# Practice Set #3

## Purpose, Process, Product

Various `R` features and finance topics will be reprised in this chapter. Specifically we will practice reading in data, exploring time series, estimating auto and cross correlations, and investigating volatility clustering in financial time series. We will summarize our experiences in debrief.

### Set A

In this set we will build and explore a data set using filters and `if` and `diff` statements. We will then answer some questions using plots and a pivot table report. We will then review a function to house our approach in case we would like to run some of the same analysis on other data sets.

### Problem

Marketing and accounts receivableds managers at our company continue to note we have a significant exposure to exchange rates. Our customer base is located in the United Kingdom, across the European Union, and in Japan. The exposure hits the gross revenue line of our financials. Cash flow is further affected by the ebb and flow of accounts receivable components of working capital. of producing several products. When exchange rates are volatile, so is earnings, and more importantly, our cash flow. Our company has also missed earnings forecasts for five straight quarters. To get a handle on exchange rate exposures we download this data set and review some basic aspects of the exchange rates.

```
# Read in data
require(zoo)
require(xts)
require(ggplot2)
# Read and review a csv file from
# FRED
exrates <- na.omit(read.csv("data/exrates.csv",
    header = TRUE))
head(exrates)
```

```
##        DATE  USD.EUR  USD.GBP USD.CNY USD.JPY
## 1 1/29/2012 0.763678 0.638932 6.29509 77.1840
## 2  2/5/2012 0.760684 0.633509 6.29429 76.3930
## 3 2/12/2012 0.757491 0.632759 6.29232 77.2049
## 4 2/19/2012 0.760889 0.634166 6.29644 78.7109
## 5 2/26/2012 0.750301 0.632641 6.29710 80.3373
## 6  3/4/2012 0.750474 0.629771 6.29873 81.1607
```

```
tail(exrates)
```

```
##           DATE  USD.EUR  USD.GBP USD.CNY USD.JPY
## 255 12/11/2016 0.938872 0.791554 6.93141 114.397
## 256 12/18/2016 0.950478 0.796572 6.93042 116.796
## 257 12/25/2016 0.958288 0.810481 6.94908 117.469
## 258   1/1/2017 0.954067 0.813594 6.94929 117.100
## 259   1/8/2017 0.951493 0.812388 6.92820 116.968
## 260  1/15/2017 0.943352 0.820854 6.91781 115.287
```

```
str(exrates)
```

```
## 'data.frame':    260 obs. of  5 variables:
##  $ DATE   : Factor w/ 260 levels "1/1/2017","1/10/2016",..: 15 103 89 94 100 126 109 114 120 130 ...
##  $ USD.EUR: num  0.764 0.761 0.757 0.761 0.75 ...
##  $ USD.GBP: num  0.639 0.634 0.633 0.634 0.633 ...
##  $ USD.CNY: num  6.3 6.29 6.29 6.3 6.3 ...
##  $ USD.JPY: num  77.2 76.4 77.2 78.7 80.3 ...
```

**Questions**

1. What is the nature of exchange rates? We want to reflect the ups and downs of rate movements, known to managers as currency appreciation and depreciation. First, we calculate percentage changes as log returns. Our interest is in the ups and downs. To look at that we use `if` and `else` statements to define a new column called `direction`. We will build a data frame to house this analysis.

```
# Compute log differences percent
# using as.matrix to force numeric
# type
exrates.r <- diff(log(as.matrix(exrates[,
    -1]))) * 100
head(exrates.r)
```

```
##       USD.EUR    USD.GBP     USD.CNY    USD.JPY
## 2 -0.39282058 -0.8523826 -0.01270912 -1.0301113
## 3 -0.42063724 -0.1184583 -0.03130311  1.0571858
## 4  0.44758304  0.2221127  0.06545522  1.9318720
## 5 -1.40130272 -0.2407629  0.01048156  2.0452375
## 6  0.02305476 -0.4546859  0.02588158  1.0197119
## 7  1.19869144  0.7988383  0.22598055  0.6384155
```

```
tail(exrates.r)
```

```
##        USD.EUR    USD.GBP      USD.CNY    USD.JPY
## 255 -0.1234763 -0.4555311  0.602265010  0.9803397
## 256  1.2285861  0.6319419 -0.014283828  2.0753968
## 257  0.8183343  1.7310378  0.268885929  0.5745646
## 258 -0.4414460  0.3833571  0.003021937 -0.3146198
## 259 -0.2701570 -0.1483412 -0.303945688 -0.1127877
## 260 -0.8592840  1.0367203 -0.150079365 -1.4475722
```

```
str(exrates.r)
```

```
##  num [1:259, 1:4] -0.3928 -0.4206 0.4476 -1.4013 0.0231 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:259] "2" "3" "4" "5" ...
##   ..$ : chr [1:4] "USD.EUR" "USD.GBP" "USD.CNY" "USD.JPY"
```

```
# Create size and direction
size <- na.omit(abs(exrates.r))  # size is indicator of volatility
colnames(size) <- paste(colnames(size),
    ".size", sep = "")
direction <- ifelse(exrates.r > 0, 1,
    ifelse(exrates.r < 0, -1, 0))  # another indicator of volatility
colnames(direction) <- paste(colnames(direction),
    ".dir", sep = "")
# Convert into a time series object:
```

```
# 1. Split into date and rates
dates <- as.Date(exrates$DATE[-1], "%m/%d/%Y")
values <- cbind(exrates.r, size, direction)
# for dplyr pivoting we need a data
# frame
exrates.df <- data.frame(dates = dates,
    returns = exrates.r, direction = direction)
str(exrates.df)  # notice the returns.* and direction.* prefixes
```

```
## 'data.frame':    259 obs. of  9 variables:
##  $ dates               : Date, format: "2012-02-05" "2012-02-12" ...
##  $ returns.USD.EUR     : num  -0.3928 -0.4206 0.4476 -1.4013 0.0231 ...
##  $ returns.USD.GBP     : num  -0.852 -0.118 0.222 -0.241 -0.455 ...
##  $ returns.USD.CNY     : num  -0.0127 -0.0313 0.0655 0.0105 0.0259 ...
##  $ returns.USD.JPY     : num  -1.03 1.06 1.93 2.05 1.02 ...
##  $ direction.USD.EUR.dir: num  -1 -1 1 -1 1 1 1 -1 -1 1 ...
##  $ direction.USD.GBP.dir: num  -1 -1 1 -1 -1 1 1 -1 -1 1 ...
##  $ direction.USD.CNY.dir: num  -1 -1 1 1 1 1 1 -1 -1 -1 ...
##  $ direction.USD.JPY.dir: num  -1 1 1 1 1 1 1 -1 -1 -1 ...
```

```
# 2. Make an xts object with row
# names equal to the dates
exrates.xts <- na.omit(as.xts(values,
    dates))  #order.by=as.Date(dates, '%d/%m/%Y')))
str(exrates.xts)
```

