Econometrics III
Assignment Part 3, 4, 5
Tinbergen Insitute

Stanislav Avdeev
590050sa
stnavdeev@gmail.com

Bas Machielsen
590049bm
590049bm@eur.nl

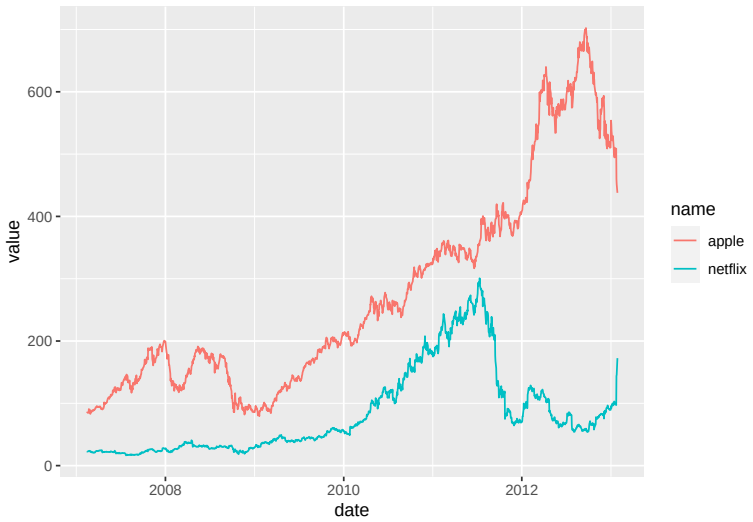
April 8, 2021

Question 3

Part 1:

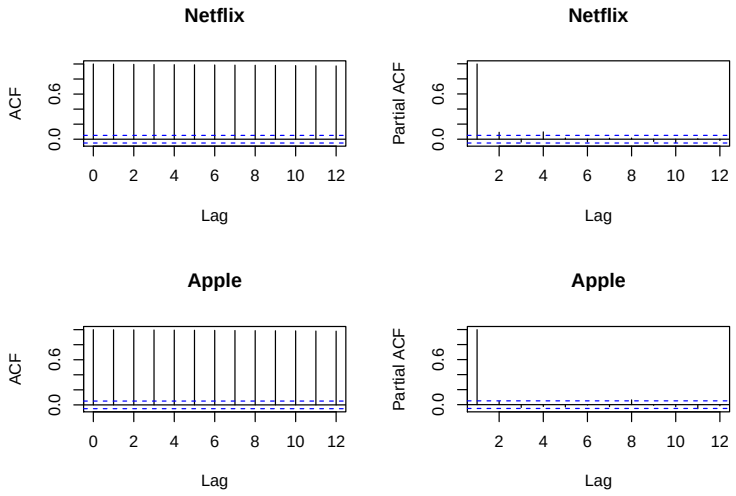
First, we plot the two time series:

```
df3 <- readr::read_csv("./data/data_assign_p3.csv")

twostocks <- c("apple", "netflix")

df3_twostocks <- df3 %>%
    janitor::clean_names() %>%
    pivot_longer(-date) %>%
    dplyr::filter(is.element(name, twostocks)) %>%
    mutate(date = lubridate::dmy(date))

df3_twostocks %>%
    ggplot(aes(x = date, y = value,
               group = name, color = name)) + geom_line()
```



Then, we show the acf and pacf-functions:

```
n1 <- df3$NETFLIX %>%
    acf(lag.max = 12, plot = F)

n2 <- df3$NETFLIX %>%
    pacf(lag.max = 12, plot = F)

a1 <- df3$APPLE %>%
    acf(lag.max = 12, plot = F)

a2 <- df3$APPLE %>%
    pacf(lag.max = 12, plot = F)

par(mfrow=c(2,2))

plot(n1, main = "Netflix");plot(n2, main = "Netflix")
plot(a1, main = "Apple"); plot(a2, main = "Apple")
```



The ACF's tell us that the stock price is highly dependent on the past stock price, and this dependence decays only very slowly: even the 50 or 100-period lag still shows significant autocorrelation.

