Recitation 9 Notes: Time Series

Matthew Alampay Davis

December 12, 2021

Cleaning and manipulating data for time series

Problem Set 8 uses data from the Federal Reserve and we are tasked with cleaning it up a certain way so we can use it for time series regressions. I did the problem set in both R and Stata to prep for this recitation and it's an absolute nightmare to do in Stata (I believe they have you use Excel as an intermediate step because Stata is unintuitive for beginners in this regard).

Fortunately, R is excellent at this. What's more, R also has a package called *Quandl* that lets you download data directly from the Federal Reserve website so do install the *Quandl* package to make your life easier. I would just caution that Quandl limits you to 50 uses per day or else you have to make a (free) account and get your own API key.

Downloading data from FRED using Quandl

Once you've installed Quandl, you can very easily use it to download data from FRED, the Federal Reserve database. The problem set tells you the ID for each dataset we'll need (GDPC1 for GDP, M2REAL for M2 money, and GCEC1 for government spending). I'll be using different FRED datasets so that you'll have to apply the lessons here to the new ones.

Let's download GDP data, which has the ID "GDP", the consumer price index ("CPIAUCSL"), and the federal funds rate ("FEDFUNDS"), basically an interest rate:

```
Quandl.api_key('xHu2y3xExQ6bGkGqcYEi') # This is my personal API key
gdp <- Quandl('FRED/GDP')
cpi <- Quandl('FRED/CPIAUCSL')
interest <- Quandl('FRED/FEDFUNDS')

head(gdp)
```

```
##         Date    Value
## 1 2021-07-01 23187.04
## 2 2021-04-01 22740.96
## 3 2021-01-01 22038.23
## 4 2020-10-01 21477.60
## 5 2020-07-01 21138.57
## 6 2020-04-01 19477.44
```

```
head(cpi)
```

```
##         Date   Value
## 1 2021-11-01 278.880
## 2 2021-10-01 276.724
## 3 2021-09-01 274.138
## 4 2021-08-01 273.012
## 5 2021-07-01 272.265
## 6 2021-06-01 270.981
```

```
head(interest)
```

```
##         Date Value
## 1 2021-11-01  0.08
## 2 2021-10-01  0.08
## 3 2021-09-01  0.08
## 4 2021-08-01  0.09
## 5 2021-07-01  0.10
## 6 2021-06-01  0.08
```

Let's rename the "Value" columns so we know what they're describing:

```
names(gdp)[2] <- 'gdp'
names(cpi)[2] <- 'cpi'
names(interest)[2] <- 'interest'
```

Converting monthly data to quarterly data

Clearly these are all time series. However, the GDP data is quarterly with observations only given on the first day of every quarter: January 1, April 1, July 1, and October 1. The CPI and interest rate are given on a monthly basis. We will want to make sure the frequencies are the same before merging them so this will entail transforming the monthly data to quarterly data.

CPI and interest rates are not stock variables so we are fine using just the first observations of each quarter as representative of the price index and interest rate for that quarter. By contrast, if it had been the other way around with GDP monthly and PCPI and interest rates quarterly and had we been looking to aggregate GDP to the quarterly level, we would want to add the GDP of all three months in the quarter to arrive at quarterly GDP.

So let's take a subset of the CPI and interest-rate series, keeping only the observations for the months that bein each quarter. To do this, we'll want to use the *lubridate* package so install that if you haven't yet. This package is basically the go-to in R for dealing with dates.

```
cpi %<>% filter(month(Date) %in% c(1,4,7,10))
interest %<>% filter(month(Date) %in% c(1,4,7,10))
```

So a few things going on here. We've used the "<>" pipe a few times now but just in case it hasn't caught on, the "<>" pipe in the first like takes the cpi dataset we loaded above and automatically uses it as the first argument in the *filter* function and assigns it whatever the output of that function is. It's just a shorter version of

```
cpi <- filter(cpi, month(Date) %in% c(1,4,7,10))
```

Next, the "month(Date)" part makes use of the month function in the lubridate package. It says to look at the Date column in the cpi data, and only keep the observations whose months are "in" the vector (1, 4, 7, 10) (i.e. January, April, July, and October), exactly what we want.

Our CPI and interest-rate data is now effectively quarterly.

Merging datasets

Now we want to combine our GDP, CPI, and interest rate data into one dataset that we'll call "merged.data". We want to match them by date, which we can do by using the "merge" function:

```
merged.data <- merge(gdp, cpi, by = 'Date')
head(merged.data)
```

```
##         Date     gdp   cpi
## 1 1947-01-01 243.164 21.48
## 2 1947-04-01 245.968 22.00
## 3 1947-07-01 249.585 22.23
## 4 1947-10-01 259.745 22.91
## 5 1948-01-01 265.742 23.68
## 6 1948-04-01 272.567 23.82
```

So the merge function merges two datasets at a time. The first two arguments are the data objects in the order you want them to appear, and the third argument "by" is the name of the column that both data objects share and that you want to match them by. The result will be the subset of dates that both datasets have in common. If GDP data spanned 1950-1980 and CPI data spanned 1960-1990, then the merged dataset would not include any data from 1950-1959 or 1981-1990.

Of course, we want to also merge in the interest rate data. We can merge the new merged.data object with the cpi object in the exact same way. If we wanted to combine all three datasets in one command we could have run:

```
merged.data <- merge(gdp, cpi, by = 'Date') %>%
  merge(interest, by = 'Date')
head(merged.data)
```

```
##         Date     gdp   cpi interest
## 1 1954-07-01 390.996 26.86     0.80
## 2 1954-10-01 399.734 26.72     0.85
## 3 1955-01-01 413.073 26.77     1.39
## 4 1955-04-01 421.532 26.79     1.43
## 5 1955-07-01 430.221 26.76     1.68
## 6 1955-10-01 437.092 26.82     2.24
```