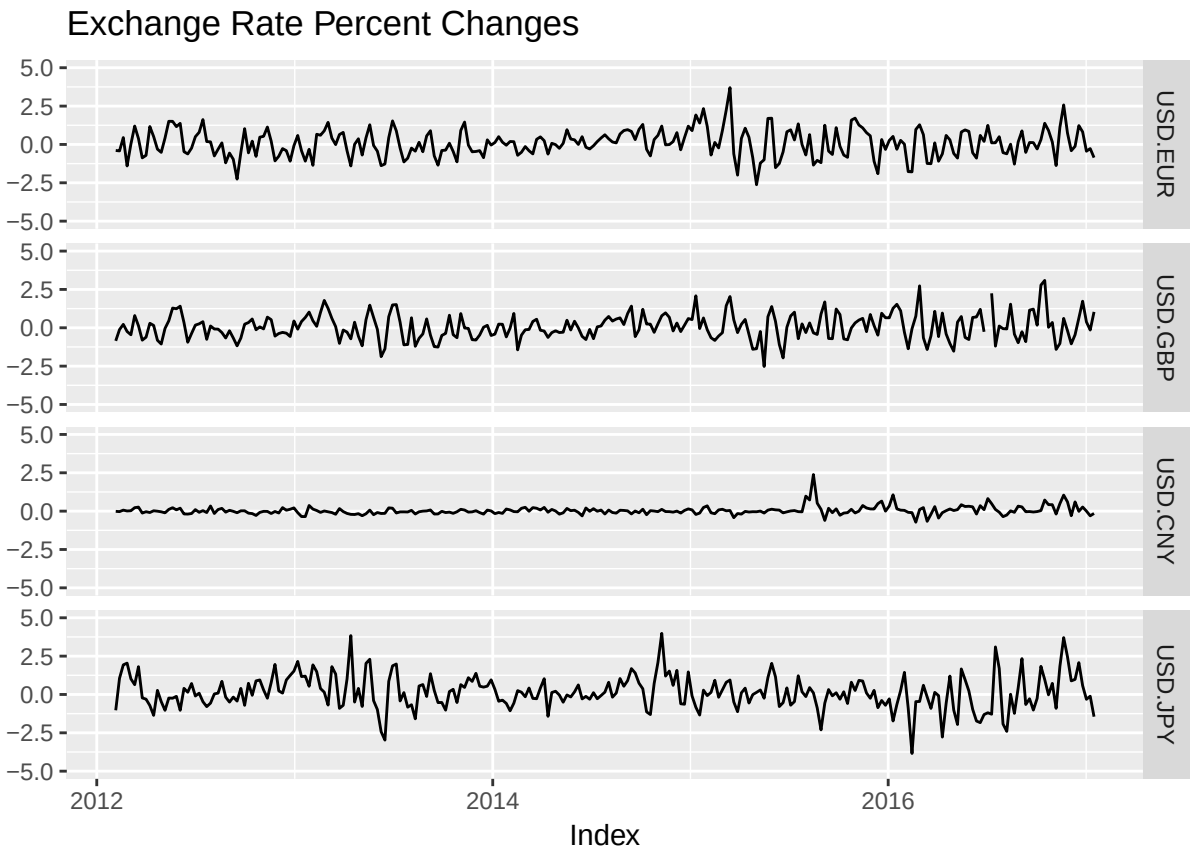
```
## An 'xts' object on 2012-02-05/2017-01-15 containing:
##   Data: num [1:259, 1:12] -0.3928 -0.4206 0.4476 -1.4013 0.0231 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:12] "USD.EUR" "USD.GBP" "USD.CNY" "USD.JPY" ...
##   Indexed by objects of class: [Date] TZ: UTC
##   xts Attributes:
##  NULL
```

```
exrates.zr <- as.zooreg(exrates.xts)
str(exrates.zr)
```

```
## 'zooreg' series from 2012-02-05 to 2017-01-15
##   Data: num [1:259, 1:12] -0.3928 -0.4206 0.4476 -1.4013 0.0231 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:12] "USD.EUR" "USD.GBP" "USD.CNY" "USD.JPY" ...
##   Index:  Date[1:259], format: "2012-02-05" "2012-02-12" "2012-02-19" "2012-02-26" ...
##   Frequency: 0.142857142857143
```

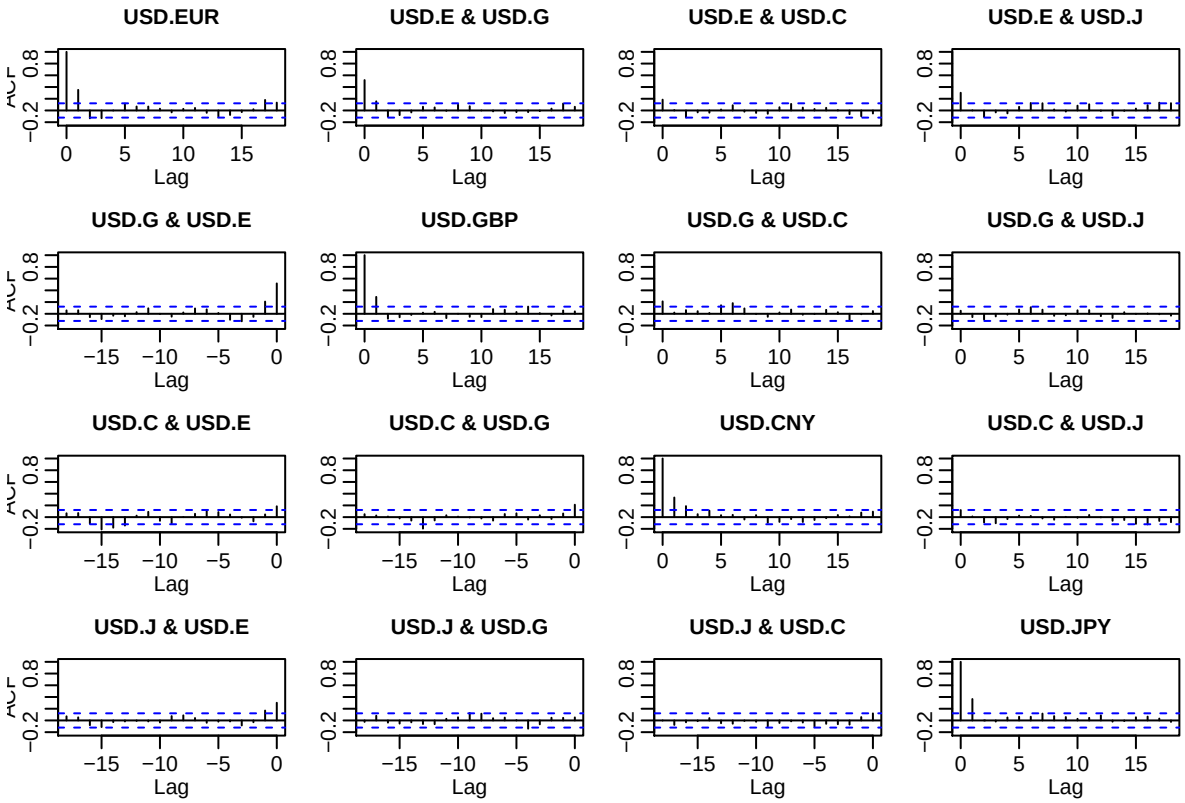
We can plot with the `ggplot2` package. In the `ggplot` statements we use `aes`, "aesthetics", to pick `x` (horizontal) and `y` (vertical) axes. Use `group =1` to ensure that all data is plotted. The added (`+`) `geom_line` is the geometrical method that builds the line plot.

```
require(ggplot2)
title.chg <- "Exchange Rate Percent Changes"
autoplot.zoo(exrates.xts[, 1:4]) + ggtitle(title.chg) +
    ylim(-5, 5)
```