Part 2

We now implement a general to specific unit root test function:

```
unit_root_test <- function(column, order, critical_value){

  # Make the dataset
  series <- ts(column)
  first_differences <- diff(series, differences = 1)
  laggedvar <- stats::lag(series, -1)

  #other lagged first differences, delta x_{t-1}, .., delta x_{t-p+1}
  lagged_fds <- list()

  for(i in 1:(order-1)){

    lagged_fds[[i]] <- stats::lag(first_differences, k = -i)

  }

  df <- cbind(first_differences, laggedvar, purrr::reduce(lagged_fds, cbind)) %>%
    as_tibble()

  colnames(df) <- c("dxt", "xtm1", paste("dxtm", 1:(order-1), sep = ""))

  df <- df %>%
    na.omit()

  # run the stepwise regression - find the best model
  null = lm(data = df, formula = "dxt ~ xtm1")
  full = lm(data = df, formula = paste("dxt ~ xtm1 +",
                                       paste(paste("dxtm", 1:(order-1), sep = ""),
                                             collapse = ' + '),
                                       collapse = " ")
  )

  bestmodel <- step(full,
       scope = list(lower = null, upper = full),
       direction = "backward",
       criterion = "BIC",
       k = log(nrow(df)),
       trace = 0)

  # perform the unit root test (MacKinnon, 2010)
  b_critical <- critical_value

  t_value <- bestmodel %>%
    summary() %>%
    .$coefficients %>%
    .[,3] %>%
    .["xtm1"]

  significant = abs(t_value) > abs(b_critical)

  data.frame(stock = deparse(substitute(column)),
             best_model = as.character(bestmodel$call[2]),
             t_value = t_value,
             sig = significant)
}
```

```
summary_tests <- list()

for(i in 1:length(colnames(df3[,-1]))){
  df <- unit_root_test(df3[, i+1], 12, -1.6156)
  summary_tests[[i]] <- df %>%
    mutate(name = colnames(df3[,-1][i]))
}

summary_tests <- summary_tests %>%
  purrr::reduce(rbind) %>%
  select(-stock)

rownames(summary_tests) <- NULL
```

```
knitr::kable(summary_tests)
```

best_model	t_value	sig	name
dxt ~ xtm1 + dxtm1 + dxtm3 + dxtm7	-0.6984454	FALSE	APPLE
dxt ~ xtm1 + dxtm1 + dxtm2	-2.0475011	TRUE	EXXON_MOBIL
dxt ~ xtm1	-1.1619768	FALSE	FORD
dxt ~ xtm1 + dxtm1 + dxtm3	-1.5455395	FALSE	GEN_ELECTRIC
dxt ~ xtm1	-2.2598273	TRUE	INTEL
dxt ~ xtm1	-2.3999971	TRUE	MICROSOFT
dxt ~ xtm1 + dxtm2	-1.0642387	FALSE	NETFLIX
dxt ~ xtm1 + dxtm2 + dxtm3	-0.6373731	FALSE	NOKIA
dxt ~ xtm1 + dxtm1	-1.3045653	FALSE	SP500
dxt ~ xtm1 + dxtm1 + dxtm6 + dxtm11	-2.6283893	TRUE	YAHOO

The test seems to be significant for a number of stocks, indicating that for these stocks, the null hypothesis of a unit root is rejected in favor of stationarity. With a 10% α-level, we expect to see a type I-error (rejecting the null while it is true) about one tenth of the time for every test. This means that the probability of having at least 1 type-I error is very large: $(1 - 0.9^{10}) = 0.6513216$. It would be better to use some kind of Bonferroni correction to correct for these compounding type I-errors, but alternatively, we could also lower the α-level.

Part 3

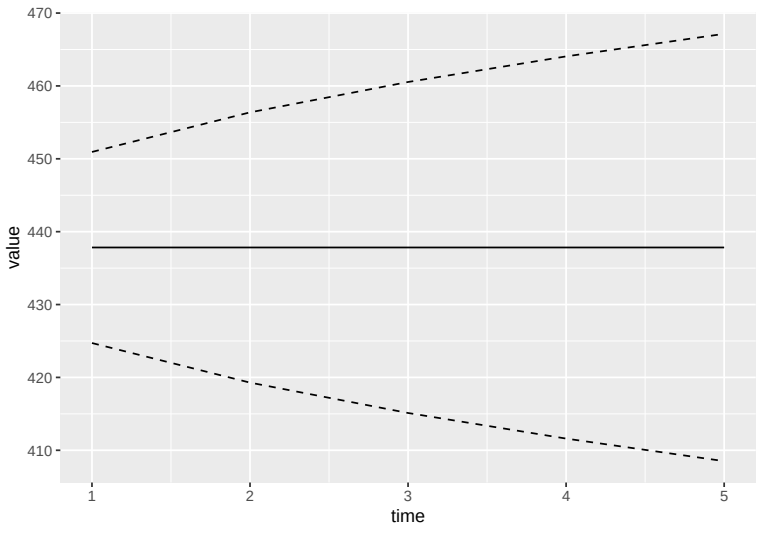
The forecast $\mathbb{E}[P_{t+1}|D_t] = \mathbb{E}[P_{t+1}|P_t] = \mathbb{E}[P_t + \epsilon_t] = P_t$. Similarly, the forecast $\mathbb{E}[P_{t+2}] = \mathbb{E}[P_{t+1} + \epsilon_{t+1}] = \mathbb{E}[P_{t+1}] = P_t$. Generalizing this pattern, the forecast for $P_{t+h} = P_t$. The variance of the forecast is derived using the distribution:

$$
\begin{aligned}
P_{t+1} &= P_t + \epsilon_t \Rightarrow P_{t+1}|P_t \sim N(P_t, \sigma^2) \\
P_{t+2} &= P_{t+1} + \epsilon_{t+1} = \\
&= P_t + \epsilon_t + \epsilon_{t+1} \Rightarrow P_{t+2}|P_t \sim N(P_t, 2\sigma^2)
\end{aligned}
$$

