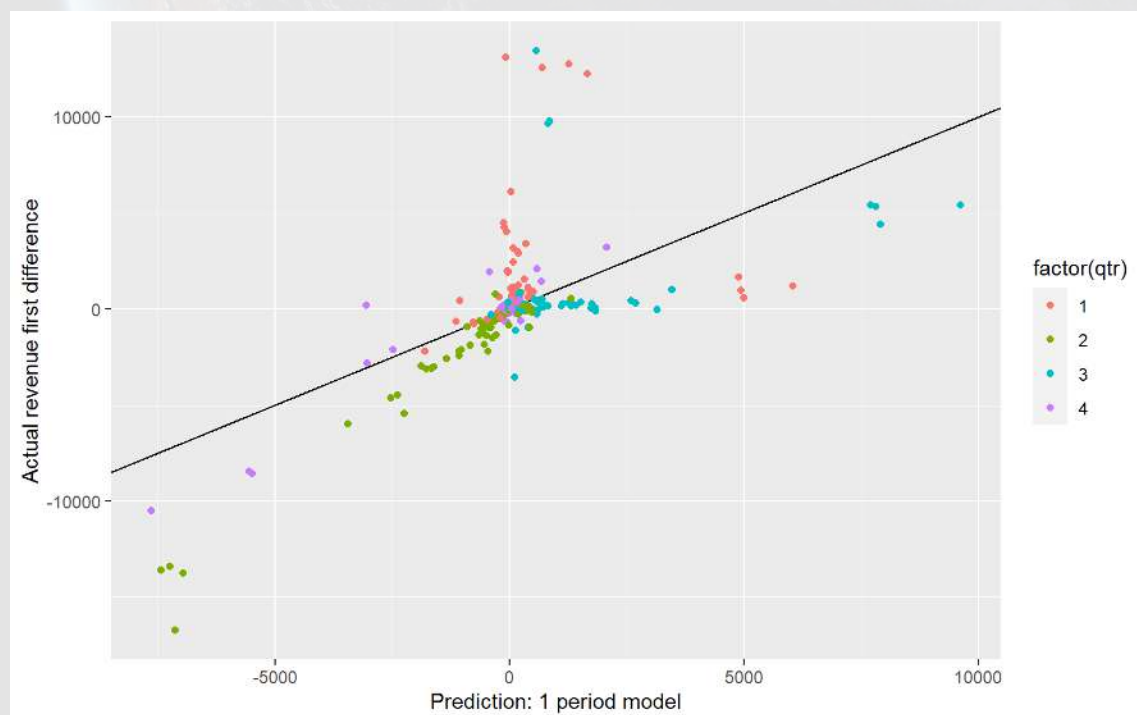


What about for first difference?

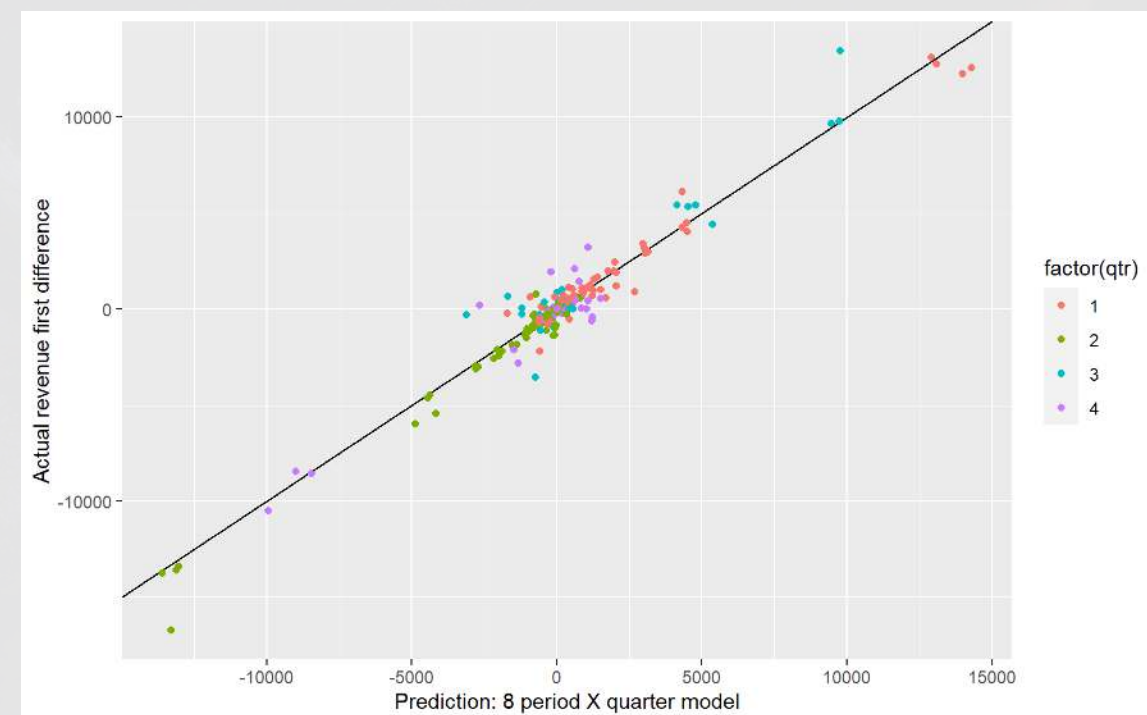
Predicting first difference in revenue itself