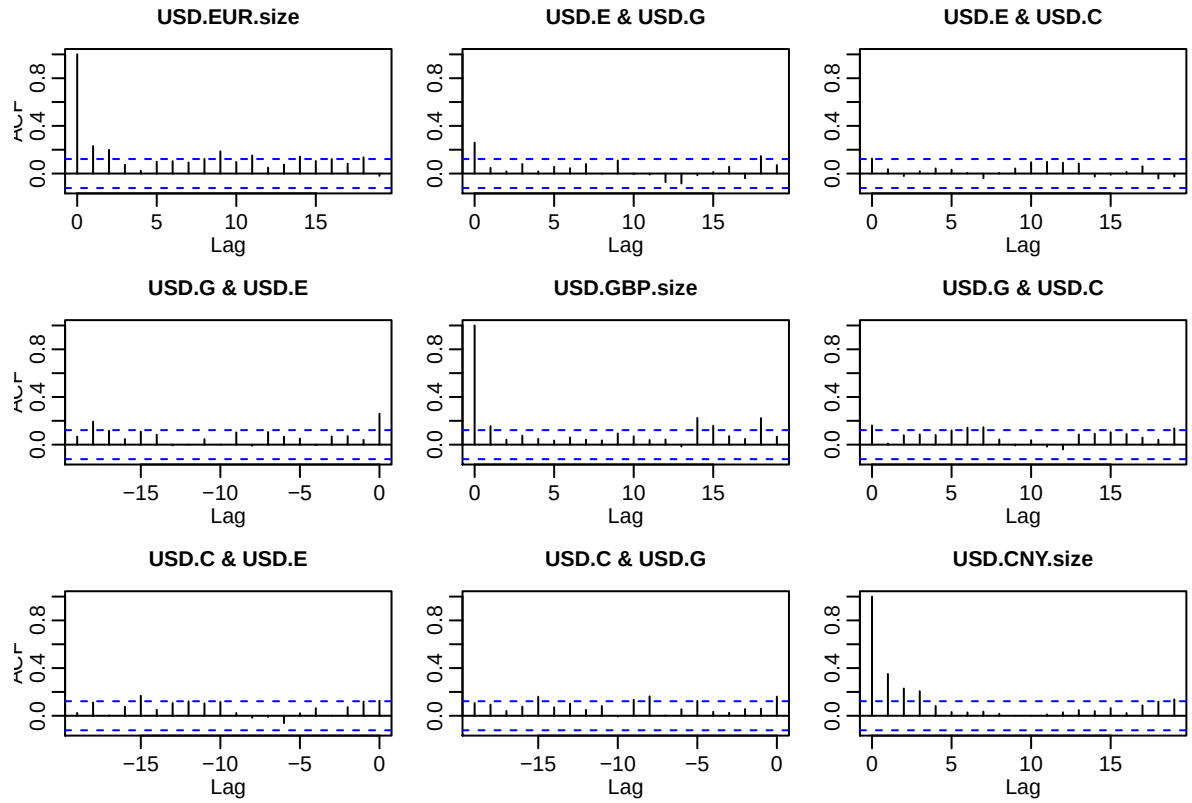



2. Let's dig deeper and compute mean, standard deviation, etc. Load the `data_moments()` function. Run the function using the `HO2.df$return` subset and write a `knitr::kable()` report.

```
acf(coredata(exrates.xts[, 1:4]))
```




```
acf(coredata(exrates.xts[, 5:7]))
```




```
# Load the data_moments() function
# data_moments function INPUTS: r
# vector OUTPUTS: list of scalars
# (mean, sd, median, skewness,
# kurtosis)
data_moments <- function(data) {
    require(moments)
    mean.r <- mean(data)
    sd.r <- sd(data)
    median.r <- median(data)
    skewness.r <- skewness(data)
    kurtosis.r <- kurtosis(data)
    result <- data.frame(mean = mean.r,
        std_dev = sd.r, median = median.r,
        skewness = skewness.r, kurtosis = kurtosis.r)
    return(result)
}
# Run data_moments()
answer <- data_moments(exrates.xts)
# Build pretty table
answer <- round(answer, 4)
knitr::kable(answer)
```

| | mean | std_dev | median | skewness | kurtosis |
|---|---|---|---|---|---|
| USD.EUR | 0.2424 | 0.8832 | 0.2447 | 0.1690 | 3.5901 |
| USD.GBP | 0.2424 | 0.8832 | 0.2447 | 1.6627 | 12.9297 |

| | mean | std_dev | median | skewness | kurtosis |
|---|---|---|---|---|---|
| USD.CNY | 0.2424 | 0.8832 | 0.2447 | 2.9623 | 23.2779 |
| USD.JPY | 0.2424 | 0.8832 | 0.2447 | 0.1906 | 4.3916 |
| USD.EUR.size | 0.2424 | 0.8832 | 0.2447 | 1.3773 | 6.3808 |
| USD.GBP.size | 0.2424 | 0.8832 | 0.2447 | 4.0555 | 34.3779 |
| USD.CNY.size | 0.2424 | 0.8832 | 0.2447 | 4.9157 | 41.4959 |
| USD.JPY.size | 0.2424 | 0.8832 | 0.2447 | 1.6373 | 6.3185 |
| USD.EUR.dir | 0.2424 | 0.8832 | 0.2447 | -0.1627 | 1.0265 |
| USD.GBP.dir | 0.2424 | 0.8832 | 0.2447 | 0.0232 | 1.0005 |
| USD.CNY.dir | 0.2424 | 0.8832 | 0.2447 | 0.0232 | 1.0005 |
| USD.JPY.dir | 0.2424 | 0.8832 | 0.2447 | -0.1471 | 1.0216 |

### Set B

We will use the data from Set A to investigate the interactions of the distribution of exchange rates.

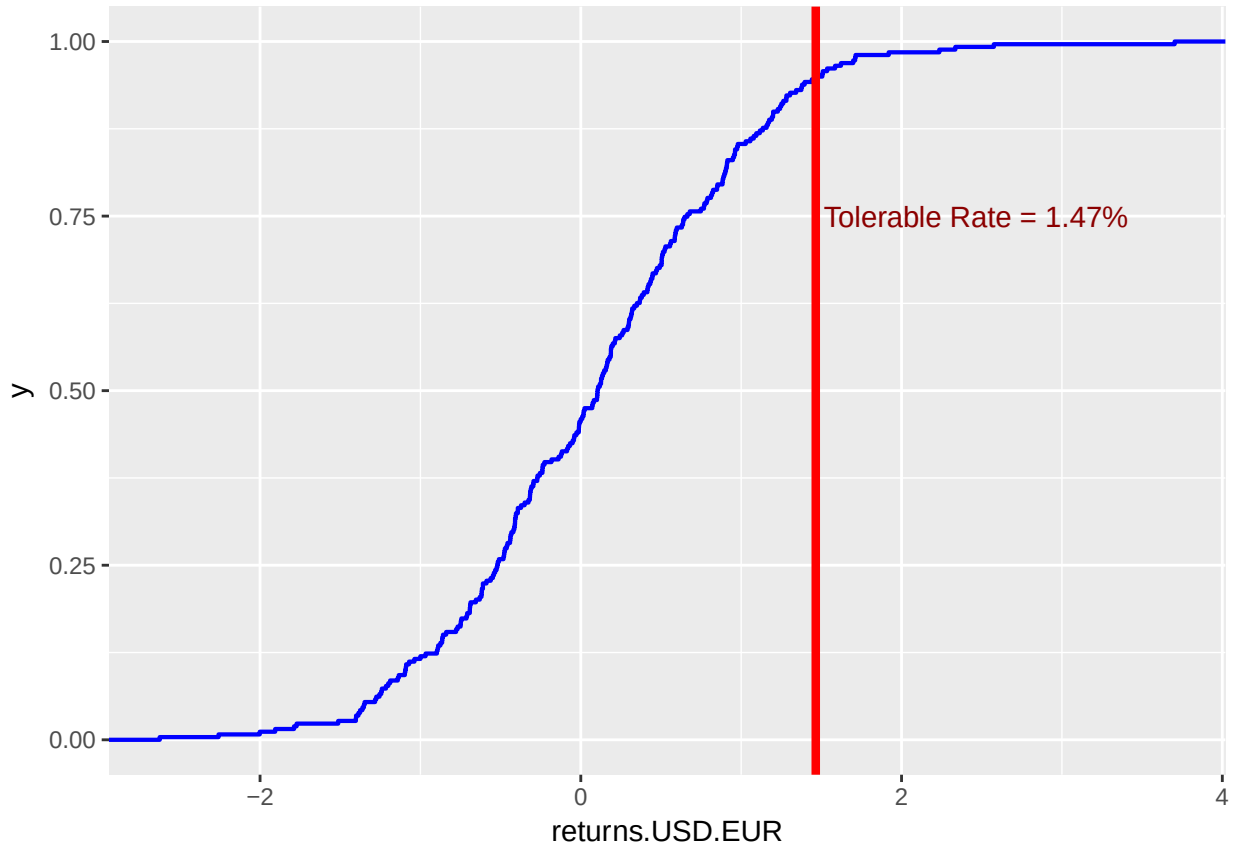
### Problem

We want to characterize the distribution of up and down movements visually. Also we would like to repeat the analysis periodically for inclusion in management reports.