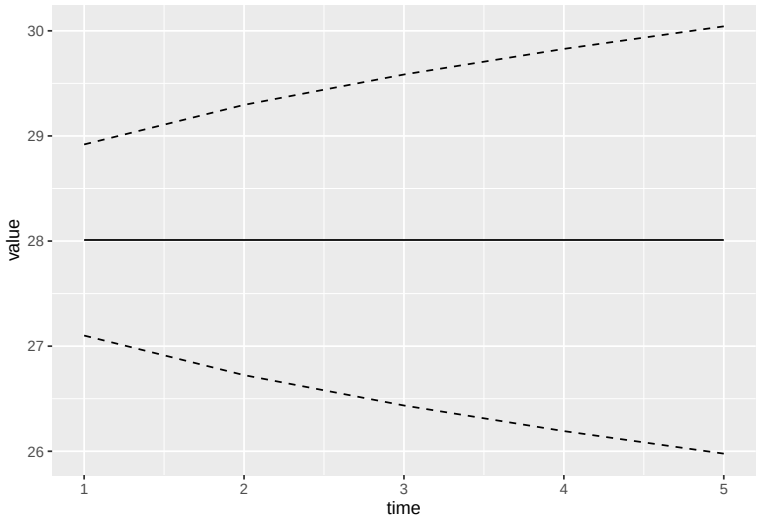
Generalizing this pattern, we can see that $\text{Var}(P_{t+h}) = h \cdot \sigma^2$. Hence, we can implement our forecasts in the following way:

```
#forecast code
p_tplush <- df3 %>%
  slice_tail(n=1) %>%
```

```
  select(c("APPLE", "MICROSOFT"))

apple_t <- matrix(df3$APPLE, ncol=1)
apple_lags <- cbind(apple_t, c(NA, apple_t))

var_apple <- lm(apple_lags[,1] ~ apple_lags[,2]) %>%
  .$residuals %>%
  var()

microsoft_t <- matrix(df3$MICROSOFT, ncol = 1)
microsoft_lags <- cbind(microsoft_t, c(NA, microsoft_t))

var_microsoft <- lm(microsoft_lags[,1] ~ microsoft_lags[,2]) %>%
  .$residuals %>%
  var()

forecasts_apple <- data.frame(time = 1:5,
                              value = rep(p_tplush %>%
                                            pull(1), 5),
                              var = var_apple * 1:5)

forecasts_microsoft <- data.frame(time = 1:5,
                                  value = rep(p_tplush %>%
                                                pull(2), 5),
                                  var = var_microsoft * 1:5)

forecasts_apple %>%
  ggplot(aes(x = time)) +
  geom_line(aes(y = value)) +
  geom_line(aes(y = value + 1.96*sqrt(var)), lty = "dashed") +
  geom_line(aes(y = value - 1.96*sqrt(var)), lty = "dashed")
```




```
forecasts_microsoft %>%
  ggplot(aes(x = time)) +
  geom_line(aes(y = value)) +
  geom_line(aes(y = value + 1.96*sqrt(var)), lty = "dashed") +
  geom_line(aes(y = value - 1.96*sqrt(var)), lty = "dashed")
```



Hence, there is no investment advice that we can give: the value is predicted to remain constant, and for all predictions, the probability of an increase in stock price equals the probability of a decrease in stock price. The expected value of any investment strategy is 0, and the value is neither expected to increase, nor to decrease.

Part 4

Do you find a statistically significant contemporaneous relation between Microsoft and Exxon Mobile stock prices?

```
lm(df3, formula = MICROSOFT ~ EXXON_MOBIL) %>%
  stargazer(header = F, omit.stat = c("adj.rsq", "ser", "f"))
```

Table 2:

	Dependent variable:
	MICROSOFT
EXXON_MOBIL	0.203^{***}
	(0.009)
Constant	11.245^{***}
	(0.713)
Observations	1,499
R^2	0.251
Note:	*p<0.1; **p<0.05; ***p<0.01

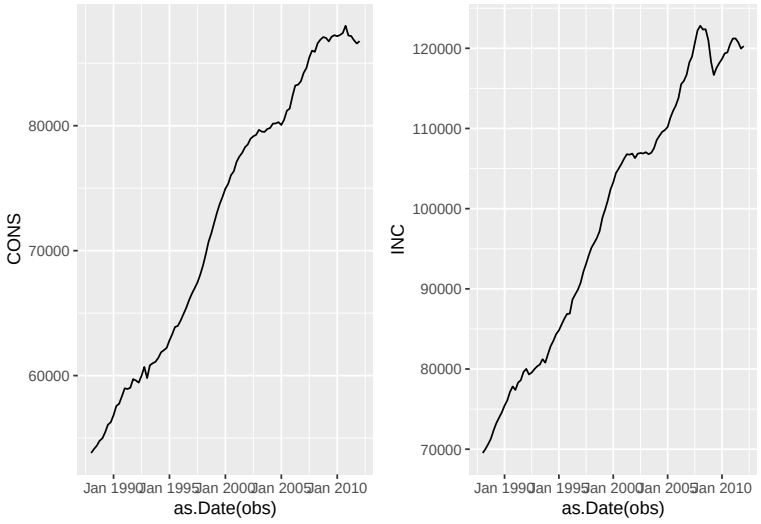
Do you agree that changes in Microsoft stock prices are largely explained by fluctuations in the stock price of Exxon Mobile?