Transforming time series data: lags and growth rates

So our merge.data dataset is now one dataset comprised of three time series. Now suppose we wanted to create a couple more time series: GDP growth and inflation, which is the growth rate of the price index. Remember, for a variable X_t, we can approximate its growth rate x_t using the following transformation:

$$x_t = \log(X_t) - \log(X_{t-1})$$

So we'll need the log of each variable, its one-period lag, and to take their differences. Let's use the familiar *mutate* function:

```
merged.data %<>% mutate(log.y = log(gdp),
                        growth = log.y - lag(log.y))
head(merged.data)
```

```
##         Date     gdp   cpi interest    log.y     growth
## 1 1954-07-01 390.996 26.86     0.80 5.968697         NA
## 2 1954-10-01 399.734 26.72     0.85 5.990799 0.02210200
## 3 1955-01-01 413.073 26.77     1.39 6.023624 0.03282501
## 4 1955-04-01 421.532 26.79     1.43 6.043896 0.02027136
## 5 1955-07-01 430.221 26.76     1.68 6.064299 0.02040334
## 6 1955-10-01 437.092 26.82     2.24 6.080144 0.01584467
```

Alternatively:

```
merged.data %<>% mutate(growth2 = c(NA, diff(log(gdp))))
head(merged.data)
```

```
##         Date     gdp   cpi interest    log.y     growth    growth2
## 1 1954-07-01 390.996 26.86     0.80 5.968697         NA         NA
## 2 1954-10-01 399.734 26.72     0.85 5.990799 0.02210200 0.02210200
## 3 1955-01-01 413.073 26.77     1.39 6.023624 0.03282501 0.03282501
## 4 1955-04-01 421.532 26.79     1.43 6.043896 0.02027136 0.02027136
## 5 1955-07-01 430.221 26.76     1.68 6.064299 0.02040334 0.02040334
## 6 1955-10-01 437.092 26.82     2.24 6.080144 0.01584467 0.01584467
```

Here, the *diff* function takes first differences. The result has one observation less than the original time series (since we don't have a previous observation to take differences with for the earliest observation we have) so we must include an NA for it to fit into our dataset. We can see they produce the exact same time series.

As another alternative, we might consider creating a lagged variable first. Let's do this to create the inflation series:

```
merged.data %<>% mutate(cpi.l1 = lag(cpi),
                        inflation = 4*(log(cpi)-log(cpi.l1)))
head(merged.data)
```

```
##         Date     gdp   cpi interest    log.y     growth    growth2 cpi.l1
## 1 1954-07-01 390.996 26.86     0.80 5.968697         NA         NA     NA
## 2 1954-10-01 399.734 26.72     0.85 5.990799 0.02210200 0.02210200  26.86
## 3 1955-01-01 413.073 26.77     1.39 6.023624 0.03282501 0.03282501  26.72
## 4 1955-04-01 421.532 26.79     1.43 6.043896 0.02027136 0.02027136  26.77
## 5 1955-07-01 430.221 26.76     1.68 6.064299 0.02040334 0.02040334  26.79
## 6 1955-10-01 437.092 26.82     2.24 6.080144 0.01584467 0.01584467  26.76
##      inflation
## 1           NA
## 2 -0.020903370
## 3  0.007478035
## 4  0.002987304
## 5 -0.004481793
## 6  0.008958570
```

The multiplying by four here converts it to an annualized inflation rate (we could also multiply it by 100 to get it in terms of percentages)

Finally, we may also just want to produce lagged versions of an existing time series or even lagged versions of a differenced time series for distributed lag models for example. Let's do this for the interest rate:

```
merged.data %<>% mutate(di = c(NA, diff(interest)),
                        di1 = lag(di),
                        di2 = lag(di, 2),
                        di3 = lag(di, 3),
                        di4 = lag(di, 4),
                        i5 = lag(interest, 5))
head(merged.data)
```

```
##         Date     gdp   cpi interest    log.y     growth    growth2 cpi.l1
## 1 1954-07-01 390.996 26.86     0.80 5.968697         NA         NA     NA
## 2 1954-10-01 399.734 26.72     0.85 5.990799 0.02210200 0.02210200  26.86
## 3 1955-01-01 413.073 26.77     1.39 6.023624 0.03282501 0.03282501  26.72
## 4 1955-04-01 421.532 26.79     1.43 6.043896 0.02027136 0.02027136  26.77
## 5 1955-07-01 430.221 26.76     1.68 6.064299 0.02040334 0.02040334  26.79
## 6 1955-10-01 437.092 26.82     2.24 6.080144 0.01584467 0.01584467  26.76
##      inflation   di  di1  di2  di3  di4  i5
## 1           NA   NA   NA   NA   NA   NA  NA
## 2 -0.020903370 0.05   NA   NA   NA   NA  NA
## 3  0.007478035 0.54 0.05   NA   NA   NA  NA
## 4  0.002987304 0.04 0.54 0.05   NA   NA  NA
## 5 -0.004481793 0.25 0.04 0.54 0.05   NA  NA
## 6  0.008958570 0.56 0.25 0.04 0.54 0.05 0.8
```

Notice that we're taking the first four lags of the differenced interest rate but the last lag is for the non-differenced interest rate. These are the kinds of transformations you might make in a distributed lag model to estimate dynamic causal effects.

Finally, suppose we're only interested in observations from 1960 to 1989:

```
merged.data %<>% filter(Date >= '1960-01-01' & Date < '1990-01-01')
```

Time series regressions with HAC standard errors

These are very simple in R. All we need is the standard "lm" function, and the "NeweyWest" function from the *sandwich* package. Suppose we wanted to run a model like the following:

$$Growth_t = \alpha + \beta_1 Inflation_t + \beta_2 Inflation_{t-1} + \gamma_1 \Delta Interest_t + \gamma_2 \Delta Interest_{t-1} + \gamma_3 \Delta Interest_{t-2} + u_t$$