	adj_r_sq	rmse_in	mae_in	rmse_out	mae_out
1 period	0.3532044	896.7969	287.77940	2252.7605	1022.0960
1 and 4 periods	0.8425348	454.8651	115.52694	734.8120	377.5281
8 periods	0.9220849	333.0054	95.95924	651.4967	320.0567
8 periods w/ quarters	0.9312580	312.2140	88.24559	661.4063	331.0617

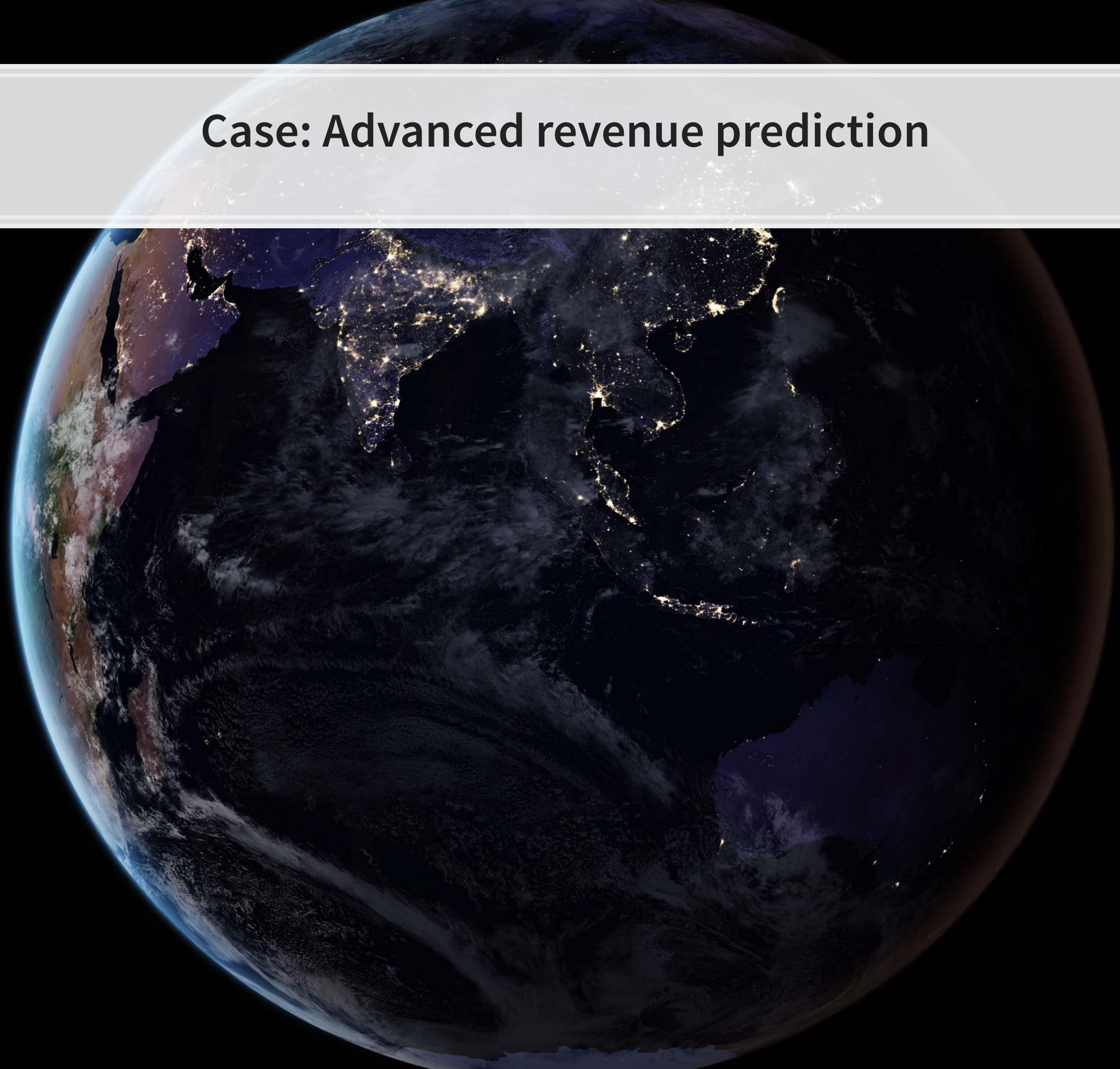
1 quarter model



8 period model, by quarter



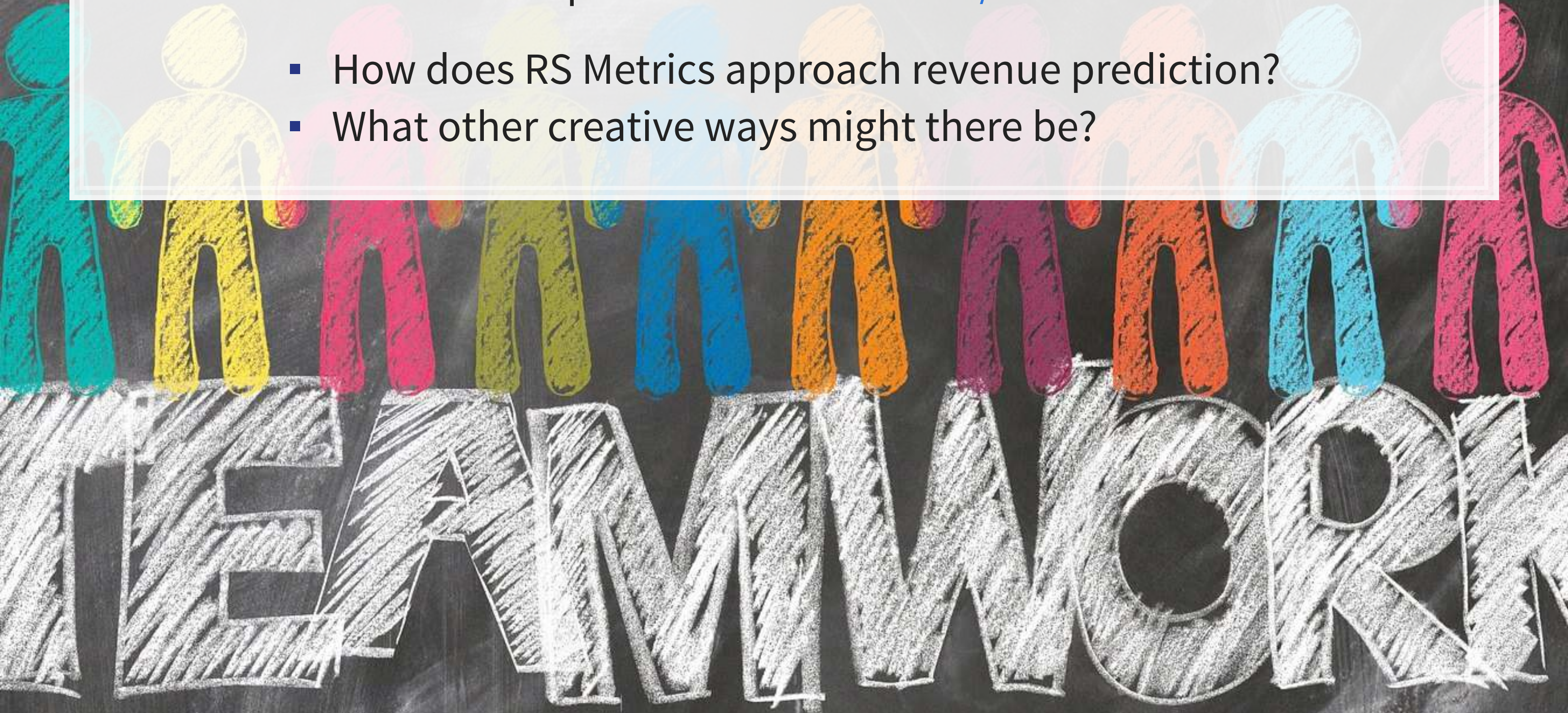
Case: Advanced revenue prediction



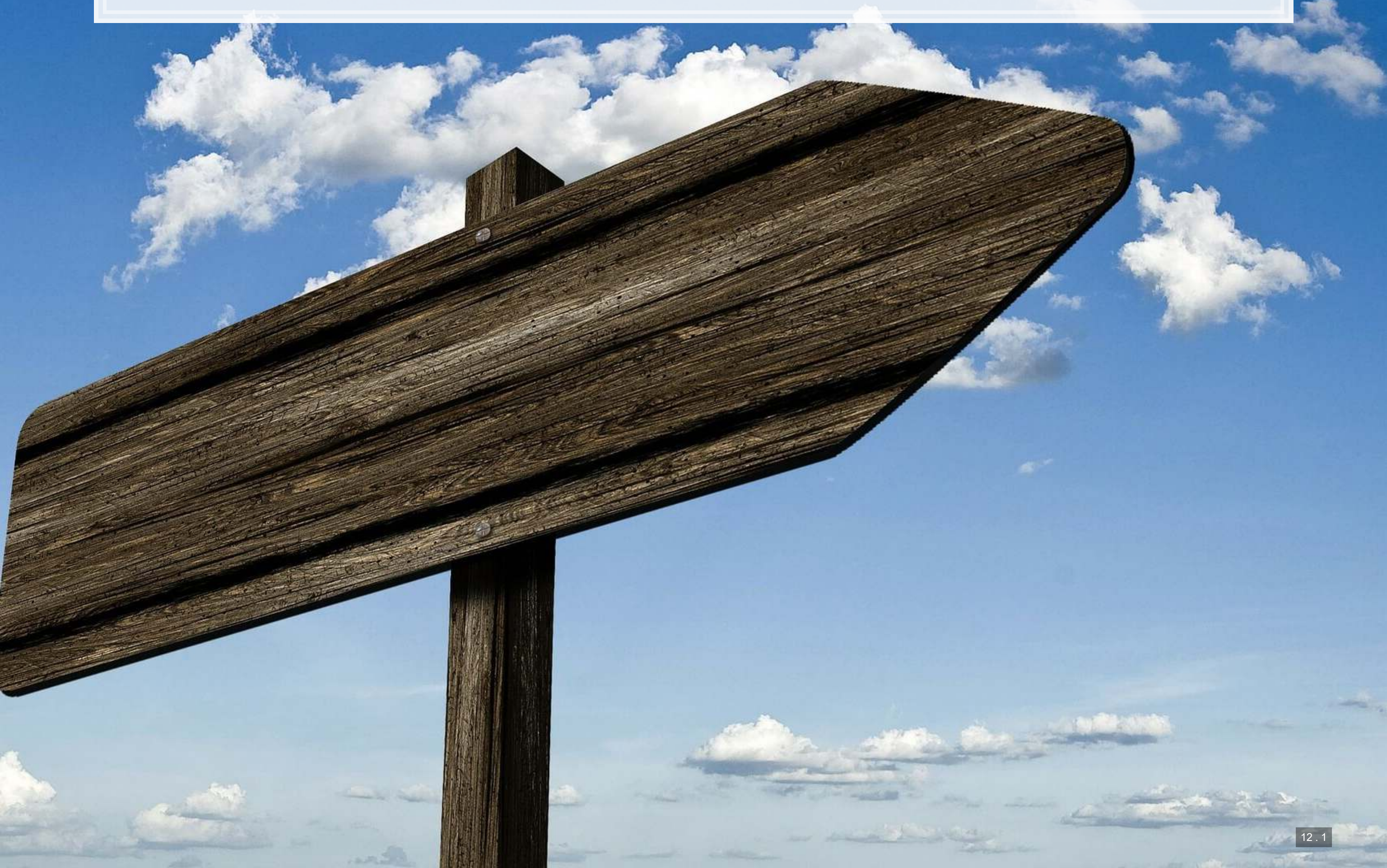
RS Metrics' approach

Read the press release: rmc.link/420class3

- How does RS Metrics approach revenue prediction?
- What other creative ways might there be?



End matter



For next week

- For next week:
 - First individual assignment
 - Finish by next class
 - Submit on eLearn
 - Datacamp
 - Practice a bit more to keep up to date