We do not agree that changes in Microsoft stock prices are largely explained by fluctuations in the stock price of Exxon Mobile. It is likely that these results are driven by a shared stochastic trend in both of variables, in other words, the variables might be cointegrated. It can be shown that if two variables share a stochastic trend, the t-value of the estimated coefficient tends to infinity, and the probability of obtaining statistical significance to 1, even under the assumption of completely unrelated trends.

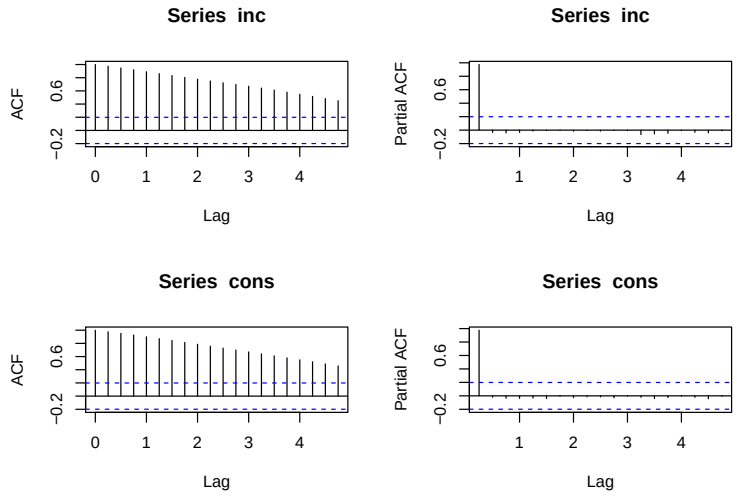
Question 4

Part 1

```
df4 <- read_csv("data/data_assign_p4.csv")

df4$obs <- ts(df4$obs,
              frequency = 4,
              start = c(1988, 1))

p_cons <- ggplot(data = df4, aes(x = as.Date(obs), y = CONS)) +
  geom_line() +
  scale_x_date(date_labels = "%b %Y")

p_inc <- ggplot(data = df4, aes(x = as.Date(obs), y = INC)) +
  geom_line() +
  scale_x_date(date_labels = "%b %Y")

cowplot::plot_grid(p_cons, p_inc)
```




```
inc <- ts(df4$INC,
   frequency = 4,
   start = c(1988, 1))

cons <- ts(df4$CONS,
   frequency = 4,
   start = c(1988, 1))

inc_acf <- acf(inc, plot = F); inc_pcf <- pacf(inc, plot = F)
cons_acf <- acf(cons, plot = F); cons_pcf <- pacf(cons, plot = F)

par(mfrow=c(2,2))
plot(inc_acf);plot(inc_pcf);plot(cons_acf);plot(cons_pcf)
```



We observe that in both cases, the autocorrelation function shows extremely high estimates for each subsequent lag: even for a large number of periods, there is still a large degree of autocorrelation in the data. The partial autocorrelations, however, show a completely different view: there is no partial autocorrelation when controlled for other influences. This means that there is a lot of dependence in the series unconditionally, but conditionally on previous values, there seems to be no correlation. This behavior seems to be consistent with random walk behavior.

Part 2

We use the same function as in the previous question to find the best model (according to BIC), and compute the t-value and conduct a Dickey-Fuller test:

```
urcons <- unit_root_test(df4$CONS, order = 12, -1.9393)
urinc <- unit_root_test(df4$INC, order = 12, -1.9393)

unitroots <- rbind(urcons, urinc) %>%
  mutate(stock = str_replace(stock, "df4\\$", ""))

rownames(unitroots) <- NULL
```

```
knitr::kable(unitroots)
```

stock	best_model	t_value	sig
CONS	dxt ~ xtm1 + dxtm3	-1.619306	FALSE
INC	dxt ~ xtm1 + dxtm1	-1.169998	FALSE

As we can see in the table, the unit root hypothesis is not rejected at a 5% level in both cases.

Part 3

Now, we perform a unit root test on the first differences of both series:

```
urcons2 <- unit_root_test(diff(df4$CONS), order = 12, -1.9393)
urinc2 <- unit_root_test(diff(df4$INC), order = 12, -1.9393)

unitroots2 <- rbind(urcons2, urinc2) %>%
  mutate(stock = str_replace(stock, "df4\\$", ""))
```

```
rownames(unitroots2) <- NULL
```

```
knitr::kable(unitroots2)
```

stock	best_model	t_value	sig
diff(CONS)	dxt ~ xtm1 + dxtm1 + dxtm2	-2.382383	TRUE
diff(INC)	dxt ~ xtm1 + dxtm8	-5.231291	TRUE

Now, we see that both hypotheses are rejected! We conclude that both of these series are integrated at order $I(1)$, because the hypothesis of a unit root in the first differences is rejected.