### Questions

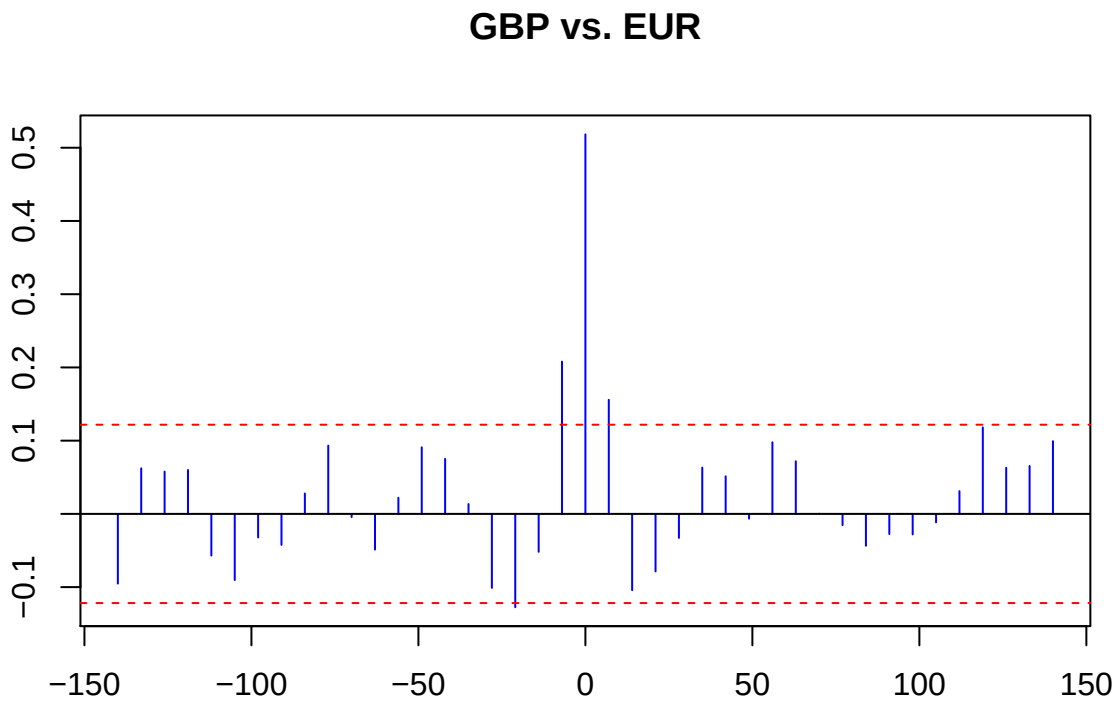
1. How can we show the shape of our exposure to euros, especially given our tolerance for risk? Suppose corporate policy set tolerance at 95%. Let's use the `exrates.df` data frame with `ggplot2` and the cumulative relative frequency function `stat_ecdf`.

```
exrates.tol.pct <- 0.95
exrates.tol <- quantile(exrates.df$returns.USD.EUR,
    exrates.tol.pct)
exrates.tol.label <- paste("Tolerable Rate = ",
    round(exrates.tol, 2), "%", sep = "")
ggplot(exrates.df, aes(returns.USD.EUR,
    fill = direction.USD.EUR.dir)) +
    stat_ecdf(colour = "blue", size = 0.75) +
    geom_vline(xintercept = exrates.tol,
        colour = "red", size = 1.5) +
    annotate("text", x = exrates.tol +
        1, y = 0.75, label = exrates.tol.label,
        colour = "darkred")
```

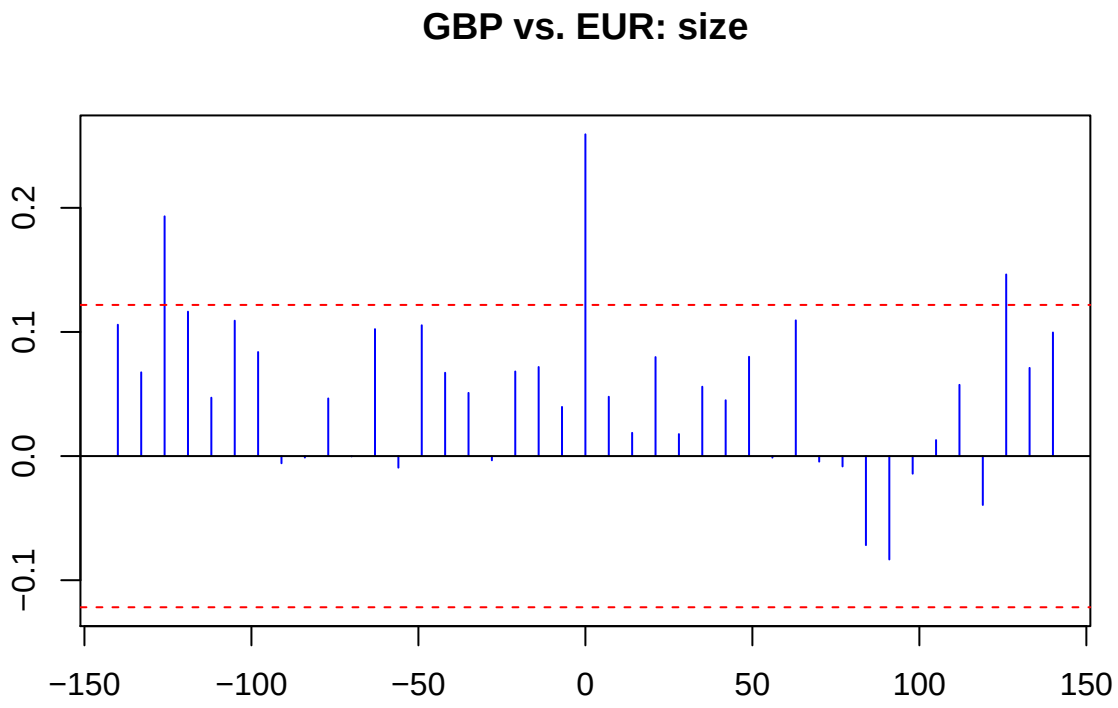



2. What is the history of correlations in the exchange rate markets? If this is a "history," then we have to manage the risk that conducting business in one country will definitely affect business in another. Further that bad things will be followed by more bad things more often than good things. We will create a rolling correlation function, `corr.rolling`, and embed this function into the `rollapply()` function (look this one up!).

```
ccf(exrates.zr[, 1], exrates.zr[, 2],
    main = "GBP vs. EUR", lag.max = 20,
    ylab = "", xlab = "", col = "blue",
    ci.col = "red")
```




```
ccf(abs(exrates.zr[, 1]), abs(exrates.zr[,
    2]), main = "GBP vs. EUR: size",
    lag.max = 20, ylab = "", xlab = "",
    col = "blue", ci.col = "red")
```




```
#' We see some small raw correlations across time with raw returns. More revealing, we see volatility o
#' \t
#' \t
corr.rolling <- function(x) {
    dim <- ncol(x)
    corr.r <- cor(x)[lower.tri(diag(dim),
        diag = FALSE)]
    return(corr.r)
}
ALL.r <- exrates.zr[, 1:4]
corr.returns <- rollapply(ALL.r, width = 250,
    corr.rolling, align = "right", by.column = FALSE)
head(corr.returns)
```