In particular, we are worried about serial correlation in the error term so we want to use Newey-West standard errors. First, we need to compute the lag we want to use according to the truncation rule of thumb in the textbook:

```
m <- 0.75*nrow(merged.data)^(1/3)
m
```

```
## [1] 3.699318
```

```
m <- ceiling(m) # Round up
```

```
model <- lm(growth ~ inflation + lag(inflation) + di + lag(di) + lag(di, 2), merged.data)
hac.se <- NeweyWest(model, lag = m, prewhite = F, adjust = T)
coeftest(model, vcov = hac.se)
```

```
##
## t test of coefficients:
##
##                   Estimate  Std. Error t value  Pr(>|t|)
## (Intercept)     0.01631544  0.00162050 10.0682 < 2.2e-16 ***
## inflation       0.09730787  0.02494895  3.9003 0.0001643 ***
## lag(inflation) -0.02097267  0.03373591 -0.6217 0.5354209
## di              0.00119083  0.00086842  1.3713 0.1730381
## lag(di)        -0.00058628  0.00054338 -1.0789 0.2829323
## lag(di, 2)     -0.00113766  0.00052097 -2.1837 0.0310683 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

It is important to use these exact arguments for prewhite and adjust in the NeweyWest function so that our answers are the same as what would come out of Stata

Empirical Exercise 16.1

Note that the textbook leads you to the data hosted on the official textbook website. This data does not match what's in the textbook or the solutions! It doesn't even have the variables you need to answer any of these subquestions. I had to find the original dataset on an unofficial website so download it from my recitation folder instead:

```
macro <- read_xlsx('UsMacro_Monthly.xlsx')
head(macro)
```

```
## # A tibble: 6 x 6
##    Year Month    IP   Oil   CPI  PCED
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1  1947     1  13.6    NA  21.5    NA
## 2  1947     2  13.7    NA  21.6    NA
## 3  1947     3  13.7    NA  22      NA
## 4  1947     4  13.6    NA  22      NA
## 5  1947     5  13.7    NA  22.0    NA
## 6  1947     6  13.7    NA  22.1    NA
```

Part a: Compute the monthly growth rate in IP, expressed in percentage points. What are the mean and standard deviation of IP growth over the 1960:M1–2017:M12 sample period? What are the units for IP growth (percent, percent per annum, percent per month, or something else)?

```
macro %<>% mutate(ip.growth = 100*log(IP/lag(IP)))
macro.sub <- filter(macro, Year >= 1960 & Year <= 2017)
mean(macro.sub$ip.growth)
```

```
## [1] 0.2235496
```

```
sd(macro.sub$ip.growth)
```

```
## [1] 0.7781159
```

So we have a mean of about 0.22 and a standard deviation of about 0.75. This is slightly different from the provided solutions because they seem to be using the period 1952-2009 for whatever reason (old edition?) rather than 1960-2017.

Part b: Plot the value of O_t. Why are so many values of O_t equal to 0? Why aren't some values of O_t negative?

We want to plot the value of oil. Let's create a Date variable that combines the info in Year and Month first:

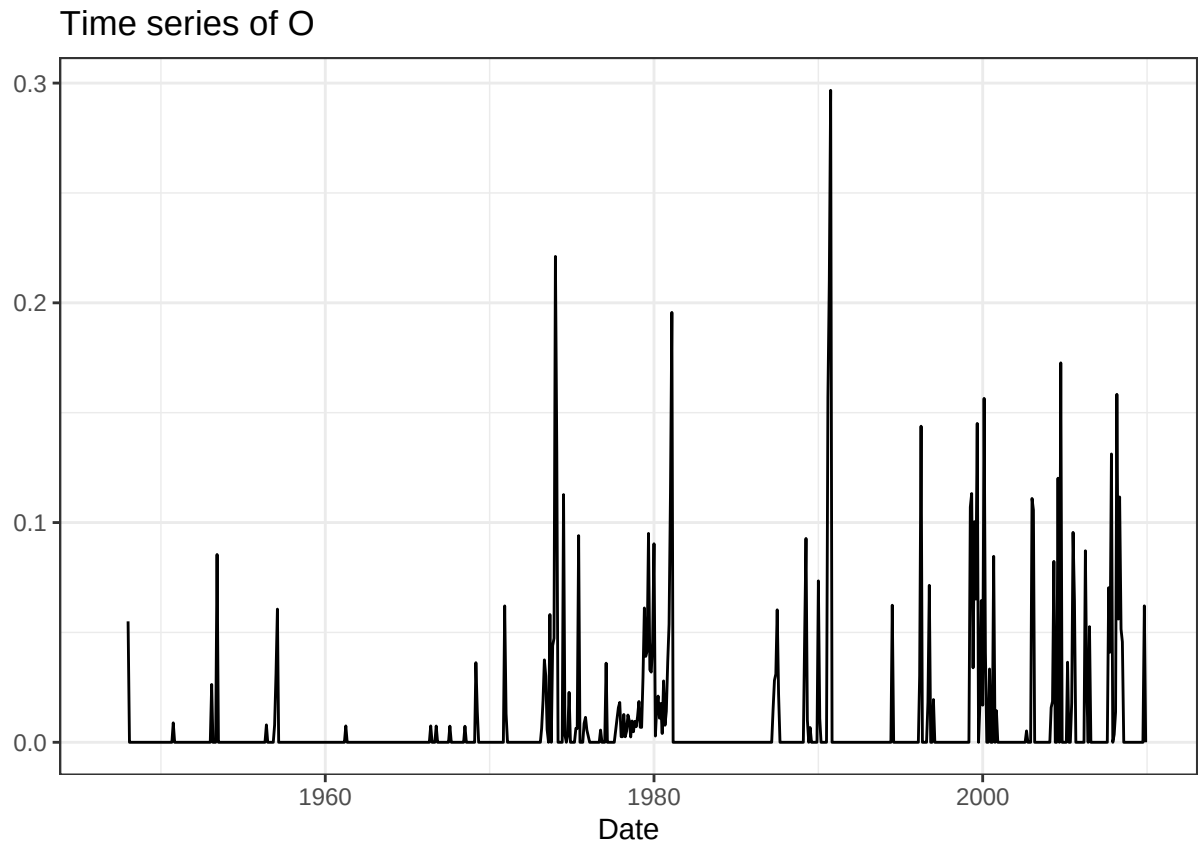
```
macro %<>% mutate(Date = as.Date(ISOdate(Year, Month, 1)))
```

Here, the 1 just tells R to assign it the first day of the month.

Now let's plot the oil time series:

```
ggplot(macro, aes(x = Date, y = Oil)) +
  theme_bw() + # A black-and-white theme I like over the gray default
  geom_line() + # Draw a line graph using Date on the x and Oil on the y
  xlab('Date') + ggtitle('Time series of O') + ylab('')
```