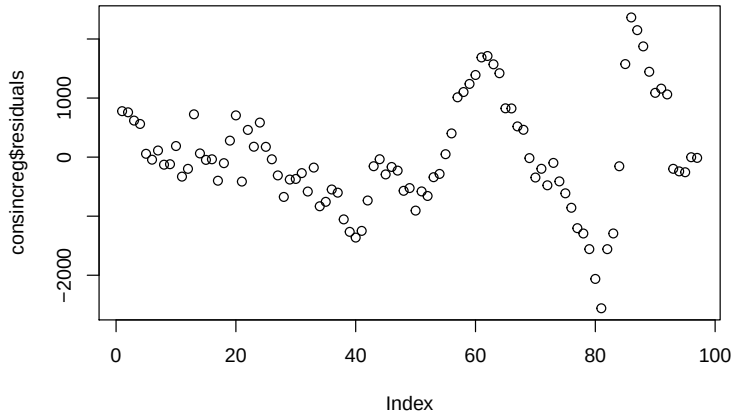
Part 4

```
consincreg <- lm(data = df4,
   formula = CONS ~ INC)

consincreg %>%
  stargazer(header = F,
            omit.stat = c("adj.rsq", "ser", "f"))
```

Table 5:

	Dependent variable:
	CONS
INC	0.665***
	(0.006)
Constant	6,783.373***
	(554.991)
Observations	97
R^2	0.993
Note:	*p<0.1; **p<0.05; ***p<0.01

```
plot(consincreg$residuals)
```




```
urt2 <- unit_root_test(consincreg$residuals,
                order = 12,
                -1.6156)

rownames(urt2) <- NULL
```

```
knitr::kable(urt2)
```

stock	best_model	t_value	sig
consincreg$residuals	dxt ~ xtm1 + dxtm2 + dxtm8	-2.127911	TRUE

We observe that the hypothesis of a unit root in the residuals is rejected, indicating that the two series are cointegrated: $Cons, Inc \sim CI(1,1)$. Using general-to-specific approach with BIC, a determined number of lags is two.

Part 5

As the series are cointegrated, we can estimate an error correction model, incorporate long-run equilibrium information as well as short-run dynamics. The error-correction model is of the following general form:

$$\Delta Y_t = \alpha - \gamma \bar{Z}_{t-1} + \phi_1 \Delta Y_{t-1} + \cdots + \phi_p \Delta Y_{t-p} + \\ \beta_0 \Delta X_t + \cdots + \beta_q \Delta X_{t-q} + u_t$$

We first construct the necessary data matrix in Python:

```
import pandas as pd

# p for dep var, q for indep var
def generate_datamatrix(p, q, y, x):
  # dependent variable
  fdy = r.df4[str(y)] - r.df4[str(y)].shift(1)
  #ytm1
  ytm1 = r.df4[str(y)].shift(1)
  #xtm1
```

```
  xtm1 = r.df4[str(x)].shift(1)

  d = {}

  d['xt'] = r.df4[str(x)]
  d['xtm1'] = xtm1
  # x first differences
  d['dxtm0'] = r.df4[str(x)] - r.df4[str(x)].shift(1)
  # lagged first differences
  for i in range(1, q+1):
    d['dxtm{0}'.format(i)] = d['dxtm0'].shift(i)

  # y first differences
  e = {}

  e['yt'] = r.df4[str(y)]
  e['fdy'] = fdy
  e['ytm1'] = ytm1

  for j in range(1, p+1):
    e['dytm{0}'.format(j)] = e['fdy'].shift(j)

  # put everything in a dataframe
  dfx = pd.DataFrame.from_dict(d)
  dfy = pd.DataFrame.from_dict(e)

  df_out = pd.concat([dfx.reset_index(drop=True), dfy], axis=1)

  return df_out

data_ecm = generate_datamatrix(4, 4, 'CONS', 'INC')
```

Now, we implement the general-to-specific procedure in R. First, we estimate the residuals $\bar{Z}_t$, and compute $\bar{Z}_{t-1}$ which we then lag to add as a predictor in the general-to-specific procedure.

```
zt <- lm(data = py$data_ecm,
   formula = yt ~ xt)

residuals <- zt$residuals

ztm1 <- lag(residuals, 1)
```

Now, we add $\bar{Z}_{t-1}$ to the dataset, and formulate the g2s procedure (again according to the BIC):

```
data_ecm <- py$data_ecm %>%
  mutate(ztm1 = ztm1) %>%
  na.omit()

indep_vars <- py$data_ecm %>%
  colnames()

indep_vars <- indep_vars[grepl("dxt|dyt", indep_vars)]

longformula <- paste("fdy ~ ztm1 + 0 +", paste(indep_vars, collapse = " + "))
# run the stepwise regression - find the best model
null = lm(data = data_ecm, formula = "fdy ~ ztm1 + 0")
full = lm(data = data_ecm, formula = longformula)
```

```
bestmodel <- step(full,
       scope = list(lower = null, upper = full),
       direction = "backward",
       criterion = "BIC",
       k = log(nrow(data_ecm)))
```