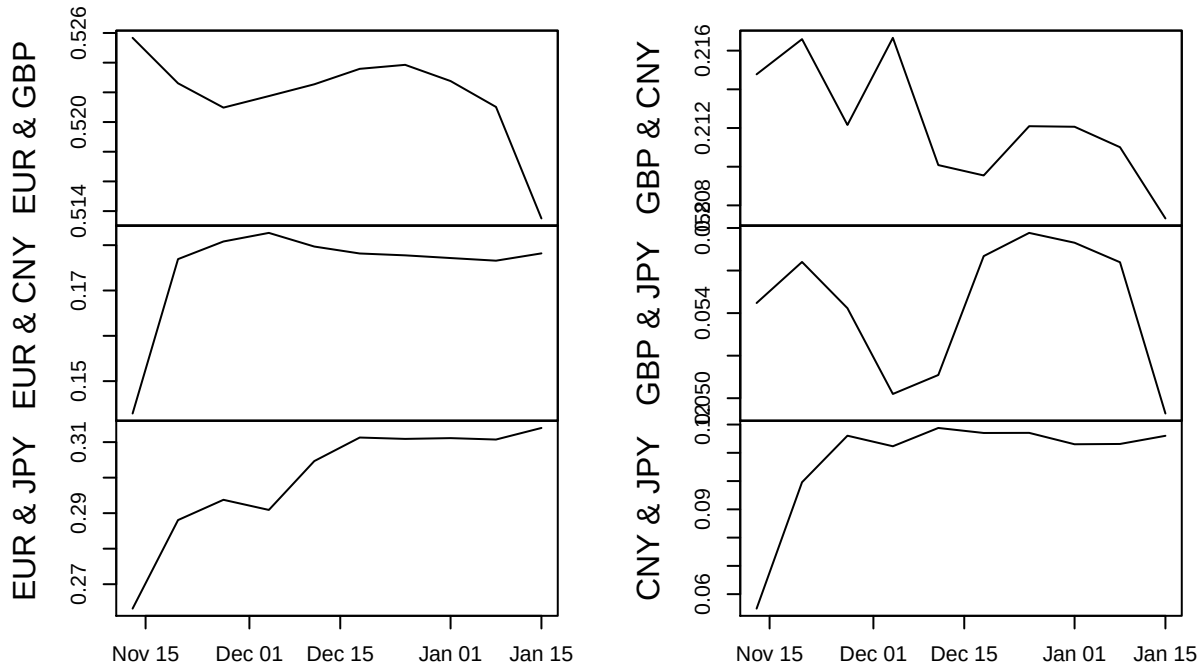
```
##
## 2016-11-13 0.5256724 0.1428576 0.2631368 0.2147757 0.05447472 0.05489404
## 2016-11-20 0.5226087 0.1769266 0.2880611 0.2165933 0.05640353 0.09959976
## 2016-11-27 0.5209718 0.1808128 0.2937643 0.2121587 0.05423030 0.11605636
## 2016-12-04 0.5217494 0.1827067 0.2909089 0.2166618 0.05020015 0.11232346
## 2016-12-11 0.5225435 0.1796925 0.3046657 0.2100831 0.05109035 0.11879143
## 2016-12-18 0.5235834 0.1781655 0.3112950 0.2095466 0.05667866 0.11701808
```

```
str(corr.returns)
```

```
## 'zooreg' series from 2016-11-13 to 2017-01-15
##   Data: num [1:10, 1:6] 0.526 0.523 0.521 0.522 0.523 ...
##   Index:  Date[1:10], format: "2016-11-13" "2016-11-20" "2016-11-27" "2016-12-04" ...
```

```
##   Frequency: 0.142857142857143
```

```
colnames(corr.returns) <- c("EUR & GBP",
    "EUR & CNY", "EUR & JPY", "GBP & CNY",
    "GBP & JPY", "CNY & JPY")
plot(corr.returns, xlab = "", main = "")
```




```
#' \t
```

4. How related are correlations and volatilities? Put another way, do we have to be concerned that inter-market transactions (e.g., customers and vendors transacting in more than one currency) can affect transactions in a single market? Let's take the `exrate` data to understand how dependent correlations and volatilities depend upon one another.

```
require(matrixStats)
R.corr <- apply.monthly(as.xts(ALL.r),
    FUN = cor)
str(R.corr)
```

```
## An 'xts' object on 2012-02-26/2017-01-15 containing:
##   Data: num [1:60, 1:16] 1 1 1 1 1 1 1 1 1 1 ...
##   Indexed by objects of class: [Date] TZ: UTC
##   xts Attributes:
##  NULL
```

```
head(ALL.r)
```

```
##                 USD.EUR    USD.GBP     USD.CNY    USD.JPY
## 2012-02-05 -0.39282058 -0.8523826 -0.01270912 -1.0301113
```

```
## 2012-02-12 -0.42063724 -0.1184583 -0.03130311  1.0571858
## 2012-02-19  0.44758304  0.2221127  0.06545522  1.9318720
## 2012-02-26 -1.40130272 -0.2407629  0.01048156  2.0452375
## 2012-03-04  0.02305476 -0.4546859  0.02588158  1.0197119
## 2012-03-11  1.19869144  0.7988383  0.22598055  0.6384155
```

```
head(R.corr)
```

```
##            [,1]      [,2]       [,3]        [,4]      [,5] [,6]       [,7]
## 2012-02-26    1 0.3779270  0.4926042 -0.08431313 0.3779270    1  0.6104888
## 2012-03-25    1 0.9718639  0.8787135  0.49720832 0.9718639    1  0.8796893
## 2012-04-29    1 0.9059433 -0.2923581 -0.57795431 0.9059433    1 -0.1252051
## 2012-05-27    1 0.6763763  0.9451004  0.89692382 0.6763763    1  0.8182297
## 2012-06-24    1 0.8708443  0.9960030 -0.88635953 0.8708443    1  0.8583950
## 2012-07-29    1 0.6310372 -0.2679904  0.25154628 0.6310372    1  0.3465303
##                  [,8]       [,9]      [,10] [,11]       [,12]       [,13]
## 2012-02-26  0.8769294  0.4926042  0.6104888     1  0.5351171 -0.08431313
## 2012-03-25  0.4101965  0.8787135  0.8796893     1  0.7796609  0.49720832
## 2012-04-29 -0.7075547 -0.2923581 -0.1252051     1 -0.1424010 -0.57795431
## 2012-05-27  0.8293420  0.9451004  0.8182297     1  0.8506507  0.89692382
## 2012-06-24 -0.7526198  0.9960030  0.8583950     1 -0.9238693 -0.88635953
## 2012-07-29  0.6477883 -0.2679904  0.3465303     1 -0.1533164  0.25154628
##                 [,14]      [,15] [,16]
## 2012-02-26  0.8769294  0.5351171     1
## 2012-03-25  0.4101965  0.7796609     1
## 2012-04-29 -0.7075547 -0.1424010     1
## 2012-05-27  0.8293420  0.8506507     1
## 2012-06-24 -0.7526198 -0.9238693     1
## 2012-07-29  0.6477883 -0.1533164     1
```

```
R.vols <- apply.monthly(ALL.r, FUN = colSds)  # from MatrixStats\t
head(R.corr, 3)
```

```
##            [,1]      [,2]       [,3]        [,4]      [,5] [,6]       [,7]
## 2012-02-26    1 0.3779270  0.4926042 -0.08431313 0.3779270    1  0.6104888
## 2012-03-25    1 0.9718639  0.8787135  0.49720832 0.9718639    1  0.8796893
## 2012-04-29    1 0.9059433 -0.2923581 -0.57795431 0.9059433    1 -0.1252051
##                  [,8]       [,9]      [,10] [,11]       [,12]       [,13]
## 2012-02-26  0.8769294  0.4926042  0.6104888     1  0.5351171 -0.08431313
## 2012-03-25  0.4101965  0.8787135  0.8796893     1  0.7796609  0.49720832
## 2012-04-29 -0.7075547 -0.2923581 -0.1252051     1 -0.1424010 -0.57795431
##                 [,14]      [,15] [,16]
## 2012-02-26  0.8769294  0.5351171     1
## 2012-03-25  0.4101965  0.7796609     1
## 2012-04-29 -0.7075547 -0.1424010     1
```

```
head(R.vols, 3)
```

```
##              USD.EUR   USD.GBP    USD.CNY   USD.JPY
## 2012-02-26 0.7559750 0.4483734 0.04195577 1.4242570
## 2012-03-25 0.8595039 0.7034406 0.18111483 0.8461026
## 2012-04-29 0.7891844 0.5999862 0.04331953 0.5956700
```

```
# Form correlation matrix for one
# month
R.corr.1 <- matrix(R.corr[1, ], nrow = 4,
    ncol = 4, byrow = FALSE)
```

```
rownames(R.corr.1) <- colnames(ALL.r[,
    1:4])
colnames(R.corr.1) <- rownames(R.corr.1)
head(R.corr.1)
```

```
##             USD.EUR   USD.GBP   USD.CNY     USD.JPY
## USD.EUR  1.00000000 0.3779270 0.4926042 -0.08431313
## USD.GBP  0.37792703 1.0000000 0.6104888  0.87692936
## USD.CNY  0.49260422 0.6104888 1.0000000  0.53511710
## USD.JPY -0.08431313 0.8769294 0.5351171  1.00000000
```

```
#
R.corr <- R.corr[, c(2, 3, 4, 7, 8, 12)]
head(R.corr)
```

```
##                 [,1]       [,2]        [,3]       [,4]       [,5]
## 2012-02-26 0.3779270  0.4926042 -0.08431313  0.6104888  0.8769294
## 2012-03-25 0.9718639  0.8787135  0.49720832  0.8796893  0.4101965
## 2012-04-29 0.9059433 -0.2923581 -0.57795431 -0.1252051 -0.7075547
## 2012-05-27 0.6763763  0.9451004  0.89692382  0.8182297  0.8293420
## 2012-06-24 0.8708443  0.9960030 -0.88635953  0.8583950 -0.7526198
## 2012-07-29 0.6310372 -0.2679904  0.25154628  0.3465303  0.6477883
##                  [,6]
## 2012-02-26  0.5351171
## 2012-03-25  0.7796609
## 2012-04-29 -0.1424010
## 2012-05-27  0.8506507
## 2012-06-24 -0.9238693
## 2012-07-29 -0.1533164
```

```
colnames(R.corr) <- colnames(corr.returns)
colnames(R.vols) <- c("EUR.vols", "GBP.vols",
    "CNY.vols", "JPY.vols")
head(R.corr, 3)
```

```
##            EUR & GBP  EUR & CNY   EUR & JPY  GBP & CNY  GBP & JPY
## 2012-02-26 0.3779270  0.4926042 -0.08431313  0.6104888  0.8769294
## 2012-03-25 0.9718639  0.8787135  0.49720832  0.8796893  0.4101965
## 2012-04-29 0.9059433 -0.2923581 -0.57795431 -0.1252051 -0.7075547
##             CNY & JPY
## 2012-02-26  0.5351171
## 2012-03-25  0.7796609
## 2012-04-29 -0.1424010
```

```
head(R.vols, 3)
```