```
## Warning: Removed 12 row(s) containing missing values (geom_path).
```



From Exercise 16.1, O here is defined as "the greater of 0 or the percentage point difference between oil prices at date t and their maximum value during the past three years." Thus it equals 0 whenever the price of oil is less than the maximum during the previous three years.

Part c: Estimate a distributed lag model by regressing IP growth onto the current value and 18 lagged values of O, including an intercept. What value of the HAC standard error truncation parameter did you choose? Why?

We are estimating a distributed lag model of IP growth onto the current and 18 lagged values of O.

First, let's calculate the HAC truncation parameter m:

```
m <-  0.75*nrow(macro)^(1/3)
m <- ceiling(m)
```

Part d: Taken as a group, are the coefficients on O statistically significantly different from 0?

Then the regression:

```
oil.model <- lm(ip.growth ~ Oil + lag(Oil) + lag(Oil, 2) + lag(Oil, 3) + lag(Oil, 4) +
                  lag(Oil, 5) + lag(Oil, 6) + lag(Oil, 7) + lag(Oil, 8) + lag(Oil, 9) +
                  lag(Oil, 10) + lag(Oil, 11) + lag(Oil, 12) + lag(Oil, 13) + lag(Oil, 14) +
                  lag(Oil, 15) + lag(Oil, 16) + lag(Oil, 17) + lag(Oil, 18),
```

```
               data = macro)
hac.se <- NeweyWest(oil.model, lag = m, prewhite = F, adjust = T)
coeftest(oil.model, vcov = hac.se)
```

```
##
## t test of coefficients:
##
##                Estimate Std. Error t value  Pr(>|t|)
## (Intercept)   0.4275475  0.0674593  6.3379 4.153e-10 ***
## Oil           0.1553167  0.8532205  0.1820   0.85561
## lag(Oil)     -0.9760037  0.9558976 -1.0210   0.30759
## lag(Oil, 2)  -1.4030477  0.7918010 -1.7720   0.07683 .
## lag(Oil, 3)  -0.8315475  0.9295158 -0.8946   0.37130
## lag(Oil, 4)  -0.4064382  0.8823558 -0.4606   0.64521
## lag(Oil, 5)  -0.4201911  0.7929348 -0.5299   0.59633
## lag(Oil, 6)  -2.5550729  1.4715218 -1.7363   0.08294 .
## lag(Oil, 7)  -0.2285942  0.9797307 -0.2333   0.81558
## lag(Oil, 8)   0.8799744  1.0083214  0.8727   0.38312
## lag(Oil, 9)  -1.6392365  1.0284963 -1.5938   0.11142
## lag(Oil, 10) -3.9233937  1.8721357 -2.0957   0.03647 *
## lag(Oil, 11) -2.6074098  1.9539566 -1.3344   0.18249
## lag(Oil, 12) -0.2247408  1.2920913 -0.1739   0.86197
## lag(Oil, 13) -1.5546828  1.1310431 -1.3746   0.16971
## lag(Oil, 14) -1.4831538  0.9119501 -1.6264   0.10432
## lag(Oil, 15) -1.4787743  0.8443107 -1.7515   0.08030 .
## lag(Oil, 16) -0.1002288  0.9036673 -0.1109   0.91172
## lag(Oil, 17)  0.4980770  0.7492743  0.6647   0.50643
## lag(Oil, 18)  0.0096065  0.9699462  0.0099   0.99210
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
coefci(oil.model, vcov = hac.se)
```

```
##                    2.5 %     97.5 %
## (Intercept)   0.2951025  0.5599924
## Oil          -1.5198367  1.8304700
## lag(Oil)     -2.8527459  0.9007386
## lag(Oil, 2)  -2.9576141  0.1515188
## lag(Oil, 3)  -2.6564935  0.9933985
## lag(Oil, 4)  -2.1387937  1.3259173
## lag(Oil, 5)  -1.9769836  1.1366014
## lag(Oil, 6)  -5.4441556  0.3340098
## lag(Oil, 7)  -2.1521288  1.6949403
## lag(Oil, 8)  -1.0996930  2.8596418
## lag(Oil, 9)  -3.6585139  0.3800409
## lag(Oil, 10) -7.5990134 -0.2477739
## lag(Oil, 11) -6.4436710  1.2288515
## lag(Oil, 12) -2.7615422  2.3120606
## lag(Oil, 13) -3.7752934  0.6659279
## lag(Oil, 14) -3.2736127  0.3073050
## lag(Oil, 15) -3.1364347  0.1788861
## lag(Oil, 16) -1.8744258  1.6739682
## lag(Oil, 17) -0.9729956  1.9691497
```

```
## lag(Oil, 18) -1.8947179  1.9139309
```

Since the formula for thie regression is quite long, it's helpful to press enter after every few regressors so it's visible in the output

F test on all oil coefficients:

```
# Get the names of all the oil coefficients
oil.names <- names(coef(oil.model))[-1] # All but the intercept

linearHypothesis(oil.model,
                 oil.names,
                 test = 'F',
                 vcov = hac.se)
```

```
## Linear hypothesis test
##
## Hypothesis:
## Oil = 0
## lag(Oil) = 0
## lag(Oil, 2) = 0
## lag(Oil, 3) = 0
## lag(Oil, 4) = 0
## lag(Oil, 5) = 0
## lag(Oil, 6) = 0
## lag(Oil, 7) = 0
## lag(Oil, 8) = 0
## lag(Oil, 9) = 0
## lag(Oil, 10) = 0
## lag(Oil, 11) = 0
## lag(Oil, 12) = 0
## lag(Oil, 13) = 0
## lag(Oil, 14) = 0
## lag(Oil, 15) = 0
## lag(Oil, 16) = 0
## lag(Oil, 17) = 0
## lag(Oil, 18) = 0
##
## Model 1: restricted model
## Model 2: ip.growth ~ Oil + lag(Oil) + lag(Oil, 2) + lag(Oil, 3) + lag(Oil,
##     4) + lag(Oil, 5) + lag(Oil, 6) + lag(Oil, 7) + lag(Oil, 8) +
##     lag(Oil, 9) + lag(Oil, 10) + lag(Oil, 11) + lag(Oil, 12) +
##     lag(Oil, 13) + lag(Oil, 14) + lag(Oil, 15) + lag(Oil, 16) +
##     lag(Oil, 17) + lag(Oil, 18)
##
## Note: Coefficient covariance matrix supplied.
##
##   Res.Df Df      F  Pr(>F)
## 1    725
## 2    706 19 1.7399 0.02607 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The F-statistic is 1.74, corresponding to a p-value of 0.02607. This means the joint hypothesis cannot be

rejected at a 1% level but can be rejected at a 5% level. (Again, this is not how I like to summarize hypothesis tests, but that's how we're taught it in this course)

Part e: Construct graphs like those in Figure 16.2, showing the estimated dynamic multipliers, cumulative multipliers, and 95% confidence intervals. Comment on the real-world size of the multipliers.