Starting with $p = q = 4$, we get the following model:

```
stargazer(bestmodel, header = F,
          omit.stat = c("adj.rsq", "ser", "f"))
```

Table 7:

	Dependent variable:
	fdy
ztm1	0.036
	(0.037)
dxtm0	0.205***
	(0.046)
dytm2	0.222***
	(0.078)
dytm3	0.410***
	(0.083)
Observations	92
R^2	0.659
Note:	*p<0.1; **p<0.05; ***p<0.01

Report and interpret the short-run and long-run multipliers. Report and interpret the error correction coefficient.

The error correction coefficient $\gamma = 0.0363067$ means that if consumption and income are away from their long-run relationship, there is a correction towards the equilibrium value. For example, if there is a positive residual, meaning consumption is too high relative to the income at a given point in time, in the next period, we tend to observe a *decrease* in consumption to correct for that.

The short-run multipliers can be derived from the coefficient belonging to ΔX_t. An (positive) exogenous shock in income causes an increase in consumption (0.20). The effects of that shock also appear three periods later (reflected by the coefficient on ΔY_{t-3}).

The long-run multipliers can be derived from the equilibrium relationship in the "first-stage" regression:

$\bar{Y} = 6783.3732069 + 0.6651404 \cdot \bar{X}$.

This indicates that people tend to consume a proportion of 66% of their long-run equilibrium income, plus a fixed amount of 6783.3732069, which can (kind of) be interpreted as the cost of living (even if the long-run equilibrium income is 0, they theoretically consume this amount).

6. How strong is the correction to equilibrium? Is there over-shooting?

The error correction coefficient $\gamma = 0.0363067$ is very close to zero. Hence, there is no overshooting $\gamma < -1$, and only a partial error correction. The error correction is also quite slow, indicated by the low magnitude of the coefficient.

Do you find evidence of Granger causality? Justify your answer.

We can formally test for Granger causality by taking a stationary version of both series (in this case, the first differences) and investigate if there is any predictive power from the lagged first differences of X_t (income), conditional on the inclusion of lagged Y (first differences of consumption) variables. To do this, we again use the Python code to generate a datamatrix containing first differences of both variables, and lagged versions. We start with 10 lags:

```
data_granger = generate_datamatrix(10, 10, 'CONS', 'INC')
```

We now use this dataset in R to perform stepwise regression:

```
indep_vars <- py$data_granger %>%
  colnames()

indep_vars <- indep_vars[grepl("dxtm|dytm", indep_vars)]

longformula <- paste("fdy ~ ", paste(indep_vars, collapse = " + "))
# run the stepwise regression - find the best model
null = lm(data = py$data_granger %>%
            na.omit(), formula = "fdy ~ dytm1")
full = lm(data = py$data_granger %>%
            na.omit(), formula = longformula)

bestmodel <- step(full,
       scope = list(lower = null, upper = full),
       direction = "backward",
       criterion = "BIC",
       k = log(nrow(data_ecm)))
```

```
stargazer(bestmodel,
          omit.stat = c("f", "ser", "adj.rsq"),
          header = FALSE)
```

We can find very clear evidence of Income Granger-causing Consumption: there is at least 1 (in the case) two lagged variables that are significant: ΔX_{t-4} and ΔX_{t-8}. This is probably because there is a seasonality effect. In any case, if at least one of the lags is significant, then ΔX granger-causes ΔY. The p-values are small enough, so that we also don't have two worry about type I-errors while making this conclusion.

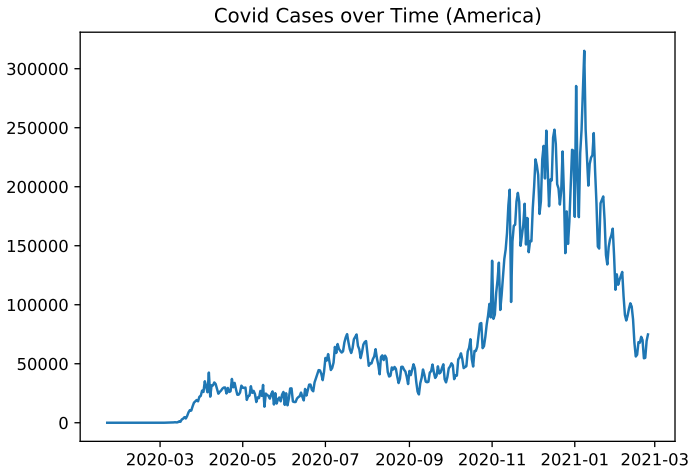
Question 5

First, we import and plot the data:

```
import dateutil
import pandas as pd
import matplotlib.pyplot as plt

coviddixneuf = pd.read_csv("./data/data_assign_p5.csv", skiprows=2)
coviddixneuf['Date'] = coviddixneuf['Date'].apply(dateutil.parser.parse, dayfirst=True)

plt.plot(coviddixneuf['Date'], coviddixneuf['New Cases'])
plt.title("Covid Cases over Time (America)")
```