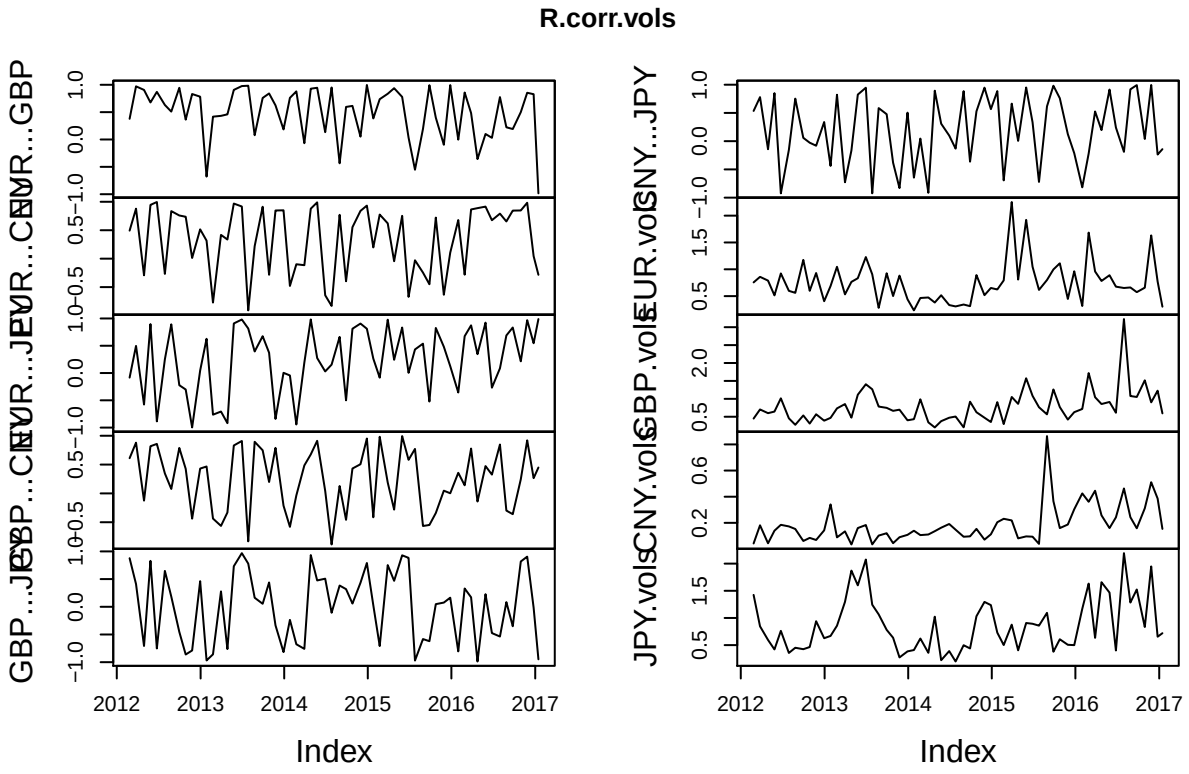
```
##             EUR.vols  GBP.vols   CNY.vols  JPY.vols
## 2012-02-26 0.7559750 0.4483734 0.04195577 1.4242570
## 2012-03-25 0.8595039 0.7034406 0.18111483 0.8461026
## 2012-04-29 0.7891844 0.5999862 0.04331953 0.5956700
```

```
R.corr.vols <- merge(R.corr, R.vols)
head(R.corr.vols)
```

```
##            EUR...GBP  EUR...CNY   EUR...JPY  GBP...CNY  GBP...JPY
## 2012-02-26 0.3779270  0.4926042 -0.08431313  0.6104888  0.8769294
## 2012-03-25 0.9718639  0.8787135  0.49720832  0.8796893  0.4101965
```

```
## 2012-04-29 0.9059433 -0.2923581 -0.57795431 -0.1252051 -0.7075547
## 2012-05-27 0.6763763  0.9451004  0.89692382  0.8182297  0.8293420
## 2012-06-24 0.8708443  0.9960030 -0.88635953  0.8583950 -0.7526198
## 2012-07-29 0.6310372 -0.2679904  0.25154628  0.3465303  0.6477883
##             CNY...JPY  EUR.vols  GBP.vols   CNY.vols  JPY.vols
## 2012-02-26  0.5351171 0.7559750 0.4483734 0.04195577 1.4242570
## 2012-03-25  0.7796609 0.8595039 0.7034406 0.18111483 0.8461026
## 2012-04-29 -0.1424010 0.7891844 0.5999862 0.04331953 0.5956700
## 2012-05-27  0.8506507 0.5140025 0.6435821 0.13962184 0.4205204
## 2012-06-24 -0.9238693 0.9228469 1.0141171 0.18477342 0.7627554
## 2012-07-29 -0.1533164 0.5985276 0.4552383 0.17257629 0.3560885
```

```
#'
plot.zoo(R.corr.vols)
```




```
#' \t
EUR.vols <- as.numeric(R.corr.vols[,
    "EUR.vols"])
GBP.vols <- as.numeric(R.vols[, "GBP.vols"])
CNY.vols <- as.numeric(R.vols[, "CNY.vols"])
length(EUR.vols)
```

```
## [1] 60
```

```
#' \t
#' Smooth data volatility
#'
fisher <- function(r) {
```

```
    0.5 * log((1 + r)/(1 - r))
}
rho.fisher <- matrix(fisher(as.numeric(R.corr.vols[,
    1:6])), nrow = length(EUR.vols),
    ncol = 6, byrow = FALSE)
#' \t
#' \t
#' ***\t
#' Here is the quantile regression part of the package.\t
#' \t
#' ## Notice\t
#' 1. We set `taus` as the quantiles of interest.\t
#' 2. We run the quantile regression using the `quantreg` package and a call to the `rq` function.\t
#' 3. We can overlay the quantile regression results onto the standard linear model regression.\t
#' 4. We can sensitize our analysis with the range of upper and lower bounds on the parameter estimates
#' \t
#'
require(quantreg)
taus <- seq(0.05, 0.95, 0.05)
fit.rq.EUR.GBP <- rq(rho.fisher[, 1] ~
    EUR.vols, tau = taus)
fit.lm.EUR.GBP <- lm(rho.fisher[, 1] ~
    EUR.vols)
#' \t
summary(fit.rq.EUR.GBP, se = "boot")
```

```
##
## Call: rq(formula = rho.fisher[, 1] ~ EUR.vols, tau = taus)
##
## tau: [1] 0.05
##
## Coefficients:
##             Value    Std. Error t value  Pr(>|t|)
## (Intercept) -1.30970  0.77376   -1.69264  0.09589
## EUR.vols     1.11618  0.60032    1.85931  0.06806
##
## Call: rq(formula = rho.fisher[, 1] ~ EUR.vols, tau = taus)
##
## tau: [1] 0.1
##
## Coefficients:
##             Value    Std. Error t value  Pr(>|t|)
## (Intercept) -0.76669  0.59570   -1.28704  0.20319
## EUR.vols     0.87530  0.53871    1.62481  0.10962
##
## Call: rq(formula = rho.fisher[, 1] ~ EUR.vols, tau = taus)
##
## tau: [1] 0.15
##
## Coefficients:
##             Value    Std. Error t value  Pr(>|t|)
## (Intercept) -0.41944  0.31988   -1.31123  0.19495
## EUR.vols     0.72127  0.29578    2.43848  0.01783
##
```

```
## Call: rq(formula = rho.fisher[, 1] ~ EUR.vols, tau = taus)
##
## tau: [1] 0.2
##
## Coefficients:
##             Value    Std. Error t value  Pr(>|t|)
## (Intercept) -0.34183  0.20375   -1.67769  0.09879
## EUR.vols     0.68684  0.16074    4.27308  0.00007
##
## Call: rq(formula = rho.fisher[, 1] ~ EUR.vols, tau = taus)
##
## tau: [1] 0.25
##
## Coefficients:
##             Value    Std. Error t value  Pr(>|t|)
## (Intercept) -0.20006  0.15355   -1.30285  0.19777
## EUR.vols     0.62395  0.16373    3.81086  0.00034
##
## Call: rq(formula = rho.fisher[, 1] ~ EUR.vols, tau = taus)
##
## tau: [1] 0.3
##
## Coefficients:
##             Value    Std. Error t value  Pr(>|t|)
## (Intercept) -0.16748  0.17361   -0.96469  0.33870
## EUR.vols     0.62911  0.22460    2.80105  0.00691
##
## Call: rq(formula = rho.fisher[, 1] ~ EUR.vols, tau = taus)
##
## tau: [1] 0.35
##
## Coefficients:
##             Value    Std. Error t value  Pr(>|t|)
## (Intercept) -0.05569  0.16850   -0.33052  0.74220
## EUR.vols     0.57094  0.20947    2.72559  0.00847
##
## Call: rq(formula = rho.fisher[, 1] ~ EUR.vols, tau = taus)
##
## tau: [1] 0.4
##
## Coefficients:
##             Value   Std. Error t value Pr(>|t|)
## (Intercept) 0.08166 0.21492    0.37997 0.70535
## EUR.vols    0.49946 0.27739    1.80059 0.07697
##
## Call: rq(formula = rho.fisher[, 1] ~ EUR.vols, tau = taus)
##
## tau: [1] 0.45
##
## Coefficients:
##             Value    Std. Error t value  Pr(>|t|)
## (Intercept) -0.11248  0.27306   -0.41192  0.68192
## EUR.vols     0.83120  0.30796    2.69902  0.00910
##
```

```
## Call: rq(formula = rho.fisher[, 1] ~ EUR.vols, tau = taus)
##
## tau: [1] 0.5
##
## Coefficients:
##             Value   Std. Error t value Pr(>|t|)
## (Intercept) 0.07989 0.33773    0.23655 0.81384
## EUR.vols    0.71696 0.39070    1.83508 0.07162
##
## Call: rq(formula = rho.fisher[, 1] ~ EUR.vols, tau = taus)
##
## tau: [1] 0.55
##
## Coefficients:
##             Value   Std. Error t value Pr(>|t|)
## (Intercept) 0.44312 0.36481    1.21466 0.22942
## EUR.vols    0.50126 0.40198    1.24699 0.21741
##
## Call: rq(formula = rho.fisher[, 1] ~ EUR.vols, tau = taus)
##
## tau: [1] 0.6
##
## Coefficients:
##             Value   Std. Error t value Pr(>|t|)
## (Intercept) 0.61565 0.43776    1.40638 0.16495
## EUR.vols    0.40224 0.53385    0.75348 0.45421
##
## Call: rq(formula = rho.fisher[, 1] ~ EUR.vols, tau = taus)
##
## tau: [1] 0.65
##
## Coefficients:
##             Value   Std. Error t value Pr(>|t|)
## (Intercept) 0.87137 0.43880    1.98578 0.05179
## EUR.vols    0.24695 0.52676    0.46880 0.64097
##
## Call: rq(formula = rho.fisher[, 1] ~ EUR.vols, tau = taus)
##
## tau: [1] 0.7
##
## Coefficients:
##             Value   Std. Error t value Pr(>|t|)
## (Intercept) 0.94211 0.44626    2.11111 0.03908
## EUR.vols    0.26943 0.57988    0.46462 0.64394
##
## Call: rq(formula = rho.fisher[, 1] ~ EUR.vols, tau = taus)
##
## tau: [1] 0.75
##
## Coefficients:
##             Value   Std. Error t value Pr(>|t|)
## (Intercept) 0.80393 0.47282    1.70030 0.09443
## EUR.vols    0.83269 0.60709    1.37162 0.17547
##
```