The above model captured non-cumulative effects. Let's collect the coefficient estimates:

```
# Get the coefficients
dyna.coef <- coef(oil.model)[-1] # Take out the intercept coefficient
# Get the confidence intervals for each coefficient
dyna.ci <- coefci(oil.model, vcov = hac.se)[-1,] # Take out the first row for the intercept coefficient

dyna.multipliers <- data.frame(lags = 0:18,
                               coef = dyna.coef,
                               low = dyna.ci[,1],
                               high = dyna.ci[,2])
dyna.multipliers
```

```
##              lags       coef        low       high
## Oil             0  0.1553167 -1.5198367  1.8304700
## lag(Oil)        1 -0.9760037 -2.8527459  0.9007386
## lag(Oil, 2)     2 -1.4030477 -2.9576141  0.1515188
## lag(Oil, 3)     3 -0.8315475 -2.6564935  0.9933985
## lag(Oil, 4)     4 -0.4064382 -2.1387937  1.3259173
## lag(Oil, 5)     5 -0.4201911 -1.9769836  1.1366014
## lag(Oil, 6)     6 -2.5550729 -5.4441556  0.3340098
## lag(Oil, 7)     7 -0.2285942 -2.1521288  1.6949403
## lag(Oil, 8)     8  0.8799744 -1.0996930  2.8596418
## lag(Oil, 9)     9 -1.6392365 -3.6585139  0.3800409
## lag(Oil, 10)   10 -3.9233937 -7.5990134 -0.2477739
## lag(Oil, 11)   11 -2.6074098 -6.4436710  1.2288515
## lag(Oil, 12)   12 -0.2247408 -2.7615422  2.3120606
## lag(Oil, 13)   13 -1.5546828 -3.7752934  0.6659279
## lag(Oil, 14)   14 -1.4831538 -3.2736127  0.3073050
## lag(Oil, 15)   15 -1.4787743 -3.1364347  0.1788861
## lag(Oil, 16)   16 -0.1002288 -1.8744258  1.6739682
## lag(Oil, 17)   17  0.4980770 -0.9729956  1.9691497
## lag(Oil, 18)   18  0.0096065 -1.8947179  1.9139309
```

Remember, to capture cumulative effects, we would run a similar same model up to *seventeen* lags of *differenced* Oil values and then include the 18th lag of *non-differenced* Oil. So let's first create the differenced oil time series:

```
macro %<>% mutate(d.Oil = c(NA, diff(Oil)))
```

Now run the corresponding regression

```
cumu.model <- lm(ip.growth ~ d.Oil + lag(d.Oil) + lag(d.Oil, 2) + lag(d.Oil, 3) + lag(d.Oil, 4) +
                   lag(d.Oil, 5) + lag(d.Oil, 6) + lag(d.Oil, 7) + lag(d.Oil, 8) + lag(d.Oil, 9) +
```

```
                  lag(d.Oil, 10) + lag(d.Oil, 11) + lag(d.Oil, 12) + lag(d.Oil, 13) + lag(d.Oil, 14) +
                  lag(d.Oil, 15) + lag(d.Oil, 16) + lag(d.Oil, 17) + lag(Oil, 18),
                data = macro)
```

To emphasize again, notice the last term here is the 18th lag of *non-differenced* Oil while all the other oil terms are lagged differenced Oil.

I believe you can also do the last two regressions we've just run very compactly using the dynlm function from the dynlm package (see the econometrics-with-r website for examples), but I am less familar with it so do not know how it interacts with the linearHypothesis function so I left it out for now.