Table 8:

	Dependent variable:
	fdy
dxtm0	0.289^{***}
	(0.046)
dxtm4	0.110^{**}
	(0.046)
dxtm8	0.260^{***}
	(0.045)
dytm1	-0.153
	(0.094)
Constant	37.828
	(50.922)
Observations	86
R^2	0.467
Note:	*p<0.1; **p<0.05; ***p<0.01



We observe that both the mean and the variance tend to vary over time. This means we cannot use models of the ARMA(p,q) class to model the process, because the data is non-stationary. We formally test for this in the following code chunk, using the function unit_root_test from question 4.

```
# Unit root test showing the data is non-stationary and the first differences are stationary
covid19 <- py$coviddixneuf %>%
  janitor::clean_names() %>%
  arrange(date) %>%
  mutate(diffcases = c(NA, diff(new_cases)))

unit_root_test(covid19$new_cases, 4, 2.0) %>%
```

```
  knitr::kable()
```

	stock	best_model	t_value	sig
xtm1	covid19$new_cases	dxt ~ xtm1 + dxtm1 + dxtm2	-1.7324	FALSE

```
unit_root_test(covid19$diffcases, 4, 2.0) %>%
  knitr::kable()
```

	stock	best_model	t_value	sig
xtm1	covid19$diffcases	dxt ~ xtm1 + dxtm1 + dxtm2 + dxtm3	-16.05064	TRUE

We observe that the first differences are stationary. Hence, we can use a regular ARMA process on the first differences. However, the ARMA-class of models assumes a constant error variance, whereas in the plot, we can observe that the variance is non-constant. To model this behavior, we might resort to the GARCH-class of models. From a lemma based on stack overflow, we know that the conditional variance of a first-differenced series ΔX_t is equal to the conditional variance of the original series X_t. Hence, we can use the original data to generate the variance and standard errors corresponding to our predictions (point estimates) made by the AR(I)MA class of models. Hence, in the end, we deliver predictions that can be applied to original, levels, i.e. the non-stationary data series, and predict New Cases, rather than predicting first differences.

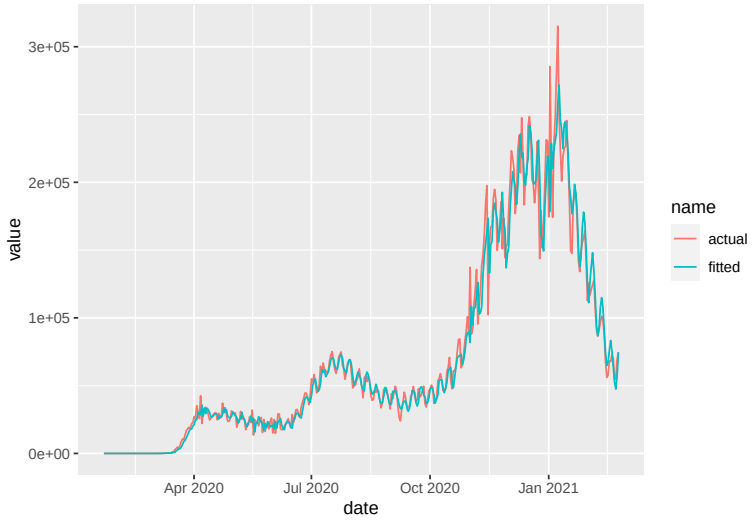
Hence, first, we estimate an Arima model on the first-differences data, using the AIC-criterion to go from general to specific models. The selection is implemented in Python, the rest is implemented in R. Then, we convert back the estimates to compute the in-sample fit, and visually assess in-sample accuracy.

```
from statsmodels.tsa.arima.model import ARIMA
import numpy as np
import warnings
warnings.filterwarnings('ignore')

best_p = None
best_q = None
best_aic = 10000

for i in reversed(range(1,5)):

  for j in reversed(range(1, 5)):

    arima = ARIMA(coviddixneuf['New Cases'], order =(i, 1, j))
    arima_fit = arima.fit()

    if arima_fit.aic < best_aic:
      best_aic = arima_fit.aic
      best_p = i
      best_q = j

print("Best p:", best_p, "\n", "Best q:", best_q)
```

```
## Best p: 4
##  Best q: 3
```