 - Using R more will make it more natural

Packages used for these slides

- `kableExtra`
- `knitr`
- `lubridate`
- `magrittr`
- `revealjs`
- `tidyverse`

Custom code

Custom code

```
# These functions are a bit ugly, but can construct many charts quickly  
# eval(parse(text=var)) is just a way to convert the string name to a variable reference  
# Density plot for 1st to 99th percentile of data  
plt_dist <- function(df,var) {  
  df %>%  
    filter(eval(parse(text=var)) < quantile(eval(parse(text=var)),0.99, na.rm=TRUE),  
           eval(parse(text=var)) > quantile(eval(parse(text=var)),0.01, na.rm=TRUE)) %>%  
    ggplot(aes(x=eval(parse(text=var)))) +  
    geom_density() + xlab(var)  
}
```

```
# Density plot for 1st to 99th percentile of both columns  
plt_bar <- function(df,var) {  
  df %>%  
    filter(eval(parse(text=var)) < quantile(eval(parse(text=var)),0.99, na.rm=TRUE),  
           eval(parse(text=var)) > quantile(eval(parse(text=var)),0.01, na.rm=TRUE)) %>%  
    ggplot(aes(y=eval(parse(text=var)), x=qtr)) +  
    geom_bar(stat = "summary", fun.y = "mean") + xlab(var)  
}
```


Custom code

```
# Scatter plot with lag for 1st to 99th percentile of data
plt_sct <- function(df,var1, var2) {
  df %>%
    filter(eval(parse(text=var1)) < quantile(eval(parse(text=var1)),0.99, na.rm=TRUE),
           eval(parse(text=var2)) < quantile(eval(parse(text=var2)),0.99, na.rm=TRUE),
           eval(parse(text=var1)) > quantile(eval(parse(text=var1)),0.01, na.rm=TRUE),
           eval(parse(text=var2)) > quantile(eval(parse(text=var2)),0.01, na.rm=TRUE)) %>%
    ggplot(aes(y=eval(parse(text=var1)), x=eval(parse(text=var2)), color=factor(qtr))) +
    geom_point() + geom_smooth(method = "lm") + ylab(var1) + xlab(var2)
}
```

```
# Calculating various in and out of sample statistics
models <- list(mod1,mod2,mod3,mod4)
model_names <- c("1 period", "1 and 4 period", "8 periods", "8 periods w/ quarters")

df_test <- data.frame(adj_r_sq=sapply(models, function(x)summary(x)[["adj.r.squared"]]),
                      rmse_in=sapply(models, function(x)rmse(train$revtq, predict(x,train))),
                      mae_in=sapply(models, function(x)mae(train$revtq, predict(x,train))),
                      rmse_out=sapply(models, function(x)rmse(test$revtq, predict(x,test))),
                      mae_out=sapply(models, function(x)mae(test$revtq, predict(x,test))))

rownames(df_test) <- model_names
html_df(df_test) # Custom function using knitr and kableExtra
```