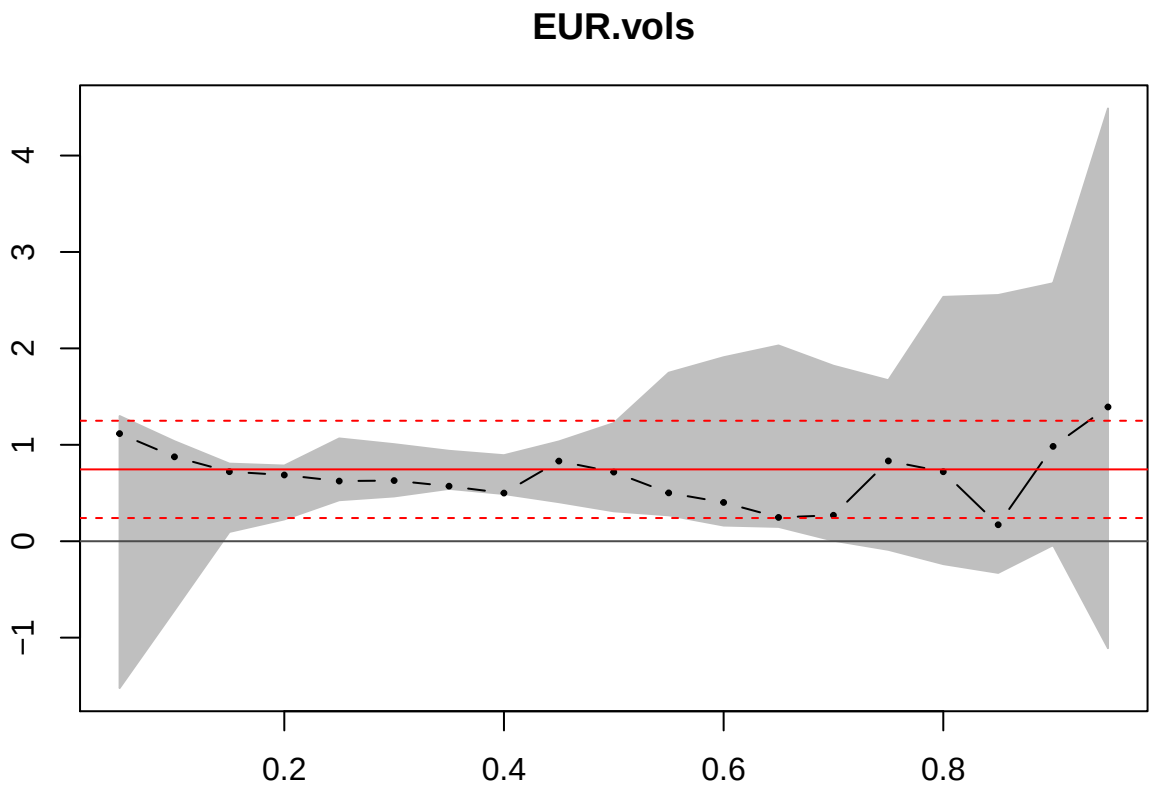
```
## Call: rq(formula = rho.fisher[, 1] ~ EUR.vols, tau = taus)
##
## tau: [1] 0.8
##
## Coefficients:
##             Value   Std. Error t value Pr(>|t|)
## (Intercept) 0.93575 0.49289    1.89849 0.06261
## EUR.vols    0.72058 0.63478    1.13516 0.26098
##
## Call: rq(formula = rho.fisher[, 1] ~ EUR.vols, tau = taus)
##
## tau: [1] 0.85
##
## Coefficients:
##             Value   Std. Error t value Pr(>|t|)
## (Intercept) 1.58199 0.49471    3.19779 0.00224
## EUR.vols    0.17092 0.72666    0.23522 0.81487
##
## Call: rq(formula = rho.fisher[, 1] ~ EUR.vols, tau = taus)
##
## tau: [1] 0.9
##
## Coefficients:
##             Value   Std. Error t value Pr(>|t|)
## (Intercept) 1.27929 0.58930    2.17085 0.03405
## EUR.vols    0.98377 0.83170    1.18283 0.24170
##
## Call: rq(formula = rho.fisher[, 1] ~ EUR.vols, tau = taus)
##
## tau: [1] 0.95
##
## Coefficients:
##             Value   Std. Error t value Pr(>|t|)
## (Intercept) 1.41097 1.02427    1.37754 0.17364
## EUR.vols    1.39234 1.09940    1.26645 0.21041
```

```
#'
summary(fit.lm.EUR.GBP, se = "boot")
```

```
##
## Call:
## lm(formula = rho.fisher[, 1] ~ EUR.vols)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.8826 -0.5731 -0.1813  0.5917  2.4763
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.2396     0.2613   0.917   0.3630
## EUR.vols      0.7450     0.3067   2.429   0.0183 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9209 on 58 degrees of freedom
```

```
## Multiple R-squared:  0.09233,    Adjusted R-squared:  0.07668
## F-statistic:   5.9 on 1 and 58 DF,  p-value: 0.01826
```

```
plot(summary(fit.rq.EUR.GBP), parm = "EUR.vols")
```