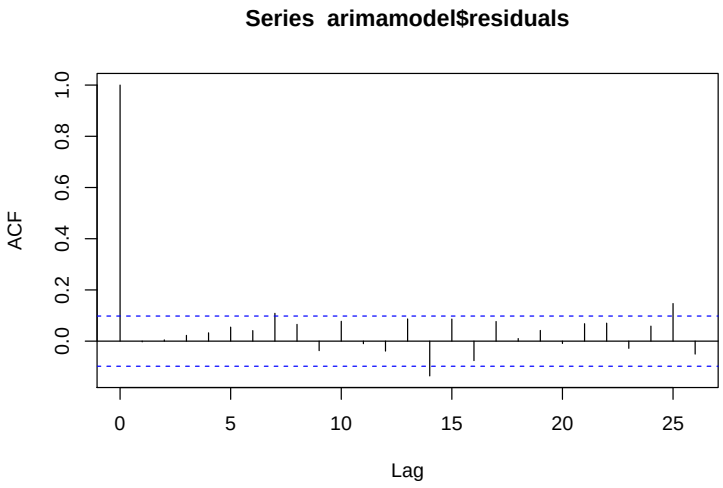
Hence, the best ARIMA-model is of the order (4, 1, 3). We now implement this in R:

```
arimamodel <- forecast::Arima(covid19$new_cases, order = c(4, 1, 3),
                              include.constant = FALSE)

evalarima <- data.frame(date = covid19$date,
                        actual = covid19$new_cases,
                        fitted = fitted(arimamodel))

evalarima %>%
  pivot_longer(c(actual, fitted)) %>%
  ggplot(aes(x = date, y = value, group = name, color = name)) + geom_line()
```



We can verify whether the model is well-specified by checking for autocorrelation in the residuals, from which we conclude that the model is well-specified:

```
acf(arimamodel$residuals)
```



We then predict the values for the Arima model:

```
how_many_days_predict <- 14

predictions <- predict(arimamodel, how_many_days_predict)$pred
```

Then, we estimate a Garch model to find an estimate of the conditional variances:

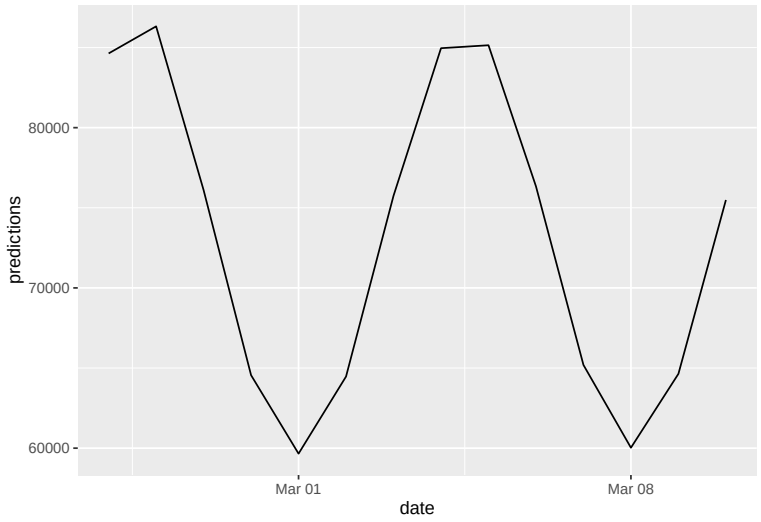
```
library(fGarch)
# Same order as optimal model
garchmodel <- fGarch::garchFit(formula = ~ arma(4,1) + garch(1,1),
                               data = covid19$new_cases %>%
                                 na.omit(),
                               trace = F)
```

We use the Garch model to compute σ_t for the predictions:

```
# Calculate predicted sigma_t's for the next X periods
omega <- garchmodel@fit$params$params['omega']
alpha1 <- garchmodel@fit$params$params['alpha1']
beta1 <- garchmodel@fit$params$params['beta1']

create_sigma_t <- function(alpha, beta, omega, predict_periods){

  sigma_t = vector(length = predict_periods)

  for(i in 1:predict_periods){

    if(i == 1){

      sigma_t[i] <- sqrt(beta * garchmodel@sigma.t[400]^2+ alpha * garchmodel@data[400]^2 + omega)

    } else {

      sigma_t[i] <- sqrt(beta * sigma_t[i-1]^2 + alpha * predictions[i-1]^2 + omega)

    }

  }

  return(sigma_t)
}

sigma_formodel <- create_sigma_t(alpha1, beta1, omega, how_many_days_predict)
```

Finally, we provide predicted values with confidence intervals to investigate out-of-sample fit:

```
# Graph with confidence bounds
preddata <- data.frame(predictions = predictions,
           date = seq(
             from = lubridate::as_date(covid19$date[400]) + 1,
             length.out = how_many_days_predict,
             by = "1 day"),
           sigma_t = sigma_formodel
)


preddata %>%
  ggplot() + geom_line(aes(x = date, y = predictions))
```



Hence, we expect to see a sharp drop in cases following the end of February, after which we expect a stabilization of number of cases around a mean of around 70,000 new cases a day. Of course, there are limitations to this analysis: being atheoretical, there is no way to incorporate the vaccination of the population, nor is there any component that incorporates government activity. Secondly, as is evident from the modeling of the volatility, estimation of the variance is very large, effectively excluding no possible data points from happening. In sum, a prediction based on these values, especially over a longer period, is not very valuable.