Calculate Newey-West standard errors and display the output:

```
hac.se <- NeweyWest(cumu.model, lag = m, prewhite = F, adjust = T)
coeftest(cumu.model, vcov = hac.se)
```

```
##
## t test of coefficients:
##
##                  Estimate Std. Error t value  Pr(>|t|)
## (Intercept)      0.427547   0.067459  6.3379 4.153e-10 ***
## d.Oil            0.155317   0.853221  0.1820 0.8556069
## lag(d.Oil)      -0.820687   1.150362 -0.7134 0.4758237
## lag(d.Oil, 2)   -2.223735   1.391207 -1.5984 0.1103968
## lag(d.Oil, 3)   -3.055282   1.786075 -1.7106 0.0875920 .
## lag(d.Oil, 4)   -3.461720   1.963932 -1.7626 0.0783925 .
## lag(d.Oil, 5)   -3.881911   2.297537 -1.6896 0.0915466 .
## lag(d.Oil, 6)   -6.436984   2.884135 -2.2319 0.0259373 *
## lag(d.Oil, 7)   -6.665579   2.813145 -2.3694 0.0180830 *
## lag(d.Oil, 8)   -5.785604   2.707564 -2.1368 0.0329549 *
## lag(d.Oil, 9)   -7.424841   2.903326 -2.5574 0.0107550 *
## lag(d.Oil, 10) -11.348234   3.579153 -3.1706 0.0015868 **
## lag(d.Oil, 11) -13.955644   3.917228 -3.5626 0.0003918 ***
## lag(d.Oil, 12) -14.180385   4.258568 -3.3298 0.0009141 ***
## lag(d.Oil, 13) -15.735068   4.472010 -3.5186 0.0004617 ***
## lag(d.Oil, 14) -17.218222   4.682386 -3.6772 0.0002537 ***
## lag(d.Oil, 15) -18.696996   4.653303 -4.0180 6.499e-05 ***
## lag(d.Oil, 16) -18.797225   4.493089 -4.1836 3.231e-05 ***
## lag(d.Oil, 17) -18.299148   4.441045 -4.1205 4.230e-05 ***
## lag(Oil, 18)   -18.289541   4.478197 -4.0841 4.931e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We interpret these coefficients as cumulative multipliers. Let's plot them by first creating a table of these coefficients:

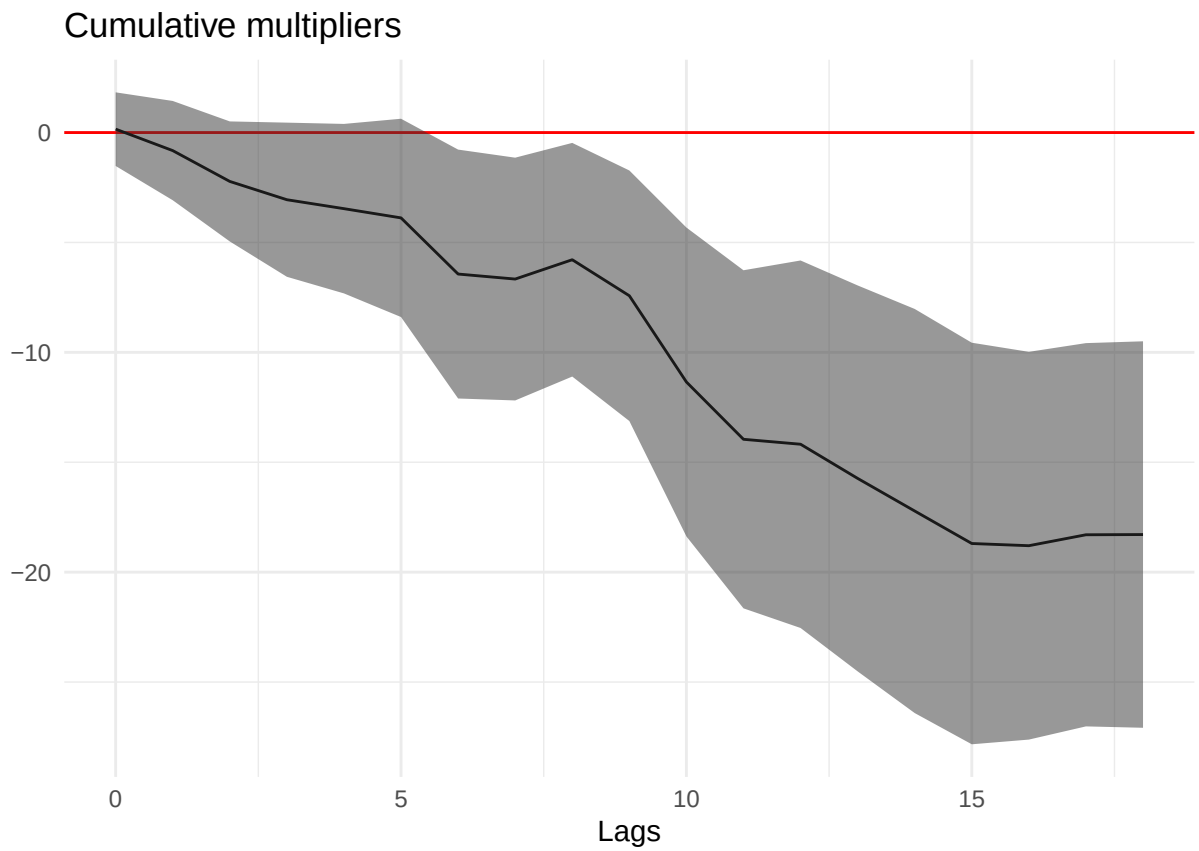
```
# Get the coefficients
cumu.coef <- coef(cumu.model)[-1] # Take out the intercept coefficient

# Get the confidence intervals for each coefficient
cumu.ci <- coefci(cumu.model, vcov = hac.se)[-1,] # Take out the first row for the intercept coefficien

cumu.multipliers <- data.frame(lags = 0:18,
```

```
                              coef = cumu.coef,
                              low = cumu.ci[,1],
                              high = cumu.ci[,2])
cumu.multipliers
```

```
##                lags        coef        low       high
## d.Oil             0   0.1553167  -1.519837  1.8304700
## lag(d.Oil)        1  -0.8206870  -3.079226  1.4378520
## lag(d.Oil, 2)     2  -2.2237347  -4.955134  0.5076641
## lag(d.Oil, 3)     3  -3.0552822  -6.561937  0.4513724
## lag(d.Oil, 4)     4  -3.4617204  -7.317566  0.3941254
## lag(d.Oil, 5)     5  -3.8819115  -8.392734  0.6289109
## lag(d.Oil, 6)     6  -6.4369844 -12.099493 -0.7744761
## lag(d.Oil, 7)     7  -6.6655786 -12.188709 -1.1424478
## lag(d.Oil, 8)     8  -5.7856043 -11.101445 -0.4697632
## lag(d.Oil, 9)     9  -7.4248408 -13.125026 -1.7246550
## lag(d.Oil, 10)   10 -11.3482344 -18.375292 -4.3211767
## lag(d.Oil, 11)   11 -13.9556442 -21.646455 -6.2648334
## lag(d.Oil, 12)   12 -14.1803850 -22.541359 -5.8194110
## lag(d.Oil, 13)   13 -15.7350678 -24.515098 -6.9550376
## lag(d.Oil, 14)   14 -17.2182216 -26.411290 -8.0251534
## lag(d.Oil, 15)   15 -18.6969959 -27.832965 -9.5610268
## lag(d.Oil, 16)   16 -18.7972247 -27.618641 -9.9758086
## lag(d.Oil, 17)   17 -18.2991477 -27.018384 -9.5799114
## lag(Oil, 18)     18 -18.2895412 -27.081719 -9.4973629
```

Now let's plot:

```
ggplot(cumu.multipliers, aes(x = lags)) +
  theme_minimal() + # A minimalistic theme to override the default gray plots
  ggtitle('Cumulative multipliers') +
  xlab('Lags') + ylab('') +
  geom_hline(yintercept = 0, color = 'red') + # optional: draw a line at y = 0
  geom_line(aes(y = coef)) + # Line graph where the y is given by the estimate variable
  geom_ribbon(aes(ymin = low, ymax = high), alpha = 0.5) # Confidence intervals
```

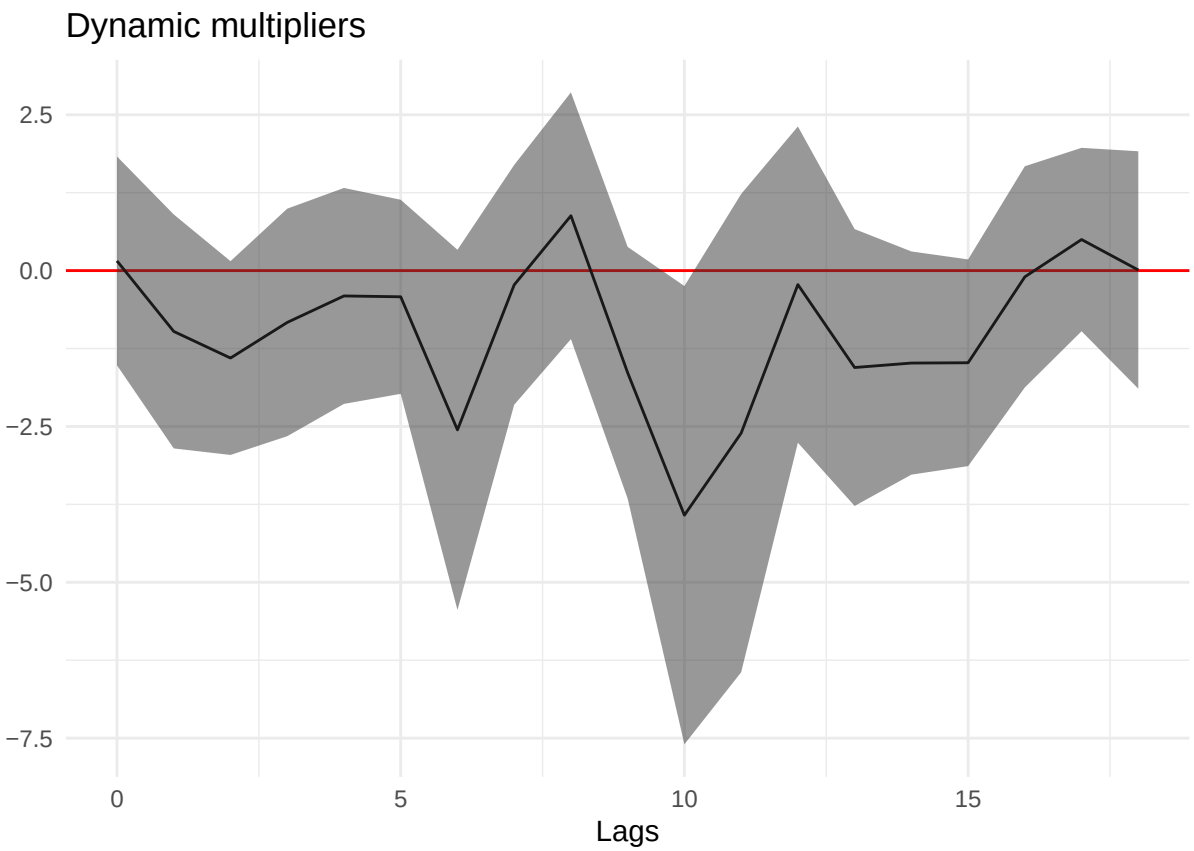


geom_ribbon here allows us to draw confidence intervals by providing the lower and higher ends. Note we want to set alpha between 0 and 1 so that it's transparent and we can see the line graph.

The cumulative multipliers show a persistent and large decrease in industrial production following an increase in oil prices above their previous 12 month peak price. Specifically a 100% increase in oil prices is associated with a roughtly 18% decline in industrial production after 18 months.

Now let's do the dynamic multipliers:

```
ggplot(dyna.multipliers, aes(x = lags)) +
  theme_minimal() + # A minimalistic theme to override the default gray plots
  ggtitle('Dynamic multipliers') +
  xlab('Lags') + ylab('') +
  geom_hline(yintercept = 0, color = 'red') + # optional: draw a line at y = 0
  geom_line(aes(y = coef)) + # Line graph where the y is given by the estimate variable
  geom_ribbon(aes(ymin = low, ymax = high), alpha = 0.5) # Confidence intervals
```



You can also plot the cumulative and dynamic multipliers in the same plot. To do so here, let's combine these into one data.frame