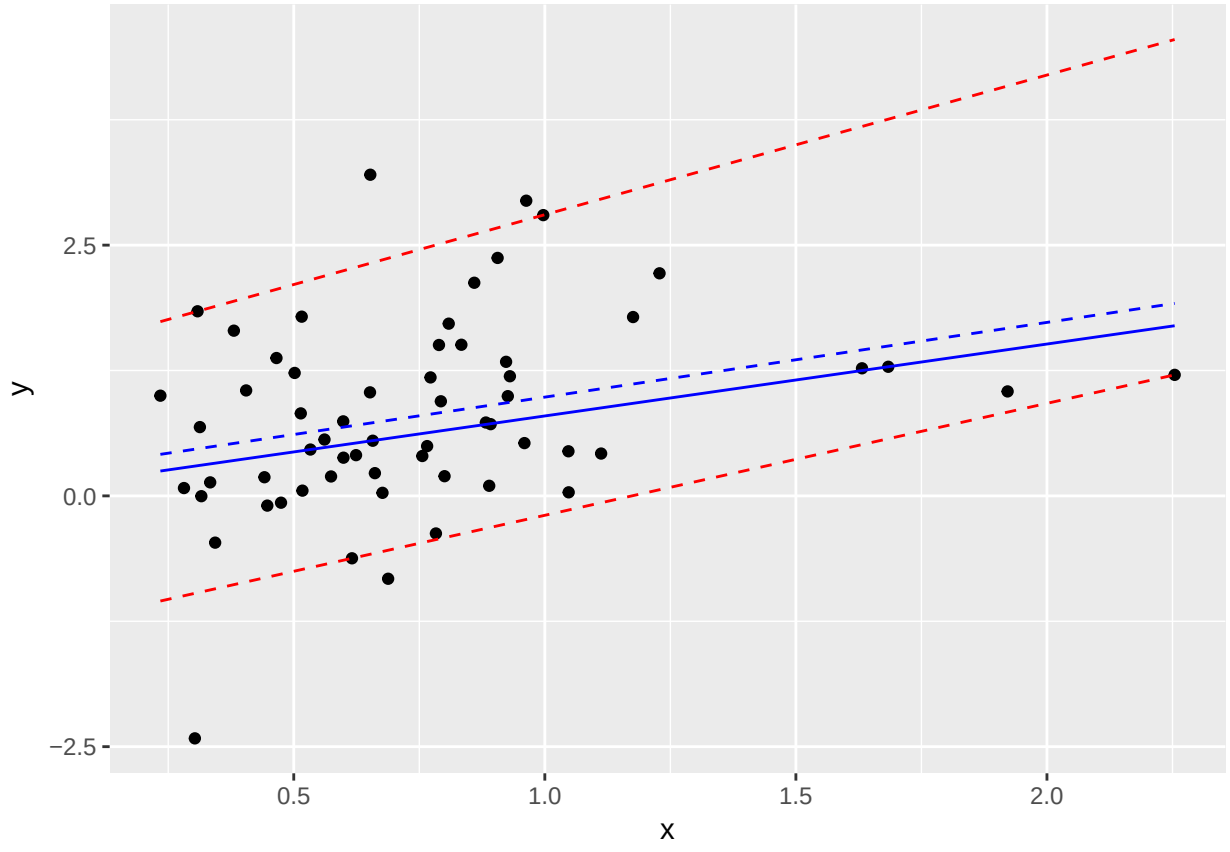

```
#' ***\t
#' Here we build the estimations and plot the upper and lower bounds.\t
#' \t
taus1 <- c(0.05, 0.5, 0.95)  # fit the confidence interval (CI)
EUR.GBP.p <- predict(rq(rho.fisher[,
    1] ~ EUR.vols, tau = taus1))
EUR.GBP.lm.p <- predict(lm(rho.fisher[,
    1] ~ EUR.vols))
colnames(EUR.GBP.p) <- c(paste("tau",
    taus1[1] * 100, sep = ""), paste("tau",
    taus1[2] * 100, sep = ""), paste("tau",
    taus1[3] * 100, sep = ""))
head(EUR.GBP.p)
```

```
##            tau5     tau50    tau95
## [1,] -0.4658975 0.6218969 2.463547
## [2,] -0.3503409 0.6961234 2.607695
## [3,] -0.4288300 0.6457068 2.509786
## [4,] -0.7359815 0.4484114 2.126639
## [5,] -0.2796389 0.7415380 2.695890
## [6,] -0.6416366 0.5090129 2.244327
```

```
EUR.GBP.CI <- data.frame(x = EUR.vols,
    y = rho.fisher[, 1], y.5 = EUR.GBP.p[,
        1], y.50 = EUR.GBP.p[, 2], y.95 = EUR.GBP.p[,
        3], y.lm <- EUR.GBP.lm.p)
head(EUR.GBP.CI)
```

```
##           x         y        y.5      y.50     y.95 y.lm....EUR.GBP.lm.p
## 1 0.7559750 0.3976391 -0.4658975 0.6218969 2.463547             0.8027790
## 2 0.8595039 2.1248414 -0.3503409 0.6961234 2.607695             0.8799036
## 3 0.7891844 1.5044171 -0.4288300 0.6457068 2.509786             0.8275186
## 4 0.5140025 0.8224042 -0.7359815 0.4484114 2.126639             0.6225199
## 5 0.9228469 1.3365632 -0.2796389 0.7415380 2.695890             0.9270914
## 6 0.5985276 0.7431378 -0.6416366 0.5090129 2.244327             0.6854875
```

```
ggplot(EUR.GBP.CI, aes(x, y)) + geom_point() +
    geom_line(aes(y = y.5), colour = "red",
        linetype = "dashed") + geom_line(aes(y = y.95),
    colour = "red", linetype = "dashed") +
    geom_line(aes(y = y.50), colour = "blue") +
    geom_line(aes(y = y.lm), colour = "blue",
        linetype = "dashed")
```




**Practice Set Debrief**

1. List the R skills needed to complete these practice sets.

2. What are the packages used to compute and graph results. Explain each of them.
3. How well did the results begin to answer the business questions posed at the beginning of each practice set?