```
dyna.multipliers %<>% mutate(type = 'dynamic')
cumu.multipliers %<>% mutate(type = 'cumulative')
multipliers <- rbind(dyna.multipliers, cumu.multipliers)
multipliers
```

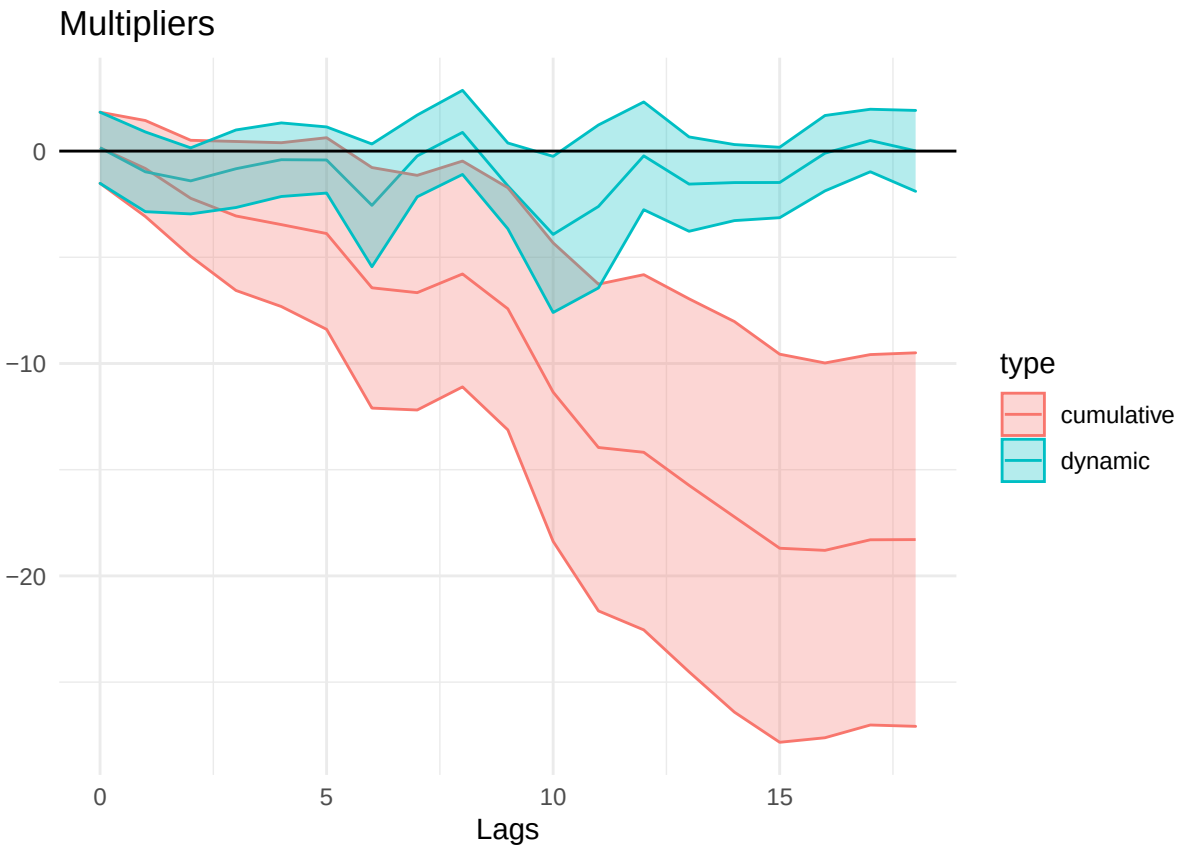
```
##                lags        coef         low       high       type
## Oil               0   0.1553167  -1.5198367  1.8304700    dynamic
## lag(Oil)          1  -0.9760037  -2.8527459  0.9007386    dynamic
## lag(Oil, 2)       2  -1.4030477  -2.9576141  0.1515188    dynamic
## lag(Oil, 3)       3  -0.8315475  -2.6564935  0.9933985    dynamic
## lag(Oil, 4)       4  -0.4064382  -2.1387937  1.3259173    dynamic
## lag(Oil, 5)       5  -0.4201911  -1.9769836  1.1366014    dynamic
## lag(Oil, 6)       6  -2.5550729  -5.4441556  0.3340098    dynamic
## lag(Oil, 7)       7  -0.2285942  -2.1521288  1.6949403    dynamic
## lag(Oil, 8)       8   0.8799744  -1.0996930  2.8596418    dynamic
## lag(Oil, 9)       9  -1.6392365  -3.6585139  0.3800409    dynamic
## lag(Oil, 10)     10  -3.9233937  -7.5990134 -0.2477739    dynamic
## lag(Oil, 11)     11  -2.6074098  -6.4436710  1.2288515    dynamic
## lag(Oil, 12)     12  -0.2247408  -2.7615422  2.3120606    dynamic
## lag(Oil, 13)     13  -1.5546828  -3.7752934  0.6659279    dynamic
## lag(Oil, 14)     14  -1.4831538  -3.2736127  0.3073050    dynamic
## lag(Oil, 15)     15  -1.4787743  -3.1364347  0.1788861    dynamic
## lag(Oil, 16)     16  -0.1002288  -1.8744258  1.6739682    dynamic
```

```
## lag(Oil, 17)     17   0.4980770  -0.9729956  1.9691497    dynamic
## lag(Oil, 18)     18   0.0096065  -1.8947179  1.9139309    dynamic
## d.Oil             0   0.1553167  -1.5198367  1.8304700 cumulative
## lag(d.Oil)        1  -0.8206870  -3.0792261  1.4378520 cumulative
## lag(d.Oil, 2)     2  -2.2237347  -4.9551335  0.5076641 cumulative
## lag(d.Oil, 3)     3  -3.0552822  -6.5619368  0.4513724 cumulative
## lag(d.Oil, 4)     4  -3.4617204  -7.3175661  0.3941254 cumulative
## lag(d.Oil, 5)     5  -3.8819115  -8.3927339  0.6289109 cumulative
## lag(d.Oil, 6)     6  -6.4369844 -12.0994927 -0.7744761 cumulative
## lag(d.Oil, 7)     7  -6.6655786 -12.1887094 -1.1424478 cumulative
## lag(d.Oil, 8)     8  -5.7856043 -11.1014454 -0.4697632 cumulative
## lag(d.Oil, 9)     9  -7.4248408 -13.1250265 -1.7246550 cumulative
## lag(d.Oil, 10)   10 -11.3482344 -18.3752922 -4.3211767 cumulative
## lag(d.Oil, 11)   11 -13.9556442 -21.6464550 -6.2648334 cumulative
## lag(d.Oil, 12)   12 -14.1803850 -22.5413591 -5.8194110 cumulative
## lag(d.Oil, 13)   13 -15.7350678 -24.5150980 -6.9550376 cumulative
## lag(d.Oil, 14)   14 -17.2182216 -26.4112899 -8.0251534 cumulative
## lag(d.Oil, 15)   15 -18.6969959 -27.8329650 -9.5610268 cumulative
## lag(d.Oil, 16)   16 -18.7972247 -27.6186409 -9.9758086 cumulative
## lag(d.Oil, 17)   17 -18.2991477 -27.0183840 -9.5799114 cumulative
## lag(Oil, 18)1    18 -18.2895412 -27.0817195 -9.4973629 cumulative
```

We added a column here called "type" which tells us what kind of multiplier the row corresponds to. The "rbind" function (short for "row bind"), just stacks one data.frame on top of the other as long as they have the same number of columns.

Now we can plot:

```
ggplot(multipliers, aes(x = lags, y = coef, ymin = low, ymax = high, color = type, fill = type)) +
  theme_minimal() +
  ggtitle('Multipliers') +
  xlab('Lags') + ylab('') +
  geom_line() +
  geom_ribbon(alpha = 0.3) +
  geom_hline(yintercept = 0, color = 'black') # optional: draw a line at y = 0
```


Multipliers
0
−10
−20
0
5
10
15
Lags
type
cumulative
dynamic